FLEXLOOP: DEPTH-ELASTIC LOOPED POLICIES FOR ADAPTIVE TEST-TIME COMPUTATION IN DEEP RL

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped architectures scale computation by reusing the same parameters across recurrent steps, and recent work shows that they substantially improve deep reinforcement learning policies on long-horizon tasks. Since recurrent depth directly controls computation, one may expect looped policies to naturally support elastic inference across recurrent depths. Surprisingly, we find that pretrained looped policies exhibit severe recurrent-depth specialization: reliable decisions are concentrated near the full trained depth, tying deployment computation to this depth even when less computation may suffice. Achieving depth elasticity, i.e., reliable decisions across recurrent depths with adaptive computation at deployment, therefore remains a key challenge. To address this, we propose FlexLoop, a novel post-training framework that converts pretrained fixed-depth looped policies into depth-elastic policies. FlexLoop keeps training on the original RL objective to preserve full-depth capability while performing adjacent-depth policy distillation to progressively transfer decision quality from deeper to shallower recurrent steps. The resulting policy supports reliable inference across recurrent depths and enables state-wise adaptive inference through recurrent-depth consistency. Experiments on 30 online and offline long-horizon goal-conditioned environments show that FlexLoop preserves full-depth performance while making shallower depths effective. Keeping competitive performance, FlexLoop reduces average recurrent depth by up to $\mathbf{43\%}$ and achieves up to $\mathbf{1.34}\times$ wall-clock speedup in a stress test.

1 INTRODUCTION

Scaling model capacity has been a central driver of progress in modern machine learning, with larger models enabling increasingly capable behavior across language modeling (Kaplan et al., 2020; Wei et al., 2022), robotics (Zitkovich et al., 2023; Kim et al., 2024), and deep reinforcement learning (DRL) (Wang et al., 2025). Beyond increasing parameter count, a complementary line of work has explored looped architectures (Dehghani et al., 2019; Yang et al., 2024; Goyal et al., 2026), which repeatedly apply a shared computation block to increase effective computational depth while keeping the number of trainable parameters fixed. By scaling computation rather than model size, such architectures provide a parameter-efficient way to expand computational capacity.

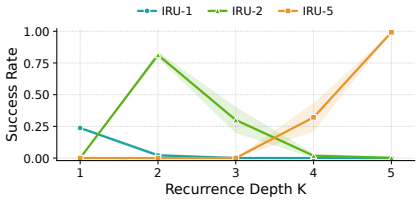


Figure 1: Fixed-depth training induces recurrent-depth specialization. Each IRU-K is trained at the K-th recurrent depth. Reliable decisions are concentrated near the full trained depth.

Recent work on looped recurrent policies, such as IRU (Ghugare et al., 2026), has shown that recurrent computation provides an effective scaling dimension for DRL, with additional recurrent steps substantially improving planning and decision making on long-horizon tasks without increasing policy parameter count. This naturally suggests recurrent depth as a deployment-time computation budget, analogous to recent looped language models that support budget-aware or adaptive inference (Jeddi et al., 2026; Popescu et al., 2026; Schwethelm et al., 2026). Ideally, such policies shall exhibit *depth elasticity*, namely, the ability of a single policy to maintain reliable decision quality across multiple recurrent depths and flexibly adjust its inference computation at deployment.

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.

Surprisingly, as shown in Figure 1, standard RL training does not yield such elasticity. Instead, looped policies trained with a fixed recurrent depth exhibit severe recurrent-depth specialization, with reliable decision-making concentrated at the full training depth and performance deteriorating sharply at shallower depths. This specialization ties deployment to the full recurrent depth. Although the full-depth policy itself is strong, always executing the maximum recurrent depth may use more computation than necessary for some decisions and limit adaptation to deployment resource constraints. This motivates a key question:

Can we preserve the full-depth capability of a looped RL policy while making its shallower recurrent depths reliable for decision making?

Addressing this question is nontrivial. Empirically, we find that looped architectures have sufficient capacity to support useful policies across multiple recurrent depths, yet naively training these depths jointly suppresses full-depth performance, while relying on distillation alone can degrade generalization to unseen tasks.

To resolve this dilemma, we propose FlexLoop, a novel post-training framework that converts a pretrained fixed-depth looped policy, which is reliable primarily at its training depth, into an elastic-depth policy that remains effective across recurrent depths. Starting from the pretrained policy, FlexLoop keeps training on the original RL objective to preserve full-depth performance and out-of-distribution generalization, while introducing adjacent-depth policy distillation to progressively propagate decision quality toward shallower recurrent steps. The resulting consistency across successive depths further provides a natural signal for state-wise adaptive inference, allowing computation to terminate once the policy has sufficiently stabilized.

Across 30 online and offline long-horizon goal-conditioned environments with discrete and continuous action spaces, FlexLoop preserves full-depth performance while making shallower recurrent depths reliable, improving the deployment-time computation–performance trade-off. With state-wise adaptive inference, FlexLoop automatically allocates recurrent computation according to state difficulty. It reduces the average recurrent depth by up to 43% and achieves up to $1.34\times$ wall-clock speedup in a large-scale stress test, while maintaining competitive or better task performance.

In summary, our main contributions are:

- We identify recurrent-depth specialization in existing looped RL policies: despite their capacity to operate across recurrent depths, fixed-depth training fails to induce depth elasticity, concentrating reliable decisions at the training depth and limiting adaptation to deployment computation constraints.
- We propose FlexLoop, a novel post-training framework that induces depth elasticity in pretrained looped RL policies. It keeps training on the original RL objective to preserve full-depth capability and performs adjacent-depth policy distillation to transfer decision quality from deeper to shallower recurrent depths, enabling state-wise adaptive inference at deployment without retraining.
- Across 30 online and offline long-horizon goal-conditioned environments, FlexLoop preserves full-depth performance while enabling reliable decisions at shallower recurrent depths. State-wise adaptive inference reduces average recurrent computation by up to 43% and achieves up to $1.34\times$ wall-clock speedup, while maintaining competitive or better task performance.

2 Related Work

Computation Scaling in DRL. Prior work has incorporated additional computation into DRL through planning and iterative value estimation (Tamar et al., 2016; Silver et al., 2017; Farquhar et al., 2018), while recent studies show that increasing network depth can substantially improve long-horizon decision making (Wang et al., 2025). More recently, Ghugare et al. (2026) showed that recurrent computation can scale RL capability without increasing policy parameter count. We build on this direction by investigating whether recurrent policies can remain effective across inference depths. We find that standard fixed-depth training instead leads to recurrent-depth specialization, highlighting the need for elastic recurrent policies.

Looped Architectures and Elastic Depth. Looped architectures, which reuse the same parameters across multiple computation steps, have been studied for iterative reasoning and parameter-efficient computation (Dehghani et al., 2019; Yang et al., 2024; Zhu et al., 2025; Du et al., 2026). Recent work extends looped models to variable inference budgets. For example, LoopFormer trains language models across different recurrent depths (Jeddi et al., 2026), while ELT uses intra-loop self-distillation to transfer behavior from high-compute to intermediate depths in visual generation (Goyal et al., 2026). Others study adaptive stopping and efficient depth-adaptive inference in looped language models (Popescu et al., 2026; Schwethelm et al., 2026). However, directly extending such depth elasticity to RL is challenging. Unlike settings with explicit targets, RL provides no ground-truth intermediate decisions, as policy quality is ultimately determined by long-horizon return rather than local output agreement. Achieving reliable depth elasticity for looped RL policies therefore remains underexplored.

3 PRELIMINARY

Goal-Conditioned RL. We consider a goal-conditioned Markov decision process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{G}, P, r, p_0, \gamma)$, where $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{G}$ denote the state, action, and goal spaces, respectively; $P(\cdot \mid s, a)$ is the transition kernel; $r(s, a, g)$ is the goal-conditioned reward; $p_0(s, g)$ is a joint distribution over initial states and goals; and $\gamma \in [0, 1)$ is the discount factor. At the beginning of each episode, $(s_0, g) \sim p_0$, and a goal-conditioned policy $\pi(a \mid s, g)$ induces trajectories according to $a_t \sim \pi(\cdot \mid s_t, g)$ and $s_{t+1} \sim P(\cdot \mid s_t, a_t)$. The objective is to maximize

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t, g)\right].$$

We consider both online and offline settings. In the latter, the agent learns from a fixed dataset $\mathcal{D}$ without further environment interaction.

Looped Policies and IRU. Looped architecture (Jeddi et al., 2026; Zhu et al., 2025; Du et al., 2026) increase computational depth by repeatedly applying a shared computation block, without introducing depth-specific parameters. Let x denote the network input and h_k the representation after k recurrent steps. We write a looped computation abstractly as

$$h_{k+1} = F_\theta(h_k, x),$$

where F_θ is the computation block whose parameters θ are shared across recurrent depth.

Recent work (Ghugare et al., 2026) introduced the Interpolation Recurrent Unit (IRU) as a minimal instantiation of this architecture and demonstrated that scaling its recurrent depth can substantially improve long-horizon decision making in DRL. We therefore focus primarily on IRU-based policies. Given recurrent state c_k, IRU computes

$$\begin{aligned} f_k &= \sigma(\mathrm{FORGET}_\theta([x, \tanh(c_k)])), \\ I_k &= \tanh(\mathrm{INPUT}_\theta([x, \tanh(c_k)])), \\ c_{k+1} &= f_k \odot c_k + (1 - f_k) \odot I_k. \end{aligned}$$

At each depth k, an output head produces $z_k = G_\theta(c_k)$, which may correspond to action values $Q_k(s, g, \cdot)$, a policy $\pi_k(\cdot \mid s, g)$, or a value estimate $V_k(s, g)$ depending on the RL algorithm. We denote the full recurrent depth by K. While standard IRU policies are optimized through z_K, we aim to make z_k reliable for $k < K$ while preserving full-depth capability.

4 METHOD: FLEXLOOP

As shown in Figure 2, we propose **FlexLoop**, a novel post-training framework for converting a depth-specialized looped RL policy into an elastic one. Starting from a pretrained loop policy, FlexLoop keeps training on the original RL training while introducing adjacent-depth policy self-distillation. The RL objective preserves the task capability acquired at full depth, while depth-wise distillation progressively makes shallower recurrent depths reliable for decision making, and induces a natural policy-consistency signal for state-wise adaptive-depth inference.

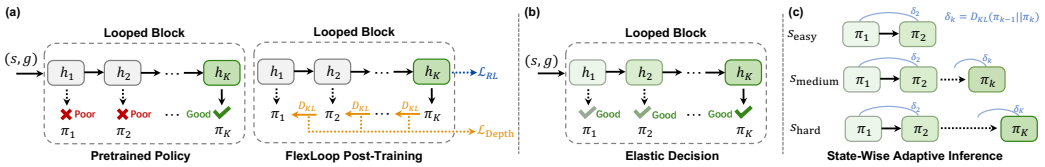


Figure 2: Framework of FlexLoop.**(a)** Starting from a depth-specialized looped policy, FlexLoop preserves full-depth capability with the original RL objective while propagating decision quality to shallower depths via adjacent-depth policy distillation. **(b)** The resulting policy supports reliable decisions across recurrent depths. **(c)** Adjacent-depth policy discrepancy serves as a stopping signal for state-wise adaptive inference, allocating more computation to harder states.

4.1 Elastic Post-Training

FlexLoop starts from a pretrained looped policy with full recurrent depth K, inheriting all pretrained model parameters. During post-training, it continues optimization with the original algorithm-specific RL objective $\mathcal{L}_{\mathrm{RL}}$. We denote the policy induced after k recurrent steps by $\pi_k(\cdot \mid s, g)$.

Full-Depth RL Objective. FlexLoop inherits all parameters from the pretrained depth-specialized model and continues to optimize its full-depth value functions and policy under the same algorithm-specific RL objective $\mathcal{L}_{\mathrm{RL}}$ as in pretraining. This retains the capability acquired at depth K and preserves the out-of-distribution generalization of the pretrained policy while shallower recurrent depths are elasticized.

Progressive Depth-Wise Policy Distillation. To make shallower recurrent depths directly executable, FlexLoop progressively transfers decision quality from deeper to shallower recurrent policies. Let $\pi_k(\cdot \mid s, g)$ denote the policy induced after k recurrent steps. We optimize

$$\mathcal{L}_{\mathrm{depth}} = \frac{1}{K-1} \sum_{k=1}^{K-1} D(\mathrm{sg}[\pi_{k+1}], \pi_k)\,, \tag{1}$$

where D is an algorithm-specific policy discrepancy and $\mathrm{sg}[\cdot]$ denotes stop-gradient. This progressive distillation propagates full-depth decision quality toward shallower recurrent steps, while explicitly controlling policy variation across adjacent depths. The resulting discrepancy between successive policies provides a natural signal for state-wise adaptive inference, allowing computation to stop once the policy has sufficiently stabilized.

The full FlexLoop objective is

$$\mathcal{L}_{\mathrm{FlexLoop}} = \mathcal{L}_{\mathrm{RL}} + \lambda_{\mathrm{depth}} \mathcal{L}_{\mathrm{depth}} \tag{2}$$

4.2 Instantiating FlexLoop across RL Algorithms

Following the setting of Ghugare et al. (2026), we instantiate FlexLoop on three representative DRL algorithms for goal-conditioned long-horizon tasks, spanning online and offline settings as well as discrete and continuous action spaces.

Goal-Conditioned DQN (GC-DQN). We first instantiate FlexLoop with a goal-conditioned variant of DQN (Mnih et al., 2015). Since a discrete-action policy is induced directly by its action values, we convert each depth-k value output into a soft policy

$$\pi_k^Q(a \mid s, g) = \frac{\exp(Q_k(s, g, a)/\tau)}{\sum_{a'} \exp(Q_k(s, g, a')/\tau)}, \tag{3}$$

where τ is a temperature parameter. We then apply progressive policy distillation between adjacent depths,

$$\mathcal{L}_{\mathrm{depth}}^{\mathrm{DQN}} = \frac{1}{K-1} \sum_{k=1}^{K-1} \mathbb{E}_{(s,g)\sim\mathcal{D}} \left[D_{\mathrm{KL}} \left(\mathrm{sg}[\pi_{k+1}^Q] \,\|\, \pi_k^Q \right) \right], \tag{4}$$

where $\mathcal{D}$ denotes the sampled transition batch from the replay buffer.

Goal-Conditioned PPO (GC-PPO). We next instantiate FlexLoop with a goal-conditioned variant of PPO (Schulman et al., 2017). We transfer decision quality across recurrent depth by distilling adjacent policies:

$$\mathcal{L}_{\text{depth}}^{\text{PPO}} = \frac{1}{K-1} \sum_{k=1}^{K-1} \mathbb{E}_{(s,g)\sim\mathcal{B}} \left[D_{\text{KL}} \left(\text{sg}[\pi_{k+1}(\cdot \mid s, g)] \,\|\, \pi_k(\cdot \mid s, g) \right) \right], \tag{5}$$

where $\mathcal{B}$ denotes the on-policy rollout batch.

Goal-Conditioned IQL (GC-IQL). We finally instantiate FlexLoop with a goal-conditioned variant of IQL (Kostrikov et al., 2022), where learning is restricted to a fixed offline dataset $\mathcal{D}$. We distill the Gaussian policy across adjacent recurrent steps,

$$\mathcal{L}_{\text{depth}}^{\text{IQL}} = \frac{1}{K-1} \sum_{k=1}^{K-1} \mathbb{E}_{(s,g)\sim\mathcal{D}} \left[\widetilde{D}_{\text{KL}} \left(\text{sg}[\pi_{k+1}(\cdot \mid s, g)] \,\|\, \pi_k(\cdot \mid s, g) \right) \right], \tag{6}$$

where $\widetilde{D}_{\text{KL}}$ denotes the normalized policy KL used for continuous-action distillation, avoiding the scale-inflation degeneracy of the raw KL objective.

4.3 State-Wise Adaptive Inference

Once a policy produces meaningful decisions at multiple recurrent depths, the remaining question is when additional computation is useful. We first observe that good endpoint performance alone does not, in general, guarantee good shallower decisions.

Proposition 4.1 (No Shallow-Depth Guarantee)**.** *For any MDP $\mathcal{M}$ with bounded returns, recurrent depth $K \geq 2$, and $\epsilon > 0$, there exists a shared-weight recurrent policy Π such that*

$$J(\Pi_K) \geq \sup_{\pi} J(\pi) - \epsilon, \qquad J(\Pi_k) \leq \inf_{\pi} J(\pi) + \epsilon, \quad \forall 1 \leq k < K,$$

where $\sup_\pi$ and $\inf_\pi$ are taken over all stationary goal-conditioned Markov policies.

Proposition 4.1 shows that strong full-depth performance alone does not imply useful intermediate decisions. The recurrent states may instead serve purely as internal computation. The proof is given in Appendix A.1.

Through $\mathcal{L}_{\text{depth}}$, FlexLoop explicitly controls how the policy evolves across recurrent depth. Define the adjacent-depth discrepancy

$$\epsilon_j = \sup_{s,g} D_{\text{TV}} \left(\pi_j(\cdot \mid s, g), \pi_{j+1}(\cdot \mid s, g) \right), \tag{7}$$

then we have the following proposition.

Proposition 4.2 (Shallow-Depth Performance Bound)**.** *Assume $\forall (s, a, g) \in \mathcal{S}\times\mathcal{A}\times\mathcal{G}, |r(s, a, g)| \leq R_{\max}$ and $\gamma < 1$. Define $\delta_{\max} = \max_{1\leq j<K} \sup_{(s,g)} D_{\text{KL}} \left(\pi_{j+1}(\cdot \mid s, g) \,\|\, \pi_j(\cdot \mid s, g) \right)$. Then, for any $1 \leq k < K$,*

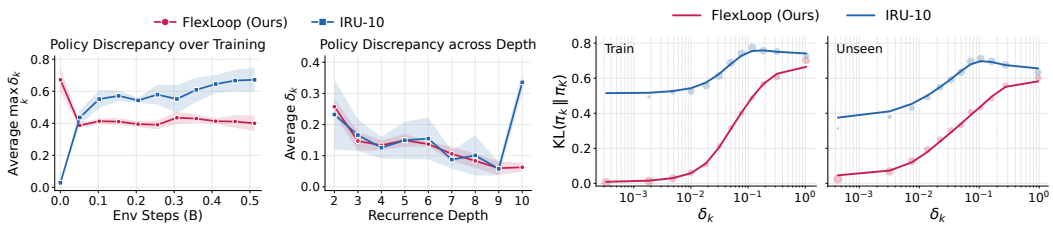
$$|J(\pi_k) - J(\pi_K)| \leq \frac{2R_{\max}}{(1-\gamma)^2}(K-k)\sqrt{\frac{\delta_{\max}}{2}}.$$

Proposition 4.2 shows that controlling the largest adjacent-depth policy discrepancy directly bounds the performance gap between any shallow policy and the full-depth policy. Thus, when FlexLoop keeps neighboring recurrent policies close, shallow-depth performance cannot drift arbitrarily far from the full-depth endpoint. The proof is given in Appendix A.2.

State-Wise Adaptive Inference. Motivated by Proposition 4.2, we monitor the depth-wise change in the policy output,

$$\delta_k(s, g) = D \left(\pi_{k-1}(\cdot \mid s, g), \pi_k(\cdot \mid s, g) \right), \quad k > 1 \tag{8}$$

where D is a policy-space divergence.



(a) Reduction of adjacent-depth policy discrepancy.

(b) Local-to-endpoint policy consistency.

Figure 3: (a) FlexLoop reduces adjacent-depth policy discrepancy during post-training and across depths. (b) Smaller δ_k corresponds to smaller $D_{\mathrm{KL}}(\pi_k \| \pi_K)$ on both training and unseen tasks. Marker size denotes the sample fraction in each pooled quantile bin.

Figure 3a shows that FlexLoop reduces adjacent-depth policy divergence during training and across recurrent depths, while Figure 3b shows that smaller local discrepancies are consistently associated with smaller divergence to the full-depth policy on both training and unseen tasks. This suggests that δ_k is informative of endpoint proximity, motivating its use as a practical stopping signal for state-wise adaptive inference. Accordingly, given a threshold ε, minimum depth $k_{\min}$, inference terminates at

$$k^\star(s, g) = \min\left(\{k > k_{\min} : \delta_k(s, g) < \varepsilon\} \cup \{K\}\right). \tag{9}$$

Varying ε yields different computation–performance trade-offs from a single trained policy.

5 Experiments

In this section, we evaluate FlexLoop across diverse online and offline goal-conditioned RL settings to answer three questions: (i) can it preserve full-depth capability while enabling reliable shallow-depth decisions? (ii) how does it trade off performance and inference compute? and (iii) how do training design choices and recurrent architectures affect depth elasticity?

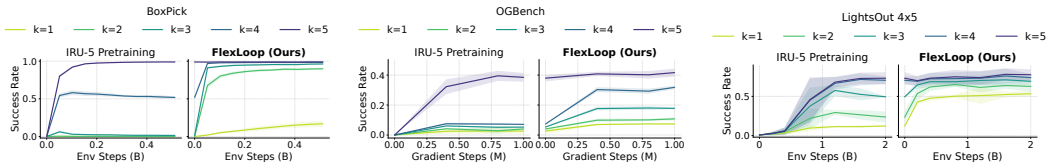
5.1 Experimental Setup

Benchmarks. Following Ghugare et al. (2026), we evaluate FlexLoop on 30 long-horizon goal-conditioned environments spanning online and offline RL, discrete and continuous action spaces, as well as in-distribution learning and out-of-distribution generalization. Specifically, we consider four environments from the BoxPick stitching benchmark (Bortkiewicz et al., 2025), where an agent moves boxes on 2-D maps to designated goal locations; one LightsOut environment (Anderson & Feil, 1998), formulated as a 4×5 binary switch puzzle; and 25 environments from OGBench (Park et al., 2025), covering three manipulation scenarios and two navigation scenarios. For BoxPick and LightsOut, we additionally evaluate generalization to unseen instances. Detailed environment descriptions are provided in Appendix B.1. We use goal-conditioned DQN for BoxPick, PPO for LightsOut, and goal-conditioned IQL for OGBench as the underlying RL algorithms.

Baselines and Implementation. We use IRU as the common looped-policy backbone and denote a pretrained depth-K IRU policy as IRU-K. Following the results reported by Ghugare et al. (2026), we choose $K = 5$ as the default setting, as it provides strong performance while maintaining a practical computational cost. FlexLoop is initialized from the corresponding IRU-5 checkpoint and post-trained for a fixed number of optimization steps to acquire depth elasticity. Our implementation builds on the codebase of Ghugare et al. (2026), retaining the same environment configurations, RL algorithms, and evaluation protocols unless otherwise stated. We report results over three random seeds. Additional implementation details and hyperparameters are provided in Appendix B.2.

5.2 Analysis of Learned Depth Elasticity

We evaluate FlexLoop on 30 long-horizon goal-conditioned environments using pretrained IRU-5 policies. Figure 4 tracks the evolution of the recurrent-depth performance during IRU pretraining



(a) Depth-wise decision performance on training tasks.



(b) Depth-wise decision performance on unseen tasks.

Figure 4: Depth-wise decision performance of pretrained IRU-5 and FlexLoop on training and unseen tasks. Solid lines denote mean success rates and shading indicates one standard deviation.

and FlexLoop post-training, showing the emergence of reliable shallow-depth policies and their generalization to unseen tasks in the BoxPick and LightsOut environments. Full environment-wise curves are provided in Appendix C.1. We further compare the depth-wise performance curves of the final pretrained and post-trained policies in Appendix C.2, providing a direct characterization of their depth elasticity. These results reveal several key observations:

FlexLoop improves shallow-depth decisions while preserving full-depth capability. Across almost all tasks in Figure 4a, pretrained IRU policies concentrate reliable decision performance at the full depth, whereas shallower recurrent depths remain ineffective. After post-training, FlexLoop maintains or further improves the full-depth performance while rapidly increasing the performance of shallow recurrent depths. This shows that FlexLoop successfully converts a depth-specialized policy into an elastic policy without sacrificing the capability acquired by the pretrained model.

FlexLoop generalizes depth elasticity beyond the training distribution. We evaluate the post-trained policies on unseen tasks to examine whether the learned depth elasticity generalizes beyond the training environments. As shown in Figure 4b, FlexLoop improves shallow-depth performance on unseen tasks while preserving the full-depth capability of the pretrained policy. This suggests that the improved shallow policies retain the general decision capability of the pretrained policy rather than simply overfitting to the training environments.

Shallower recurrent depths are more challenging to elasticize. In Figure 4, the improvement speed during post-training decreases monotonically with shallower recurrent depths. A possible explanation is that a reliable elastic policy needs to simultaneously support effective standalone decision making at shallow depths, maintain consistency across recurrent depths, and preserve robustness under distribution shifts. With fewer recurrent steps available, shallow policies have less computation capacity to refine intermediate representations and satisfy these requirements, making the shallowest depths harder to elasticize.

The underlying RL formulation influences how FlexLoop acquires depth elasticity. FlexLoop achieves the largest improvements on the GC-DQN-based BoxPick benchmark, while gains are smaller on PPO and offline IQL tasks. We hypothesize that this difference arises from the coupling between value learning and policy execution. In value-based methods, improved value estimation directly determines action selection across recurrent depths. In contrast, methods with separately parameterized policies, such as PPO and IQL, rely on indirect policy updates from value learning, making depth-wise adaptation more challenging.

5.3 State-wise Adaptive Inference

FlexLoop's learned depth elasticity enables adaptive recurrent computation during deployment. We first evaluate whether adaptive inference can reduce computation while preserving performance, and then analyze the computation-performance trade-offs induced by different stopping thresholds.

Adaptive Inference Performance. By tuning the stopping threshold ε in Equation (9), we select adaptive inference policies for each environment. Table 1 summarizes the resulting performance and average recurrent depth. FlexLoop maintains the performance of the full-depth policy while reducing the average recurrent depth by up to 43%, demonstrating that learned depth elasticity enables effective inference computation reduction without sacrificing performance. Full environment-wise results are provided in Appendix C.3.

Table 1: State-wise adaptive inference results across environments. $\pm$ indicates one standard deviation.

Environment	FlexLoop		IRU-5
	Avg. Depth	Succ.	Succ.
OGBench	4.19 ± 0.08	**0.410** ± 0.020	0.385 ± 0.029
BoxPick (Train)	4.50 ± 0.02	**0.989** ± 0.002	**0.989** ± 0.003
BoxPick (Unseen)	4.46 ± 0.02	**0.778** ± 0.055	0.760 ± 0.043
LightsOut-4x5 (Train)	2.84 ± 0.08	**0.747** ± 0.081	0.732 ± 0.076
LightsOut-4x5 (Unseen)	2.93 ± 0.21	**0.053** ± 0.031	0.013 ± 0.004

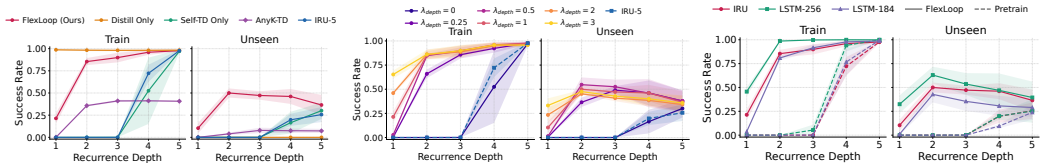
Computation-Performance Trade-off. We further analyze how the stopping threshold controls the trade-off between computation and performance. By varying ε, we obtain the Pareto curves (See Figure 11 in Appendix C.4). In general, smaller thresholds allocate deeper recurrent computation and tend to achieve higher performance, while larger thresholds reduce the average inference depth with controllable performance trade-offs. The resulting Pareto curves show that FlexLoop often matches the pretrained IRU policy performance with reduced average recurrent depth. In addition, this favorable trade-off is preserved on unseen tasks, demonstrating that FlexLoop's learned depth elasticity enables generalizable adaptive computation beyond the training distribution.

Wall-Clock Time Saving. To evaluate the practical inference speedup enabled by adaptive recurrent depth, we additionally train an IRU-10 policy and apply FlexLoop post-training to obtain the corresponding elastic variant. We evaluate both IRU-5 and IRU-10 fixed-depth policies and their FlexLoop variants on BoxPick-Exact-4 with a batch of $32,768$ parallel environments. As shown in Table 2, FlexLoop maintains competitive performance relative to the pretrained policies while using fewer recurrent steps, achieving up to $1.34\times$ wall-clock speedup compared with fixed-depth inference.

Table 2: Wall-clock comparison on BoxPick-Exact-4. $\pm$ indicates one standard deviation. Bold numbers indicate the shortest wall-clock latency.

	Model	Success	Avg. Depth	Wall-clock (ms)
Train	IRU-5	0.995 ± 0.001	5	170.1 ± 17.2
	FlexLoop	0.993 ± 0.001	3.85 ± 0.04	**133.0** ± 7.7
	IRU-10	0.997 ± 0.000	10	333.2 ± 36.5
	FlexLoop	0.996 ± 0.001	9.35 ± 0.04	**319.9** ± 22.2
Unseen	IRU-5	0.511 ± 0.085	5	170.1 ± 17.2
	FlexLoop	0.543 ± 0.093	3.80 ± 0.05	**127.4** ± 6.3
	IRU-10	0.664 ± 0.080	10	333.2 ± 36.5
	FlexLoop	0.675 ± 0.066	9.37 ± 0.04	**321.1** ± 22.7

5.4 Ablation Study

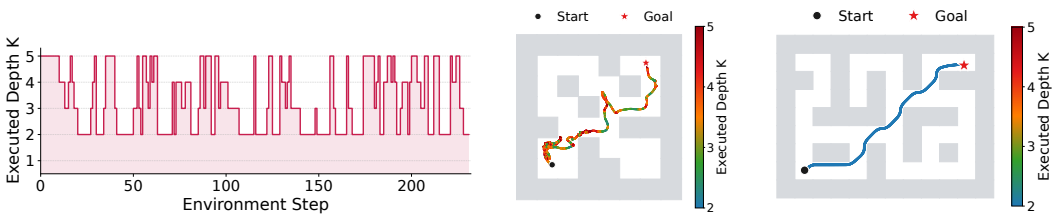


(a) Component Ablation. (b) Sensitivity to λ_{depth}. (c) Architectural Generality.

Figure 5: Ablation studies of FlexLoop. Solid lines denote mean performance, while shaded regions indicate one standard deviation.

Based on BoxPick-Exact-4, we study FlexLoop's key components, λ_{depth} sensitivity, and architectural generality. Additional results on from-scratch training and pretraining recurrent depth are provided in Appendix D.

Contribution of Key Components. We ablate the key components of FlexLoop, including *Distill Only*, which performs depth-wise distillation from the pretrained full-depth policy, *Self-TD Only*, which solely keeps the original RL training, and *AnyK-TD*, which jointly optimizes all recurrent depths with TD learning. As shown in Figure 5a, *Self-TD Only* preserves the pretrained policy's generalization but provides limited shallow-depth improvement, showing that depth elasticity does



(a) Scene. (b) AntMaze. (c) PointMaze.

Figure 6: State-wise adaptive recurrent-depth allocation across OGBench tasks. FlexLoop allocates more recurrent computation to states requiring more complex decisions.

not emerge solely from the original RL objective. Meanwhile, *AnyK-TD* and *Distill Only* each suffer from limitations in full-depth capability or unseen-task generalization. These highlight the distinct roles of the two objectives. Continued RL optimization preserves full-depth capability and generalization, while depth-wise policy distillation transfers decision quality to shallower depths.

Sensitivity to λ_{depth}. We further study the effect of the depth distillation weight λ_{depth}. As shown in Figure 5b, FlexLoop remains effective across a broad range of λ_{depth} values. Increasing λ_{depth} strengthens depth-wise distillation and consistently improves shallow-depth performance on training tasks. However, excessive distillation can hurt generalization to unseen tasks, where a moderate λ_{depth} achieves a better balance between shallow-depth adaptation and generalization.

Generality across Architectures. We further apply FlexLoop to LSTM-based policies (Hochreiter & Schmidhuber, 1997) to evaluate its architectural generality. LSTM-256 and LSTM-184 denote backbones with hidden sizes of 256 and 184, respectively, where LSTM-184 has a comparable parameter count to IRU. As shown in Figure 5c, both LSTM variants exhibit depth specialization during pretraining, while FlexLoop consistently improves shallow-depth performance and preserves full-depth capability across architectures. Moreover, LSTM-256 benefits more from FlexLoop than LSTM-184, suggesting that larger capacity allows a policy to acquire stronger depth elasticity.

5.5 Visualization of Adaptive Inference

To qualitatively understand how FlexLoop utilizes its learned depth elasticity, we visualize the recurrent computation allocated during successful trajectories on three OGBench tasks. Figure 6 shows the recurrent depth selected by FlexLoop at each environment step, with representative trajectory segments visualized frame by frame in Appendix C.5. The allocated computation appears to track state-level decision difficulty. In Scene, FlexLoop increases recurrent depth at skill boundaries, such as goal switching, while using only three or two recurrent steps during simpler execution phases such as button pressing. Similarly, deeper computation is concentrated near turns and goals in AntMaze, whereas straight navigation uses shallower inference. On the simpler PointMaze task, FlexLoop consistently terminates after two recurrent steps. These trajectories suggest that FlexLoop reallocates recurrent computation toward challenging decisions and terminates earlier once adjacent-depth policies have stabilized, rather than uniformly reducing computation.

6 Conclusion and Future Directions

In this work, we identify recurrent-depth specialization in pretrained looped RL policies and propose FlexLoop, a post-training framework that induces depth elasticity through continued RL optimization and adjacent-depth policy distillation. Across 30 online and offline long-horizon goal-conditioned environments, FlexLoop preserves full-depth capability while enabling reliable shallower-depth inference, reducing average recurrent computation by up to 43% and achieving up to $1.34\times$ wall-clock speedup, while maintaining competitive or better task performance. Future work includes extending depth elasticity beyond the trained depth, developing learning-based adaptive inference mechanisms beyond the current depth-consistency criterion, and studying depth elasticity in larger-scale looped architectures such as transformer-based models.

AI USE STATEMENT

In this work, we used generative AI tools for providing feedback on research methodology and experiments, formulating mathematical claims, assisting in the writing of proofs, and assisting with translation. We have not used generative AI tools for helping develop theoretical models or conceptual frameworks, providing critical ingredients for proving mathematical claims, proposing or refining hypotheses, designing research methodology or experiments, and supporting qualitative and thematic data analysis, interpreting results, and generating synthetic data sets, and cleaning and reformatting dataset are not applicable to this work. Additionally, we used generative AI tools for creating or editing software code, drafting parts of a research paper, summarizing existing literature, and identifying relevant literature. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness by 2 authors. LLM-generated text and LLM-suggested references were also checked manually. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Marlow Anderson and Todd Feil. Turning lights out with linear algebra. *Mathematics magazine*, 71 (4):300–303, 1998.

Michał Bortkiewicz, Władysław Pałucki, Mateusz Ostaszewski, and Benjamin Eysenbach. Is temporal difference learning the gold standard for stitching in rl?, 2025. URL https://arxiv.org/abs/2510.21995.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Hanwen Du, Yuxin Dong, and Xia Ning. Latent thinking optimization: Your latent reasoning language model secretly encodes reward signals in its latent thoughts. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 101000–101032, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/a4b1a25e1b37a17733c1fea50d0b7446-Paper-Conference.pdf.

Gregory Farquhar, Tim Rocktaeschel, Maximilian Igl, and Shimon Whiteson. TreeQN and ATreec: Differentiable tree planning for deep reinforcement learning. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1dh6Ax0Z.

Raj Ghugare, Michał Bortkiewicz, Alicja Ziarko, and Benjamin Eysenbach. On the role of computation in reinforcement learning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=HKzMnBX5B2.

Sahil Goyal, Swayam Agrawal, Gautham Govind Anil, Prateek Jain, Sujoy Paul, and Aditya Kusupati. Elt: Elastic looped transformers for visual generation. In Paolo Favaro, Zuzana Kukelova, Atsuto Maki, Anna Rohrbach, Konrad Schindler, and Federico Tombari (eds.), *Computer Vision – ECCV 2026*, pp. 308–326, Cham, 2026. Springer Nature Switzerland. ISBN 978-3-032-37324-3.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RzYXb5YWBs.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pp. 267–274, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1558608737.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *8th Annual Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=ZMnD6QZAE6.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=68n2s9ZJWF8.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 94937–94982, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/ecd92623ac899357312aaa8915853699-Paper-Conference.pdf.

Andrei Cristian Popescu, Haitz Sáez de Ocáriz Borde, and Pietro Liò. Adaptive depth in looped transformers: Diagnosing learned halting gates and trajectory readouts, 2026. URL https://arxiv.org/abs/2607.20519.

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

Kristian Schwethelm, Daniel Rueckert, and Georgios Kaissis. Depth-adaptive inference of looped language models via continuous depth batching, 2026. URL https://arxiv.org/abs/2608.09444.

David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, and Thomas Degris. The predictron: End-to-end learning and planning. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3191–3199. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/silver17a.html.

Aviv Tamar, YI WU, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/c21002f464c5fc5bee3b98ced83963b8-Paper.pdf.

Kevin Wang, Ishaan Javali, MichałBortkiewicz, Tomasz Trzcinski, and Benjamin Eysenbach. 1000 layer networks for self-supervised rl: Scaling depth can enable new goal-reaching capabilities. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 157643–157670. Curran Associates, Inc., 2025. doi: 10.52202/085713-5268. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/e74ee34cc0f2d0780f34ee77d8fba25b-Paper-Conference.pdf.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=yzkSU5zdwD. Survey Certification.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 42195–42214, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/b8402301e7f06bdc97a31bfaa653dc32-Paper-Conference.pdf.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling latent reasoning via looped language models, 2025. URL https://arxiv.org/abs/2510.25741.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R Sanketi, Grecia Salazar, Michael S Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J Joshi, Alex Irpan, brian ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *7th Annual Conference on Robot Learning*, 2023. URL https://openreview.net/forum?id=XMQgwiJ7KSX.

Technical Appendices

The organization of the appendix is as follows:

- Appendix A: We provide the proofs of Propositions 4.1 and 4.2.
- Appendix B: We provide details of the environments (B.1) and implementation (B.2).
- Appendix C: We provide extended results, including training curves (C.1), elasticity curves (C.2), state-wise adaptive inference results (C.3), Pareto curves (C.4), and visualized trajectory segments (C.5).
- Appendix C: We provide additional ablations of FlexLoop on training from scratch (D.1) and different pretraining recurrent depth (D.2).

A Proofs of the Propositions

A.1 Proof of Proposition 4.1

Proof. Since the return is bounded, $\sup_\pi J(\pi)$ and $\inf_\pi J(\pi)$ are both finite, where $\sup_\pi$ and $\inf_\pi$ are taken over all stationary goal-conditioned Markov policies. Hence, for any $\epsilon > 0$ there exist policies π^+ and π^- such that

$$J(\pi^+) \geq \sup_\pi J(\pi) - \epsilon, \qquad J(\pi^-) \leq \inf_\pi J(\pi) + \epsilon. \tag{10}$$

We construct a shared-weight recurrent policy $\Pi = (F, G)$, where F denotes the shared recurrent block and G the policy output head. Let the recurrent state contain a scalar h_k, initialized as $h_0 = 0$, and define

$$h_{k+1} = F(h_k, s, g) = h_k + 1, \tag{11}$$

where F simply ignores (s, g), yielding $h_k = k$ at recurrent depth k.

Define the shared policy output head as

$$G(\cdot \mid s, g, h) = \begin{cases} \pi^-(\cdot \mid s, g), & h < K, \\ \pi^+(\cdot \mid s, g), & h \geq K. \end{cases} \tag{12}$$

The depth-k policy induced by Π is

$$\Pi_k(\cdot \mid s, g) := G(\cdot \mid s, g, h_k). \tag{13}$$

Therefore, the policy obtained after k recurrent steps is

$$\Pi_k = \pi^-, \qquad \forall k < K,$$

while at the full recurrent depth,

$$\Pi_K = \pi^+.$$

Using Equation (10),

$$J(\Pi_K) = J(\pi^+) \geq \sup_\pi J(\pi) - \epsilon,$$

and, for every $k < K$,

$$J(\Pi_k) = J(\pi^-) \leq \inf_\pi J(\pi) + \epsilon.$$

Thus, a shared-weight recurrent policy can be arbitrarily close to optimal at its full depth while all of its shallower policies are arbitrarily close to the minimum achievable return. □

A.2 Proof of Proposition 4.2

Proof. For analysis, we use the equivalent discounted infinite-horizon formulation with a zero-reward absorbing terminal state, augmenting the state with time when needed, and write $x = (s, g)$ for compactness. Define the worst-case total-variation discrepancy between adjacent depths as

$$\alpha_j = \sup_x D_{\mathrm{TV}}\left(\pi_j(\cdot \mid x), \pi_{j+1}(\cdot \mid x)\right), \quad 1 \leq j < K.$$

By Pinsker's inequality,

$$\alpha_j \le \sup_x \sqrt{\frac{1}{2} D_{\mathrm{KL}}\left(\pi_{j+1}(\cdot \mid x) \,\|\, \pi_j(\cdot \mid x)\right)} \le \sqrt{\frac{\delta_{\max}}{2}}.$$

For any $1 \le k < K$, the triangle inequality across recurrent depths gives

$$D_{\mathrm{TV}}\left(\pi_k(\cdot \mid x), \pi_K(\cdot \mid x)\right) \le \sum_{j=k}^{K-1} \alpha_j \le (K-k)\sqrt{\frac{\delta_{\max}}{2}}. \tag{14}$$

We next relate this policy discrepancy to return. By the performance difference lemma (Kakade & Langford, 2002), for any two policies π and π',

$$J(\pi) - J(\pi') = \frac{1}{1-\gamma} \mathbb{E}_{x \sim d_\pi}\left[\mathbb{E}_{a \sim \pi(\cdot|x)} A^{\pi'}(x,a)\right], \tag{15}$$

where d_π denotes the normalized discounted state–goal occupancy measure under π, and $A^{\pi'}$ is the advantage of π'.

Taking $\pi = \pi_k$ and $\pi' = \pi_K$, and using $\mathbb{E}_{a \sim \pi_K(\cdot|x)} A^{\pi_K}(x,a) = 0$, we have

$$\begin{aligned}\left|\mathbb{E}_{a \sim \pi_k(\cdot|x)} A^{\pi_K}(x,a)\right| &= \left|\mathbb{E}_{a \sim \pi_k(\cdot|x)} A^{\pi_K}(x,a) - \mathbb{E}_{a \sim \pi_K(\cdot|x)} A^{\pi_K}(x,a)\right| \\ &= \left|\mathbb{E}_{a \sim \pi_k(\cdot|x)} Q^{\pi_K}(x,a) - \mathbb{E}_{a \sim \pi_K(\cdot|x)} Q^{\pi_K}(x,a)\right|.\end{aligned}$$

Since $|r| \le R_{\max}$, $\gamma < 1$,

$$|Q^{\pi_K}(x,a)| \le \frac{R_{\max}}{1-\gamma},$$

and hence

$$\left|\mathbb{E}_{a \sim \pi_k(\cdot|x)} Q^{\pi_K}(x,a) - \mathbb{E}_{a \sim \pi_K(\cdot|x)} Q^{\pi_K}(x,a)\right| \le \frac{2R_{\max}}{1-\gamma} D_{\mathrm{TV}}\left(\pi_k(\cdot \mid x), \pi_K(\cdot \mid x)\right).$$

Therefore,

$$\left|\mathbb{E}_{a \sim \pi_k(\cdot|x)} A^{\pi_K}(x,a)\right| \le \frac{2R_{\max}}{1-\gamma} D_{\mathrm{TV}}\left(\pi_k(\cdot \mid x), \pi_K(\cdot \mid x)\right). \tag{16}$$

Substituting this into Equation (15) yields

$$|J(\pi_k) - J(\pi_K)| \le \frac{2R_{\max}}{(1-\gamma)^2} \sup_x D_{\mathrm{TV}}\left(\pi_k(\cdot \mid x), \pi_K(\cdot \mid x)\right). \tag{17}$$

Combining with Equation (14) gives

$$|J(\pi_k) - J(\pi_K)| \le \frac{2R_{\max}}{(1-\gamma)^2}(K-k)\sqrt{\frac{\delta_{\max}}{2}},$$

which proves the claim. □

B Experiment Details

B.1 Environments

Following Ghugare et al. (2026), we evaluate FlexLoop on 30 long-horizon goal-conditioned environments from BoxPick (Bortkiewicz et al., 2025), LightsOut (Anderson & Feil, 1998), and OGBench (Park et al., 2025).

BoxPick. BoxPick is a discrete-action benchmark, where an agent navigates in a 6×6 grid, picks up boxes, and places them at designated goal locations. An episode is successful only when all boxes are correctly placed. We consider two types of configurations. (i) *BoxPick-Exact-n* requires the agent to transport n boxes. The training tasks involve moving boxes between two adjacent quadrants of the map, while the unseen tasks require transportation between diagonal quadrants. We evaluate $n \in \{3, 4\}$. (ii) *BoxPick-Generalize-n-m* requires transporting n boxes in total, where training tasks only require moving m boxes and unseen tasks require moving all n boxes to their target locations. We evaluate $(n, m) \in \{(4,1), (4,2)\}$. These four environments provide a challenging test of long-horizon planning and generalization. We use goal-conditioned DQN as the underlying RL algorithm for this environment.

LightsOut. *LightsOut-4x5* is a large discrete-action decision-making benchmark with a 4×5 grid of 20 binary switches. At each step, the agent selects one switch, which toggles its own state and the states of all adjacent switches. An episode is successful when all switches are turned on. The training tasks consist of initial configurations that can be solved with relatively fewer steps, while the unseen tasks contain configurations requiring longer solution sequences. We use goal-conditioned PPO as the underlying RL algorithm for this environment.

OGBench. OGBench is a continuous-action offline RL benchmark, where agents learn to achieve diverse test goals solely from static offline datasets. We evaluate five scenarios, including two navigation scenarios, *AntMaze-Medium-Stitch* and *PointMaze-Large-Stitch*, which test the ability to compose experiences through trajectory stitching, and three manipulation scenarios, *Scene-Play*, *Cube-Double-Play*, and *Puzzle-4x4-Play*, which evaluate fine-grained control and compositional generalization. Following the evaluation protocol of Ghugare et al. (2026), we report the average success rate over five task environments for each scenario. We use goal-conditioned IQL as the underlying RL algorithm for these environments.

B.2 Implementation Details

We build our implementation upon the official codebase by Ghugare et al. (2026) and reproduce the IRU baseline before implementing FlexLoop. Unless otherwise stated, we use the same training steps, environment configurations, and hyperparameters as reported in the IRU paper. For BoxPick, the temperature τ used in Equation (3) is set to 0.25. For OGBench, we use a hidden size of 256 instead of the hidden size of 64 reported in the original paper, as the former produces results closer to those reported for the benchmark. For on-policy GC-PPO, we additionally replace the GAE-based value targets used in the value loss with value estimates from the pretrained IRU model to stabilize FlexLoop training. The policy optimization objective remains unchanged and continues to use standard on-policy PPO updates.

The key hyperparameter of FlexLoop is the adjacent-depth policy distillation weight λ_{depth}. The selected values for each environment are reported in Table 3.

Table 3: Adjacent-depth policy distillation weight used for each environment.

Environment	λ_{depth}
BoxPick-Exact-3	1.0
BoxPick-Exact-4	1.0
BoxPick-Generalize-4-1	1.0
BoxPick-Generalize-4-2	1.0
LightsOut	0.05
Scene	0.05
Cube	0.02
Puzzle	0.2
AntMaze	0.03
PointMaze	0.3

C Extended Results

In this section, we provide detailed environment-wise evaluation results, complementing the aggregated results reported in the main paper.

C.1 FULL TRAINING CURVES FOR FIGURE 4

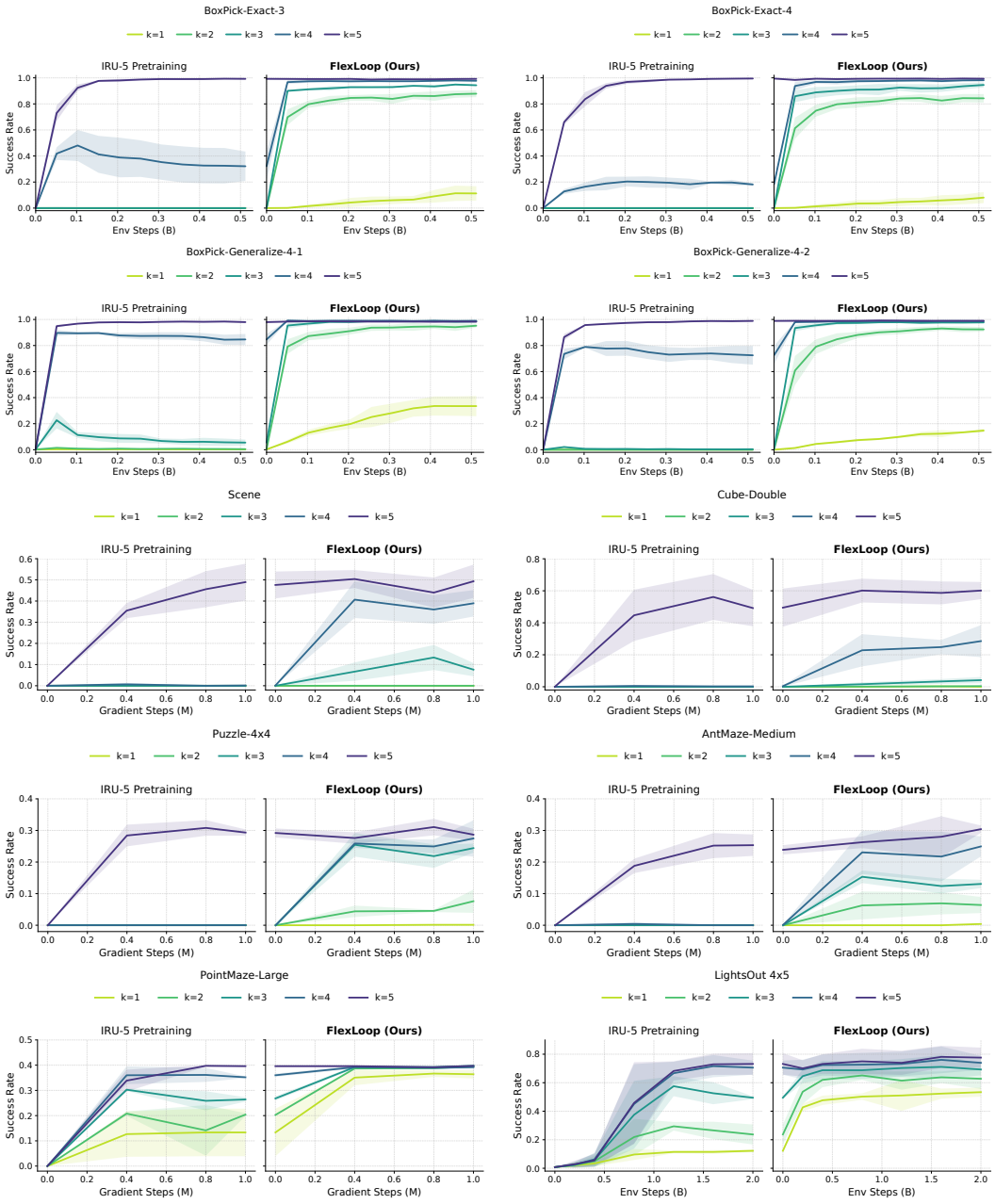


Figure 7: Depth-wise decision performance during IRU pretraining and FlexLoop post-training. Solid lines denote mean success rates and shading indicates one standard deviation. FlexLoop rapidly improves shallow-depth performance while preserving the full-depth capability of pretrained policies.

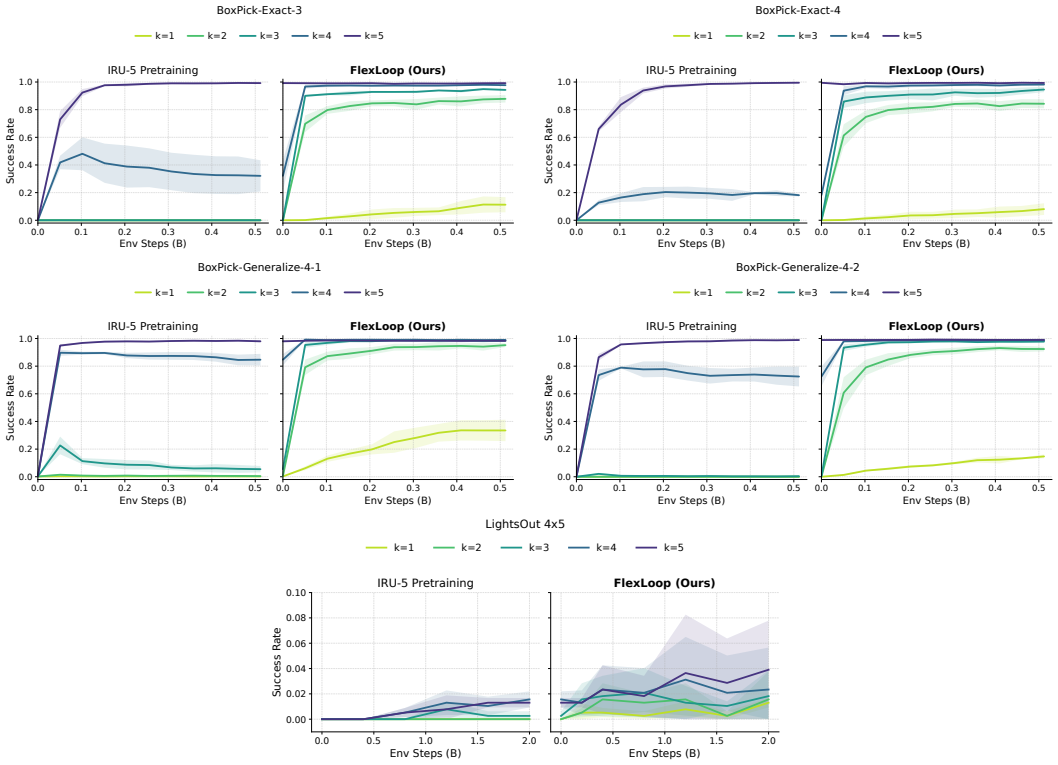


Figure 8: Depth-wise decision performance of pretrained IRU-5 and FlexLoop on unseen tasks. Solid lines denote mean success rates and shading indicates one standard deviation. FlexLoop improves shallow-depth decision elasticity while maintaining out-of-distribution generalization.

C.2 ELASTICITY CURVES

We evaluate the depth-wise decision performance of the pretrained IRU-5 checkpoint and the corresponding FlexLoop post-trained checkpoint at each recurrent depth, and visualize the resulting depth-performance curves for comparison. Figure 9 presents the curves on training tasks, while Figure 10 presents the corresponding results on unseen tasks. These curves provide a direct characterization of the depth elasticity learned by FlexLoop.

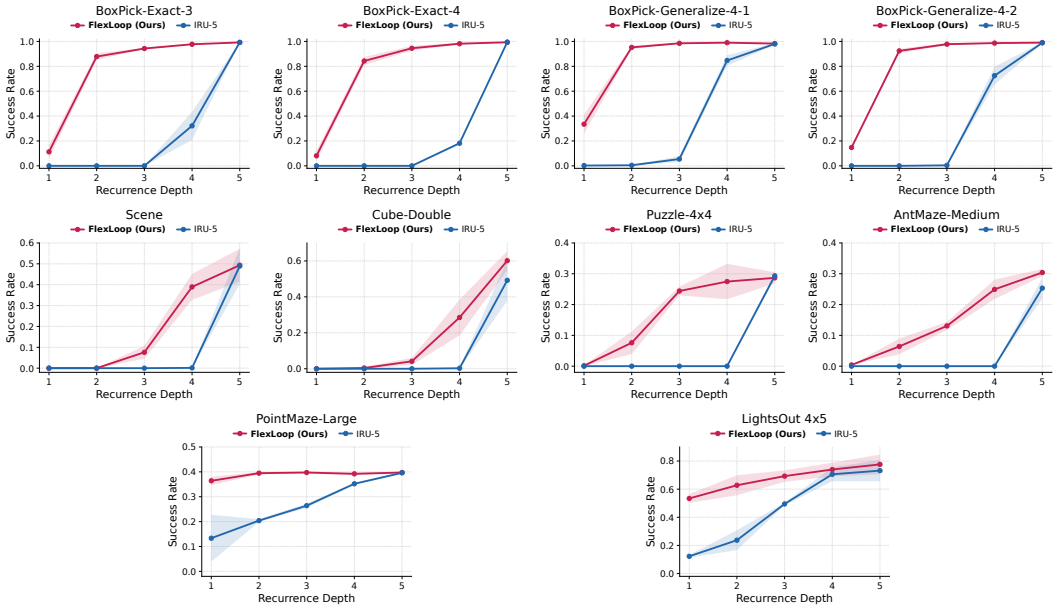


Figure 9: Depth-wise decision performance on training tasks before and after FlexLoop post-training. Markers denote mean success rates and shading indicates one standard deviation.

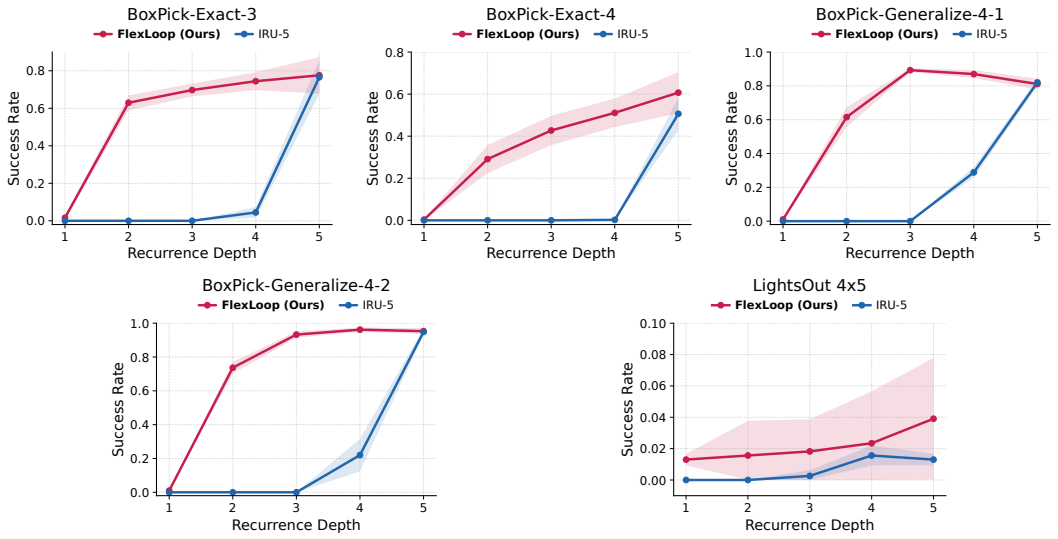


Figure 10: Depth-wise decision performance on unseen tasks before and after FlexLoop post-training. Markers denote mean success rates and shading indicates one standard deviation.

C.3 Full results for Table 1

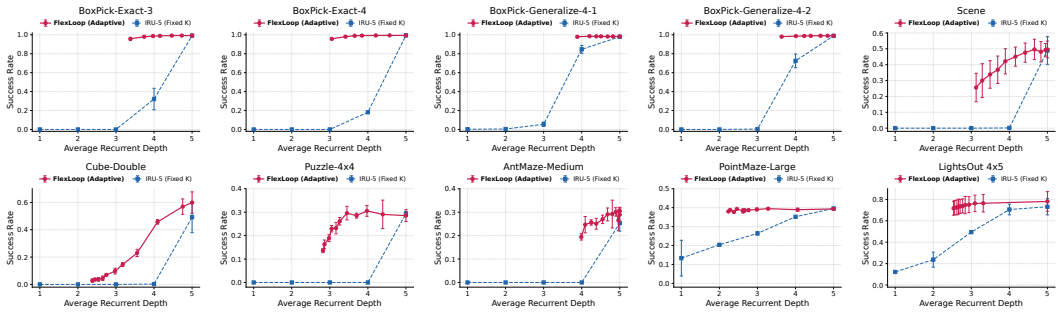
Table 4: Full state-wise adaptive inference results on all environments. ± indicates one standard deviation.

Task	FlexLoop		IRU-5
	Avg. Depth	Succ.	Succ.
Scene	4.66 ± 0.06	0.496 ± 0.064	0.489±0.087
Cube	4.75 ± 0.03	0.569 ± 0.056	0.492±0.114
Puzzle	3.45 ± 0.03	0.296 ± 0.029	0.293±0.01
AntMaze	4.80 ± 0.03	0.292 ± 0.059	0.253±0.034
PointMaze	3.28 ± 0.30	0.395 ± 0.004	0.396±0.003
LightsOut-4x5 (Train)	2.84 ± 0.08	0.747 ± 0.081	0.732 ± 0.076
LightsOut-4x5 (Unseen)	2.93 ± 0.21	0.053 ± 0.031	0.013 ± 0.004
BoxPick-Exact-3(Train)	4.47 ± 0.02	0.992 ± 0.004	0.992 ± 0.003
BoxPick-Exact-3(Unseen)	4.45 ± 0.05	0.768 ± 0.101	0.766 ± 0.091
BoxPick-Exact-4(Train)	4.21 ± 0.04	0.993 ± 0.002	0.994±0.003
BoxPick-Exact-4(Unseen)	4.25 ± 0.04	0.567 ± 0.098	0.506±0.077
BoxPick-Gen-4-1(Train)	4.68 ± 0.03	0.982 ± 0.007	0.98 ± 0.007
BoxPick-Gen-4-1(Unseen)	4.57 ± 0.02	0.82 ± 0.033	0.82±0.022
BoxPick-Gen-4-2(Train)	4.66 ± 0.01	0.991 ± 0.004	0.989±0.004
BoxPick-Gen-4-2(Unseen)	4.57 ± 0.01	0.956 ± 0.017	0.948±0.019

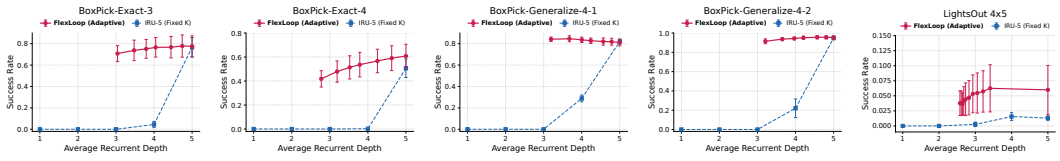
-Gen means -Generalize

C.4 Pareto Curve

Each Pareto curve is obtained by sweeping the stopping threshold ε in Equation (9), where each threshold corresponds to one computation–performance point. The ε grids are selected according to the characteristics of different environment families. For BoxPick, we use a 7-point grid $\{0, 0.005, 0.01, 0.02, 0.03, 0.05, 0.1\}$. For LightsOut, we use an 11-point uniformly spaced grid over $[0, 0.05]$. For OGBench, we use an 11-point uniformly spaced grid for all environments: manipulation tasks (Scene, Cube, and Puzzle) use a finer grid over $[0, 0.005]$, while navigation tasks (AntMaze and PointMaze) use a grid over $[0, 0.05]$.



(a) Pareto trade-offs on training tasks.

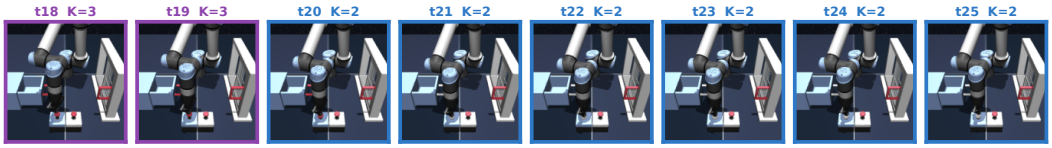


(b) Pareto trade-offs on unseen tasks.

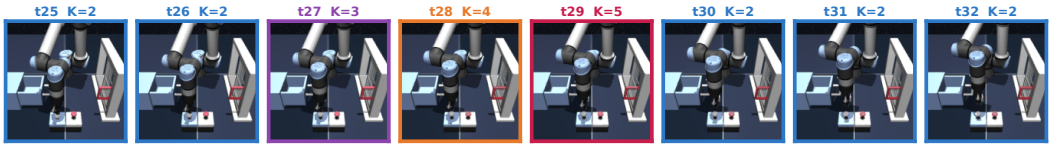
Figure 11: Pareto trade-offs between task performance and average recurrent depth under different adaptive inference thresholds. Markers denote mean success rates and error bars indicate one standard deviation.

C.5 Visualized Trajectory Segments of Scene

Figure 12 visualizes selected trajectory segments of FlexLoop in the Scene environment to illustrate its state-wise adaptive inference behavior. FlexLoop uses shallower recurrent depths for relatively simple and monotonic actions, such as lowering the robotic arm to press a button or rotating the gripper orientation, while allocating deeper computation to more critical decision points, such as switching goals after pressing the button and moving toward the drawer.



(a) Lowering the arm to press the button.



(b) Transitioning from button pressing to moving toward the drawer.



(c) Rotating the gripper orientation.

Figure 12: Selected trajectory segments from a single episode in the Scene environment. FlexLoop adaptively uses fewer recurrent steps for simple, monotonic motions and increases computation at challenging decision points, such as goal switching.

D Additional Ablations

D.1 FlexLoop from Scratch

In this section, we further consider a more restrictive setting where the pretrained looped policy weights are unavailable and only its outputs can be queried. We ask whether FlexLoop can still acquire depth elasticity from random initialization while recovering the teacher's full-depth capability.

We instantiate this setting on BoxPick-Exact-4 with GC-DQN. Since the scratch policy does not inherit the pretrained weights, we make two modifications to FlexLoop. First, at the full depth, we replace the standard TD target with the value estimate queried from the pretrained looped policy:

$$\mathcal{L}_{\mathrm{TD}}^{T} = \mathbb{E}_{\mathcal{D}}\left[\left(Q_K^{\theta}(s,g,a) - \mathrm{sg}[y^T]\right)^2\right], \qquad y^T = r + \gamma(1-d)\max_{a'} Q^T(s',g,a'). \tag{18}$$

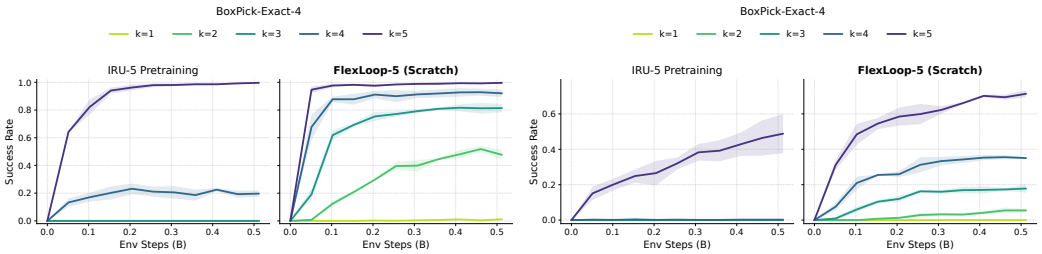
Second, we introduce an inter-depth Bellman refinement objective for shallow recurrent steps. Using the FlexLoop's target network $\bar{\theta}$, we define

$$y_k = r + \gamma(1-d)\max_{a'} Q_k^{\bar{\theta}}(s',g,a'), \tag{19}$$

and optimize

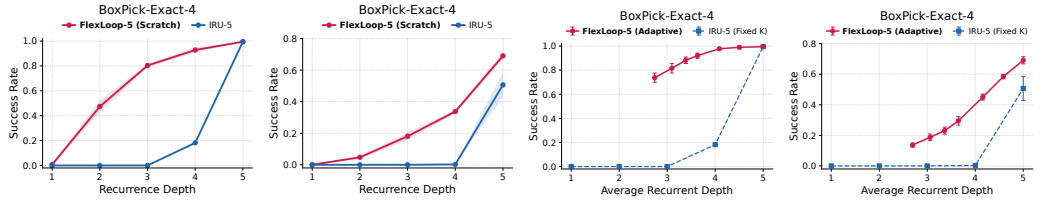
$$\mathcal{L}_{\mathrm{aux}}^{\mathrm{DQN}} = \frac{1}{K-2}\sum_{k=1}^{K-2}\mathbb{E}_{\mathcal{D}}\left[\rho\big(Q_{k+1}^{\theta}(s,g,a) - \mathrm{sg}[y_k]\big)\right]. \tag{20}$$

We set the weight of $\mathcal{L}_{\mathrm{aux}}^{\mathrm{DQN}}$ to 0.1.



(a) Training Tasks. (b) Unseen Tasks.

Figure 13: Depth-wise performance evolution of pretrained IRU-5 and FlexLoop from scratch on both training and unseen tasks. Solid lines denote mean success rates and shading indicates one standard deviation.



(a) Elasticity (Training). (b) Elasticity (Unseen). (c) Pareto (Training). (d) Pareto (Unseen).

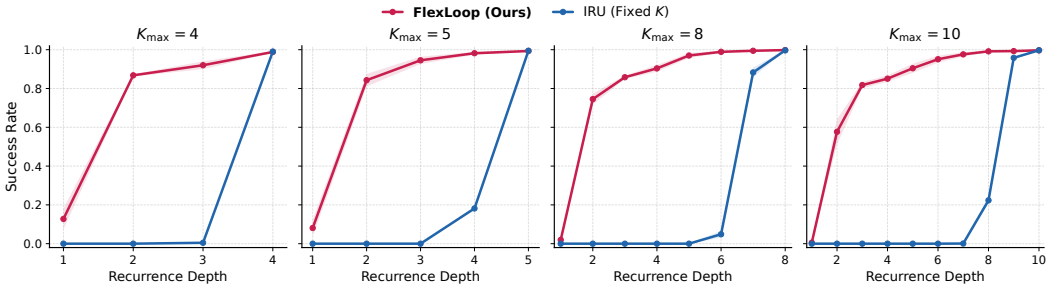
Figure 14: Elasticity and Pareto curves of pretrained IRU-5 and FlexLoop from scratch on both training and unseen tasks. Markers denote mean success rates and the shading and error bars indicate one standard deviation.

Notably, these modifications are introduced only to compensate for the absence of pretrained initialization. We observe no additional benefit in the teacher-initialized setting.

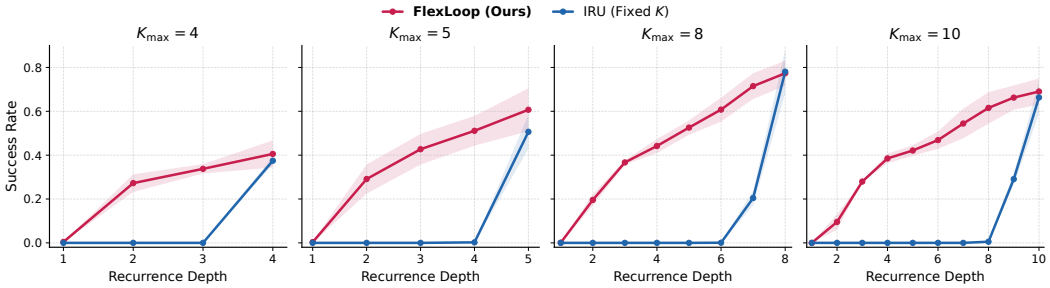
As shown in Figures 13 and 14, FlexLoop from scratch rapidly develops depth elasticity and recovers strong full-depth performance, while still supporting effective state-wise adaptive inference. Together, these results broaden the applicability of FlexLoop to settings where only output-level access to a pretrained policy is available.

D.2 FlexLoop on Policies with Different Pretraining Depths

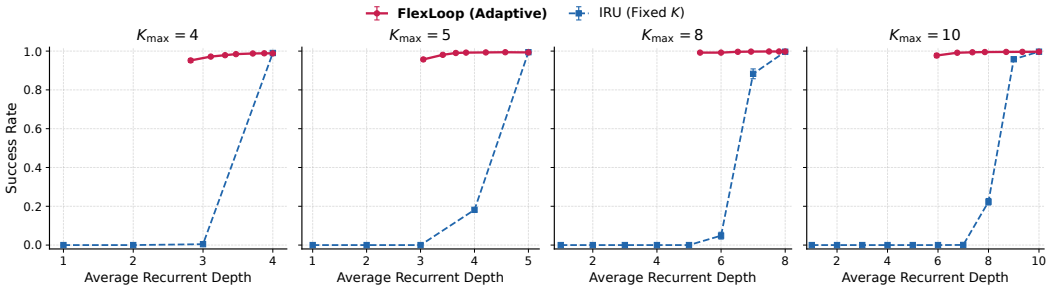
We further apply FlexLoop to IRU policies pretrained with different recurrent depths, $K \in \{4, 5, 8, 10\}$. Figure 15 shows that, across all pretraining depths, FlexLoop substantially improves shallow-depth performance while preserving full-depth capability, with the same trend observed on unseen tasks. The corresponding Pareto curves further show that state-wise adaptive inference remains effective across different pretraining depths. We note that deeper pretrained policies generally require smaller stopping thresholds ε, suggesting that the appropriate threshold scale depends on the pretrained recurrent depth.



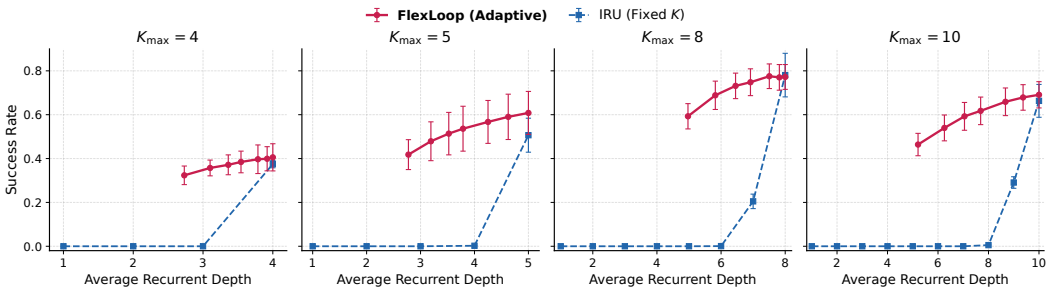
(a) Elasticity on Training Tasks.



(b) Elasticity on Unseen Tasks.



(c) Pareto on Training Tasks.



(d) Pareto on Unseen Tasks.

Figure 15: Elasticity and Pareto Curves of the policies with different pretraining recurrent depths before and after FlexLoop post-training. Markers denote mean success rates and shading indicates one standard deviation.