LEARNING TO LOOP: REINFORCEMENT LEARNING FOR LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped Transformers can increase computation by repeatedly applying existing layers, but many approaches use fixed repetition patterns across inputs. We ask whether a model can instead learn, for each question, when additional computation is useful and where it should occur. We introduce a lightweight controller that observes the model's hidden states and decides after each block execution whether to repeat the layer or proceed forward. The controller is trained by comparing alternative computation routes for the same question using signals from final-answer correctness and the model's likelihood of a reference solution, while the adapted language model remains frozen during controller training. Unlike fixed-depth looping, our method allows different questions to use different numbers and locations of repetitions under a maximum computation budget, while retaining admissible fixed-depth routes as special cases. We evaluate the approach across three 7–8B language models and four mathematical reasoning benchmarks: GSM8K and MathQA for in-domain evaluation, and SVAMP and MATH-500 for frozen transfer. Adaptive routing improves reasoning performance over fixed repetition schedules while allocating different amounts and locations of computation across questions. Controlled comparisons with exact-depth routing and shuffled question–route assignments further support the value of question-specific computation paths beyond simply executing more layers.

1 INTRODUCTION

Additional inference computation can improve language-model reasoning, but its benefit depends on how that computation is used. Recurrent-depth models perform additional computation by repeatedly applying shared Transformer blocks (Geiping et al., 2025; Zhu et al., 2025). These approaches separate inference computation from parameter count and raise a common question: which additional computations are useful for a particular input? Studies of test-time scaling show that effective allocation can depend on problem difficulty rather than following one uniform strategy (Snell et al., 2024). For layer repetition, this question has two parts: *where* to repeat and *how often*. Routes with the same execution depth can repeat different layers, whereas a fixed repetition schedule applies the same allocation to every question. Each repetition changes the hidden state used by later routing decisions, motivating a sequential decision formulation. We ask whether a controller trained from the outcomes of its own sampled routes can learn useful repetition policies over an adapted, fixed backbone. Our contribution is the route-learning and execution design together with a controlled evaluation of online reward learning, updated-state observations, and question-dependent depth.

Figure 1 summarizes the routing and training procedure. We introduce an RL-trained controller for monotone layer repetition under a hard maximum depth. After each block execution, the controller observes the updated prompt representations and chooses whether to repeat the current block or advance to the next. Every original block is executed at least once; advancing beyond the final block terminates routing, even when repetition budget remains. This lets the controller adapt both repetition locations and realized depth. The language model weights remain frozen during controller training, following a separate source-task adaptation stage. Routing is performed once while processing the prompt, and the selected route is reused for all generated tokens without further controller decisions.

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

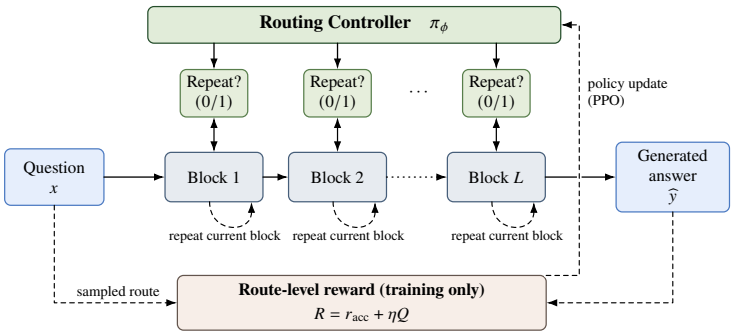


Figure 1: Adaptive loop routing with a frozen language-model backbone.

We train the controller by sampling several complete routes for the same question and comparing their outcomes. Each route receives a terminal reward combining generated-answer correctness with a bounded score derived from the language model's likelihood of a reference solution. We form leave-one-out route advantages and update the controller with a PPO-style clipped objective.

A larger depth cap makes more routes available, but does not guarantee that a finite controller learns a better policy. It can also admit longer decision sequences and more repetition allocations, complicating exploration and terminal-reward credit assignment. We therefore treat the cap as an empirical design choice rather than a guarantee of improvement.

Our evaluation covers three 7–8B models, two source tasks, and two frozen-transfer tasks. Fixed-depth and shuffled-route comparisons test repetition placement and question–route matching. We also compare a validation-selected common depth, whole-route prediction from initial prompt features, search-supervised routing, and alternative rewards. The compute analysis accounts for route depth, answer length, controller work, and occurrence-specific KV storage (Appendices B and C).

2 Related Work

2.1 Looped Transformers and latent reasoning

Parameter sharing provides a way to increase effective depth without introducing an independent parameter set at every execution. Universal Transformers combine shared transformations with recurrence across depth (Dehghani et al., 2019), while ALBERT uses cross-layer parameter sharing to reduce model size (Lan et al., 2019). Takase & Kiyono (2023) study how sharing and execution patterns affect Transformer performance. Beyond compression, Giannou et al. (2023) construct looped Transformers that execute programs, and Yang et al. (2023) investigate their inductive bias for learning iterative algorithms in context. Saunshi et al. (2025) provide theoretical and empirical evidence that repeated computation can support reasoning tasks requiring substantial effective depth. These studies motivate recurrence, but do not establish that arbitrary repetitions of a pretrained block improve a particular answer.

Recent work brings latent recurrence to larger language models. The recurrent-depth approach of Geiping et al. (2025) scales computation through a shared recurrent core, and Ouro incorporates iterative latent computation and learned depth allocation into pretraining (Zhu et al., 2025). Coconut instead feeds continuous hidden states back as input embeddings, replacing some explicit reasoning tokens with latent steps (Hao et al., 2024). SMELT studies looped mixture-of-experts architectures under compute-matched comparisons (Wang et al., 2026). Closer to frozen-backbone execution, Chen et al. (2026) retrofit loops onto pretrained checkpoints using damped repeated updates and report that naive block reapplication can degrade performance. Our focus is complementary: rather than training the recurrent representation or modifying the block update, we train a controller to select repetition locations and counts while holding the adapted backbone fixed.

2.2 Adaptive computation and layer routing

Learning how much computation to use has a long history. Adaptive Computation Time learns recurrent halting decisions (Graves, 2016), and PonderNet learns a distribution over computation steps while balancing prediction and computation objectives (Banino et al., 2021). SkipNet uses learned gates and reinforcement learning to select execution paths in convolutional networks (Wang et al., 2018). In language models, Depth-Adaptive Transformers make predictions at different network depths (Elbayad et al., 2019); CALM allocates computation through confidence-based early exits (Schuster et al., 2022); and LayerSkip trains intermediate exits for early-exit inference and self-speculative decoding (Elhoushi et al., 2024). These studies establish learned stopping and skipping mechanisms. Our controller instead retains every original layer and selects additional executions.

Adaptive recurrence also operates at finer granularity. Mixture-of-Depths routes selected tokens through Transformer blocks (Raposo et al., 2024), while Mixture-of-Recursions learns token-dependent recursive depths (Bae et al., 2025). Our policy instead constructs one prompt-dependent route and reuses it throughout answer generation. Popescu et al. (2026) further distinguish the quality of intermediate recurrent representations from the quality of a learned halting rule, highlighting that successful adaptive computation depends on more than the availability of extra depth. Freezing the backbone in our controller stage separates learning a routing policy from changing the underlying block weights.

The closest work also changes the execution of pretrained layers. Dr.LLM learns hidden-state routers with a frozen backbone and sequence-level skip, execute, or repeat decisions; it preserves layer order and reuses prompt-level routing during generation, but trains its routers using offline search-derived supervision (Heakl et al., 2026). PoLar predicts complete input-specific programs that skip or repeat pretrained layer segments, learning from programs found through offline search (Li et al., 2026). In contrast, we formulate layer repetition as a sequential reinforcement-learning problem: the controller repeatedly decides whether to repeat or advance, samples complete routes from its current policy, receives a terminal route-level reward, and is updated directly from these sampled outcomes. Our design further requires every original block to be executed and renews the routing decision after each repetition.

2.3 Reinforcement learning and reasoning-time computation

Reasoning-time computation is often allocated in output space. Chain-of-thought prompting generates intermediate reasoning steps (Wei et al., 2022), self-consistency samples multiple reasoning paths and aggregates their answers (Wang et al., 2022), and Tree of Thoughts searches over intermediate textual states (Yao et al., 2023). Snell et al. (2024) study how test-time computation should be allocated across problems of different difficulty. Our controller acts on a different axis: the sequence of layer executions used by a fixed language model. It does not search over candidate answers at deployment, although different routes can change the answer produced by the same greedy decoder.

Our optimization draws on established policy-gradient methods. REINFORCE provides the terminal-reward formulation (Williams, 1992), and PPO provides the clipped probability-ratio surrogate for reusing collected trajectories (Schulman et al., 2017). The within-question comparison uses a leave-one-out baseline of the form studied for language-model RL by Ahmadian et al. (2024). DeepSeekMath introduces group-relative policy optimization, and DeepSeek-R1 demonstrates outcome-reward training for language-model reasoning (Shao et al., 2024; DeepSeek-AI, 2025). Our sampled object is a computation route rather than a token sequence: advantages are assigned to routing decisions, and gradients update only the controller.

The source of credit also matters. Process supervision evaluates intermediate reasoning steps (Lightman et al., 2023), whereas our reference-solution likelihood is an auxiliary score for a complete route, not supervision of individual latent steps. Reinforcement learning for looped models is already explored by RLTT, which trains on next-token distributions across latent iterations (Williams & Tureci, 2026). These methods update the looped language model. We instead learn discrete execution choices over frozen blocks, using one terminal mixed reward per route.

3 ADAPTIVE LOOP ROUTING VIA REINFORCEMENT LEARNING

3.1 SETTING

Let $F_1, \ldots, F_L$ be the frozen complete attention-and-FFN blocks. A monotone route $\rho = (\rho_1, \ldots, \rho_D)$ visits every layer at least once and only repeats the current layer or advances to the next, with $D = L + \sum_{\ell=1}^{L} k_\ell \le D_{\max}$ and $0 \le k_\ell \le K$, where k_ℓ is the number of extra executions of layer ℓ. At execution t, ρ_t is the block index and the residual stream obeys $H_{t+1} = F_{\rho_t}(H_t)$. Repetition shares block weights but each occurrence has its own KV-cache slot during generation. After t executions, suppose the current layer is ℓ. There have been $e = t - \ell$ extras, since layers $1, \ldots, \ell$ have each executed at least once. Reserve one execution for each of the $L - \ell$ future layers. The remaining extra allowance is $b = (D_{\max} - L) - e$. The action mask is

$$\begin{aligned} \text{REPEAT allowed} &\iff b > 0 \text{ and } k_\ell < K, \\ \text{ADVANCE allowed} &\iff \text{always}. \end{aligned}$$

REPEAT consumes one unit of b. If $\ell < L$, ADVANCE moves to layer $\ell + 1$ without consuming an extra execution. If $\ell = L$, ADVANCE instead terminates route selection. Thus, at the final layer the controller effectively chooses between REPEAT and STOP, and any remaining allowance may be left unused. Throughout, K counts *extra* executions. The exact-depth control masks out any action that would make it impossible to finish at the prescribed depth; Appendix A.2 gives the full initialization.

3.2 ACTOR

The *actor* is a controller that decides whether to repeat the current Transformer block or advance after each block execution. Its trainable parameters are denoted by ϕ. The language-model parameters θ remain frozen during actor training. The actor selects one prompt-dependent route per rollout. Its inputs are representations computed from the prompt and the current routing state.

For a prompt with T valid tokens and residual width d, each block execution updates the full residual stream $H_t \in \mathbb{R}^{T \times d}$, after which the actor selects one action for the entire prompt; t indexes block executions rather than token positions. After execution t, the actor uses $h_t = H_{t+1}[T, :]$, $\delta h_t = H_{t+1}[T, :] - H_t[T, :]$, and $\bar{h}_t = \sum_{i=1}^{T} H_{t+1}[i, :]/T$ as the current last-token state, its execution-induced change, and the mean prompt representation. Thus, h_t is the current representation at the last prompt-token position; δh_t is the change at that same position caused by the latest block execution; and $\bar{h}_t$ is the mean representation across the prompt. The fourth vector, $h_{\text{entry},\ell}$, is the last prompt-token residual saved immediately before the first execution of F_ℓ.

For each residual vector v, define $\widetilde{v} = v / \max\{\mathrm{RMS}(v), 10^{-6}\}$ and $m(v) = \log(1 + \mathrm{RMS}(v))$. The actor receives both the normalized vector and its magnitude, with $\mathbf{m}_t = [m(h_t); m(\delta h_t); m(h_{\text{entry},\ell}); m(\bar{h}_t)] \in \mathbb{R}^4$. The current layer index ℓ is encoded by a trainable lookup embedding $\mathbf{e}_\ell \in \mathbb{R}^{d_{\text{layer}}}$, with $d_{\text{layer}} = 32$. Each Transformer layer has its own embedding vector, learned jointly with the actor. The remaining routing inputs are $\mathbf{u}_t = [k_\ell / \max\{1, K\}; b / \max\{1, D_{\max} - L\}] \in \mathbb{R}^2$, where the denominators also cover $K = 0$ and $D_{\max} = L$. The complete observation is $o_t = [\widetilde{h}_t; \widetilde{\delta h}_t; \widetilde{h}_{\text{entry},\ell}; \widetilde{\bar{h}}_t; \mathbf{m}_t; \mathbf{e}_\ell; \mathbf{u}_t] \in \mathbb{R}^{4d + d_{\text{layer}} + 6}$.

How the actor chooses an action. A shared one-hidden-layer network maps o_t to two logits:

$$\mathbf{z}_t = W_2 \,\mathrm{GELU}(W_1 o_t + c_1) + c_2 \in \mathbb{R}^2.$$

Here $W_1 \in \mathbb{R}^{512 \times (4d + d_{\text{layer}} + 6)}$ and $c_1 \in \mathbb{R}^{512}$, while $W_2 \in \mathbb{R}^{2 \times 512}$ and $c_2 \in \mathbb{R}^2$. The same network is used at every layer and repetition; its parameters and the learned layer embeddings belong to ϕ. The first logit represents REPEAT, and the second represents C, where $C =$ ADVANCE for $\ell < L$ and STOP for $\ell = L$. Let $\mathcal{A}(o_t)$ be the feasible action set under the free or exact mask. The policy is the softmax of the two logits restricted to $\mathcal{A}(o_t)$: infeasible actions have probability zero, and a unique feasible action has probability one. This definition also covers exact-depth states in which ADVANCE is forbidden. During RL collection, the actor samples from this masked policy; evaluation chooses the feasible action with the largest logit.

How the route is constructed. Route construction begins with the mandatory execution of F_1, so $\rho_1 = 1$. After executing F_ℓ at step t, the actor chooses the next action according to $\pi_\phi(\cdot \mid o_t)$. If it chooses REPEAT, then $\rho_{t+1} = \ell$, $k_\ell \leftarrow k_\ell + 1$, and $b \leftarrow b - 1$, while $h_{\mathrm{entry},\ell}$ remains unchanged. Let C denote ADVANCE for $\ell < L$ and STOP for $\ell = L$. If $\ell < L$ and C is chosen, then $\rho_{t+1} = \ell + 1$, $h_{\mathrm{entry},\ell+1}$ stores the current last-token residual, and $k_{\ell+1} = 0$; ADVANCE does not consume b. If $\ell = L$, C terminates routing, possibly leaving unused allowance. The actor repeats this process after each execution until STOP, producing $\rho = (\rho_1, \ldots, \rho_D)$ with $L \le D \le D_{\max}$. Once ρ is finalized, the occurrence-specific KV caches are retained for decoding.

3.3 Answer generation and mixed terminal reward

Generation with the selected route. After route selection, decoding begins directly from the retained occurrence-specific KV caches, without additional prompt prefill, and the frozen language model generates $\widehat{y} = g_\theta(x; \rho)$. The route ρ is fixed throughout generation.

Final-answer correctness. A task-specific scorer extracts and normalizes the final answer from $\widehat{y}$ and defines $r_{\mathrm{acc}}(x, y, \rho) = \mathbf{1}\{\mathrm{score}(g_\theta(x; \rho), y) = 1\} \in \{0, 1\}$. The scorer compares only the extracted answer with reference answer y, not the full generated response with a reference solution. For arithmetic tasks, normalized numerical values are compared under the declared extraction rule; extraction failures receive $r_{\mathrm{acc}} = 0$, and formatting correctness alone earns no reward.

Reference-solution likelihood. To provide additional feedback beyond correctness, let $s^\star = (s_1^\star, \ldots, s_S^\star)$ be the tokenized reference solution, including its final answer, distinct from the reference answer y used by the correctness scorer. For route ρ, a teacher-forced scoring pass computes $\mathcal{L}_{\mathrm{ref}}(x, s^\star; \rho) = -\sum_{j=1}^{S} \log p_\theta(s_j^\star \mid x, s_{<j}^\star; \rho)/S$, using only reference-solution tokens. We define $Q(x, s^\star, \rho) = \exp[-\mathcal{L}_{\mathrm{ref}}(x, s^\star; \rho)] \in [0, 1]$, so lower reference loss gives higher Q. This measures the likelihood of one reference solution.

Mixing correctness and reference likelihood. For each question x_i, sample M routes $\rho_{i1}, \ldots, \rho_{iM}$ and let $r_{im}^{\mathrm{acc}} = r_{\mathrm{acc}}(x_i, y_i, \rho_{im})$ and $Q_{im} = Q(x_i, s_i^\star, \rho_{im})$. The route-level terminal reward is $R_{im} = r_{im}^{\mathrm{acc}} + \eta Q_{im}$ with $0 < \eta < 1$, where correctness captures the downstream task outcome and Q_{im} provides a dense auxiliary signal for how strongly the route supports the reference solution.

The auxiliary score Q is the geometric mean of reference-token probabilities, rather than the unnormalized probability of the entire solution. Since $0 < \eta < 1$, every individually correct route has higher mixed reward than every individually incorrect route. This ordering does not guarantee that maximizing expected mixed reward maximizes expected accuracy across stochastic policies. We therefore select checkpoints by generated-answer accuracy and, in Appendix B, test the reward using correctness alone, reference-solution likelihood alone, and different values of the mixing coefficient η.

3.4 On-policy route collection and clipped policy optimization

Collecting route groups. At each collection round, we freeze the current policy as ϕ_{old}, sample N training questions, and for each x_i draw M independent routes $\rho_{im} \sim \pi_{\phi_{\mathrm{old}}}(\cdot \mid x_i)$, where i, m, and t index the question, route, and block execution. Each route is generated by sequential REPEAT/ADVANCE/STOP decisions, followed by answer generation and computation of the mixed reward. For each decision, we store the residual features, routing metadata, layer index, feasibility mask, sampled action, and old-policy log-probability. Let $\mathcal{T}_{im}$ denote the execution steps along route ρ_{im} at which both REPEAT and ADVANCE/STOP are feasible. Steps with only one feasible action are forced and are excluded from the policy-gradient update, because their action probability is fixed at one and therefore provides no learning signal about the actor's preference.

Comparing routes within a question. For each question, we compare each sampled route with the other $M - 1$ routes using the leave-one-out baseline $B_{im} = \sum_{n \neq m} R_{in}/(M-1)$ and advantage $A_{im} = R_{im} - B_{im}$. Thus A_{im} measures whether route m has higher or lower mixed reward than the other routes for the same question. The same A_{im} is assigned to all decisions in $\mathcal{T}_{im}$. Because reward

is observed only after completing and evaluating the full route, we use the route-level advantage A_{im} as the credit signal for every non-forced decision along that route. The same route-level advantage is assigned to every non-forced routing decision along the route; we do not attempt to determine which individual decision caused the terminal outcome.

Clipped router update. Write $o_{imt}(\phi)$ for the observation defined in the preceding section, formed from the stored residual. The probability ratio for the stored action is

$$w_{imt}(\phi) = \frac{\pi_\phi(a_{imt} \mid o_{imt}(\phi))}{\pi_{\phi_{\text{old}}}(a_{imt} \mid o_{imt}(\phi_{\text{old}}))}. \tag{1}$$

The denominator is fixed throughout optimization. This ratio concerns routing actions, not generated answer-token probabilities. We maximize the clipped surrogate

$$\widehat{\mathcal{J}}_{\text{clip}}(\phi) = \frac{1}{NM} \sum_{i=1}^{N} \sum_{m=1}^{M} \sum_{t \in \mathcal{T}_{im}} \min\Big\{ w_{imt}(\phi) A_{im}, \; \text{clip}\big(w_{imt}(\phi), 1-\epsilon, 1+\epsilon\big) A_{im} \Big\}, \tag{2}$$

where $0 < \epsilon < 1$ is the clipping threshold. Gradients update the actor network and its learned layer embeddings.

Collection and update cycle. For each collection round, routes and advantages are generated using the frozen policy ϕ_{old}. We then perform U PPO updates on this batch while keeping ϕ_{old} and the advantages fixed, discard the batch, and collect fresh routes with the updated actor.

4 Experimental Setting

Models and computation budget. We evaluate three instruction-tuned language models: Llama-3.1-8B, Qwen2.5-7B, and Qwen3-8B, with $L = 32$, 28, and 36 Transformer layers, respectively. Qwen3 is evaluated with thinking disabled. Our primary free-choice controller uses a maximum execution depth of $D_{\max} = 3L$ and a per-layer repetition cap of $K = 4$, corresponding to maximum depths of 96, 84, and 108 block executions for the three backbones. The realized depth D is selected by the controller for each question and may be smaller than $D_{\max}$. During controller training, the language-model backbone and its LoRA adaptation weights remain frozen.

Datasets and evaluation. We use GSM8K and MathQA for in-domain training and evaluation, and SVAMP and MATH-500 for frozen transfer. Evaluation uses all 1,319 GSM8K test questions, 2,985 MathQA test questions, 1,000 SVAMP questions, and 500 MATH-500 questions. GSM8K and MathQA use task-specific source training data. For transfer, the GSM8K-trained adapter and controller are applied directly to SVAMP and MATH-500 without target-task training, checkpoint selection, or hyperparameter tuning. GSM8K and SVAMP are evaluated by normalized final-number accuracy, MathQA by final-option accuracy, and MATH-500 by mathematical-equivalence scoring.

Controller training. The actor is initialized by imitating the Sequence-$2L$ routing policy before reinforcement learning. Each collection round samples 32 training questions and four independent routes per question from the current policy. We optimize the controller using the mixed terminal reward described in Section 3, while keeping the backbone fixed. We use AdamW with learning rate 10^{-5}, zero weight decay, and gradient-norm clipping at 1. Training runs for up to 200 collection rounds, corresponding to at most 25,600 sampled routes, with evaluation on a fixed validation set every 20 rounds. At test time, routing is deterministic and selects the highest-probability feasible action.

Generation. All methods use the same prompts, decoding procedure, and answer scorer within each dataset. We use greedy decoding with a maximum of 1,024 generated positions for GSM8K, MathQA, and SVAMP, and 2,048 positions for MATH-500. The controller determines a single computation route from the prompt, and the resulting route is reused throughout answer generation; no additional routing decisions are made on generated tokens.

Random seeds and execution schedule. Each experimental block is executed independently using five random seeds. Seed indices may be shared across blocks, but each block constitutes a separate

run; stochastic adaptation, controller training, and evaluation are rerun independently. Consequently, repeated configurations reported in different tables are independent replications and need not have identical means or standard deviations unless explicitly stated otherwise. Within an experimental block, methods intended for paired comparison share the same adapted checkpoint, data partition, and evaluation instances for each seed.

Hardware and runtime. Experiments are run in BF16 on two NVIDIA H100 80GB GPUs. A complete actor-training run requires approximately 40 hours of wall-clock time on two H100s, corresponding to roughly 80 GPU-hours. The training budget is separated into rollout generation, reference scoring, controller updates, and validation. Adaptation, initialization, and baseline search are accounted for separately.

5 RESULTS

5.1 ACCURACY WITH THE SAME UPPER BOUND

Table 1 compares our free-choice RL controller with fixed and learned-routing baselines under a common maximum depth of $3L$. **Sequence** executes each layer three times consecutively, whereas **Cycle** executes the complete Transformer stack three times. We additionally compare against a validation-selected fixed route, providing a stronger input-independent baseline than the hand-designed Sequence and Cycle schedules. **Exact-depth RL** uses the same sequential actor and training procedure as our method but is constrained to exactly $3L$ block executions for every question. In contrast, **free-choice RL** treats $3L$ only as an upper bound and may terminate earlier, allowing the controller to adapt both the locations of repetitions and the total amount of computation.

We report the mean and standard deviation over five independent seeds. For each seed, all methods within a backbone share the same adapted language-model checkpoint and evaluation split; stochastic backbone adaptation, route collection, and controller optimization are repeated across seeds. Exact-depth RL versus fixed $3L$ routes tests the learned policy against selected schedules at equal block depth. Free-choice versus exact-$3L$ routing compares two policies, but a better common depth may explain part of the difference. We therefore include validation-selected common-depth actors in Appendix B. For free-choice RL, we additionally report the mean realized depth D/L across test questions and seeds.

Table 1: Accuracy (%) under a common maximum depth of $3L$. Results are mean $\pm$ standard deviation over five seeds. For free-choice RL, mean D/L is reported across seeds.

Method	GSM8K	Mean D/L	MathQA	Mean D/L
Llama-3.1-8B				
Native (L)	84.9 ± 0.3	1.00	71.0 ± 0.4	1.00
Sequence	85.6 ± 0.5	3.00	74.6 ± 0.2	3.00
Cycle	86.1 ± 0.7	3.00	75.9 ± 0.4	3.00
Validation-selected fixed route	86.7 ± 0.3	3.00	77.0 ± 0.5	3.00
Exact-depth RL	88.9 ± 0.4	3.00	81.1 ± 0.2	3.00
Free-choice RL	$\mathbf{89.8 \pm 0.4}$	$\mathbf{2.23 \pm 0.06}$	$\mathbf{82.7 \pm 0.5}$	$\mathbf{2.35 \pm 0.07}$
Qwen2.5-7B				
Native (L)	92.1 ± 0.2	1.00	75.5 ± 0.3	1.00
Sequence	92.1 ± 0.3	3.00	78.2 ± 0.5	3.00
Cycle	92.5 ± 0.4	3.00	78.7 ± 0.2	3.00
Validation-selected fixed route	92.9 ± 0.5	3.00	79.4 ± 0.3	3.00
Exact-depth RL	93.5 ± 0.3	3.00	83.8 ± 0.5	3.00
Free-choice RL	$\mathbf{94.9 \pm 0.4}$	$\mathbf{2.16 \pm 0.05}$	$\mathbf{85.9 \pm 0.4}$	$\mathbf{2.28 \pm 0.09}$
Qwen3-8B				
Native (L)	92.8 ± 0.4	1.00	79.3 ± 0.3	1.00
Sequence	92.9 ± 0.5	3.00	81.4 ± 0.7	3.00
Cycle	93.2 ± 0.2	3.00	82.2 ± 0.4	3.00
Validation-selected fixed route	93.5 ± 0.7	3.00	83.0 ± 0.2	3.00
Exact-depth RL	94.6 ± 0.3	3.00	86.6 ± 0.4	3.00
Free-choice RL	$\mathbf{95.2 \pm 0.2}$	$\mathbf{2.03 \pm 0.05}$	$\mathbf{87.6 \pm 0.5}$	$\mathbf{2.17 \pm 0.06}$

Across all three backbones and both datasets, exact-depth RL exceeds the strongest fixed-route baseline at depth $3L$. Free-choice routing further improves over exact-$3L$ routing while using fewer than $3L$ executions on average. These results support useful learned repetition and the benefit of avoiding compulsory $3L$ execution. The validation-selected common-depth comparison further distinguishes this benefit from question-dependent stopping. An operating range near two passes is also reported in the different compute-matched setting of SMELT (Wang et al., 2026).

5.2 Does question–route matching matter?

At exactly $D = 2L$, Table 2 directly tests whether question–route matching matters. We compare the learned policy with its most frequent source-validation route and with a random reassignment of its test routes. The modal-route control tests whether the most frequent learned schedule accounts for the improvement; it is not an exhaustive search for the best input-independent policy. The shuffled-route control preserves the route multiset while breaking question–route correspondence. This experiment tests matching, while the static-feature controller in Appendix B tests the additional contribution of observing updated hidden states.

Table 2: GSM8K accuracy (%) at exactly $D = 2L$. Results are mean $\pm$ standard deviation over five seeds.

Method, exactly $2L$	Llama-3.1-8B	Qwen2.5-7B	Qwen3-8B
Sequence	85.1 ± 0.3	91.8 ± 0.2	92.7 ± 0.2
Validation-selected fixed route	86.0 ± 0.5	92.3 ± 0.4	93.0 ± 0.3
Sequential RL: modal tune route	86.3 ± 0.4	92.2 ± 0.2	93.2 ± 0.2
Sequential RL: shuffled routes	86.2 ± 0.4	90.2 ± 0.6	92.9 ± 0.4
Sequential RL: repeated decisions	$\mathbf{89.9 \pm 0.6}$	$\mathbf{93.6 \pm 0.4}$	$\mathbf{94.2 \pm 0.3}$

Across all three backbones, shuffling the learned routes across questions consistently lowers accuracy relative to the matched assignment. This pattern supports a useful correspondence between each question and its selected computation route.

5.3 Transfer and sensitivity to the cap

We next test two questions. First, does a routing policy learned on GSM8K transfer to related but unseen reasoning benchmarks without further training? Second, how does performance change as the maximum allowed computation increases?

We evaluate the GSM8K-trained adapter and controller directly on SVAMP and MATH-500. The **native** row executes the frozen adapted model once through all L layers. The **exact** $3L$ controller must use exactly $3L$ block executions, whereas the **free** $\leq 3L$ controller may stop earlier.

Table 3: Frozen transfer accuracy (%) and realized depth. Mean $\pm$ standard deviation over five seeds.

Backbone / actor	SVAMP	Mean D/L	MATH-500	Mean D/L
Llama-3.1-8B / native L	89.2 ± 0.6	1.00	55.1 ± 0.6	1.00
Llama-3.1-8B / exact $3L$	91.8 ± 0.5	3.00	57.4 ± 0.8	3.00
Llama-3.1-8B / free $\leq 3L$	$\mathbf{92.6 \pm 0.7}$	1.88 ± 0.03	$\mathbf{58.6 \pm 0.7}$	2.55 ± 0.08
Qwen2.5-7B / native L	92.7 ± 0.4	1.00	74.6 ± 0.6	1.00
Qwen2.5-7B / exact $3L$	94.3 ± 0.4	3.00	79.5 ± 0.7	3.00
Qwen2.5-7B / free $\leq 3L$	$\mathbf{95.0 \pm 0.3}$	1.78 ± 0.06	$\mathbf{80.7 \pm 0.5}$	2.49 ± 0.07
Qwen3-8B / native L	93.2 ± 0.6	1.00	86.9 ± 0.5	1.00
Qwen3-8B / exact $3L$	94.5 ± 0.8	3.00	88.7 ± 0.4	3.00
Qwen3-8B / free $\leq 3L$	$\mathbf{95.3 \pm 0.7}$	1.62 ± 0.05	$\mathbf{89.4 \pm 0.3}$	2.37 ± 0.09

Transfer behavior. Table 3 shows that both learned routing policies outperform native execution on the two transfer tasks, despite receiving no target-task updates. Free-choice routing provides a further improvement over exact-depth routing while using fewer than $3L$ executions. The controller selects lower mean execution depth on SVAMP than on MATH-500. This experiment examines

whether a controller trained on GSM8K selects different execution depths on SVAMP and MATH-500 under the same maximum budget. Differences in average depth alone do not establish that the controller estimates question difficulty.

Sensitivity to the computation cap. We next vary the maximum depth $D_{\max}$. A separate free-choice actor is trained for each cap. Thus, this experiment studies how much useful computation the controller can exploit when given a larger feasible route space.

Table 4: GSM8K accuracy (%) and realized depth as the maximum computation budget changes. Mean $\pm$ standard deviation over five seeds.

Cap	Llama-3.1-8B		Qwen2.5-7B		Qwen3-8B	
	Acc.	Mean D/L	Acc.	Mean D/L	Acc.	Mean D/L
$1L$	84.9 ± 0.7	1.00	92.1 ± 0.2	1.00	92.8 ± 0.5	1.00
$2L$	88.1 ± 0.6	1.64 ± 0.05	94.0 ± 0.3	1.59 ± 0.04	94.5 ± 0.4	1.52 ± 0.04
$3L$	89.8 ± 0.4	2.23 ± 0.06	94.9 ± 0.4	2.16 ± 0.05	95.2 ± 0.2	2.03 ± 0.05
$4L$	$\mathbf{90.1 \pm 0.3}$	2.34 ± 0.04	$\mathbf{95.1 \pm 0.2}$	2.28 ± 0.06	$\mathbf{95.3 \pm 0.4}$	2.12 ± 0.03

Effect of increasing the cap. As shown in Table 4, performance improves substantially when the controller gains access to repetitions beyond the native depth, followed by diminishing gains as the maximum computation budget increases. Moving from L to $2L$ produces the largest improvement, whereas increasing the cap from $3L$ to $4L$ yields only a small additional gain. The $3L$ results are the same free-choice RL results reported in Table 1. Meanwhile, realized depth increases much more slowly than the allowed maximum: even under a $4L$ cap, the controllers use only $2.34L$, $2.28L$, and $2.12L$ executions on average for Llama-3.1-8B, Qwen2.5-7B, and Qwen3-8B, respectively. Thus, the cap functions as an upper bound on available computation rather than as a target execution depth.

6 Discussion and limitations

The exact-depth and shuffled-route comparisons support useful question-specific repetition placement over an adapted, fixed backbone. At equal depth, learned routing outperforms fixed schedules, while shuffling learned routes across questions reduces accuracy, supporting the value of matching computation paths to individual inputs. Free routing also improves over compulsory-$3L$ execution while using fewer block executions on average. Compared with a validation-selected common-depth policy, free routing achieves higher mean accuracy at lower mean depth, although this comparison alone does not establish that question-dependent depth variation causes the improvement. The additional controls show that sequential routing with updated hidden states outperforms the tested initial-feature and search-supervised alternatives, providing evidence for these design choices without establishing that either sequential feedback or reinforcement learning is uniquely necessary.

For a fixed monotone route family and repetition cap, allowing variable depth includes its admissible fixed-depth routes. This is a representational observation, not a policy-improvement or convergence guarantee; Cycle is outside the monotone family. Sparse rewards, limited exploration, imperfect state features, repetition-induced degradation, and checkpoint selection can all limit the trained policy. The mixed reward provides no explicit incentive to minimize compute among equally rewarded routes, and reference likelihood can disagree with generated-answer accuracy.

The controller selects a prompt-level route and cannot reconsider it after observing generated reasoning. Transfer is evaluated only on mathematical benchmarks. Varying routes complicates batching and increases occurrence-specific KV memory, while equal depth need not imply equal total generation work. Route-level rewards do not identify which individual repetitions cause an improvement, and a depth–difficulty correlation alone would not establish a causal effect of extra computation.

AI USE STATEMENT

AI assistance was used for manuscript editing, coding/debugging support, and formatting the presentation of experimental results. The authors remain fully responsible for verifying the final code, analyses, reported results, and empirical claims.

REPRODUCIBILITY STATEMENT

Sections 3 and 4 specify the routing algorithm, optimization procedure, models, datasets, and evaluation setup. Appendix A provides the complete implementation configuration, data splits, adaptation and initialization procedures, decoding and scoring rules, and evaluation protocol. Appendices B–D report controlled comparisons, computation and memory measurements, and aggregation procedures. An anonymized implementation and configuration files are provided in the supplementary material.

REFERENCES

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting REINFORCE style optimization for learning from human feedback in LLMs. *arXiv preprint arXiv:2402.14740*, 2024. doi: 10.48550/arXiv.2402.14740. URL https://arxiv.org/abs/2402.14740.

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. *arXiv preprint arXiv:2507.10524*, 2025. doi: 10.48550/arXiv.2507.10524. URL https://arxiv.org/abs/2507.10524.

Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021. doi: 10.48550/arXiv.2107.05407. URL https://arxiv.org/abs/2107.05407.

Lizhang Chen, Jonathan Li, Chen Liang, Ni Lao, and Qiang Liu. Training-free looped transformers. *arXiv preprint arXiv:2605.23872*, 2026. doi: 10.48550/arXiv.2605.23872. URL https://arxiv.org/abs/2605.23872.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. doi: 10.48550/arXiv.2501.12948. URL https://arxiv.org/abs/2501.12948.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1807.03819.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. *arXiv preprint arXiv:1910.10073*, 2019. doi: 10.48550/arXiv.1910.10073. 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 A. Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024. doi: 10.48550/arXiv.2404.16710. URL https://arxiv.org/abs/2404.16710.

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. *arXiv preprint arXiv:2502.05171*, 2025. doi: 10.48550/arXiv.2502.05171. 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. In *Proceedings of the 40th*

International Conference on Machine Learning, volume 202 of *Proceedings of Machine Learning Research*, pp. 11398–11442. PMLR, 2023. URL https://proceedings.mlr.press/v202/giannou23a.html.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016. doi: 10.48550/arXiv.1603.08983. 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. *arXiv preprint arXiv:2412.06769*, 2024. doi: 10.48550/arXiv.2412.06769. URL https://arxiv.org/abs/2412.06769.

Ahmed Heakl, Martin Gubri, Salman Khan, Sangdoo Yun, and Seong Joon Oh. Dr.LLM: Dynamic layer routing in LLMs. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/774164b966cc277c82a960934445140d-Paper-Conference.pdf.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019. doi: 10.48550/arXiv.1909.11942. URL https://arxiv.org/abs/1909.11942.

Ziyue Li, Yang Li, and Tianyi Zhou. Skip a layer or loop it? learning program-of-layers in llms. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2606.06574.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023. doi: 10.48550/arXiv.2305.20050. URL https://arxiv.org/abs/2305.20050.

Andrei Cristian Popescu, Haitz Sáez de Ocáriz Borde, and Pietro Liò. Adaptive depth in looped transformers: Diagnosing learned halting gates and trajectory readouts. *arXiv preprint arXiv:2607.20519*, 2026. doi: 10.48550/arXiv.2607.20519. URL https://arxiv.org/abs/2607.20519.

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. *arXiv preprint arXiv:2404.02258*, 2024. doi: 10.48550/arXiv.2404.02258. 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. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2502.17416.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. doi: 10.48550/arXiv.1707.06347. URL https://arxiv.org/abs/1707.06347.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *arXiv preprint arXiv:2207.07061*, 2022. doi: 10.48550/arXiv.2207.07061. URL https://arxiv.org/abs/2207.07061.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. doi: 10.48550/arXiv.2402.03300. URL https://arxiv.org/abs/2402.03300.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024. doi: 10.48550/arXiv.2408.03314. URL https://arxiv.org/abs/2408.03314.

Sho Takase and Shun Kiyono. Lessons on parameter sharing across layers in transformers. In *Proceedings of the Fourth Workshop on Simple and Efficient Natural Language Processing (SustaiNLP)*, pp. 78–90. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.sustainlp-1.5. URL https://aclanthology.org/2023.sustainlp-1.5/.

Shaowen Wang, Ge Zhang, Kairong Luo, Yuhao Wu, Shaofan Liu, Jiaheng Liu, Wenhao Huang, Shen Yan, and Jian Li. Smelt: Scaling laws for compute-matched moe looped transformers. *arXiv preprint arXiv:2609.01343*, 2026. doi: 10.48550/arXiv.2609.01343. URL https://arxiv.org/abs/2609.01343.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. SkipNet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 409–424, 2018. URL https://www.ecva.net/papers/eccv_2018/papers_ECCV/html/Xin_Wang_SkipNet_Learning_Dynamic_ECCV_2018_paper.php.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022. doi: 10.48550/arXiv.2203.11171. URL https://arxiv.org/abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022. doi: 10.48550/arXiv.2201.11903. URL https://arxiv.org/abs/2201.11903.

Jonathan Williams and Esin Tureci. Prioritize the process, not just the outcome: Rewarding latent thought trajectories improves reasoning in looped language models. *arXiv preprint arXiv:2602.10520*, 2026. doi: 10.48550/arXiv.2602.10520. URL https://arxiv.org/abs/2602.10520.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. *arXiv preprint arXiv:2311.12424*, 2023. doi: 10.48550/arXiv.2311.12424. URL https://arxiv.org/abs/2311.12424.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023. doi: 10.48550/arXiv.2305.10601. URL https://arxiv.org/abs/2305.10601.

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. *arXiv preprint arXiv:2510.25741*, 2025. doi: 10.48550/arXiv.2510.25741. URL https://arxiv.org/abs/2510.25741.

A Implementation and evaluation details

A.1 Configuration and collection

Table 5 summarizes the sequential actor.

Table 5: Configuration of the actor. The collection batch is the total across devices: $32 \times 4 = 128$ sampled routes per round. The model backbone and adapter are frozen during actor training.

Item	Specification
Actor input	$4d + 38$ features; learned layer embedding of width 32
Policy network	One width-512 GELU hidden layer; two masked logits
Primary depth settings	Free $D_{\max} = 3L$; exact $D = 3L$; at most $K = 4$ extra executions per layer (five total)
Additional settings	Exact $D = 2L$; free caps $L, 2L, 3L, 4L$
Initialization target	Sequence-$2L$ imitation on source fit prompts; the same imitation-trained parameters initialize free- and exact-depth actors
Route collection	$N = 32$ questions; $M = 4$ independently sampled routes per question; 128 routes per collection round
Training budget	At most 200 collection rounds; 25,600 sampled routes per actor
Actor optimizer	AdamW; learning rate 10^{-5}; weight decay 0; gradient-norm clip 1
Updates per batch	$U = 4$ optimizer steps per collection round, each using the complete batch of 128 collected routes
Probability-ratio clip	$\epsilon = 0.2$
Mixed-reward coefficient	$\eta = 0.1$
Exact-depth masking	Advancing is allowed only when the remaining depth budget is feasible; STOP at the final layer is allowed only when the remaining budget is zero
Checkpoint selection	Source validation every 20 rounds; highest answer accuracy; earlier checkpoint on ties
Deployment	Masked argmax routing; greedy answer decoding
Hardware configuration	Two NVIDIA H100 80GB GPUs; BF16 backbone execution

At each round, freeze the collecting actor ϕ_{old}. For every sampled route, retain its prompt-derived residual features, layer indices, routing variables, feasibility masks, actions, and old-policy action log-probabilities. Keep all four routes in each question group, including repeated routes and equal-reward outcomes.

The reward is the mixture, $R_{im} = r_{im}^{\text{acc}} + \eta Q_{im}$, and the advantage is the leave-one-out difference. The reference-solution pass scores a fixed route; it neither observes nor changes actor decisions. Reference answers, solutions, and rewards are not actor inputs.

During the U updates, keep the collecting policy, stored residual features, masks, rewards, and advantages fixed. Reconstruct each observation with the *current* learned layer embedding so that this embedding receives gradients. Apply the clipped objective to non-forced decisions, using the same mask in the old and new probabilities. Sum decision terms within a route and average over question–route pairs as in Section 3.4; do not add a route-length normalization. Discard the batch after these updates and collect again. Reused-batch updates are surrogate updates, not exact on-policy gradients of answer accuracy.

The 25,600-route count covers RL collection for one actor, not imitation, validation, baseline selection, transfer evaluation, or additional diagnostics. Reference-solution scoring is additional work per sampled route. Matching route counts does not match GPU-time.

A.2 Data, adaptation, initialization, and matched comparisons

Data and checkpoints. For each pipeline seed, we randomly permute the official source training split and partition it into disjoint adaptation, actor-fit, and validation sets (Table 6). Methods share these assignments within each seed. MathQA's separate official validation split is unused; test and

transfer questions do not select any parameter. Actor-fit questions support both initialization and route learning. The backbones and their corresponding tokenizers are

```
meta-llama/Llama-3.1-8B-Instruct
Qwen/Qwen2.5-7B-Instruct
Qwen/Qwen3-8B
```

These checkpoint families use the layer counts specified in Section 4.

Table 6: Disjoint source training allocations. The adaptation and actor-fit subsets serve different training stages; validation selects schedules and trained actors.

Source	Training total	Adaptation	Actor fit	Validation
GSM8K	7,473	5,000	1,973	500
MathQA	29,837	20,000	8,337	1,500

LoRA adaptation. We adapt one checkpoint per backbone, source task, and seed. Rank-16 LoRA modules with scale $\alpha = 32$ and dropout 0.05 modify the attention query, key, value, and output projections and the FFN gate, up, and down projections. The initial LoRA update is zero; all original parameters remain frozen. Repeated occurrences share their block's LoRA weights. We minimize reference-solution token cross-entropy for one adaptation epoch, masking prompt and padding tokens. AdamW uses learning rate 10^{-4}, $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, weight decay 0.01, gradient-norm clipping at 1, and cosine decay following 3% linear warmup. Training uses BF16, gradient checkpointing, effective batch size 32, and a combined prompt–solution limit of 2,048 tokens. Solutions are right-truncated when necessary; prompts are retained.

Prompt and solution serialization. We apply each tokenizer's chat template with no system message and one user turn. GSM8K and SVAMP use "Solve the following problem step by step. End with Final answer: $< number >$." followed by the question. MathQA appends the original ordered answer choices and substitutes "End with Final answer: $< letter >$." MATH-500 requests a final answer in `\boxed{}`. Qwen3 uses `enable_thinking=False`. Source references contain the provided rationale followed by the normalized answer or correct option letter. Cross-entropy and reference-likelihood scoring include only assistant solution tokens, excluding role prefixes, padding, and end-of-turn special tokens.

Adaptation routes. Each adaptation example independently samples q uniformly from $\{1, 2, 3, 4\}$. At $q = 1$ it uses the native route. Otherwise, equal probability selects a balanced route executing every layer q times or a random feasible monotone route of depth qL. To sample the latter, allocate $(q-1)L$ extras in layer order. With R extras remaining and n subsequent layers, sample k_ℓ uniformly from the integers $[\max(0, R - 4n), \min(4, R)]$ and reduce R accordingly. This sampler is not uniform over complete routes. All controllers share the resulting adapted checkpoint within a seed. Backbone and adapters are then frozen and placed in evaluation mode, disabling dropout throughout controller training and evaluation.

Actor initialization. Teacher observations come from actor-fit prompts executed along Sequence-$2L$, which repeats each layer once. The teacher uses the free $D_{\max} = 3L$ mask and can therefore stop legally at $2L$. We fit the actor for 500 AdamW steps at learning rate 10^{-4}, zero weight decay, gradient-norm clipping at 1, and batches of 32 prompts sampled with replacement. Cross-entropy averages over non-forced decisions with target probabilities 0.95 for the teacher action and 0.05 for the alternative. Layer embeddings follow $\mathcal{N}(0, 0.02^2)$; linear weights use Xavier uniform initialization and biases are zero. The final initialization is shared by free- and exact-depth actors before their respective RL masks apply. At cap L, the native route requires no actor training.

Exact-depth feasibility. At target depth $D \in [L, (K+1)L]$, after execution t at layer ℓ, let $e = t - \ell$ and $b_D = D - L - e$. The remaining capacities are $K - k_\ell$ extras at the current layer and $K(L - \ell)$ at later layers. Both training and evaluation apply

$$\begin{aligned} \text{REPEAT allowed} &\iff b_D \geq 1, \quad k_\ell < K, \\ &\qquad b_D - 1 \leq K - k_\ell - 1 + K(L - \ell), \\ \text{ADVANCE allowed } (\ell < L) &\iff 0 \leq b_D \leq K(L - \ell), \\ \text{STOP allowed } (\ell = L) &\iff b_D = 0. \end{aligned} \tag{A.1}$$

Each allowed action preserves a feasible completion to exactly D. The exact actor's budget feature is $b_D / \max(1, D - L)$; free actors use their remaining allowance under the selected cap. Architecture, initialization, source questions, and RL settings are shared, although masks, visited states, and learned parameters can differ. Throughout, $K = 4$ denotes four extra executions and permits at most five total executions per layer.

A.3 Fixed-schedule selection and paired evaluation

Fixed-schedule search. At exact depth D, we construct 64 distinct feasible monotone schedules. Initial candidates balance the $D - L$ extras or fill per-layer allowances in early-to-late, late-to-early, and center-out order. Center-out ordering sorts by distance to $(L + 1)/2$, breaking ties by lower layer index. The adaptation-route sampler supplies further candidates, rejecting duplicates. If fewer than 64 schedules exist, all feasible schedules are used; at $D = L$ only the native route is available. Source-validation accuracy selects the schedule, with lexicographic ties in $(k_1, \ldots, k_L)$. At $2L$ and $3L$, the search evaluates 32,000 GSM8K or 96,000 MathQA route–question pairs per backbone and seed. These costs are separate from RL collection. "Best fixed transfer" denotes this validation-selected schedule. Cycle is a separate non-monotone reference outside the search library and actor's route family.

Modal and shuffled controls. Deterministic actor inference on validation questions selects the most frequent count vector, with lexicographic ties. Shuffled-route evaluation first obtains each test question's route without consulting its answer. A seeded random derangement assigns each route to a different question index, and the recipient prompt executes the assigned route afresh. This preserves the route multiset and exact depth; distinct indices can still have identical count vectors. Comparisons pair methods by pipeline seed and question. Depth quantiles are computed across questions within each seed, separately from variation in mean depth across seeds.

A.4 Decoding, scoring, and cache accounting

We use greedy decoding throughout route-reward evaluation, validation, and testing. The maximum generation lengths are 1,024 tokens for GSM8K, MathQA, and SVAMP, and 2,048 tokens for MATH-500. For GSM8K and SVAMP, the model is prompted to terminate with `Final answer: <number>`. We extract the value from this final-answer field, normalize formatting such as thousands separators, and compare the resulting numerical values. For MathQA, the model terminates with `Final answer: <letter>`, and the extracted option is compared case-insensitively with the gold answer. For MATH-500, we extract the final boxed answer and evaluate it using Math-Verify for mathematical equivalence, applying the same extraction and comparison settings to all methods. Extraction failures are scored as incorrect and recorded separately.

If generation reaches the output-length limit, we score the available output using the same extraction and scoring procedure and additionally record a truncation flag. Thus, reaching the generation limit does not automatically make an otherwise extractable answer incorrect. We retain the generated answer, extracted answer, correctness indicator, extraction failure flag, and truncation flag for every evaluated question.

Frozen-transfer evaluation preserves the GSM8K-trained backbone, adapter, actor, and source-selected checkpoint. Dataset-specific output formats, generation limits, and scoring rules are fixed before transfer evaluation and are not tuned using SVAMP or MATH-500 outcomes. Qwen3 is evaluated with `enable_thinking=False`. Native rows evaluate the same adapted checkpoint under the standard depth-L execution path; published scores from the original instruction-tuned checkpoint are not used as substitutes for these runs.

As specified in Sections 3.2–3.3, prompt route construction retains a separate KV-cache slot for each executed block occurrence, and autoregressive generation begins directly from these caches without a second prompt prefill. Repeated occurrences share model parameters but retain occurrence-specific KV states during generation.

Inference efficiency is evaluated using direct hardware measurements rather than analytical estimates from execution depth. The hardware protocol uses 100 fixed held-out prompts, batch size 1, five warmup runs, and three timed repetitions per configuration, with synchronized GPU execution. Fresh requests do not reuse caches across examples, while normal within-request KV caching re-

mains enabled. Appendix C reports the measurement protocol and the corresponding latency and KV-cache results.

A.5 Evaluation and reporting protocol

We evaluate all methods using the same prompts, decoding procedure, and task-specific scorer within each dataset. GSM8K and SVAMP use normalized final-number accuracy, MathQA uses final-option accuracy, and MATH-500 uses mathematical-equivalence scoring. An answer that cannot be parsed under the corresponding evaluation rule is counted as incorrect.

Generation is greedy for all methods. We allow at most 1,024 generated positions on GSM8K, MathQA, and SVAMP, and 2,048 positions on MATH-500. For learned routing methods, the complete computation route is selected from the prompt before answer generation and is then reused for all generated tokens; the actor is not queried again during decoding.

All primary results are reported over five independent runs per experimental block. Runs are independent across separately executed experimental blocks. Within each block, methods intended for direct comparison on the same backbone share the corresponding adapted language-model checkpoint, data partitions, evaluation questions, prompts, and decoding configuration. We report mean and standard deviation of answer accuracy across seeds. For free-choice routing, we also report the mean realized depth D/L and its across-seed standard deviation.

Checkpoint selection is performed only on the source-task validation set using generated-answer accuracy under deterministic masked-argmax routing. Test-set outcomes are never used for checkpoint selection. For frozen transfer, the GSM8K-trained adapter and selected actor are applied directly to SVAMP and MATH-500 without target-task training, checkpoint selection, or hyperparameter tuning.

For the modal-route control, the route selected most frequently by the exact-depth actor on the tuning set is applied to every test question. For the shuffled-route control, the complete routes selected for the test questions are randomly reassigned across those questions without using test answers, after which answers are regenerated under the reassigned routes. This preserves the set of selected routes while breaking the correspondence between a question and the route chosen for it.

Depth diagnostics summarize the distribution across questions within each seed using its standard deviation, quantiles, and cap-hit frequency. These statistics are distinct from the across-seed standard deviation of mean depth. The compute analysis separately accounts for token lengths, route depth, controller work, and KV storage.

B Additional controlled comparisons

These comparisons distinguish common-depth selection, training from route rewards, and feedback from updated hidden states. The reference setting is Qwen2.5-7B on GSM8K with the source-data partitions, adapted checkpoint, and evaluation rules used in the main experiments.

B.1 Common-depth selection

The common-depth sweep trains exact-depth actors at the seven ratios in Table 7, rounding rL to an integer. Each actor receives 200 collection rounds and 25,600 terminal route evaluations. Checkpoint and common-depth selection use source-validation answer accuracy, with ties resolved by smaller depth and then earlier checkpoint. The seven-actor sweep therefore uses 179,200 terminal evaluations per seed, excluding validation. This selection cost is separate from the single free-policy training budget.

The free-policy mean lies between the $2.00L$ and $2.25L$ candidates. Accuracy is considered jointly with realized execution depth and measured inference cost; a common depth near the free policy's mean does not reproduce its question-specific allocation of computation.

Table 7: Common-depth comparison on Qwen2.5-7B/GSM8K. Validation selects the common depth from the seven candidates; accuracy here is on the test split. The reference is $D^* = 2.50L = 70$ blocks.

Exact D/L	Mean blocks	Test accuracy (%)
1.50	42	92.9 ± 0.7
1.75	49	93.3 ± 0.6
2.00	56	93.6 ± 0.4
2.25	63	93.6 ± 0.9
2.50	70	93.8 ± 0.3
2.75	77	93.5 ± 0.5
3.00	84	93.2 ± 0.4
Free routing, mean $D/L = 2.16$	60.48 ± 1.40	94.9 ± 0.4

B.2 Search supervision and the Dr.LLM reference

The Dr.LLM reference retains its original skip/execute/repeat action space and search-derived supervision, with a structural maximum depth of $2L$; because we substitute our adapted checkpoint, we treat it as a reimplementation reference rather than a fully controlled comparison. To isolate the effect of the training procedure, we additionally construct a controlled search-supervised actor that shares our controller architecture, observations, route family, initialization, reward, and training questions, but replaces policy-gradient learning with search-derived supervision.

The RL and search-supervised actors are given the same budget of 25,600 completed-route evaluations. For the search-supervised variant, PUCT explores candidate routes and retains the highest-reward route found for each training question. The same actor is then trained with masked cross-entropy to imitate these routes. Thus, this comparison primarily tests whether learning directly from route rewards with RL provides an advantage over search followed by supervised imitation.

Interpretation of the component controls. The **controlled search-supervised actor** keeps the same controller architecture, observations, and route space as our RL method, but replaces PPO with search-derived supervision, testing whether RL itself is necessary. **Whole-route prediction from initial features** chooses the complete question-specific route using only the initial prompt representation and therefore tests whether repeated feedback from updated hidden states is necessary. The **learned input-independent RL policy** removes all prompt-dependent information, testing whether routing needs to depend on the input question. The **initialization checkpoint before RL** evaluates the Sequence-$2L$ imitation-trained controller before RL updates, measuring the gain from RL beyond initialization. **Correctness-only RL** ($\eta = 0$) removes the reference-likelihood term and uses $R = r_{\text{acc}}$, while **reference-likelihood-only RL** uses $R = Q$ and removes correctness, testing the contribution and standalone usefulness of the auxiliary likelihood signal. Finally, the **mixed-reward** variants with $\eta = 0.03$ and $\eta = 0.3$ test sensitivity to weaker and stronger weighting of the reference-likelihood term in $R = r_{\text{acc}} + \eta Q$.

Table 8 contains the GSM8K component comparisons. Table 9 gives the corresponding MathQA results: a common-depth actor tests depth selection, search supervision tests the training procedure, initial features test updated-state feedback, and correctness-only reward tests the likelihood auxiliary. These comparisons share the source-task checkpoint and do not alter the main-table evaluation.

B.3 Initial-feature and input-independent controls

The **initial-feature actor** retains question-specific information but constructs the complete route using only the representation available after the first mandatory block, without observing later hidden-state updates. It uses the same controller architecture as the sequential actor, so the comparison isolates the value of updated-state feedback rather than controller capacity. The **input-independent actor** additionally removes all prompt-dependent features, causing deterministic inference to select a common route across questions and thereby testing the value of question-specific routing. To ensure fair initialization, each observation variant receives the same Sequence-$2L$ imitation procedure and subsequent RL budget and optimization settings. Our primary comparison fixes execution depth

Table 8: Qwen2.5-7B/GSM8K controlled comparisons. Maximum and realized depths are reported separately.

Method	Max. D/L	Accuracy (%)	Mean D/L
Free RL, mixed ($\eta = 0.1$)	3	94.9 ± 0.4	2.16 ± 0.05
Common-depth RL ($D^* = 2.50L$)	2.50	93.8 ± 0.3	2.50
Dr.LLM reference	2	93.1 ± 0.5	1.82 ± 0.04
Free RL, cap $2L$	2	94.0 ± 0.3	1.59 ± 0.04
Exact-depth RL, $2L$	2	93.6 ± 0.4	2.00
Controlled search-supervised actor	3	92.7 ± 0.5	2.20 ± 0.06
Whole-route prediction from initial features	3	91.8 ± 0.6	2.18 ± 0.05
Learned input-independent RL policy	3	93.1 ± 0.4	2.25 ± 0.04
Initialization checkpoint before RL	3	91.8 ± 0.3	2.00 ± 0.00
Correctness-only RL ($\eta = 0$)	3	94.1 ± 0.5	2.18 ± 0.06
Reference-likelihood-only RL ($R = Q$)	3	93.8 ± 0.4	2.26 ± 0.07
Mixed reward, $\eta = 0.03$	3	94.4 ± 0.4	2.17 ± 0.05
Mixed reward, $\eta = 0.3$	3	94.6 ± 0.4	2.22 ± 0.06

Table 9: Qwen2.5-7B/MathQA component comparisons.

Method	Accuracy (%)	Mean D/L
Common-depth RL	84.8 ± 0.5	2.50
Search-supervised actor	85.1 ± 0.4	2.31 ± 0.06
Initial-feature RL	85.2 ± 0.5	2.30 ± 0.08
Correctness-only RL	85.1 ± 0.6	2.32 ± 0.04
Free RL, mixed reward	85.9 ± 0.4	2.28 ± 0.09

to exactly $2L$, isolating route selection from differences in total backbone depth; generated answer lengths may still differ.

Interpretation of the exact-$2L$ controls. All methods in Tables 10 and 11 use exactly $2L$ block executions, so the comparisons isolate route selection rather than total backbone depth. **Sequential RL with updated states** is the reference controller, which observes the newly updated hidden representation after each block execution and makes the next repeat/advance decision sequentially. **Whole-route prediction from initial features** (or **initial-feature RL**) retains question-specific information but predicts the complete route using only the representation available after the first mandatory block, testing whether feedback from later hidden states is useful. The **controlled search-supervised actor** uses the same controller and route space but replaces RL training with search-derived routes followed by supervised imitation, testing the role of the training procedure. The **learned input-independent RL policy** removes all prompt-dependent features, so deterministic inference produces a common learned route across questions, testing whether routing needs to depend on the input. The **modal-route control** applies the single route most frequently selected by the learned controller on the validation set to every test question, testing whether one representative learned route can explain the gains. Finally, the **shuffled-route control** preserves the set of routes selected for the test questions but randomly reassigns them across questions, testing whether the particular match between each question and its selected route matters.

Table 10: Exact-$2L$ controls on Qwen2.5-7B/GSM8K.

Method	Accuracy (%)
Sequential RL with updated states	93.6 ± 0.4
Whole-route prediction from initial features	91.1 ± 0.6
Controlled search-supervised actor	92.2 ± 0.4
Learned input-independent RL policy	92.7 ± 0.3
Modal-route control	92.2 ± 0.2
Shuffled-route control	90.2 ± 0.6

Table 11 compares the search-supervised, initial-feature, and sequential updated-state controllers at exactly $2L$ across all three backbones. Comparing sequential RL with the controlled search-supervised actor isolates the effect of the training procedure, while comparing sequential RL with whole-route prediction from initial features isolates the value of feedback from updated hidden states. These controls complement the matched-versus-shuffled route comparison in the main text, which instead tests whether matching routes to individual questions matters. Fixing the depth at exactly $2L$ controls the number of block executions per token, although total generation work can still differ because generated answer lengths may vary.

Table 11: GSM8K component comparisons at exactly $2L$.

Method	Llama-3.1-8B	Qwen2.5-7B	Qwen3-8B
Controlled search-supervised actor	87.9 ± 0.5	92.2 ± 0.4	93.1 ± 0.5
Whole-route prediction from initial features	87.7 ± 0.4	91.1 ± 0.6	92.1 ± 0.3
Sequential RL with updated states	89.9 ± 0.6	93.6 ± 0.4	94.2 ± 0.3

B.4 Reward and initialization controls

The initialization control evaluates the saved parameters under each relevant mask before RL: applying an exact-$3L$ mask to Sequence-$2L$ imitation parameters changes the executed route. Separate actors use $R = r^{\mathrm{acc}}$, $R = Q$, and $R = r^{\mathrm{acc}} + \eta Q$ for $\eta \in \{0.03, 0.1, 0.3\}$ from the same initialization. The route budget and optimization are fixed, all checkpoint selection uses validation answer accuracy, and every coefficient appears in Table 8. Reward scaling can affect optimization, and accuracy differences require interpretation together with realized computation.

B.5 Depth diagnostics

Depth-distribution statistics are computed across test questions separately within each pipeline seed before averaging statistics across seeds. Table 12 distinguishes this variation from the across-seed uncertainty of mean depth. Depth variation alone does not establish its utility; the common-depth and exact-depth assignment controls address that question. Associations between depth and correctness remain descriptive.

Table 12: Qwen2.5-7B/GSM8K depth distribution under cap $3L$. The uncertainty on mean depth is across seeds; the other statistics describe variation across questions within a seed, averaged over seeds.

Statistic	Value
Mean D/L	2.16 ± 0.05
Standard deviation of D/L across questions	0.40
10th percentile of D/L	1.61
Median D/L	2.14
90th percentile of D/L	2.71
Fraction reaching $3L$	4%

C Inference latency and memory

Inference-cost measurement. We report realized execution depth together with measured inference latency and occurrence-specific KV-cache memory. Repeated block executions share model parameters but retain separate KV-cache entries. Consequently, deeper routes generally require more cache capacity and longer execution time, although neither quantity is expected to scale exactly with realized depth because of sequence length, controller overhead, cache management, and hardware execution effects.

Hardware measurement protocol. Hardware evaluation uses 100 fixed held-out prompts, five warmup runs, three timed repetitions, batch size one, and synchronized GPU execution. We measure complete per-question latency and separately profile prompt processing and controller execution.

End-to-end latency includes prompt processing, route construction, and answer decoding. Fresh requests do not reuse caches across examples, while normal within-request KV caching remains enabled. Mean KV capacity is computed over the same hardware-evaluation prompts.

Inference cost of adaptive routing. Table 13 compares native execution, compulsory exact-$3L$ routing, and free routing with maximum depth $3L$. The free controller reaches 94.9% accuracy at a mean realized depth of $2.16L$, compared with 93.5% accuracy at exactly $3L$ for the exact-depth controller. Under the hardware protocol, free routing reduces mean KV capacity from 63.49 MiB to 45.36 MiB and end-to-end latency from 18.3 s to 13.1 s relative to exact-$3L$ execution. Native execution remains substantially cheaper, requiring 21.32 MiB of KV capacity and 6.2 s per question, but achieves lower accuracy.

Table 14 extends the latency comparison to the principal controlled variants. The common-depth $2.50L$ policy requires 15.94 s per question, whereas the adaptive controllers operate in a narrower range of approximately 13.1–13.4 s. In particular, the mixed-reward free controller has the lowest measured latency among these learned adaptive variants.

Table 15 separates prompt-processing cost from controller overhead and reports the corresponding mean occurrence-specific KV capacity. The free mixed-reward controller requires 0.464 s for prompt processing and 0.22 s for controller execution, compared with 0.566 s and 0.37 s, respectively, for exact-$3L$ routing. The component controls show similar prompt-processing costs but modest differences in controller overhead and KV capacity. These measurements characterize inference cost only and are separate from the training-cost accounting reported below.

Table 13: Qwen2.5-7B/GSM8K accuracy and depth, with KV capacity and latency.

Method	Acc. (%)	Mean D/L	KV (MiB)	Time (s)
Native	92.1	1.00	21.32	6.2
Exact $3L$	93.5	3.00	63.49	18.3
Free $\leq 3L$	94.9	2.16	45.36	13.1

Table 14: Inference latency for Qwen2.5-7B/GSM8K.

Method	Time (s)
Native	6.2
Exact $3L$	18.3
Common-depth RL, $2.50L$	15.94
Search-supervised actor	13.40
Initial-feature RL	13.26
Correctness-only RL	13.28
Free RL, mixed reward	13.1

Table 15: Prompt cost and mean occurrence-specific KV capacity.

Method	Prompt (s)	Controller (s)	Mean KV (MiB)
Native	0.121	0.00	21.32
Exact $3L$	0.566	0.37	63.49
Common-depth RL, $2.50L$	0.505	0.26	54.03
Search-supervised actor	0.469	0.18	48.66
Initial-feature RL	0.446	0.18	47.24
Correctness-only RL	0.467	0.21	49.79
Free RL, mixed reward	0.464	0.22	45.36

Training costs. Table 16 compares the measured training cost of the full RL controller with its principal controls. Full RL requires 80 GPU-hours per seed, of which rollout generation accounts for most of the cost. Correctness-only RL reduces the cost to 64 GPU-hours by eliminating the teacher-forced reference-scoring pass. The controlled search-supervised actor uses the same 25,600 completed-route evaluations as full RL but requires 84 GPU-hours. The initial-feature controller has a comparable cost of 79 GPU-hours.

The validation-selected common-depth baseline is substantially more expensive to obtain because depth selection requires training seven separate exact-depth actors. The resulting sweep uses 179,200 route evaluations and 578 GPU-hours per seed, compared with 25,600 evaluations and 80 GPU-hours for a single free-routing actor. Thus, although the selected common-depth actor can achieve similar final accuracy, identifying that depth through the prescribed sweep requires substantially more training compute.

Table 16: Training-cost breakdown for Qwen2.5-7B/GSM8K actor and control training.

Component	Full RL	Acc.-only	Search	Initial	7-depth sweep
Route evaluations (count)	25,600	25,600	25,600	25,600	179,200
Rollout generation	58	58	60	57	420
Reference scoring	16	0	17	16	116
Controller updates	2	2	3	2	14
Source validation	4	4	4	4	28
Total	80	64	84	79	578

D AGGREGATION AND PAIRED COMPARISONS

For each of five seeds s, we compute accuracy $a_s = 100c_s/N$ and mean normalized depth $d_s = N^{-1}\sum_j D_{sj}/L$, and report their mean and sample standard deviation across seeds. Question-level depth statistics (standard deviation, 10th, 50th, and 90th percentiles, and cap-reaching fraction) are first computed within each seed and then averaged across seeds. For two methods A and B evaluated on the same questions, the paired gain for seed s is $\Delta_s = a_s^A - a_s^B$; its mean and standard deviation are computed across the five seeds.