Adaptive Depth in Looped Transformers: Diagnosing Learned Halting Gates and Trajectory Readouts

Anonymous authors
Paper under double-blind review

Abstract

Looped Transformers increase test-time computation by repeatedly applying a shared recurrent block. Learned halting objectives in looped Transformers typically use a single exit distribution both as the inference-time stopping rule and as the training-time weighting of per-depth losses. This entangles exit selection with trajectory formation: the gate not only chooses which recurrent state to use, but also determines how strongly each intermediate state is supervised. Consequently, poor adaptive-compute performance can arise from the readout, the induced trajectory, or their interaction. We study adaptive depth in looped Transformers through this trajectory–readout lens, across controlled synthetic tasks (modular arithmetic and binary parity) and large-scale Ouro-1.4B and 2.6B checkpoints. We find that fixed-prior depth supervision, which shapes the trajectory without an input-dependent halting policy, produces difficulty-aware trajectories whose intermediate states expose useful stopping signals, and that simple post-hoc confidence readouts often match or outperform learned linear and MLP gates. Fitting gates on frozen trajectories localizes the failure: it appears to stem mainly from the trajectory induced by joint gate training rather than from limited gate expressivity. The same pattern is present in Ouro evaluations, where pretrained ponder gates are competitive but not uniformly Pareto-optimal, and measured latency confirms that the resulting reductions in average exit depth translate into practical inference-time savings. Our systematic diagnostic evaluation reframes adaptive depth in looped Transformers as a joint problem of trajectory formation and exit readout, rather than gate learning alone, highlighting a distinction that prior learned-halting work has often left implicit.

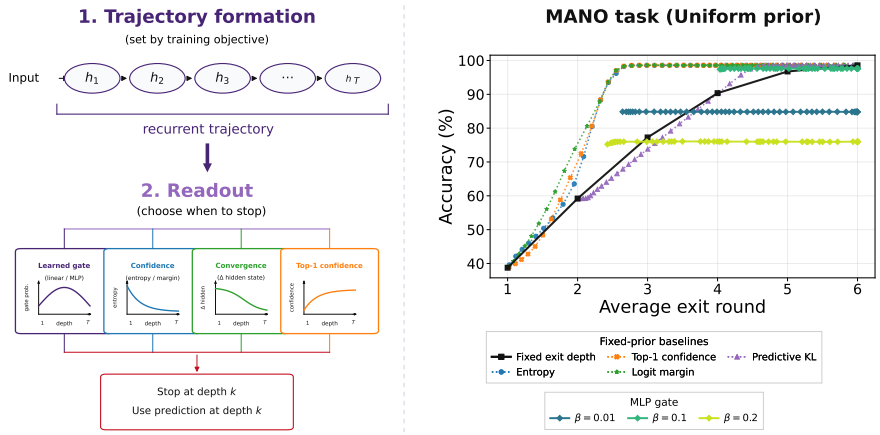


Figure 1: **Adaptive depth is a trajectory–readout problem, not just a gate-learning problem.** *Left:* a looped Transformer's recurrent trajectory $h_1, \ldots, h_T$ is shaped by the training objective, while a separate readout (a learned gate, or a simple heuristic) decides when to stop. *Right:* on MANO, post-hoc readouts on a **fixed-prior** trajectory reach near-ceiling accuracy at lower exit depth than a fixed-depth baseline, matching or outperforming an MLP gate, whose halting distribution both selects exits and shapes the trajectory it is trained on.

1 Introduction

This paper provides a systematic diagnostic study of adaptive depth in looped Transformers. Learned halting mechanisms promise adaptive computation by allocating recurrent depth to the input and stopping once further refinement is unnecessary. However, current learned halting objectives in looped Transformers couple two roles that should be distinguished. The learned exit distribution is used as an inference-time stopping rule, but it also weights the training losses across recurrent depths. Thus, a learned gate does not merely decide when to stop; it also changes the recurrent trajectory from which exits are selected.

This coupling makes adaptive depth difficult to diagnose: poor compute–quality tradeoffs may arise from the stopping rule, the induced trajectory, or their interaction. We therefore study adaptive depth as a trajectory–readout problem, varying training objectives, learned gates, post-hoc readouts, forced exits, and evaluating both controlled models trained on MANO (a synthetic modular-arithmetic task) and parity tasks, as well as pretrained Ouro checkpoints.

Our results show that adaptive depth is not only a question of learning a better halting gate. Fixed-prior depth supervision, which shapes the trajectory without learning an input-dependent halting policy, can produce difficulty-aware recurrent trajectories whose intermediate states support strong post-hoc early exits. In several settings, simple confidence readouts match or outperform learned gates in compute–accuracy tradeoff. We do not interpret this as evidence against learned halting in principle. Rather, our results suggest that current learned-gate objectives can fail because they entangle trajectory formation and exit selection, and that future adaptive-depth methods should treat both components as first-class design choices.

Our contributions are: (i) proposing a trajectory–readout view of adaptive depth in looped Transformers; (ii) systematically evaluating training objectives, gate parameterizations, post-hoc readouts, forced-exit trajectories, model scales, and latency; (iii) introducing fixed-prior depth supervision to decouple trajectory formation from exit selection; (iv) showing that fixed-prior training produces difficulty-aware recurrent trajectories on MANO; (v) demonstrating that simple post-hoc confidence readouts can match or outperform learned linear and MLP gates; (vi) showing, on Ouro-1.4B and Ouro-2.6B, that pretrained ponder gates are not uniformly Pareto-optimal; and (vii) providing latency measurements that connect reductions in average exit depth to practical inference-time speedups.

2 Related Work

Next, we review prior work along the two axes most relevant to our study: recurrent-depth Transformer architectures and adaptive computation methods for selecting how much computation to use.

2.1 Looped Transformers

Looped Transformers replace stacks of distinct layers with repeated applications of a shared block of layers, decoupling effective computational depth from parameter count. Early work in the field of looped Transformers includes the Universal Transformer (Dehghani et al., 2019), implicit-depth models such as Deep Equilibrium Models (Bai et al., 2019), and cross-layer parameter sharing in ALBERT (Lan et al., 2020). More recent work has reframed recurrence as latent computation, in which each recurrence step refines the hidden representation before performing the prediction (Saunshi et al., 2025; Geiping et al., 2025; Zhu et al., 2025; Jeddi et al., 2026). This depth-wise recurrence differs from approaches that extend the sequence with continuous hidden states or latent tokens, since computation remains on the model's depth axis rather than in context (Hao et al., 2025; Zelikman et al., 2024; Huang et al., 2025).

Looped models have been proven to have promising performance on iterative and algorithmic computation, including programmable computation, length generalization, and multi-step refinement on synthetic tasks (Giannou et al., 2023; Anil et al., 2022; Yang et al., 2024; Kohli et al., 2026). More recent language-modeling work studies whether this recurrent-depth scaling transfers to larger models, where the additional computation can be spent in latent space rather than through longer generated reasoning traces (Geiping et al., 2025; Zhu et al., 2025; Jeddi et al., 2026). Our work builds on this view of looped Transformers as a

means to increase test-time computation, but focuses on a different question: how training objectives shape the recurrent trajectories, and how exit readouts convert them into compute–quality tradeoffs.

2.2 Adaptive Computation

Adaptive computation methods aim to improve performance–compute tradeoffs by allocating different amounts of computation to different tokens. Early approaches such as ACT (Graves, 2017) and PonderNet (Banino et al., 2021) learn probabilistic halting policies, while later work applies adaptive depth to Transformers and language models (Hou et al., 2020; Elbayad et al., 2020; Fedus et al., 2021; Bae et al., 2023; Elhoushi et al., 2024; Raposo et al., 2024). This idea adapts naturally to looped Transformers. Their recurrent structure produces intermediate states, thus allowing us either to exit early and commit to one of those hidden states or to continue refining them through additional iterations. Recent looped language models therefore use learned gates or routers to allocate recurrent depth (Dehghani et al., 2019; Zhu et al., 2025; Song et al., 2026; Bae et al., 2025).

Learned gates are not the only possible stopping signal. Early-exit feedforward Transformer methods have also used confidence, entropy, or prediction stability as inexpensive criteria for terminating computation (Schuster et al., 2022; Xin et al., 2020; Zhou et al., 2020), and entropy dynamics over chain-of-thought traces have similarly been linked to reasoning correctness (Català et al., 2026). In looped Transformers, analogous signals can be computed across recurrent iterations, including changes in hidden states, logits, or predictive distributions (Geiping et al., 2025). Our work builds on this observation, but separates two roles that are often implicitly coupled: training objectives shape the recurrent trajectory, while readouts select exits from the induced trajectory.

Prior work asks how to make recurrent depth dynamic, through learned halting policies, routers, or other input-dependent allocation rules (Geiping et al., 2025; Zhu et al., 2025; Song et al., 2026; Bae et al., 2025). We ask a complementary question: when such a method succeeds or fails, what part of the system is responsible? In looped Transformers a halting mechanism is not only a stopping rule; during training it also determines which recurrent depths receive supervision, and therefore shapes the trajectory from which exits are later selected. We isolate this distinction experimentally, separating trajectory formation from exit readout.

3 Formalizing Adaptive Depth as Trajectory and Readout

In this section, we introduce the notation and mechanisms used throughout the paper. We first formalize looped Transformer trajectories as the sequence of hidden states produced by repeatedly applying a shared Transformer block. We then describe learned halting gates, which both define an exit distribution at inference time and weight losses across recurrent depths during training. Finally, we introduce fixed-prior depth supervision and post-hoc trajectory readouts, which allow us to separate trajectory formation from exit selection.

3.1 Looped Transformer Trajectories

Let $x = (x_1, \ldots, x_M)$ be an input sequence of length M and let $h_0 = \mathrm{E}(x) \in \mathbb{R}^{M \times d}$ denote its token embeddings, for an embedding function $\mathrm{E}: \mathbb{R}^{|V|} \to \mathbb{R}^d$ with vocabulary size $|V|$ and hidden size d. A looped Transformer repeatedly applies a shared block $\mathcal{F}_\theta$ of L Transformer layers to update the hidden state, where θ represents the parameters of the model: $h_t = \mathcal{F}_\theta(h_{t-1}),$ for $t = 1, ..., T$ where T is the maximum possible number of loops. The generated hidden states can be unembedded using a layer $\mathrm{lmhead} : \mathbb{R}^d \to \mathbb{R}^{|V|}$. Looped Transformers naturally induce a trajectory in the latent space: $\tau_\theta(x) = \{h_i\}_{i=1}^{T}$.

In this work, we study how this induced trajectory is shaped by the adaptive-computation training objective and whether trajectory proxies are competitive with trained gates at predicting when computation should halt. Additional architectural background and notation for looped Transformers are provided in Appendix A.1.

3.2 Gating Mechanism for Adaptive Computation

A common approach to adaptive computation in looped Transformers is to attach a PonderNet inspired halting module that predicts an exit distribution over recurrent depths (Banino et al., 2021; Zhu et al., 2025; Song et al., 2026).

Let $e_t(x) = \sigma(g_\phi(h_t)) \in (0, 1)$ denote the conditional probability of exiting at loop t, given that the model has not exited earlier. In this work, we experiment with the gate as both a linear projection and an MLP. The predicted conditional halt probabilities induce a distribution over exit depths:

$$q_t(x) = e_t(x) \prod_{j<t} (1 - e_j(x)) \quad \text{for } t = 1, ..., T-1. \qquad q_T(x) = \prod_{j<T} (1 - e_j(x)) \,.$$

Note that we assign all remaining probability mass to the final recurrent depth T, ensuring that $q(\cdot \mid x)$ is a valid distribution over exits. At inference time, the learned distribution can be used as a stopping rule, by halting computation when the cumulative exit distribution exceeds a threshold α. We can define the early exit depth d_α as follows: $d_\alpha(x) = \min \left\{ t : \sum_{s=1}^{t} q_s(x) \geq \alpha \right\}$.

During training, the exit distribution is used to weight the losses at different recurrent depths. Given per depth Cross-Entropy losses ℓ_t, the model is jointly trained with the gate using the following objective:

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{(x,y)} \left[\sum_{t=1}^{T} q_t(x) \ell_t(x, y) + \beta KL(q(x) || \pi) \right],$$

where π is a target prior over depths, and β controls the strength of the regularization. We note that since the learned weights $q_t(x)$ appear inside the loss, the gate is not only an inference-time stopping rule. It also determines how much prediction loss each recurrent depth contributes during training. Therefore, the learned trajectory $\tau_{(\theta,\phi)}(x) = \{h_t\}_{t=1}^{T}$ depends on the halting objective and on the gate parameters ϕ. This implies that training a gate entangles two effects: learning a readout $q(\cdot \mid x)$ for selecting exits, and changing the depth supervision that shapes the trajectory itself. Thus, different gate design choices affect how the model spreads the computation across loops. A central goal of our study is to separate these two effects.

3.3 Fixed-Prior Depth Supervision

The training objective above uses an input-dependent distribution $q(\cdot \mid x)$ to define an inference-time readout and to weight losses across the loops during training. To decouple these roles, we also study objectives with fixed loss weights over recurrent depth. Given a fixed distribution π over loops, fixed-prior training minimizes the following loss function:

$$\mathcal{L}_{\text{fixed}}(\theta) = \mathbb{E}_{(x,y)} \left[\sum_{t=1}^{T} \pi_t \ell_t(x, y) \right].$$

In contrast to $q(\cdot \mid x)$, the weights π_t do not depend on the input and do not define a learned stopping policy. They only specify how strongly each recurrent depth is supervised during training. Fixed-prior training provides a controlled way to shape the trajectory without simultaneously learning an adaptive readout.

In this study, we consider two families of fixed priors: the uniform and geometric priors. The uniform prior assigns equal loss weight to every recurrent depth, $\pi_t = \frac{1}{T}$. The truncated geometric prior places more weight on earlier depths. To obtain a valid probability distribution, we assign all remaining probability mass to the final recurrent depth:

$$\pi_t = \lambda(1 - \lambda)^{t-1}, \quad \text{for } t = 1, ..., T-1. \qquad \pi_T = (1 - \lambda)^{T-1}.$$

Since fixed-prior trained models do not learn a halting distribution, they need a separate readout at inference time. This makes them useful for our decomposition. If post-hoc trajectory readouts can stop fixed-prior trajectories effectively, then adaptive exit does not necessarily require a learned gate, but a trajectory whose intermediate states contain informative stopping signals.

3.4 Trajectory Readouts

After training, an adaptive-compute method must choose an exit depth from the recurrent trajectory. We use the term *readout* for any rule that maps the trajectory to an exit depth. A fixed-depth readout exits at the same loop $d(x) = t_0$ for every input. A learned gate readout uses the learned exit distribution $q(\cdot \mid x)$, for example through the CDF threshold rule $d_\alpha(x)$ defined above.

We also study post-hoc trajectory readouts that do not require a trained gate. Confidence readouts decide to exit when the model appears sufficiently certain, using quantities such as entropy, top-1 confidence, or logit margin. Convergence readouts exit when the trajectory appears to have stabilized, using quantities such as predictive KL between consecutive predictions, logit change, or hidden-state movement. All readouts are implemented as threshold rules over these scalar signals. Formal definitions are given in Appendix B.6. These readouts are applied only after training, so they can select exits from a trajectory but cannot influence how the trajectory is shaped during training. To separate trajectory quality from readout quality, we also use forced-exit evaluations. A forced exit ignores any adaptive rule and evaluates the prediction at each loop t. This gives the per-depth quality curve of the trajectory itself.

4 Methodology

We design experiments to separate trajectory formation from exit selection. In the controlled MANO experiments, we train looped Transformers under fixed-prior and learned-gate objectives, then evaluate multiple readouts on the resulting trajectories. We use Ouro 1.4B and Ouro 2.6B to validate our claims on large-scale public foundation models. In particular, we evaluate whether the pretrained gate in these public looped language model checkpoints is competitive with simple post-hoc trajectory readouts. Additional implementation details for MANO, Ouro, threshold selection, and optimization are provided in Appendix B.

4.1 Controlled MANO Experiments

Our primary controlled setting is MANO (Allen-Zhu, 2025), a synthetic modular arithmetic task over integers modulo 23. Each example is an arithmetic expression written in the prefix notation, and the model must predict the final answer. We use expressions with up to 10 operations and group examples by operation count when analyzing difficulty. Unlike setups that provide the expression length as an input token, our implementation does not include a length token, so that the model must infer difficulty from the expression itself, rather than finding a shortcut.

We focus on this setting because it exposes a nontrivial accuracy–compute tradeoff. Early recurrent states are not always sufficient, while later recurrent states remain relevant for solving harder expressions. Moreover, this dataset allows us to have a clear difficulty ordering, through the number of operations, a fact used in the analysis. All MANO models use the same looped Transformer backbone with 4 layers, 4 attention heads, hidden embedding size 512, and maximum recurrent depth $T = 6$. We evaluate models both by adaptive exits and by forced exits. In a forced-exit evaluation, the model is evaluated independently at each recurrent depth t, without using any learned or post-hoc stopping rule. This gives the per-loop trajectory of accuracy and loss, and lets us analyze whether training produced useful intermediate states.

For MANO, we generate three disjoint datasets of 50k examples using different random seeds. Models are trained on a split generated with seed 42, post-hoc readout thresholds are selected on a validation set generated with seed 43, and all reported compute–quality curves are evaluated on a test set generated with seed 44. The same validation and test splits are used for all readouts, and the test split is used only after the threshold grid has been fixed.

4.2 Training Objectives and Gate Parameterizations

We train three model families. Fixed-prior models use the objective in Section 3.3 with a uniform prior or truncated geometric priors (larger λ places more loss weight on earlier loops). Learned linear-gate models use the objective in Section 3.2 with g_ϕ a linear projection to a halt logit, trained under both priors while sweeping

β. MLP-gate models replace g_ϕ with a two-layer feedforward network, testing whether gate expressivity changes the comparison against simple trajectory readouts.

4.3 Trajectory Readouts and Threshold Selection

For each trained trajectory, we evaluate readouts that select an exit depth. Learned-gate models use their native cumulative-threshold rule d_α (Section 3.4), swept over α to trace a Pareto curve. We also evaluate post-hoc readouts that do not affect the trajectory: confidence readouts (entropy, top-1 confidence, logit margin) and convergence readouts (predictive KL, logit change, hidden-state displacement norm), each swept over a validation-calibrated grid. Fixed-depth baselines require no calibration. Grid and held-out details are in Section 4.4 and Appendix B.

4.4 Evaluation Protocol and Metrics

Our evaluation separates two questions: whether a training objective produces useful intermediate states, and whether a readout selects good exits from them. We measure trajectory quality with *forced exits*, evaluating every example at each recurrent depth without any stopping rule. We measure adaptive stopping by sweeping thresholds into compute–quality curves; each threshold gives one operating point of task quality versus average exit depth. Validation data are used only for threshold selection (MANO: select on seed 43, evaluate on seed 44; Ouro: select on the validation split or subset of Section 4.5, evaluate on the held-out split). Computation is the mean selected depth $\mathbb{E}_x[d(x)]$, the *average loops*. For MANO we summarize frontiers with D@X, the minimum average depth to reach $X\%$ test accuracy; for Ouro we report held-out accuracies at the validation-selected operating points.

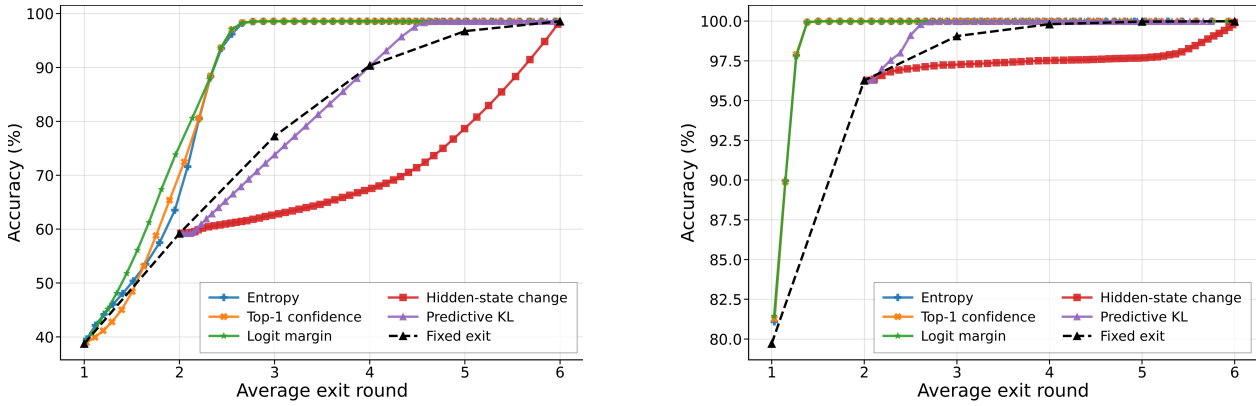
4.5 Large-Scale Looped Language-Model Evaluation

To test whether the same readout pattern appears in large-scale looped language-model checkpoints, we evaluate Ouro-1.4B and Ouro-2.6B (Zhu et al., 2025), a family of pretrained Looped Language Models that build iterative latent computation into pretraining, applying a shared Transformer block for up to four recurrent steps and using a PonderNet-style gate trained in two stages (joint entropy-regularized training, then frozen-backbone fine-tuning on a marginal-utility signal). Unlike the MANO experiments, we do not train or modify these models: we evaluate the released checkpoints as-is and compare the pretrained ponder gate against simple post-hoc readouts applied to the recurrent trajectory. Full architectural and training details are given in Appendix A.2.

We evaluate on six standard benchmarks: MMLU (Hendrycks et al., 2021), ARC-Easy (Clark et al., 2018), ARC-Challenge (Clark et al., 2018), OpenBookQA (Mihaylov et al., 2018), HellaSwag (Zellers et al., 2019), and CommonsenseQA (Talmor et al., 2019). We evaluate MMLU with 5-shot prompting, ARC-Challenge with 25-shot prompting, ARC-Easy with 8-shot prompting, OpenBookQA with 0-shot prompting, and HellaSwag and CommonsenseQA with 10-shot prompting. For benchmarks with labeled test sets, thresholds are selected on the standard validation split and evaluated on the standard test split. For HellaSwag and CommonsenseQA, where labeled test sets are not available, we split the labeled validation data into a 20% threshold-selection subset and an 80% held-out evaluation subset using seed 42. The same held-out protocol and compute metrics from Section 4.4 are used for all Ouro readouts.

5 Results

The results proceed in three stages. First, we use MANO as a controlled setting where we can change the training objective and directly inspect the recurrent trajectory. Section 5.1 asks whether adaptive exits require a learned halting policy and whether fixed-prior trajectories develop difficulty-aware dynamics. Section 5.2 compares fixed-prior trajectories against learned-gate trajectories, and Section 5.3 uses forced exits to diagnose how the objective changes intermediate states. Second, Section 5.4 freezes pretrained MANO trajectories and fits new post-hoc gates, separating gate expressivity from trajectory quality. Finally, Section 5.5 tests the same readout pattern in pretrained Ouro-1.4B and Ouro-2.6B checkpoints, and Section 5.6 measures whether reductions in average recurrent depth correspond to latency savings.



(a) Uniform fixed prior

(b) Geometric fixed prior, $\lambda = 0.3$

Figure 2: Post-hoc readout curves for fixed-prior trajectories. We plot test accuracy against average exit depth for the uniform prior and the geometric prior with $\lambda = 0.3$.

Table 1: Fixed-prior frontier summary over three seeds. D@X denotes the minimum average exit depth required to reach X% test accuracy. N/A indicates that the corresponding accuracy threshold was unattainable, so no D@X value could be computed.

Training	Readout	D@95	D@98	D@99
Uniform	Fixed depth	4.50 ± 0.71	6.00 ± 0.00	N/A
	Entropy	2.19 ± 0.51	2.66 ± 0.44	N/A
	Top-1 conf.	2.19 ± 0.51	2.66 ± 0.44	N/A
	Margin	2.19 ± 0.51	2.66 ± 0.44	N/A
Geom. ($\lambda = .3$)	Fixed depth	2.50 ± 0.71	3.50 ± 0.71	3.50 ± 0.71
	Entropy	1.38 ± 0.16	1.50 ± 0.16	1.55 ± 0.24
	Top-1 conf.	1.38 ± 0.17	1.50 ± 0.16	1.50 ± 0.16
	Margin	1.38 ± 0.16	1.50 ± 0.16	1.55 ± 0.24

5.1 Fixed-prior training produces difficulty-aware trajectories

We first ask whether adaptive computation requires a learned halting distribution. Fixed-prior-trained models provide a controlled test. Their objective shapes a recurrent trajectory, but it does not learn an input-dependent exit policy.

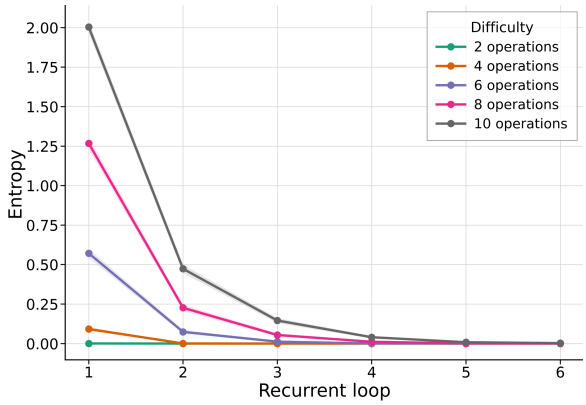
Throughout the paper, Pareto plots report task quality (y-axis) against average exit depth (x-axis), the mean number of recurrent loops executed under a given readout. Points closer to the upper left are preferable, achieving higher accuracy with less computation. Each curve is obtained by sweeping the stopping threshold of a single readout, so it represents the full compute–quality tradeoff for that readout.

Table 1 confirms the representative curves across three seeds. The geometric prior $\lambda = 0.3$ yields a favorable frontier, with confidence readouts reaching 99% accuracy at roughly 1.5 average loops. Figure 2 shows representative compute–quality curves for the uniform fixed prior and the geometric fixed prior with $\lambda = 0.3$. In both cases, simple confidence readouts such as entropy, top-1 confidence, and logit margin recover efficient early-exit behavior. Under the uniform prior, these readouts already reach high accuracy using fewer than the maximum number of loops. Under the geometric prior, useful predictions appear earlier in the trajectory, producing stronger tradeoffs at low average exit depth.

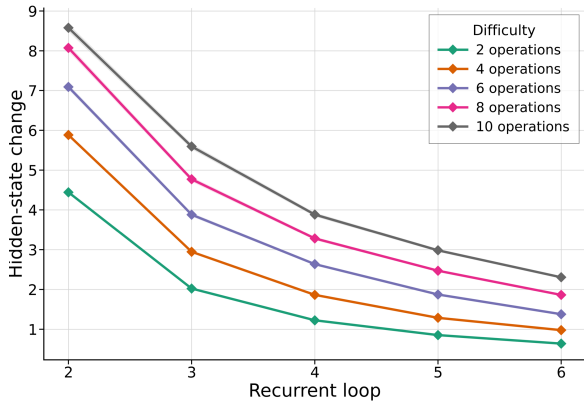
The same difficulty-aware structure persists beyond the training range: harder out-of-distribution expressions stay uncertain for more loops and converge more slowly in hidden-state space (Appendix C.4). Full sweeps and diagnostics are in Appendix C, and the same pattern holds on a second synthetic benchmark, binary parity over bit strings of length 1–40 (Appendix F), where fixed-prior trajectories support strong post-hoc readouts and diagnostics stay ordered by bit length.

5.2 Fixed-prior readouts can match or outperform learned gates

We next compare fixed-prior trajectories with jointly trained learned-gate trajectories. Fixed-prior models use post-hoc readouts, whereas learned-gate models use their native CDF-threshold readout.

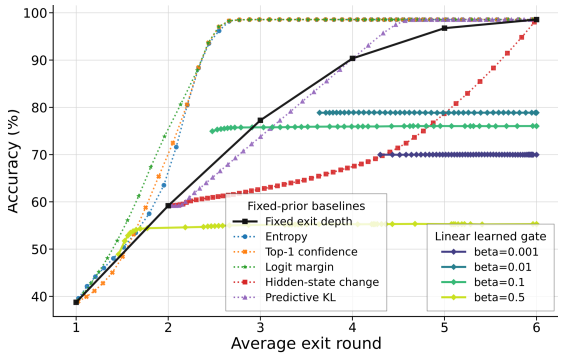


(a) Entropy

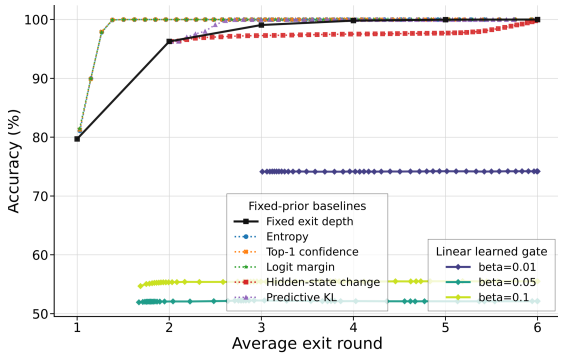


(b) Hidden-state displacement

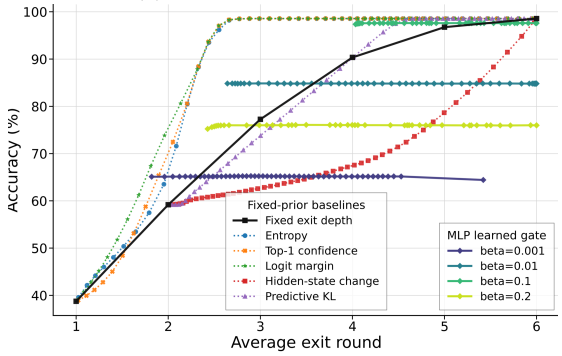
Figure 3: Trajectory signals for the geometric fixed-prior model with $\lambda = 0.3$ grouped by difficulty, which is represented as the number of operations in the input example. Entropy and hidden-state displacement values are plotted as a function of recurrent depth.



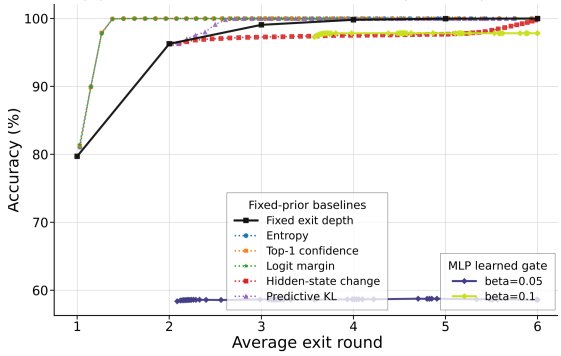
(a) Linear gate, uniform prior



(b) Linear gate, geometric prior ($\lambda = 0.3$)



(c) MLP gate, uniform prior



(d) MLP gate, geometric prior ($\lambda = 0.3$)

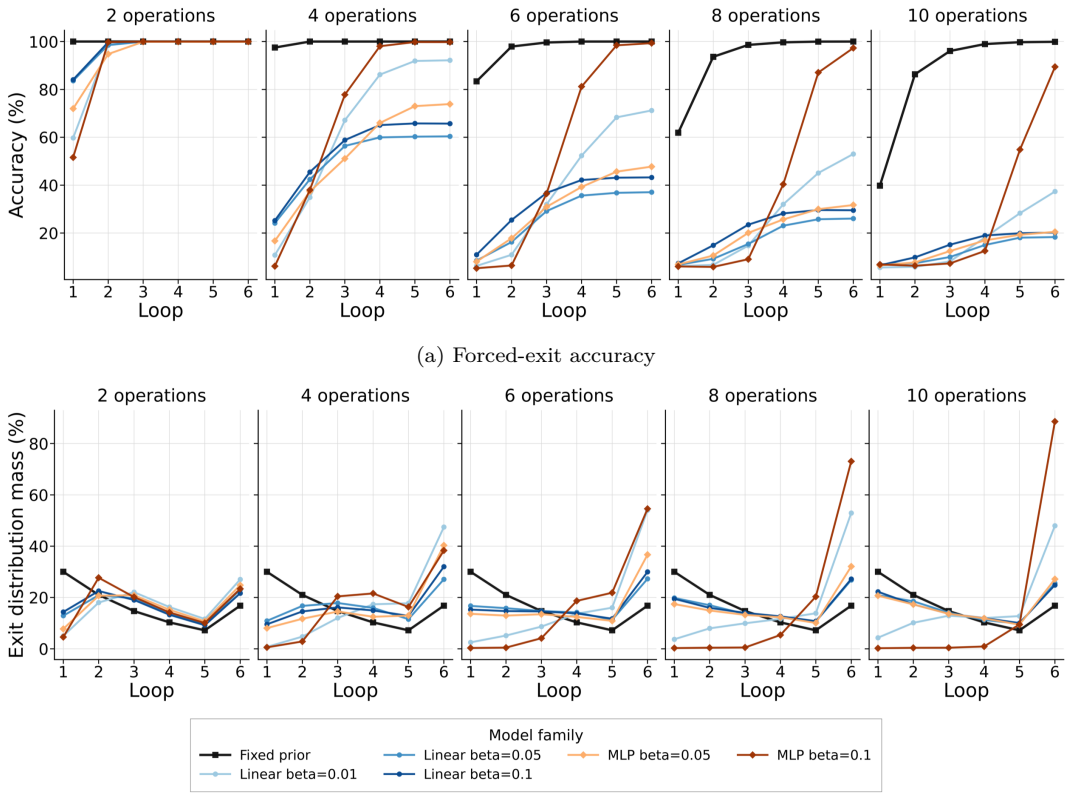
Figure 4: Fixed-prior readouts compared with learned linear and MLP gates under uniform and geometric priors.

Figure 4 shows that fixed-prior trajectories paired with simple confidence readouts consistently outperform learned linear gates. Increasing gate capacity improves the learned baseline, but MLP gates still fail to match the best fixed-prior frontiers.

Table 2 summarizes the same comparison numerically across three seeds. To avoid giving the fixed-prior models an adaptive choice of readout in this table, we use a single explicit confidence readout, logit margin, while learned-gate models use their native gate readout with the highest-performing β for each prior and gate class. Thus, the comparison favors the learned-gate side by allowing β selection, while using a fixed post-hoc readout for the fixed-prior models.

Table 2: Comparison of fixed-prior readouts and learned gates over three seeds. D@X denotes the minimum average exit depth required to reach X% test accuracy. N/A indicates that the target accuracy was not reached by any model across the three seeds.

Training	Readout	D@95	D@98	D@99
Fixed (Uniform)	Margin	2.19 ± 0.51	2.66 ± 0.44	N/A
Linear (Uniform)	Native gate	4.49 ± 0.28	4.49 ± 0.28	N/A
MLP (Uniform)	Native gate	4.30 ± 0.34	N/A	N/A
Fixed (Geom., $\lambda = .3$)	Margin	1.38 ± 0.16	1.50 ± 0.16	1.55 ± 0.24
Linear (Geom., $\lambda = .3$)	Native gate	2.80 ± 0.30	N/A	N/A
MLP (Geom., $\lambda = .3$)	Native gate	3.58 ± 0.36	N/A	N/A



(a) Forced-exit accuracy

(b) Learned exit distribution

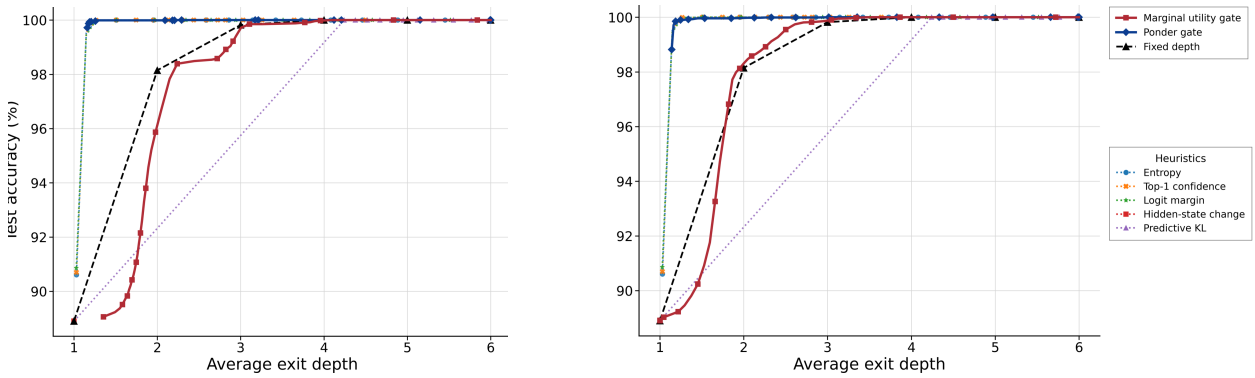
Figure 5: Forced-exit trajectories and learned exit distributions for the geometric prior with $\lambda = 0.3$. Panel (a) evaluates accuracy at each recurrent loop without using an adaptive readout. Panel (b) shows the average learned exit distribution over recurrent loops, stratified by task difficulty.

Across three seeds, the geometric fixed-prior trajectory paired with logit margin achieves a favorable frontier, requiring only 1.4–1.6 average loops to reach 95% – −99% accuracy. Learned gates consistently require greater depth, indicating that trajectory quality, not gate parameterization alone, determines adaptive-compute performance. Additional gate comparisons across all prior settings are reported in Appendix D, with separate full sweeps for the linear gate in Appendix D.1 and the MLP gate in Appendix D.2.

5.3 The training objective shapes trajectories through depth-weighted losses

We now examine why these frontiers differ. Because the learned exit distribution also weights per-depth training losses, it shapes the recurrent trajectory in addition to defining the inference-time stopping rule. We therefore compare forced exits, which evaluate every recurrent loop without adaptive stopping.

Figure 5 compares forced-exit trajectories and learned exit distributions for the geometric prior with $\lambda = 0.3$. The forced-exit curves show that the training objective changes the per-depth accuracy distribution. Fixed-prior training produces useful intermediate predictions early in the trajectory, including on harder examples. The trained gate trajectories show different depth distributions, indicating that the gate objective changes



(a) Frozen fixed-prior trajectory ($\lambda = 0.3$).

(b) Frozen learned-gate trajectory ($\lambda = 0.3$, $\beta = 0.1$).

Figure 6: Post-hoc gate training on frozen MANO trajectories: we freeze the backbone and LM head, fit new linear or MLP exit gates, and compare against simple trajectory readouts.

the trajectory rather than only changing the exit rule. The learned exit distributions are difficulty-aware, assigning harder examples more probability mass at later loops. However, the forced-exit curves show that allocating later exits alone is insufficient: the trajectory must also produce accurate intermediate states.

These results explain why adaptive-compute performance depends on both trajectory formation and exit selection. Learned gates entangle these two roles: the same distribution $q(\cdot \mid x)$ both weights the losses that shape the trajectory and defines the native stopping rule used at inference time. Fixed-prior training separates these roles by shaping the trajectory with input-independent depth supervision, after which simple readouts can select exits. The multiseed frontier summaries in Tables 1 and 2 are consistent with this trajectory-level diagnosis. Objectives that produce stronger intermediate forced exits also support stronger adaptive frontiers. Full forced-exit trajectories and learned exit-distribution diagnostics across prior settings are provided in Appendix E.

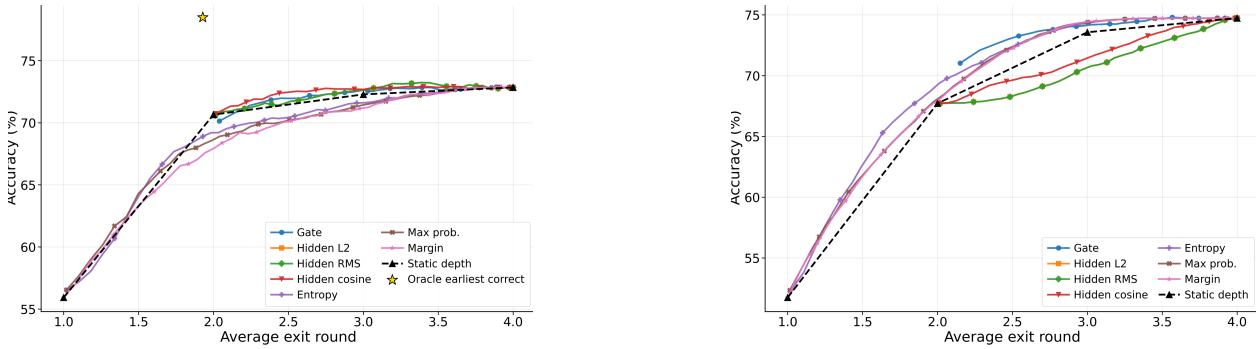
5.4 Post-hoc fitting of a learned gate over a pretrained trajectory

A remaining ambiguity in the gate results is whether the gate readout itself is weak, or whether the problem arises from jointly training the gate with the recurrent trajectory. To separate these effects, we freeze pretrained MANO trajectories and train a new exit gate on top of the frozen hidden states. In this setting, the gate can only act as a readout: it has no path to modify the recurrent states, the LM head, or the per-depth predictions.

We consider two post-hoc gate objectives. The first is the same PonderNet-style objective used for learned gates, but with the model fixed. The new gate defines an exit distribution over depths and is trained to minimize the expected per-depth cross-entropy plus the same prior regularizer. The second is a marginal-utility objective inspired by Ouro Stage-II training. For each adjacent pair of recurrent depths, we compute the realized loss improvement from continuing one more step and train the gate to predict whether continuation is worthwhile. In both cases, we train either a linear gate or a two-layer MLP gate, evaluate with the same validation-selected threshold protocol as the other readouts, and report the full linear/MLP results across fixed-prior and learned-gate trajectories in Appendix H.

Figure 6 shows two representative frozen trajectories. On the fixed-prior geometric trajectory, post-hoc gates recover strong early-exit behavior, reaching near-final-loop accuracy at much lower average depth. This shows that the gate class is capable of representing useful stopping policies when trajectory formation is separated from readout learning. However, the simple readouts remain competitive, indicating that much of the stopping signal is already exposed in the prediction trajectory.

On the frozen trajectory produced by joint learned-gate training, fitting a new post-hoc gate does not improve performance. The newly trained gates largely track the same frontier as the native gate and remain limited by the quality of the underlying trajectory. Across these frozen-trajectory experiments, the PonderNet-style post-hoc objective also gives a stronger readout than the marginal-utility objective, which tends to exit later and does not improve the frontier. This suggests that the weak gate result is not simply due to an



(a) Ouro-1.4B, HellaSwag

(b) Ouro-2.6B, MMLU

Figure 7: Large-scale Ouro readout comparisons. Each panel plots benchmark accuracy against average loops. The pretrained ponder gate is compared with confidence and convergence readouts applied to the same recurrent trajectories.

Table 3: Ouro readout comparison across benchmarks. For each model and benchmark, we compare the pretrained gate against the best post-hoc readout, both validation-selected, on the held-out split.

Model	Benchmark	Gate accuracy	Gate avg. loops	Post-hoc readout	Post-hoc accuracy	Post-hoc avg. loops
Ouro-1.4B	MMLU	67.6	3.75	Hidden cosine	67.3	3.86
Ouro-1.4B	ARC-Easy	82.9	2.72	Hidden cosine	83.2	2.69
Ouro-1.4B	ARC-Challenge	59.8	2.65	Max prob.	59.5	3.12
Ouro-1.4B	OpenBookQA	42.8	3.13	Hidden cosine	41.6	2.95
Ouro-1.4B	HellaSwag	72.8	3.94	Hidden L2	72.9	3.13
Ouro-1.4B	CommonsenseQA	74.7	3.43	Hidden cosine	74.9	3.93
Ouro-2.6B	MMLU	74.4	3.33	Max prob.	74.7	3.30
Ouro-2.6B	ARC-Easy	87.2	3.68	Hidden cosine	87.2	3.92
Ouro-2.6B	ARC-Challenge	66.6	3.38	Margin	65.8	2.90
Ouro-2.6B	OpenBookQA	43.4	3.78	Hidden cosine	41.6	2.63
Ouro-2.6B	HellaSwag	77.5	2.93	Hidden cosine	77.6	3.69
Ouro-2.6B	CommonsenseQA	81.1	2.49	Entropy	80.5	2.37

under-trained or underexpressive native gate. Instead, the main issue is the trajectory induced by the joint objective. Thus, learned gates are best understood not only as exit readouts, but also as training mechanisms that shape which recurrent states become useful.

5.5 From controlled tasks to large-scale checkpoints

We ask whether the same trajectory–readout pattern appears in larger looped language models. Figure 7 shows representative Pareto curves for the two model sizes. Because these checkpoints are fixed, we vary only the readout, not the trajectory. Unlike MANO, we cannot retrain under a different prior, so the trajectory is whatever pretraining produced. On Ouro-1.4B HellaSwag, post-hoc readouts can match or improve the gate's compute–accuracy tradeoff; on Ouro-2.6B MMLU, the ponder gate remains competitive, but post-hoc readouts still recover similar or better tradeoffs. The best readout varies across model size and benchmark, indicating that the learned gate is not necessarily learning a criterion superior to classical heuristics.

Table 3 summarizes the full benchmark grid numerically. The post-hoc readouts are highly competitive with the pretrained ponder gate across both model sizes. In five of the twelve model–benchmark pairs, the selected post-hoc readout achieves higher held-out accuracy than the gate, though several of these differences are small and reported on single runs. In seven pairs, it uses fewer average recurrent loops, and in three pairs, it improves both accuracy and compute. Several of the more promising post-hoc readouts are simple confidence or convergence criteria, such as hidden-state cosine similarity, hidden-state displacement, maximum probability, entropy, and logit margin.

Across the full evaluation, the qualitative pattern is consistent with the controlled experiments: recurrent trajectories contain stopping signals that simple confidence or convergence readouts can expose. In the controlled setting we change the training objective and show that trajectory formation matters; in the large-scale setting the trajectory is fixed by pretraining, yet alternative readouts still recover competitive early-exit behavior. This supports treating adaptive computation as a trajectory–readout problem rather than a

Table 4: Latency measurements on MANO and Ouro-1.4B. (a) Ouro-1.4B across evaluated benchmarks; for each dataset we report the final-loop baseline, the pretrained gate, and the best heuristic readout. (b) Representative fixed-prior and learned-gate models on MANO; the fixed-prior model uses a geometric prior with $\lambda = 0.3$, the learned-gate model uses the same prior with $\beta = 0.1$.

(a) Ouro-1.4B

Dataset	Readout	Acc. (%)	Avg. loops	ms/ex.	Speedup
OpenBookQA	Final loop	43.0	4.00	99.1	1.00×
OpenBookQA	Gate	42.0	2.80	69.3	1.43×
OpenBookQA	Hidden cos.	39.4	2.01	49.8	1.99×
ARC-Easy	Final loop	84.0	4.00	170.9	1.00×
ARC-Easy	Gate	82.4	2.53	108.0	1.58×
ARC-Easy	Margin	82.8	2.98	127.3	1.34×
ARC-Challenge	Final loop	60.2	4.00	616.2	1.00×
ARC-Challenge	Gate	59.5	2.65	408.8	1.51×
ARC-Challenge	Hidden L2	59.7	2.86	440.3	1.40×
CommonsenseQA	Final loop	75.6	4.00	79.9	1.00×
CommonsenseQA	Gate	75.1	3.23	64.4	1.24×
CommonsenseQA	Margin	75.4	3.31	66.1	1.21×

(b) MANO

Model	Readout	Acc. (%)	Avg. loops	ms/ex.	Speedup
Fixed prior	Final loop	100.0	6.00	0.066	1.00×
Fixed prior	Top-1 conf.	100.0	1.26	0.049	1.36×
Fixed prior	Logit margin	100.0	1.26	0.051	1.29×
Fixed prior	Pred. KL	99.9	2.32	0.060	1.11×
Fixed prior	Hidden disp.	99.5	5.80	0.074	0.90×
Learned gate	Final loop	76.1	6.00	0.066	1.00×
Learned gate	Gate	75.4	1.59	0.053	1.24×

property of the learned gate alone. Full per-benchmark Pareto curves and reasoning-tuned Ouro-Thinking diagnostics are in Appendices G and I.

5.6 Average exit depth corresponds to latency savings

Our main compute metric is average exit depth, which directly counts applications of the shared recurrent block. We additionally evaluate whether this proxy corresponds to latency savings once readout computation and adaptive-exit overhead are included. Tables 4b and 4a report measured end-to-end latency on representative models trained on the MANO task, as well as on the Ouro model. For these experiments, we use an NVIDIA RTX 5070 Ti GPU.

In the controlled MANO setting, post-hoc readouts on the fixed-prior trajectory reduce measured latency while preserving accuracy. For the geometric fixed-prior model with $\lambda = 0.3$, top-1 confidence and logit margin maintain 100.0% accuracy while reducing average depth from 6.0 to 1.26 loops, giving 1.36× and 1.29× speedups; predictive KL preserves near-perfect accuracy at 1.11×. These are smaller than the raw loop-count reduction, as expected when the small recurrent block is not the only runtime cost, but they confirm that early exits yield real savings; the hidden-displacement readout shows the opposite case, exiting late and adding overhead. The learned gate reduces average depth from 6.0 to 1.59 loops (1.24×) but operates at lower accuracy (around 75%), so here the fixed-prior trajectory with simple confidence readouts gives a more favorable accuracy–latency tradeoff.

The Ouro estimates provide the same latency-side sanity check at larger scale. Across all evaluated benchmarks, reducing the average loops produces substantial estimated latency reductions relative to the final-loop baseline. The pretrained gate consistently reduces estimated latency, with speedups ranging from 1.24× on CommonsenseQA to 1.58× on ARC-Easy. Post-hoc readouts produce similar estimated latency reductions when they exit at comparable average loops. Thus, the average exit-loops metric used in the Pareto curves corresponds to a practical latency proxy for Ouro as well.

6 Conclusion

In this systematic evaluation, we have shown that adaptive depth in looped Transformers is best understood as a trajectory–readout problem, not a gate-learning problem. The training objective shapes the recurrent trajectory, while a separate readout decides when to stop. Learned halting gates conflate these roles, using one exit distribution to both weight training losses and stop at inference. Across controlled MANO and parity experiments, and large-scale Ouro-1.4B and 2.6B checkpoints, we find that simple post-hoc readouts on well-shaped trajectories match or outperform learned gates, and that gate failures trace to the induced trajectory rather than gate expressivity. Where prior work focuses on making recurrent depth dynamic, our contribution is diagnostic: isolating the roles of trajectory formation and exit selection in determining adaptive-compute performance. We hope this trajectory–readout perspective motivates future work on alternative pretraining objectives and looped Transformer architectures.

References

Zeyuan Allen-Zhu. Physics of language models: Part 4.1, architecture design and the magic of canon layers, 2025. URL https://arxiv.org/abs/2512.17351.

Cem Anil, Yuhuai Wu, Anders Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models, 2022. URL https://arxiv.org/abs/2207.04901.

Sangmin Bae, Jongwoo Ko, Hwanjun Song, and Se-Young Yun. Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5910–5924, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.362. URL https://aclanthology.org/2023.emnlp-main.362/.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation, 2025. URL https://arxiv.org/abs/2507.10524.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models, 2019. URL https://arxiv.org/abs/1909.01377.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder, 2021. URL https://arxiv.org/abs/2107.05407.

Mar Gonzàlez I Català, Haitz Sáez de Ocáriz Borde, George D. Montañez, and Pietro Liò. The stepwise informativeness assumption: Why are entropy dynamics and reasoning correlated in llms?, 2026. URL https://arxiv.org/abs/2604.06192.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers, 2019. URL https://arxiv.org/abs/1807.03819.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer, 2020. URL https://arxiv.org/abs/1910.10073.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.681. URL http://dx.doi.org/10.18653/v1/2024.acl-long.681.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *ArXiv*, abs/2101.03961, 2021. URL https://api.semanticscholar.org/CorpusID:231573431.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach, 2025. URL https://arxiv.org/abs/2502.05171.

Angeliki Giannou, Shashank Rajput, Jy yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers, 2023. URL https://arxiv.org/abs/2301.13196.

Alex Graves. Adaptive computation time for recurrent neural networks, 2017. URL https://arxiv.org/abs/1603.08983.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space, 2025. URL https://arxiv.org/abs/2412.06769.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Dynabert: Dynamic bert with adaptive width and depth, 2020. URL https://arxiv.org/abs/2004.04037.

Wei Huang, Yizhe Xiong, Xin Ye, Zhijie Deng, Hui Chen, Zijia Lin, and Guiguang Ding. Fast quiet-star: Thinking without thought tokens, 2025. URL https://arxiv.org/abs/2505.17746.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation, 2026. URL https://arxiv.org/abs/2602.11451.

Harsh Kohli, Srinivasan Parthasarathy, Huan Sun, and Yuekun Yao. Loop, think, & generalize: Implicit reasoning in recurrent-depth transformers, 2026. URL https://arxiv.org/abs/2604.07822.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020. URL https://arxiv.org/abs/1909.11942.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering, 2018. URL https://arxiv.org/abs/1809.02789.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models, 2024. URL https://arxiv.org/abs/2404.02258.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers, 2025. URL https://arxiv.org/abs/2502.17416.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling, 2022. URL https://arxiv.org/abs/2207.07061.

Shixiang Song, He Li, Zitong Wang, Boyi Zeng, Feichen Song, Yixuan Wang, Zhiqin John Xu, Ziwei He, and Zhouhan Lin. Adaponderlm: Gated pondering language models with token-wise adaptive depth, 2026. URL https://arxiv.org/abs/2603.01914.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge, 2019. URL https://arxiv.org/abs/1811.00937.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. Deebert: Dynamic early exiting for accelerating bert inference, 2020. URL https://arxiv.org/abs/2004.12993.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms, 2024. URL https://arxiv.org/abs/2311.12424.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D. Goodman. Quiet-star: Language models can teach themselves to think before speaking, 2024. URL https://arxiv.org/abs/2403.09629.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019. URL https://arxiv.org/abs/1905.07830.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit, 2020. URL https://arxiv.org/abs/2006.04152.

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.

A Model Background

This appendix provides additional background on the architectures studied in this work. We first summarize the looped Transformer formulation used throughout the paper, then review the Ouro architecture and training procedure.

A.1 Looped Transformer Architecture

A looped Transformer replaces a stack of independently parameterized Transformer layers with repeated applications of a shared recurrent block. Let $h_0 = E(x) \in \mathbb{R}^{M\times d}$ denote the embedded input sequence. A shared block $\mathcal{F}_\theta : \mathbb{R}^{M\times d} \to \mathbb{R}^{M\times d}$, typically consisting of one or more Transformer layers, is then applied recurrently:

$$h_t = \mathcal{F}_\theta(h_{t-1}), \qquad t = 1, \ldots, T.$$

Here T is the maximum number of recurrent loops, and the same parameters θ are reused at every depth. Thus increasing T increases the amount of test-time computation without increasing the number of learned parameters. This parameter sharing distinguishes looped Transformers from standard depth scaling, where each additional layer introduces a new set of parameters.

The shared block can be written abstractly as a residual Transformer update,

$$z_t = h_{t-1} + \mathrm{Attn}_\theta(\mathrm{LN}(h_{t-1})), \qquad h_t = z_t + \mathrm{MLP}_\theta(\mathrm{LN}(z_t)),$$

or as a short stack of such layers. In either case, the important property for this paper is that the same update map is iterated multiple times. The resulting sequence

$$\tau_\theta(x) = (h_1, \ldots, h_T)$$

is the recurrent trajectory. Applying the same prediction head to every recurrent state gives per-depth logits

$$o_t = W_{\mathrm{out}} h_t,$$

and corresponding per-depth losses $\ell_t(x, y)$. These intermediate logits make it possible to evaluate the model at any recurrent loop, rather than only at the final loop.

Adaptive depth in this architecture amounts to choosing a stopping depth

$$d(x) \in \{1, \ldots, T\}$$

and returning the prediction from $h_{d(x)}$. A fixed-depth baseline chooses the same d for every input. A learned gate predicts a distribution over stopping depths from the recurrent states. A post-hoc readout instead computes a stopping signal from quantities already present in the trajectory, such as entropy, maximum probability, logit margin, predictive change, or hidden-state displacement. All of these methods operate on the same sequence of states $(h_1, \ldots, h_T)$. They differ only in how the stopping depth is selected.

This architecture is especially useful for separating trajectory formation from exit selection. The training objective determines which intermediate states receive supervision and therefore which parts of the trajectory

become predictive. The readout determines which of those states is used at inference time. Learned halting objectives couple these two roles because the gate distribution both weights the per-depth training losses and defines the inference-time stopping rule. Fixed-prior training removes the learned input-dependent gate from the training objective, allowing us to study how the trajectory is shaped independently of the adaptive readout.

In our MANO and parity experiments, we train looped Transformers from scratch with $T = 6$ recurrent loops and apply the same language-model head at every loop. In the Ouro experiments, we use pretrained looped language-model checkpoints and keep the recurrent trajectory fixed, varying only the readout used to select an exit. This common architectural view lets us compare learned gates, fixed-depth exits, and post-hoc trajectory readouts within the same trajectory–readout framework.

A.2 The Ouro Model Family

Ouro (Zhu et al., 2025) implements a Looped Language Model by repeatedly applying a shared stack of L standard decoder-only Transformer layers (multi-head attention with RoPE, SwiGLU feed-forward blocks, sandwich RMSNorm) up to $T_{\max}$ times, rather than stacking L *distinct* layers as in a conventional Transformer. The released checkpoints are Ouro-1.4B (24 layers) and Ouro-2.6B (48 layers, obtained by doubling and further training the 24-layer stack), both with hidden size 2048 and $T_{\max} = 4$ trained recurrent steps, pre-trained on 7.7T tokens spanning web, math, code, and long-context data across four stages.

At each recurrent step t, a linear exit gate produces a halt probability $\lambda_t(x) = \sigma(\mathrm{Linear}_\phi(h^{(t)}))$, which induces a discrete exit distribution $p_\phi(t \mid x)$ over $\{1, \ldots, T_{\max}\}$ in the same way as the gating mechanism in Section 3.2 of this paper. Ouro's gate is trained in two stages. **Stage I** jointly optimizes the gate and the backbone with an entropy-regularized objective,

$$\mathcal{L} = \sum_{t=1}^{T_{\max}} p_\phi(t \mid x)\, \mathcal{L}^{(t)} - \beta\, H\big(p_\phi(\cdot \mid x)\big),$$

which is equivalent to a KL penalty toward a *uniform* prior over exit depths (chosen to avoid biasing the gate toward any particular depth before depth-dependent difficulty is learned). **Stage II** then freezes the backbone and fine-tunes only the gate using a marginal-utility signal. For each adjacent pair of recurrent steps, a detached loss-improvement is calculated as follows:

$$I_i^{(t)} = \max(0,\, \mathcal{L}_{i,\mathrm{stop}}^{(t-1)} - \mathcal{L}_{i,\mathrm{stop}}^{(t)})$$

This loss improvement term is converted into a soft continue/exit label and used as a binary cross-entropy target for the gate. This Stage-II objective is the direct inspiration for the marginal-utility post-hoc gate we fit on frozen trajectories in Appendix H.

Ouro's authors report that the resulting 1.4B and 2.6B checkpoints match dense Transformer baselines 2–3× larger across general, math, and coding benchmarks, and that gains stem primarily from improved knowledge *manipulation* rather than increased knowledge capacity. They also report that post-training RLVR attempts on Ouro provided limited gains, attributing this in part to the mismatch between fast fixed-depth rollout infrastructure (e.g., vLLM/SGLang) and the model's variable-depth, gate-driven execution path. This difficulty in optimizing around the gate at the RL stage is part of our motivation for asking, in the present work, whether the gate itself is the right object to optimize versus the trajectory it induces. We use the publicly released 1.4B and 2.6B base checkpoints without modifying or retraining them.

B Experimental Details

This appendix summarizes dataset construction, model architecture, optimization settings, threshold selection, and benchmark protocols used in the experiments.

B.1 MANO Task

MANO consists of modular arithmetic expressions over integers modulo 23. Expressions are represented in prefix notation and contain up to 10 operations. We balance examples across operation-count bins and use operation count as the difficulty variable in trajectory diagnostics. Unlike MANO setups that provide the expression length as an input token, our implementation does not include a length token, in order to avoid giving the model a shortcut towards compute allocation.

B.2 MANO Architecture

All MANO experiments use the same looped Transformer backbone with 4 layers, 4 attention heads, hidden embedding size 512, block size 32, no dropout, and bias terms enabled. The model uses tied input embeddings and language-model head weights. The maximum recurrent depth is T=6. Learned exit gates map the recurrent hidden state to a scalar halt logit.

B.3 MANO Training

All MANO variants are trained with AdamW using learning rate 5×10^{-4}, $\beta = (0.9, 0.98)$, weight decay 0.1, batch size 1024, and gradient clipping at 1.0. We use a warmup-cosine learning-rate schedule with 2000 warmup steps and minimum learning rate 5×10^{-5}. Models are trained for 40,000 steps using bf16 mixed precision.

B.4 MANO Variants

Fixed-prior models use either a uniform prior or truncated geometric priors with $\lambda \in \{0.2, 0.3, 0.5, 0.7\}$. Learned linear-gate models use either a uniform prior or the same geometric priors, and sweep the KL regularization strength β. For the uniform prior, we sweep $\beta \in \{0.001, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5\}$. For geometric priors, we sweep $\beta \in \{0.01, 0.05, 0.1\}$. MLP-gate models replace the linear halt head with a two-layer feedforward gate. In the case of the MLP gate, we sweep $\beta \in \{0.05, 0.1\}$ for the geometric prior, while for the uniform prior we use $\beta \in \{0.001, 0.01, 0.1, 0.2\}$.

B.5 Threshold Selection

For each post-hoc readout, we construct a 51-point threshold grid from validation-set readout values using empirical quantiles between the 0.5th and 99.5th percentiles. The resulting grid is fixed before evaluation and then applied to the test set. Learned-gate readouts use a fixed data-independent grid of cumulative probability thresholds $\alpha \in [0.01, 1]$. Fixed-depth baselines require no threshold selection.

B.6 Post-Hoc Readout Definitions

Let $z_t(x)$ denote the task logits at recurrent loop t, $p_t(x) = \mathrm{softmax}(z_t(x))$, and let $r_t(x)$ denote the pooled hidden representation used for hidden-state readouts. For confidence readouts, we use

$$
\begin{aligned}
H_t(x) &= -\sum_k p_t(k \mid x) \log p_t(k \mid x), \\
C_t(x) &= \max_k p_t(k \mid x), \\
M_t(x) &= z_{t,(1)}(x) - z_{t,(2)}(x),
\end{aligned}
$$

where $z_{t,(1)}(x)$ and $z_{t,(2)}(x)$ are the largest and second-largest logits at loop t. For convergence readouts, we use

$$
\begin{aligned}
\Delta_{\mathrm{KL}}(t; x) &= D_{\mathrm{KL}}(p_t(x) \,\|\, p_{t-1}(x)), \\
\Delta_{\mathrm{logit}}(t; x) &= \|z_t(x) - z_{t-1}(x)\|_2, \\
\Delta_{\mathrm{hid}}(t; x) &= \|r_t(x) - r_{t-1}(x)\|_2.
\end{aligned}
$$

A confidence readout exits once the model is sufficiently confident. For top-1 confidence and logit margin, we use $d_\gamma(x) = \min\{t : S_t(x) \geq \gamma\}$, where $S_t \in \{C_t, M_t\}$. For entropy, where lower values indicate higher confidence, we instead use $d^H_\gamma(x) = \min\{t : H_t(x) \leq \gamma\}$.

In contrast to confidence readouts, a convergence readout exits once the corresponding change signal falls below a threshold: $d_\epsilon(x) = \min\{t \geq 2 : \Delta(t; x) \leq \epsilon\}$. All readouts default to T if the threshold is never reached.

B.7 Ouro Models and Benchmarks

We evaluate ByteDance Ouro-1.4B and Ouro-2.6B without further training. We use six standard benchmarks: MMLU with 5-shot prompting, ARC-Challenge with 25-shot prompting, ARC-Easy with 8-shot prompting, OpenBookQA with 0-shot prompting, CommonsenseQA with 10-shot prompting, and HellaSwag with 10-shot prompting.

B.8 Ouro Threshold Selection

For benchmarks with labeled test sets, heuristic threshold grids are selected on the validation split and evaluated on the test split. For CommonsenseQA and HellaSwag, where labeled test sets are unavailable, we split the labeled validation set into a 20% validation subset for threshold selection and an 80% held-out test subset for evaluation using a fixed random seed. As in MANO, gate thresholds use a fixed data-independent grid over cumulative gate probability, and fixed-depth baselines require no threshold selection.

C Fixed-Prior Trajectory Analysis

This appendix collects the full fixed-prior analyses supporting Section 5.1. We report the complete geometric-prior Pareto sweep across λ in Figure 8, per-difficulty trajectory diagnostics for the uniform and geometric priors in Appendices C.2 and C.3, and the difficulty extrapolation study in Appendix C.4. We use the number of operations as the difficulty variable and evaluate every recurrent loop with forced exits, so the reported signals reflect the trajectory itself rather than any adaptive stopping rule.

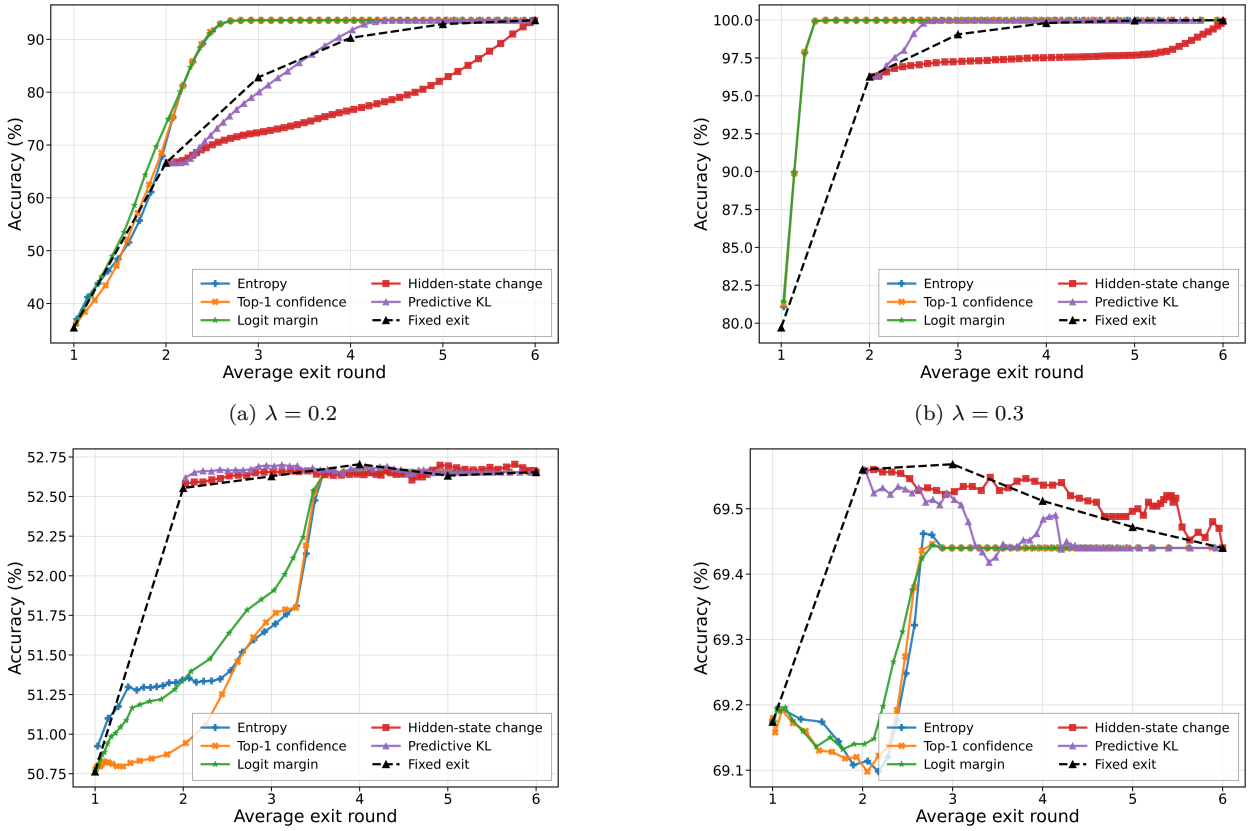
C.1 Pareto Curves for the Geometric Prior over Multiple λ Values

Figure 8 sweeps the truncated geometric prior across $\lambda \in \{0.2, 0.3, 0.5, 0.7\}$, plotting threshold-swept post-hoc readouts for each prior. The sweep reveals a clear optimum in early-depth weighting. Moderate values, and $\lambda = 0.3$ in particular, push useful predictions early in the trajectory and show a strong low-depth frontier. As λ grows, the prior concentrates almost all loss weight on the first loop, which degrades the later recurrent states and flattens the accuracy ceiling, so the trajectory no longer benefits from additional computation. The uniform prior sits at the opposite extreme, supervising all depths equally and producing useful early exits only at higher average depth. This is consistent with the main-text finding that trajectory formation, not just the readout, shapes the achievable compute–quality tradeoff.

C.2 Uniform Prior Difficulty Convergence

Figure 9 shows per-difficulty trajectory diagnostics for the uniform fixed-prior model, grouping examples by the number of operations in the input sequence. Prediction-space signals carry a clean difficulty ordering. Entropy decreases and logit margin increases with recurrent depth, and at every loop both quantities separate by operation count, with harder expressions remaining higher-entropy and lower-margin for more loops. Predictive KL between consecutive predictions follows the same trend, though with more overlap between adjacent difficulty bins. Hidden-state displacement is noisier. Harder examples tend to keep moving in representation space for longer, but the separation by operation count is weaker and less monotonic than in prediction space.

Table 5 summarizes this ordering across seeds using signed Spearman correlations.



(a) $\lambda = 0.2$ (b) $\lambda = 0.3$

(c) $\lambda = 0.5$ (d) $\lambda = 0.7$

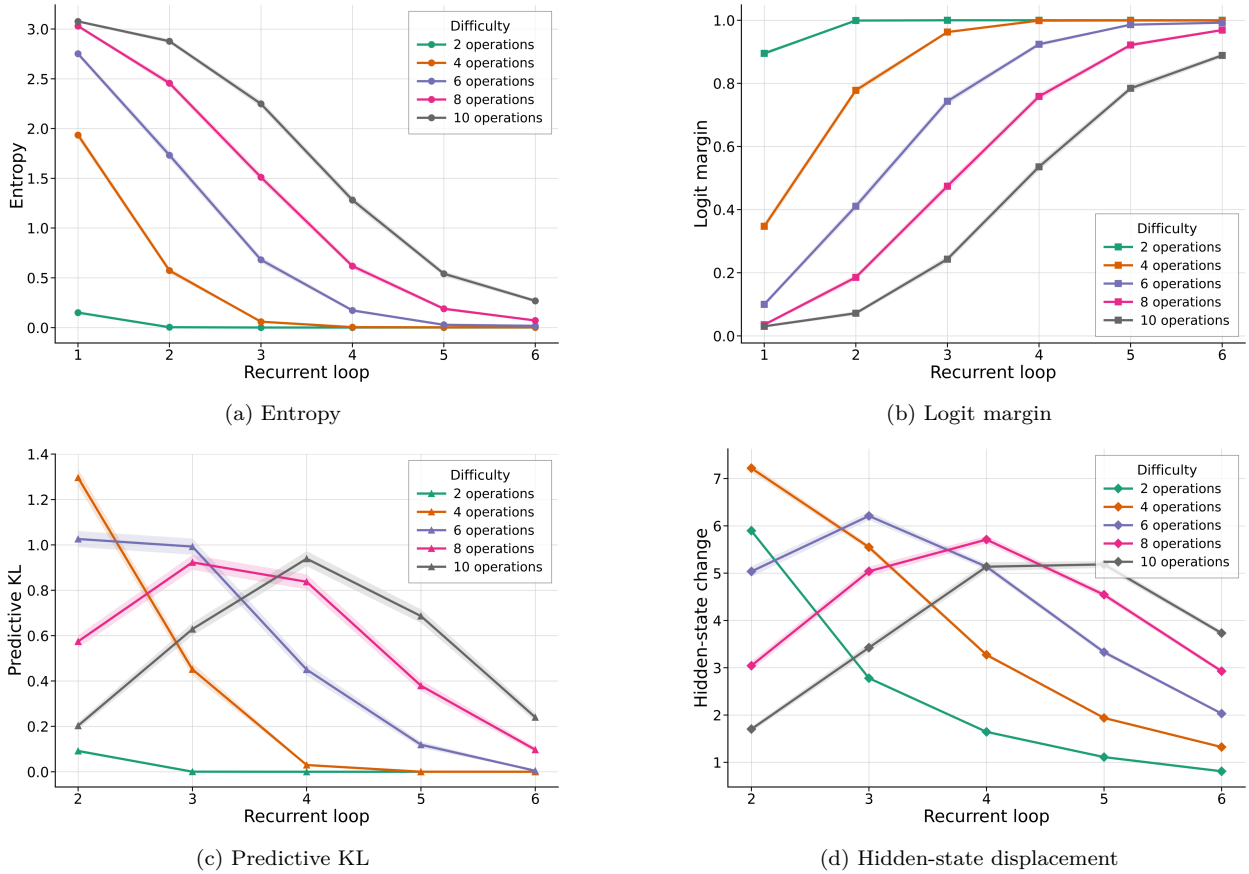
Figure 8: Full geometric fixed-prior sweep. Each panel shows threshold-swept post-hoc readouts for a different truncated geometric prior. Moderate early-depth weighting yields useful early exits, while overly aggressive early weighting can degrade the trajectory.

Table 5: Difficulty ordering of the uniform fixed-prior MANO trajectory across three seeds. For each recurrent loop, we compute a signed Spearman correlation between operation count and the trajectory signal, then average across loops and seeds. The sign is chosen so that positive values mean harder examples appear harder: higher entropy, lower logit margin, larger predictive KL, or larger hidden-state displacement.

Signal	Signed Spearman ρ
Entropy	1.00 ± 0.00
Logit margin	1.00 ± 0.00
Predictive KL	0.80 ± 0.11
Hidden-state displacement	0.35 ± 0.48

C.3 Geometric Prior Difficulty Convergence

Figure 10 repeats the diagnostic for the geometric fixed-prior model with $\lambda = 0.3$. Relative to the uniform prior, the geometric prior concentrates supervision on earlier loops and shifts the useful prediction-space signals earlier in the trajectory. Easier examples reach confident, low-entropy and high-margin states within the first one or two loops, while harder examples continue to refine for several more loops. The difficulty ordering is preserved across all four signals, and hidden-state displacement is visibly less noisy than under the uniform prior. Table 6 confirms this numerically. Entropy and logit margin remain near-perfectly ordered ($\rho = 0.98$), predictive KL improves to $\rho = 0.87$, and hidden-state displacement rises to $\rho = 0.69 \pm 0.34$, roughly double the uniform-prior value.



(a) Entropy

(b) Logit margin

(c) Predictive KL

(d) Hidden-state displacement

Figure 9: Difficulty-stratified trajectory diagnostics for the uniform fixed-prior model. Examples are grouped by the number of operations required by the task, and each panel plots a trajectory signal across recurrent loops. Prediction-space signals such as entropy, logit margin, and predictive KL show more legible difficulty structure than hidden-state displacement.

Table 6: Difficulty ordering of the geometric fixed-prior MANO trajectory ($\lambda = 0.3$) across three seeds. Conventions follow Table 5: positive signed Spearman correlation means harder examples appear harder.

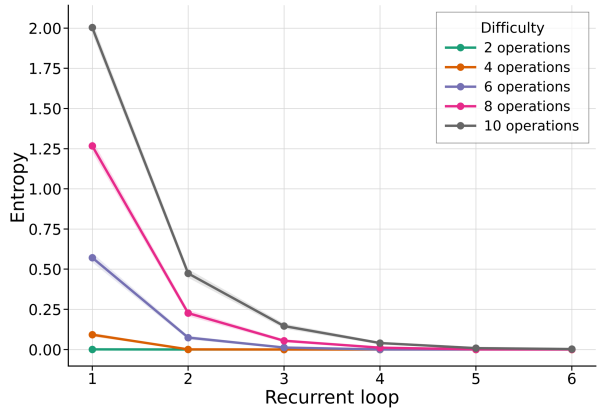
Signal	Signed Spearman ρ
Entropy	0.98 ± 0.03
Logit margin	0.98 ± 0.03
Predictive KL	0.87 ± 0.20
Hidden-state displacement	0.69 ± 0.34

C.4 Difficulty Extrapolation on MANO

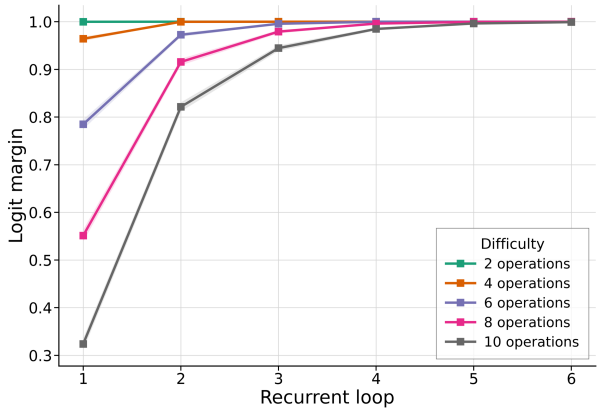
MANO provides a direct measure of input difficulty through the number of operations in each expression. We use this structure to test whether the difficulty-aware trajectory signals observed in Section 5.1 persist beyond the training difficulty range. We take our fixed-prior models and evaluate them on expressions with more operations. Thus, the harder examples are out of distribution with respect to training difficulty.

This experiment is not a length-generalization benchmark. Accuracy may degrade once expressions exceed the training range. Instead, we use it as a trajectory diagnostic. If the recurrent states encode a meaningful notion of difficulty, then harder out-of-distribution expressions should remain uncertain for more recurrent loops, develop confidence more slowly, and show slower convergence across hidden states.

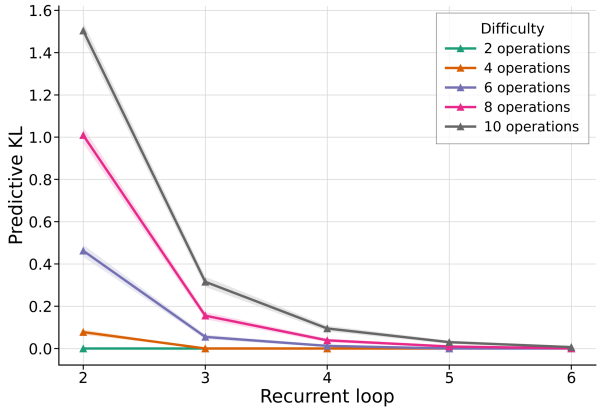
Figures 11 and 12 show this behavior for geometric and uniform fixed-prior trajectories. The geometric-prior trajectory gives a clear separation by operation count. The uniform trajectory is slightly noisier, but shows the same pattern. Across both priors, expressions with more operations remain higher-entropy for longer, develop logit margin more slowly, and tend to exhibit larger hidden-state changes across recurrent depth.



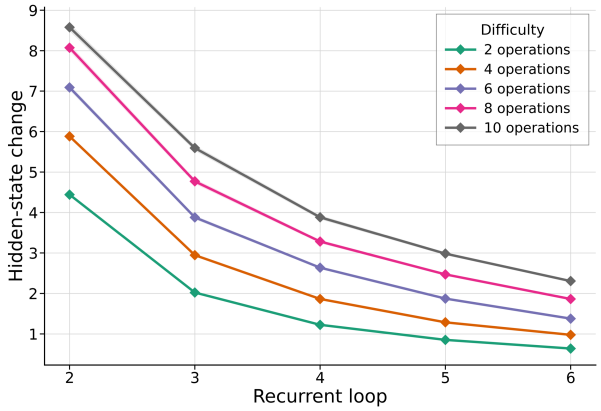
(a) Entropy



(b) Logit margin

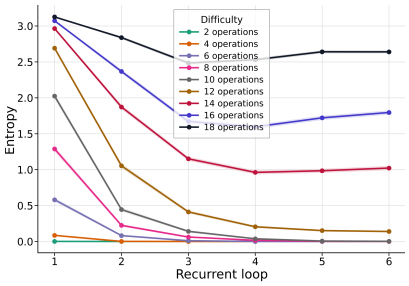


(c) Predictive KL

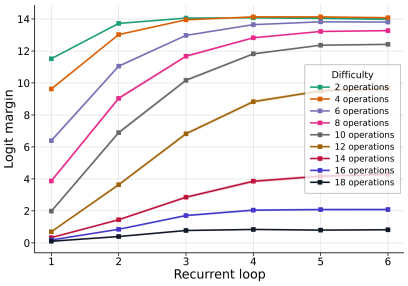


(d) Hidden-state displacement

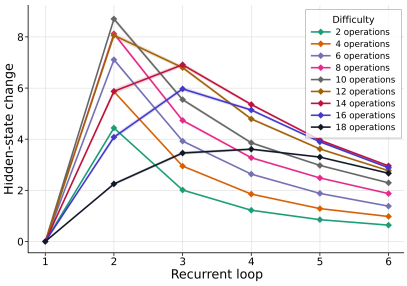
Figure 10: Difficulty-stratified trajectory diagnostics for the geometric fixed-prior model with $\lambda = 0.3$. Compared with the uniform prior, the geometric prior shifts useful prediction-space signals earlier in the recurrent trajectory. Easier examples reach confident and stable states earlier, while harder examples continue changing for more loops.



(a) Entropy.

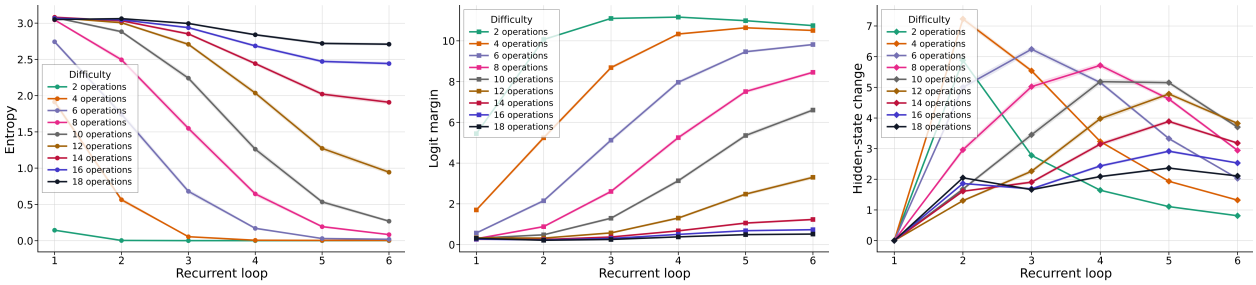


(b) Logit margin.



(c) Hidden-state displacement.

Figure 11: Difficulty extrapolation diagnostics for the geometric fixed-prior MANO model with $\lambda = 0.3$. The model is trained on expressions with at most 10 operations and evaluated on harder expressions beyond that range. Curves are grouped by operation count. Harder expressions remain uncertain for more recurrent loops, develop logit margin more slowly, and show larger hidden-state changes across depth.



(a) Entropy. (b) Logit margin. (c) Hidden-state displacement.

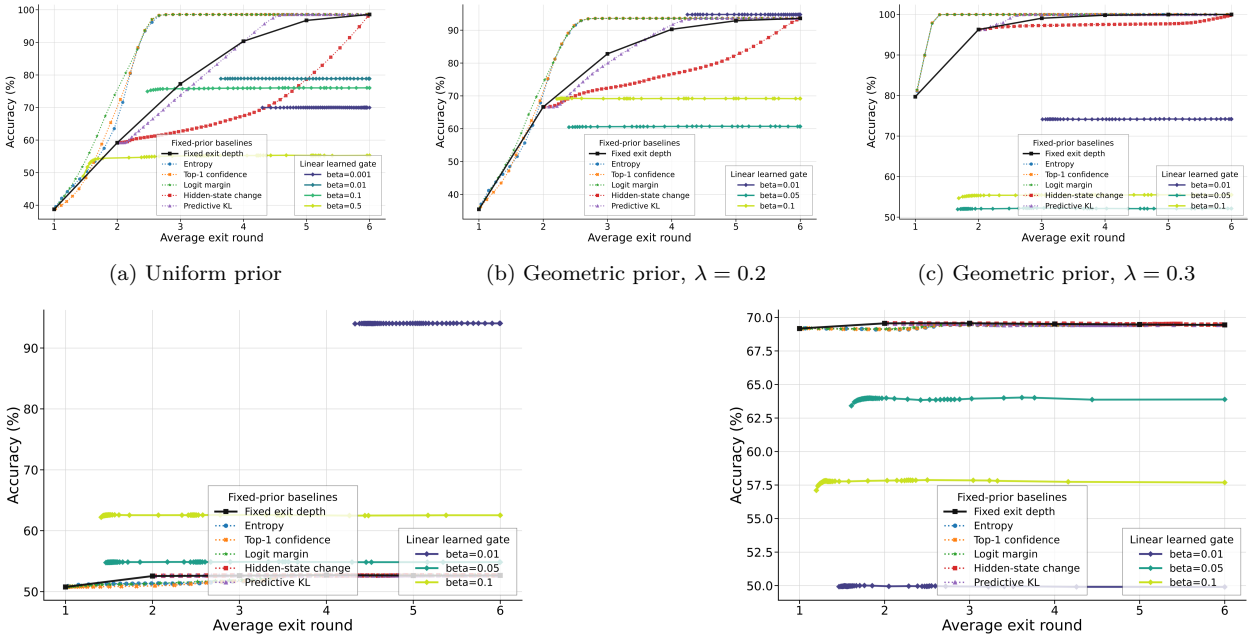
Figure 12: Difficulty extrapolation diagnostics for the uniform fixed-prior MANO model. The model is trained on expressions with at most 10 operations and evaluated on harder expressions beyond that range. The separation by operation count is less pronounced than under the geometric prior, but harder examples still tend to remain less confident and less converged for longer.

This supports the interpretation that the difficulty-aware structure in the trajectory is not only a calibration artifact of the training distribution.

D Learned Gate Comparison

This appendix reports the full gate and fixed-prior comparison across every prior setting, expanding the representative panels in Section 5.2. For each prior, we sweep the learned gate's regularization strength β and compare its native cumulative-threshold readout against post-hoc readouts applied to the fixed-prior trajectory. Appendix D.1 covers the linear gate and Appendix D.2 covers the MLP gate. Across both gate classes and all priors, the same pattern holds: fixed-prior trajectories paired with simple confidence readouts provide a strong low-depth frontier, and increasing gate capacity narrows but does not close the gap.

D.1 Linear Gate



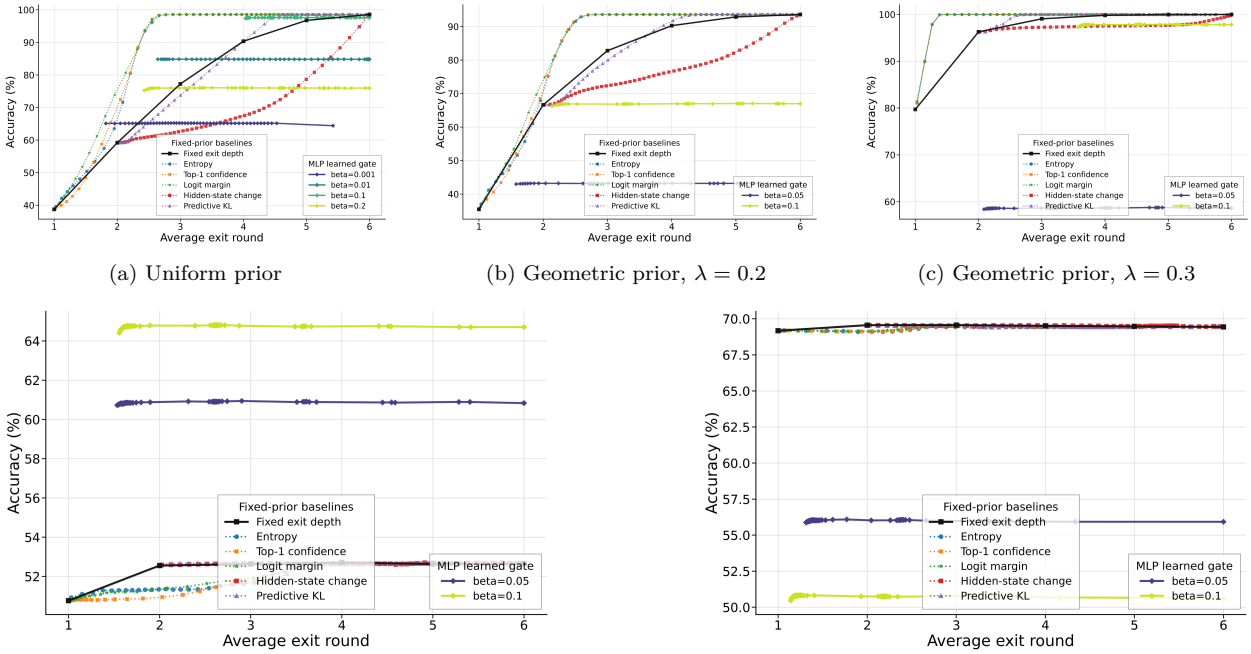
(a) Uniform prior (b) Geometric prior, $\lambda = 0.2$ (c) Geometric prior, $\lambda = 0.3$

(d) Geometric prior, $\lambda = 0.5$ (e) Geometric prior, $\lambda = 0.7$

Figure 13: Fixed-prior readouts compared with learned linear-gate models across all fixed-prior settings. Each panel plots test accuracy against average exit depth. Fixed-prior curves are obtained by sweeping post-hoc readout thresholds, while learned-gate curves are obtained by sweeping the cumulative gate threshold.

Figure 13 compares fixed-prior readouts against learned linear-gate models across all fixed-prior settings. In every panel, the fixed-prior trajectory with simple confidence readouts such as top-1 confidence and logit margin traces a good compute–quality frontier, reaching high accuracy at low average exit depth. The learned linear gate behaves differently across the regularization sweep. Small β allows the gate to spread exit mass across depths but does not produce a frontier that dominates the post-hoc readouts, while large β pulls the exit distribution toward the prior and collapses the gate onto a near-fixed-depth operating point, visible as the gate curve contracting to a small range of average depths. Under the more aggressive geometric priors ($\lambda = 0.5, 0.7$), both the gate and the trajectory saturate at a lower accuracy ceiling, consistent with the trajectory degradation noted in Figure 8. The overall message matches the main text. Learning an input-dependent halting distribution with a linear gate is not sufficient to recover the frontier that a well-shaped fixed-prior trajectory already exposes.

D.2 MLP Gate



(a) Uniform prior (b) Geometric prior, $\lambda = 0.2$ (c) Geometric prior, $\lambda = 0.3$

(d) Geometric prior, $\lambda = 0.5$ (e) Geometric prior, $\lambda = 0.7$

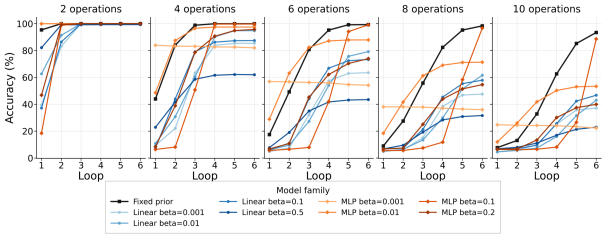
Figure 14: Fixed-prior readouts compared with learned MLP-gate models across all fixed-prior settings. Each panel plots test accuracy against average exit depth. Fixed-prior curves are obtained by sweeping post-hoc readout thresholds, while learned-gate curves are obtained by sweeping the cumulative gate threshold.

Figure 14 repeats the comparison with the learned halt head replaced by a two-layer MLP gate. Increasing gate expressivity does improve the learned-gate baseline relative to the linear gate, producing somewhat smoother frontiers and reaching higher accuracy at intermediate depths in several panels. However, the additional capacity does not change the qualitative outcome. The fixed-prior trajectory with simple post-hoc readouts still provides a solid low-depth frontier across priors, and the MLP gate remains limited by the trajectory it induces during joint training rather than by the form of the halt head. This is consistent with the frozen-trajectory experiments in Appendix H, which show that fitting a more expressive gate on a jointly trained trajectory does not recover the frontier available from a fixed-prior trajectory. Taken together, the linear and MLP comparisons indicate that gate expressivity is not the binding constraint, but the trajectory formation is.

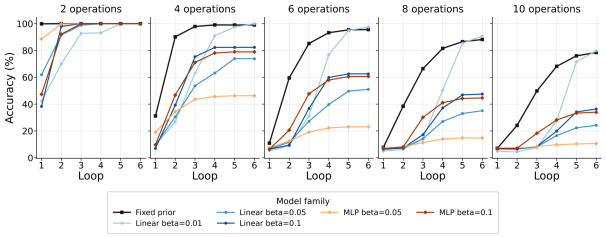
E Trajectory Diagnostics

This appendix expands the forced-exit and exit-distribution analysis of Section 5.3 to all prior and gate settings. Figure 15 reports forced-exit accuracy by difficulty, and Figure 16 reports the corresponding learned

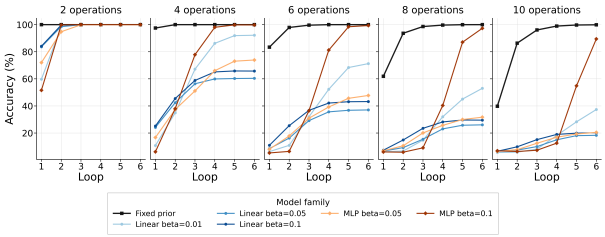
exit distributions. Forced exits evaluate the prediction at every recurrent loop without applying any adaptive stopping rule, so they measure the per-depth quality of the trajectory itself. The learned exit distributions show, for the gate models, how exit mass is allocated across depths as a function of task difficulty.



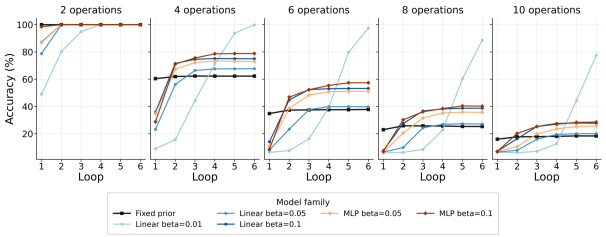
(a) Uniform prior



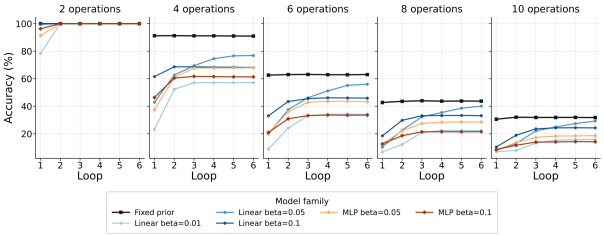
(b) Geometric prior, $\lambda = 0.2$



(c) Geometric prior, $\lambda = 0.3$



(d) Geometric prior, $\lambda = 0.5$



(e) Geometric prior, $\lambda = 0.7$

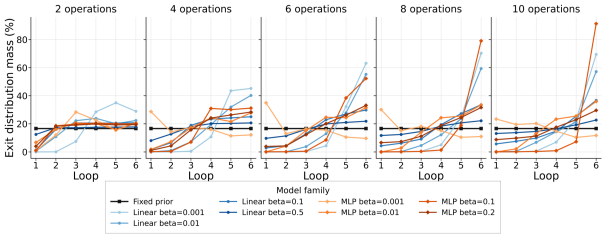
Figure 15: Forced-exit accuracy by task difficulty across fixed-prior settings. Each panel evaluates the prediction at every recurrent loop without using an adaptive readout.

Figure 15 reports forced-exit accuracy grouped by operation count across all fixed-prior and learned-gate settings. The fixed-prior trajectory produces useful intermediate predictions early in the loop, including on harder examples, and its per-depth accuracy rises smoothly with both depth and decreasing difficulty. The learned-gate trajectories show a different per-depth profile. Depending on β and gate class, accuracy at early loops is often lower, and the gain from additional loops is concentrated later in the trajectory. This confirms that the gate objective changes which recurrent states become accurate, not only the rule used to select among them. The contrast is most pronounced on the harder difficulty bins, where the fixed-prior trajectory already supports strong early exits but the jointly trained gate trajectory does not.

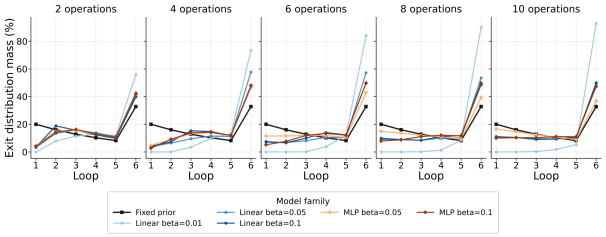
Figure 16 shows the average learned exit distribution over recurrent loops for the gate models, grouped by difficulty. The gates do learn a difficulty-aware allocation: easier examples receive more exit mass at early loops, while harder examples shift mass toward later depths. However, read together with the forced-exit curves, this allocation is not sufficient on its own. Assigning later exits to hard examples only helps if the corresponding late recurrent states are accurate, and the forced-exit panels show that the jointly trained trajectories do not always produce such states. This is the central diagnostic behind the main-text claim that adaptive-compute performance depends jointly on trajectory formation and exit selection, rather than on the exit distribution alone.

F Parity Experiments

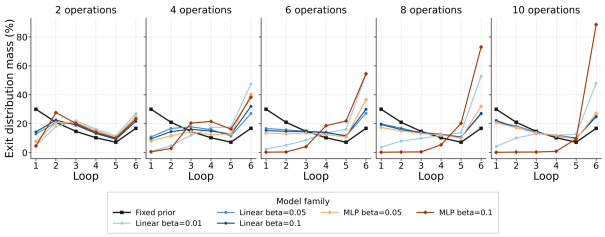
To check that the fixed-prior readout behavior is not specific to MANO, we run an additional small parity experiment. Each input is a binary sequence and the target is its parity. Unlike MANO, this task does not



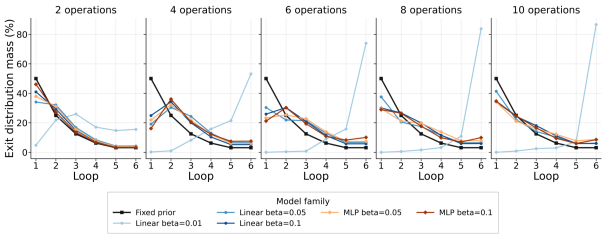
(a) Uniform prior



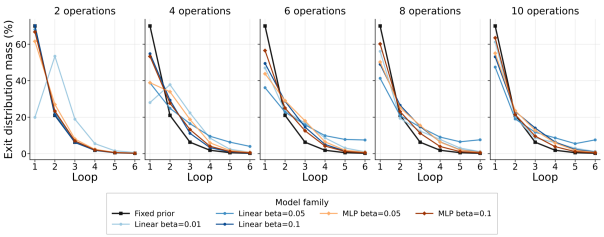
(b) Geometric prior, $\lambda = 0.2$



(c) Geometric prior, $\lambda = 0.3$



(d) Geometric prior, $\lambda = 0.5$



(e) Geometric prior, $\lambda = 0.7$

Figure 16: Learned exit distributions by task difficulty across prior settings. Each panel shows the average learned exit probability assigned to each recurrent loop for examples grouped by task difficulty.

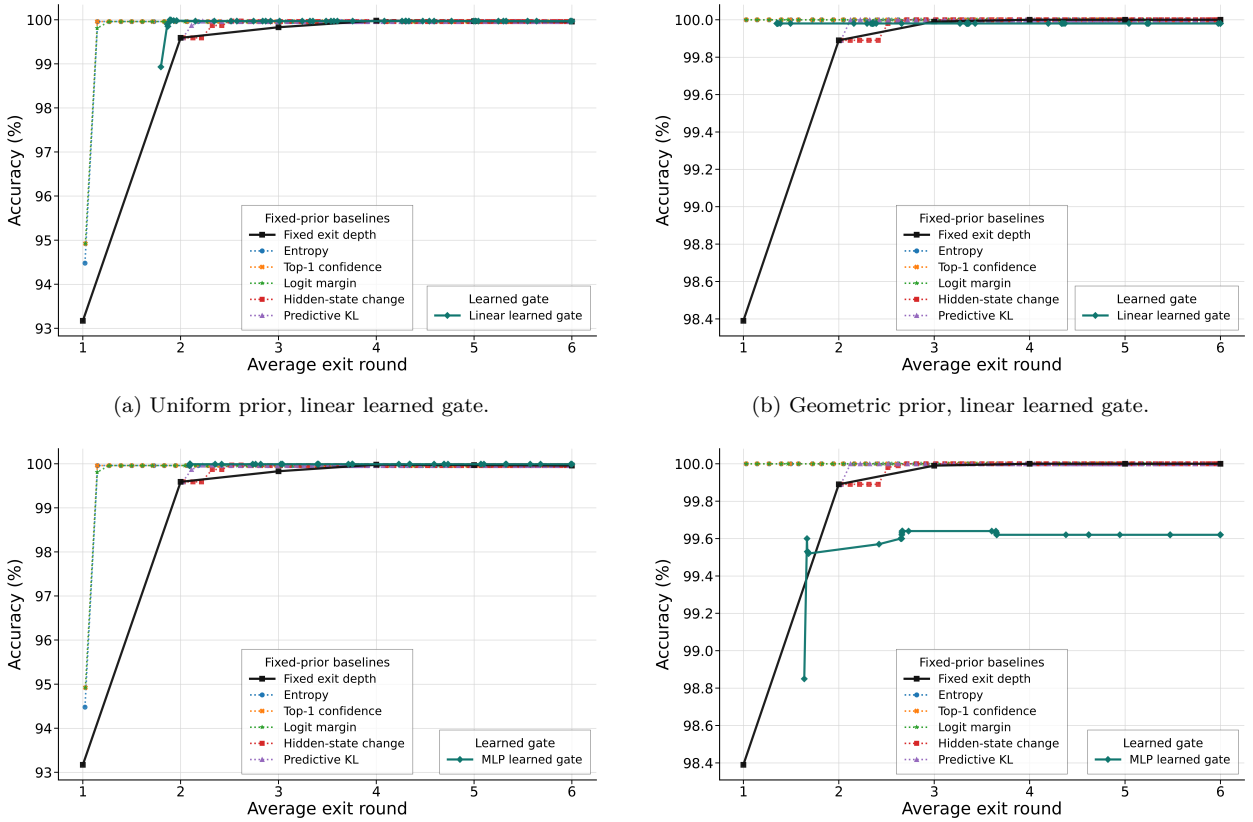
involve arithmetic expression evaluation. We use this task only as a lightweight robustness check, and do not repeat the full MANO benchmark ablation.

Table 7: Parity diagnostic over three seeds. We report mean ± standard deviation for validation-selected operating points on the held-out test split.

Training setting	Best Readout	Accuracy (%)	Avg. loops
Fixed uniform	Top-1 confidence	99.6 ± 0.7	1.30 ± 0.17
Fixed geometric, $\lambda = 0.3$	Top-1 confidence	99.0 ± 0.8	1.24 ± 0.28
Linear gate, uniform	Gate	99.6 ± 0.3	1.77 ± 0.08
MLP gate, uniform	Gate	99.6 ± 0.1	2.37 ± 0.46
Linear gate, geometric $\lambda = 0.3$	Gate	98.8 ± 0.1	1.42 ± 0.14
MLP gate, geometric $\lambda = 0.3$	Gate	98.7 ± 0.2	2.28 ± 0.65

We train looped Transformer models on parity sequences with bit lengths from 1 to 40. For each seed, training examples are generated online and balanced across bit lengths. We evaluate six settings: fixed-prior uniform, fixed-prior geometric with $\lambda = 0.3$, learned linear gates under both priors, and learned MLP gates under both priors. Learned gates use $\beta = 0.1$. All models use $T = 6$ recurrent loops.

Figure 17 shows representative compute–accuracy curves. Because parity is a simple binary classification task, most settings approach saturation, and the accuracy differences are less striking than on MANO. The relevant comparison is therefore the compute required to reach the saturated regime. Under this lens, the fixed-prior models provide useful evidence. Simple confidence readouts reach near-ceiling accuracy at average depths comparable to, and often lower than, the learned gates. Thus, even in a setting where the task is nearly solved by all methods, the recurrent trajectory contains enough stopping information for inexpensive post-hoc readouts to recover competitive adaptive computation.



(a) Uniform prior, linear learned gate.

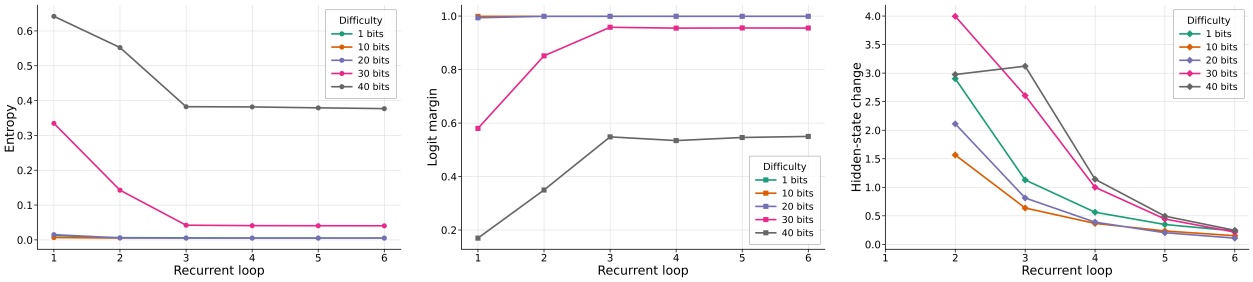
(b) Geometric prior, linear learned gate.

(c) Uniform prior, MLP learned gate.

(d) Geometric prior, MLP learned gate.

Figure 17: Parity diagnostic comparing fixed-prior readouts with learned gates.

Table 7 summarizes the same parity diagnostic across three seeds, showing that fixed-prior confidence readouts reach near-ceiling accuracy at low average depth.



(a) Entropy.

(b) Logit margin.

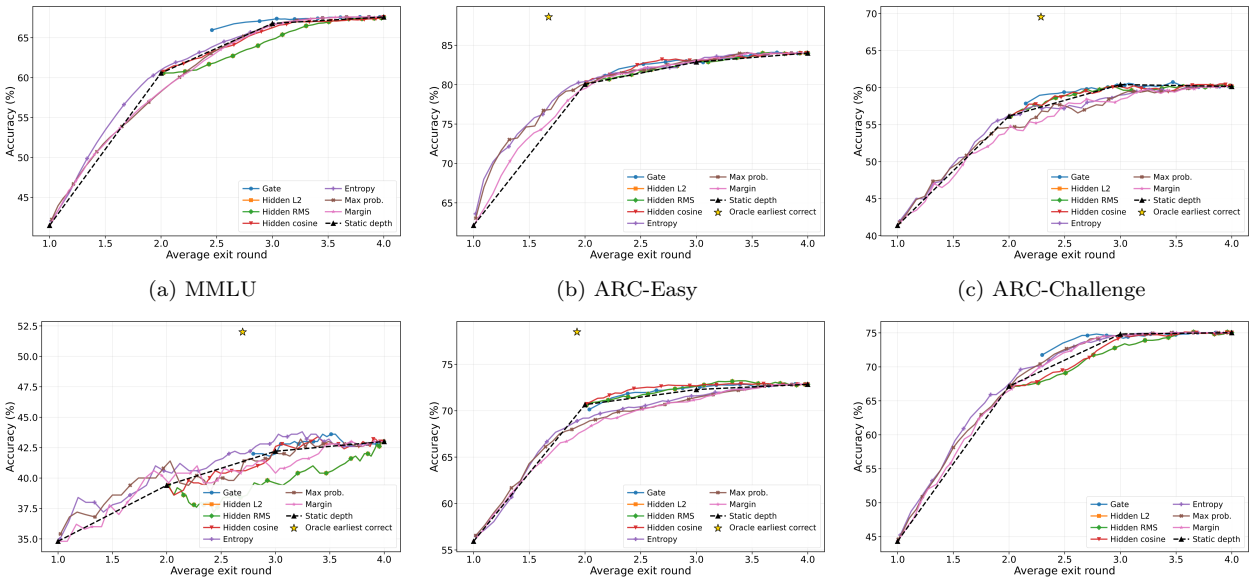
(c) Hidden-state displacement.

Figure 18: Difficulty-aware trajectory diagnostics on parity. The geometric fixed-prior model is trained on bit strings with lengths from 1 to 40. Curves are grouped by bit length. Longer sequences remain uncertain for more recurrent loops, develop logit margin more slowly, and show larger hidden-state changes across depth.

Figure 18 shows the corresponding difficulty-aware trajectory diagnostics. We use bit length as a simple difficulty variable. Although parity is not an arithmetic expression task, longer inputs require aggregating information across more positions. We therefore group validation and test examples by bit length and inspect trajectory statistics across recurrent depth. The fixed-prior trajectories show the same qualitative structure as in MANO. Longer sequences remain higher-entropy for more recurrent loops and develop logit margin more slowly. This suggests that the recurrent trajectory reflects input difficulty even in a non-MANO synthetic task.

G Additional Ouro Results

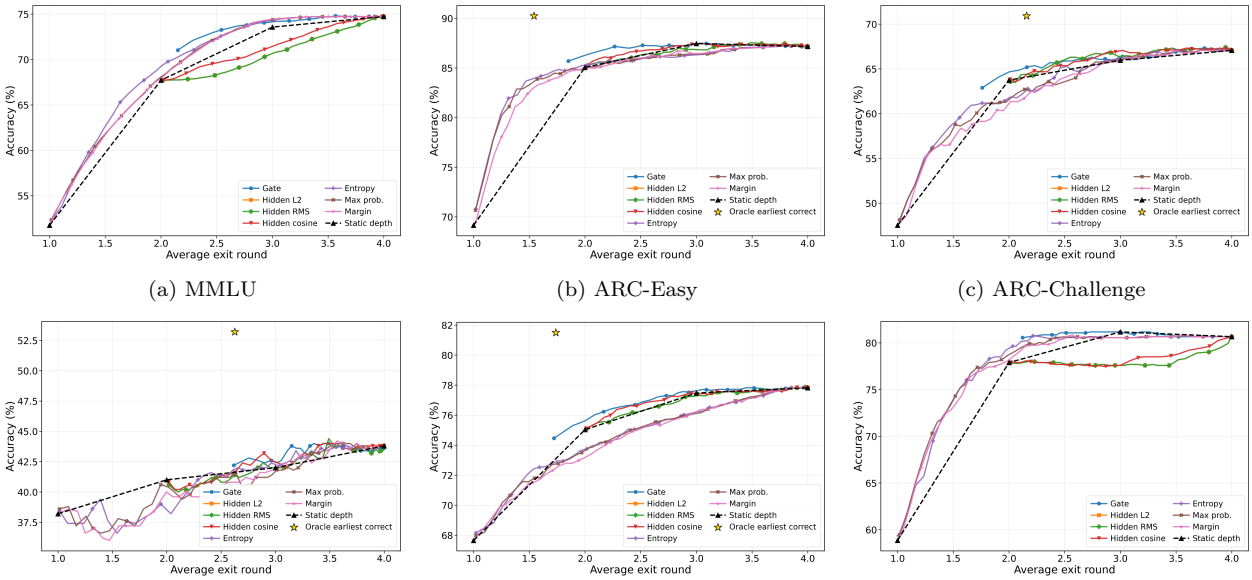
This appendix reports the full per-benchmark readout comparison for the Ouro checkpoints, expanding the two representative curves in Section 5.5. For each model and benchmark we plot benchmark accuracy against the average loops, comparing the pretrained ponder gate with confidence and convergence readouts applied to the same recurrent trajectories. We do not train or modify the models, so the trajectory is fixed by pretraining and only the readout varies.



(a) MMLU (b) ARC-Easy (c) ARC-Challenge

(d) OpenBookQA (e) HellaSwag (f) CommonsenseQA

Figure 19: Full readout comparison for Ouro-1.4B. Each panel plots benchmark accuracy against average loops. The pretrained ponder gate is compared with confidence and convergence readouts applied to the same recurrent trajectories.



(a) MMLU (b) ARC-Easy (c) ARC-Challenge

(d) OpenBookQA (e) HellaSwag (f) CommonsenseQA

Figure 20: Full readout comparison for Ouro-2.6B. Each panel plots benchmark accuracy against average loops. The pretrained ponder gate is compared with confidence and convergence readouts applied to the same recurrent trajectories.

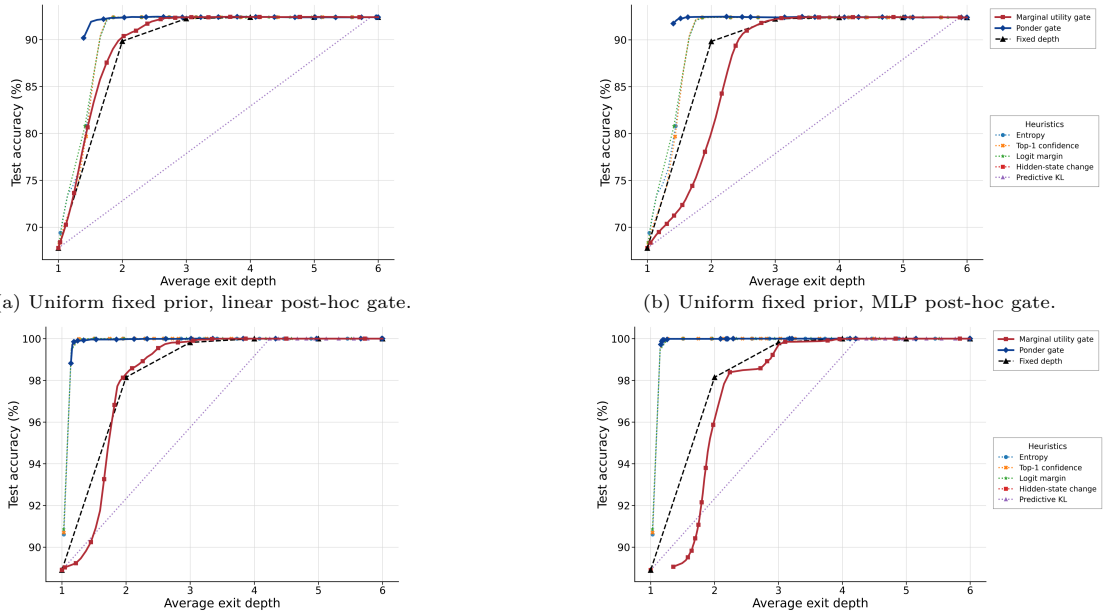
Figure 19 shows all six benchmarks for Ouro-1.4B, and Figure 20 shows the corresponding curves for Ouro-2.6B. Across both model sizes, the post-hoc readouts track the pretrained gate closely and in several bench-

marks match or improve on its compute–accuracy tradeoff. The pretrained gate remains a competitive operating point and is never far from the frontier, but it is not uniformly Pareto-optimal. The best readout varies across benchmark and model size, and simple confidence or convergence criteria such as maximum probability, entropy, logit margin, and hidden-state cosine similarity recover comparable or better trade-offs in many cases. The static-depth baseline included in each panel exits at the same loop number for every example and is generally dominated by the adaptive readouts, confirming that the gains come from input-dependent stopping rather than from a uniform reduction in depth. These full grids support the same conclusion as the numerical summary in Table 3. The recurrent trajectory exposes stopping signals that are not fully captured by the pretrained gate.

The oracle earliest-correct marker shown in several panels indicates the accuracy ceiling achievable if one could always exit at the first correct loop, and the gap between it and the realized readouts indicates the remaining room for improvement in stopping rules.

H Frozen Trajectory Gate Fitting

This appendix expands the frozen-trajectory experiments of Section 5.4. The goal is to separate two possible causes of weak learned-gate performance: an underexpressive or underfit gate readout, versus a trajectory shaped poorly by joint gate training. We freeze a pretrained recurrent trajectory, including the backbone and LM head, and train a new exit gate on top of the frozen hidden states. Because the trajectory is fixed, the new gate can only act as a readout. It cannot change the recurrent states or the per-depth predictions. We train both linear and MLP post-hoc gates, under two objectives: the PonderNet-style objective used for the native gates, and a marginal-utility objective inspired by Ouro Stage-II training that predicts, for each adjacent pair of depths, whether continuing one more loop is worthwhile.



(a) Uniform fixed prior, linear post-hoc gate.

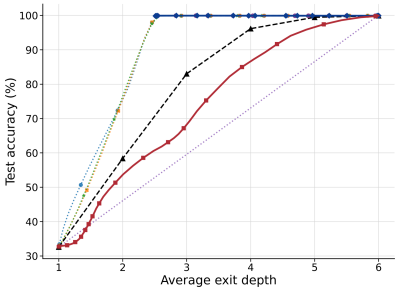
(b) Uniform fixed prior, MLP post-hoc gate.

(c) Geometric fixed prior, $\lambda = 0.3$, linear post-hoc gate.

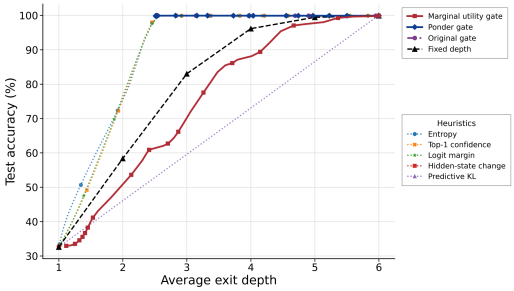
(d) Geometric fixed prior, $\lambda = 0.3$, MLP post-hoc gate.

Figure 21: Post-hoc gate training on frozen fixed-prior MANO trajectories. Both linear and MLP post-hoc gates are trained while keeping the recurrent backbone and LM head fixed.

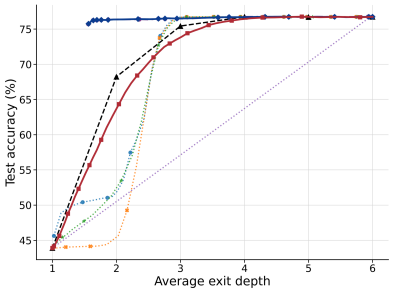
Figure 21 shows post-hoc gates trained on frozen fixed-prior trajectories. Here the post-hoc gates recover strong early-exit behavior, reaching near-final-loop accuracy at much lower average depth, which demonstrates that the gate class is capable of representing useful stopping policies when trajectory formation is separated from readout learning. The simple confidence readouts remain competitive with the fitted gates, indicating that much of the stopping signal is already exposed in the prediction trajectory and does not require a learned readout to extract.



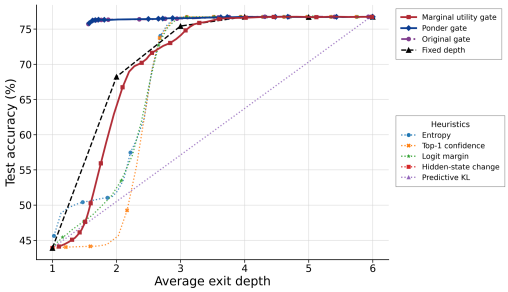
(a) Uniform linear-gate trajectory, $\beta = 0.1$, linear post-hoc gate.



(b) Uniform linear-gate trajectory, $\beta = 0.1$, MLP post-hoc gate.

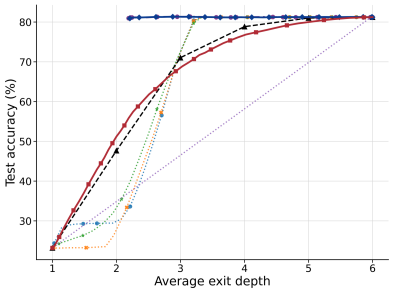


(c) Geometric linear-gate trajectory, $\lambda = 0.3$, $\beta = 0.1$, linear post-hoc gate.

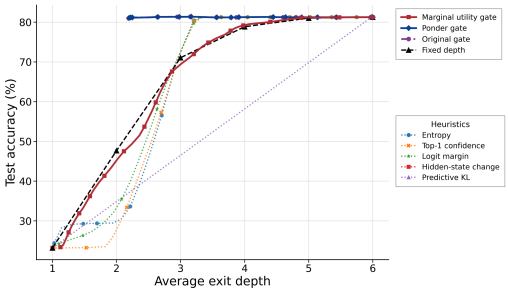


(d) Geometric linear-gate trajectory, $\lambda = 0.3$, $\beta = 0.1$, MLP post-hoc gate.

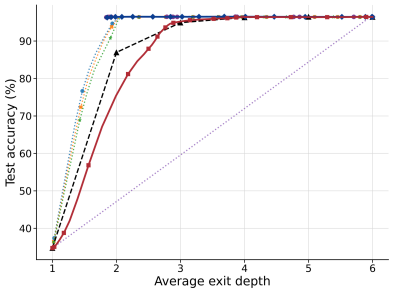
Figure 22: Post-hoc gate training on frozen MANO trajectories originally trained with a linear learned gate. The native learned gate is evaluated when available, and newly trained post-hoc gates are compared against heuristic readouts and fixed-depth exits.



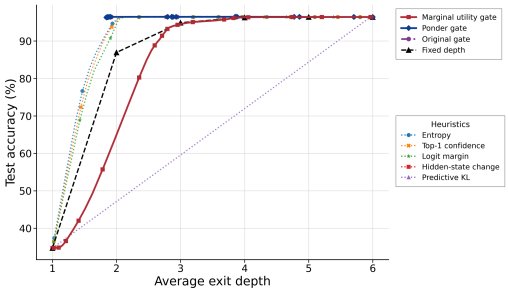
(a) Uniform MLP-gate trajectory, $\beta = 0.1$, linear post-hoc gate.



(b) Uniform MLP-gate trajectory, $\beta = 0.1$, MLP post-hoc gate.



(c) Geometric MLP-gate trajectory, $\lambda = 0.3$, $\beta = 0.1$, linear post-hoc gate.



(d) Geometric MLP-gate trajectory, $\lambda = 0.3$, $\beta = 0.1$, MLP post-hoc gate.

Figure 23: Post-hoc gate training on frozen MANO trajectories originally trained with an MLP learned gate. Across both uniform and geometric priors, post-hoc gates test whether a newly trained readout can improve on the gate learned jointly with the trajectory.

Figures 22 and 23 show post-hoc gates trained on trajectories that were themselves produced by joint learned-gate training, for the linear and MLP native gates respectively. As in Section 5.4, fitting a new post-hoc gate on these trajectories does not improve the frontier: the newly trained gates largely track the native gate and remain bounded by the quality of the underlying trajectory regardless of gate class, and the PonderNet-style objective again outperforms the marginal-utility objective, which exits later without improving the frontier.

Taken together, these frozen-trajectory results confirm the diagnosis from Section 5.4 across both gate classes: the limiting factor is not gate expressivity or post-hoc fitting, since the same gate class recovers strong frontiers on fixed-prior trajectories, but the trajectory produced by joint training itself.

I Reasoning-Tuned Ouro

The Ouro paper reports that post-SFT RLVR attempts did not bring significant gains, and identifies the dynamic early-exit mechanism as a central difficulty for RL training. Fast rollout systems assume a fixed execution path, but Ouro uses dynamic depth computation. This makes the gate an important interface between latent recurrent computation and optimization. We therefore evaluate whether trajectory readouts can recover competitive compute–quality tradeoffs on the reasoning-tuned Ouro-Thinking checkpoints.

We evaluate Ouro-1.4B-Thinking and Ouro-2.6B-Thinking on the same six benchmarks used in our main Ouro experiments. This keeps the task distribution and scoring protocol fixed, allowing us to isolate whether reasoning tuning changes the readout behavior of recurrent trajectories. Table 8 reports the best validation-selected points for each model and dataset. The same qualitative pattern persists in this experiment. The pretrained gate is a useful readout, but it is not necessarily the best. In 7 of the 12 settings, the best readout is a simple post-hoc heuristic rather than the pretrained gate. Thus, reasoning tuning does not remove the main readout effect. The pretrained gate remains useful, but in many settings a simple confidence or convergence readout gives the best performance-compute tradeoff. This suggests that even reasoning-specialized looped models expose stopping information outside the gate.

Table 8: Ouro-Thinking checkpoint diagnostic. We report the best validation-selected operating points on the held-out evaluation split.

Model	Dataset	Readout	Accuracy (%)	Avg. loops
Ouro-1.4B-Thinking	ARC-C	Hidden L2	61.9	3.31
Ouro-1.4B-Thinking	ARC-E	Gate	84.4	2.57
Ouro-1.4B-Thinking	CSQA	Entropy	73.9	2.78
Ouro-1.4B-Thinking	HellaSwag	Hidden cosine	68.9	2.23
Ouro-1.4B-Thinking	MMLU	Max prob.	67.1	3.20
Ouro-1.4B-Thinking	OpenBookQA	Max prob.	40.4	2.15
Ouro-2.6B-Thinking	ARC-C	Hidden L2	66.0	2.57
Ouro-2.6B-Thinking	ARC-E	Gate	87.6	2.28
Ouro-2.6B-Thinking	CSQA	Gate	79.5	2.30
Ouro-2.6B-Thinking	HellaSwag	Gate	73.7	2.95
Ouro-2.6B-Thinking	MMLU	Max prob.	75.7	2.89
Ouro-2.6B-Thinking	OpenBookQA	Max prob.	40.6	2.72