REDRAFT: YOUR VERIFIER IS SECRETLY A DEPTH SUPERVISOR FOR RECURRENT SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates LLM inference by having a cheap drafter propose token blocks that the target model verifies in parallel. Yet drafter depth is typically fixed by ablation and applied uniformly across blocks, despite substantial variation in draft difficulty: a per-block depth oracle gains $+1.02$ accepted tokens over the best fixed depth, while the cheapest depth suffices for 40% of blocks. Standard halting heuristics fail to exploit this variation: convergence- and confidence-based exits collapse to maximum depth, reducing throughput by 20–27%, and their tuned thresholds transfer poorly across tasks, model scales, and training data. We instead exploit a supervision signal unique to speculative decoding: the verifier reveals each draft's accepted length without additional model computation. We introduce ReDraft, a learned policy that selects drafter depth for each block. ReDraft uses a 2-layer weight-shared core iterated to variable depth and a lightweight head that predicts each block's acceptance curve from verifier-derived labels. A renewal-reward stopping rule converts these predictions into depth choices using measured hardware costs, while leaving verification—and thus losslessness—unchanged. ReDraft plans once per block with only 1.8% overhead, and a regret bound shows that throughput loss depends on errors in the relative values of candidate depths. Across five benchmarks and three targets spanning two model families, a single recipe yields tuning-free policies within 4–6% of the per-task tuned optimum, with no statistically significant difference on GSM8K, while allocating computation similarly to the oracle. The recurrent drafter achieves a $4.40\times$ lossless speedup on GSM8K, matching the released 5-layer DFlash drafter with 40% of its parameters and half its training data.

1 INTRODUCTION

Speculative decoding accelerates autoregressive generation by having a small drafter propose a block of tokens that a large target model verifies in parallel, while rejection sampling preserves the target model's output distribution (Leviathan et al., 2023; Chen et al., 2023). Recent block-diffusion drafters such as DFlash generate entire blocks in parallel and achieve over $6\times$ lossless speedup (Chen et al., 2026). Yet drafting computation remains fixed: the same depth is used for all blocks, even though blocks can differ substantially in how much computation they need. We find that this variation is substantial. Replaying one verification stream with depths 2, 5, 8, and 12, a per-block oracle gains $+1.02$ accepted tokens over the best fixed depth, while the cheapest depth already suffices on 40% of blocks (Figure 1). The useful amount of drafting computation is therefore a property of the block itself.

Can a drafter learn how much computation each block needs? Adaptive computation has been studied extensively in recurrent and iterative models (Graves, 2016; Banino et al., 2021; Schuster et al., 2022; Geiping et al., 2025). Speculative decoding gives us an unusual advantage for this problem: the computation decision is safe, and the verifier supplies its supervision. An early draft is not committed to the output. The target verifies it before any token is accepted, and allocating too little drafting computation can therefore reduce speed but cannot change the final output. At the same time, verification reveals exactly the quantity that matters for drafting: how many proposed tokens are accepted. The verifier therefore tells us not only whether a draft is correct, but how much

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.

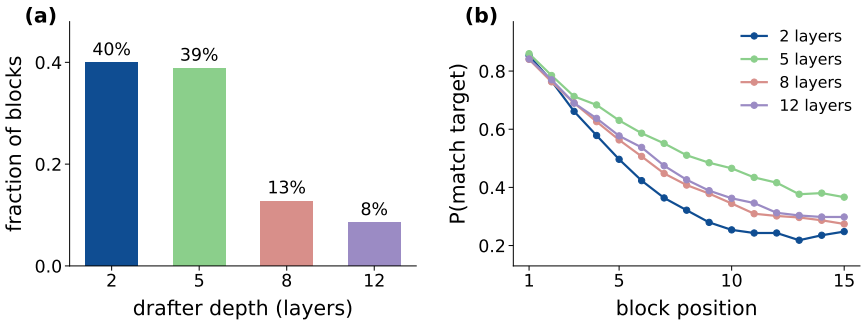

Figure 1: **Fixed drafter depth wastes computation on most blocks and starves the rest. (a)** The cheapest depth achieving the best acceptance is 2 layers on 40% of blocks and deeper than 5 on 21%, and the per-block oracle gains $+1.02$ accepted tokens over fixed-5. **(b)** Per-position correctness decays from 0.85 at the first drafted position to 0.35 at the last, the suffix-decay signature that motivates spending iterations where drafts are hard.

additional drafting computation is worth. Speculative decoding, in other words, already contains the supervision needed to adapt drafter depth. This signal also differs in kind from the internal signals adaptive computation usually relies on: a convergence criterion asks whether the drafter has stopped changing, while the verifier answers whether another iteration would produce enough accepted tokens to justify its cost.

We use this signal to build *ReDraft*, a verifier-supervised policy for adaptive drafting. We first make drafting depth a test-time computation variable by replacing DFlash's fixed-depth stack with a recurrent 2-layer weight-shared core. We train the core across depths with randomized unrolling and deep supervision, allowing one drafter to trade computation for acceptance at inference time. A lightweight predictor then estimates the block's acceptance curve from its first iteration. Finally, ReDraft chooses depth by optimizing throughput directly: a renewal-reward formulation combines predicted acceptance with measured costs, yielding a hardware-aware depth index with no task-specific threshold. The policy makes one decision per block and then runs the drafter open loop, adding only 1.8% to block latency while leaving target verification unchanged.

Across five benchmarks and three target models spanning two model families, a single ReDraft policy comes within 4%–6% of the best fixed-depth configuration tuned separately for each benchmark. It remains within 3% on four of five benchmarks for Qwen3-4B and within 5% for Qwen3-8B, with similar performance across the Llama target. ReDraft also tracks the per-block oracle: it spends fewer iterations on blocks whose acceptance saturates early and more on blocks that continue to benefit from refinement. In contrast, convergence-based halting runs to maximum depth on essentially all blocks at the published KL threshold of Geiping et al. (2025), a degeneracy our contraction analysis explains (Appendix C). These results show that the verifier can turn target-model feedback into a practical signal for allocating drafting computation. Our contributions are:

- We identify speculative decoding as a setting where adaptive computation is both *safe and directly supervised*: target-model verification provides per-block feedback on the utility of additional drafting computation.
- We introduce a recurrent-depth block-diffusion drafter that replaces DFlash's fixed-depth stack with a 2-layer weight-shared core. Randomized unrolling and deep supervision expose a variable test-time compute–acceptance trade-off within a single model.
- We develop a verifier-supervised acceptance-curve predictor and a hardware-aware depth policy. The predictor estimates the curve after one iteration, while a renewal-reward index converts it into a per-block depth choice without a tuned stopping threshold. A regret bound shows that the policy depends on differences between predicted curves instead of their absolute calibration.
- We characterize adaptive drafting across five benchmarks and three target models spanning two model families. ReDraft reaches within 4%–6% of the per-benchmark tuned fixed-depth frontier with a single tuning-free policy and 1.8% scheduling overhead.

2 Related Work

Speculative decoding and drafter design. Speculative decoding uses a cheap drafter followed by target-model verification with rejection sampling, preserving the target distribution (Leviathan et al., 2023; Chen et al., 2023), with blockwise parallel prediction predating this line (Stern et al., 2018). Modern drafters explore many ways to improve the proposals, including feature-space autoregression (Li et al., 2024; 2025), parallel prediction heads (Cai et al., 2024), target-aligned training (Zhou et al., 2024; Zhang et al., 2025), small parallel draft models (An et al., 2025), and tree-structured speculation (Miao et al., 2024; Chen et al., 2024). DFlash instead uses a target-conditioned block-diffusion drafter: per-layer KV injection provides target hidden states while the drafter generates a 16-token block in parallel (Chen et al., 2026). These approaches differ in architecture and proposal mechanism, but do not adapt the drafter's iterative computation to the difficulty of each block. We build on DFlash and make this computation adaptive.

Adaptive speculative decoding. Recent work adapts speculation along several dimensions: the number of tokens drafted before verification (Liu et al., 2024; 2025), the drafter or cascade tier, the layers used for self-speculation (Xia et al., 2025; Cha et al., 2026), speculation hyperparameters and drafter selection through online optimization (Hou et al., 2025; Hu et al., 2026b; Liu et al., 2026c), drafting policies trained with reinforcement learning (Zhang et al., 2026), and the verification procedure itself (Bachmann et al., 2025). These methods adapt the *speculation strategy*, while ReDraft adapts the computation performed *within the drafter* on each block. KnapSpec selects layer subsets for self-speculative drafting from internal similarity signals (Cha et al., 2026), and SpecBlock's deployment-time bandit uses verifier feedback to select among complete configurations (Shi et al., 2026). We use the verifier at a finer granularity: its feedback supervises the per-block depth decision itself. Orthogonal work improves what a single draft pass produces through parallel or tree-structured generation (Xiao et al., 2024; Liu et al., 2026b; Hu et al., 2026a) and repair-based correction (Liu et al., 2026a), and our depth policy can serve as a compute controller for drafters that support repeated refinement. Separately, acceptance-aware training uses verifier-derived acceptance rates as a *training objective* for the drafter (Samarin et al., 2026; Zou et al., 2026). ReDraft uses the same signal for a different purpose: deciding how much computation to spend at inference time. The two are complementary, as we show in §5.5.

Adaptive computation and learned halting. Learned halting has a long history, from stochastic computation in ACT and PonderNet (Graves, 2016; Banino et al., 2021) to depth-adaptive transformers (Elbayad et al., 2020), calibrated early exit (Schuster et al., 2022), token-level routing (Raposo et al., 2024), and gated pondering (Song et al., 2026). Recurrent-depth models provide another way to trade computation for quality by repeatedly applying shared layers (Dehghani et al., 2019; Bai et al., 2019; Geiping et al., 2025; Bae et al., 2025; Moosa et al., 2026). These methods generally infer when to stop from internal states or task-level objectives. ReDraft instead uses target-model verification to obtain an external signal for the utility of another drafting iteration. This distinction matters empirically: the KL-convergence criterion of Geiping et al. (2025) runs to maximum depth on all five benchmarks we test, a failure our contraction analysis explains (Appendix C).

Recurrent-depth and diffusion-based generation. ReDraft combines recurrent computation with block-diffusion drafting, connecting two lines of work that have largely developed independently. Geiping et al. (2026) use diffusion-style parallel sampling to accelerate a recurrent-depth model, whereas we use recurrent computation inside the drafter of a speculative decoder. Diffusion language models likewise adapt block sizes and unmasking schedules using confidence-based signals (Nie et al., 2025; Arriola et al., 2025; Wu et al., 2025). ReDraft differs in having a verifier that can directly evaluate the consequence of additional drafting computation, turning depth selection from a proxy-based halting problem into a supervised allocation problem.

3 ReDraft: Learning How Deep to Draft

The depth–acceptance trade-off. Each speculative block presents a small compute-allocation problem: additional drafting iterations cost computation, but can produce a draft that the target accepts for more tokens. Let $a_b(i)$ denote the number of tokens accepted from block b after i drafting iterations. The sequence

$$a_b(1), \ldots, a_b(R)$$

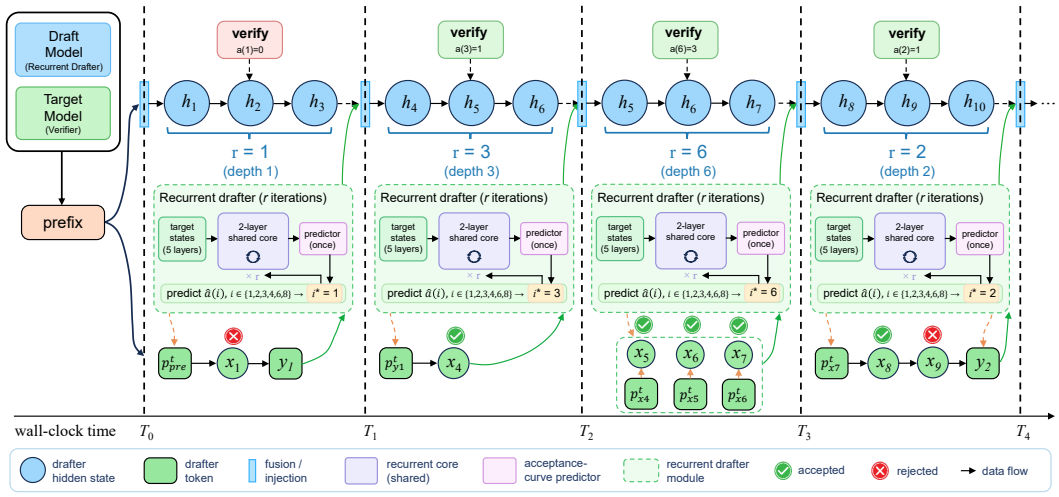

Figure 2: **ReDraft allocates drafting computation once per block.** Target hidden states are fused into a 2-layer weight-shared drafter that proposes a 16-token block. After one iteration, a lightweight predictor estimates its acceptance curve over candidate depths. ReDraft selects the depth with a throughput-optimal index, runs open loop to that depth, and verifies the draft as usual.

defines the block's *acceptance curve*, which captures the value of additional drafting computation.

Speculative decoding gives us an unusual advantage for learning this curve: the verifier already provides the supervision we need. Whenever it checks a draft at depth i, it reveals $a_b(i)$. ReDraft uses these outcomes to predict the acceptance curve after one drafting iteration, then chooses the depth that maximizes long-run decoding throughput. Because we modify only the proposal computation and leave target verification unchanged, the resulting decoder preserves speculative decoding's losslessness guarantee (Leviathan et al., 2023).

3.1 Making Depth a Computation Variable

We make drafting depth a test-time computation variable by replacing DFlash's five distinct drafter layers with a $K{=}2$-layer core whose parameters are shared across recurrent iterations (Chen et al., 2026). A block can therefore be drafted for any $r \in [1, 8]$ iterations using the same parameters. We retain DFlash's target conditioning.

We fuse hidden states from five target layers once per block,

$$H_t = \mathrm{RMSNorm}\Big(W_c[H^{(l_1)}; \ldots; H^{(l_5)}]\Big),$$

and inject them into each core layer as additional keys and values:

$$Q_i = W_i^Q h, \qquad K_i = [W_i^K H_t; W_i^K h], \qquad V_i = [W_i^V H_t; W_i^V h]. \tag{1}$$

We compute these projections once and reuse them across iterations. At iteration j, a learned iteration embedding and a gated residual update refine the draft state:

$$h^{(j+1)} = g \odot \mathrm{core}(h^{(j)} + u_j; H_t) + (1 - g) \odot h^{(j)}, \qquad g = \sigma\Big(W_g[\mathrm{core}(\cdot); h^{(j)}]\Big). \tag{2}$$

The frozen target LM head can decode a complete draft after any iteration.

Making depth variable at inference time requires the drafter to remain useful across depths. We therefore randomize the number of recurrent iterations during training and supervise the intermediate drafts. For each training example, we sample the number of iterations from a clipped log-normal distribution, $r \sim \lfloor \mathrm{LogNormal} \rfloor$, with $r \in [1, 8]$ and mean 4 (Geiping et al., 2025). We backpropagate through the final four iterations and apply deep supervision at each readout in that window $\mathcal{W}(r)$:

$$\mathcal{L} = \sum_{j \in \mathcal{W}(r)} 2^{-(r-1-j)} \sum_{k=1}^{15} w_k \, \mathrm{CE}\Big(\mathrm{head}(h_k^{(j)}), x_k\Big), \qquad w_k = e^{-(k-1)/\gamma}. \tag{3}$$

This training objective makes shallow iterations useful on their own instead of treating them as incomplete versions of the deepest draft. Removing deep supervision reduces acceptance at $r = 1$ by 1.4 tokens (§5.5). The resulting drafter has 248M parameters, just 46% of the 537M-parameter fixed-5 DFlash drafter.

3.2 Learning the Acceptance Curve

The verifier gives us the labels needed to learn when additional drafting is worthwhile. Whenever the target checks a draft at depth i, it reveals $a_b(i)$, the number of accepted tokens. We use these verifier outcomes to train a lightweight acceptance-curve predictor.

For training, we harvest $a_b(1{:}R)$ offline by evaluating all candidate depths for the same block while reusing the context KV cache. Evaluating 600 prompts yields approximately 600K labeled examples in under an hour on one GPU. We train a 5M-parameter MLP on the mean-pooled drafter state and sorted per-position top-token probabilities after the first iteration, using soft-label cross entropy over the 17 possible acceptance lengths.

At inference, the predictor runs once after the first iteration and outputs the acceptance curve $\hat{a}_b(i)$ for candidate depths $i \in \{1, 2, 3, 4, 6, 8\}$. ReDraft then runs the drafter open loop to the selected depth, avoiding intermediate LM-head reads and synchronization. On held-out blocks, the predictor achieves 1.44-token MAE, 0.70 correlation, and -0.03 mean bias.

3.3 Choosing Depth by Throughput

Predicting an acceptance curve does not by itself tell us how much computation to spend. An additional iteration is useful only when its acceptance gain justifies its latency. We therefore derive the depth decision directly from the decoding objective.

For a policy π that selects depth i_b for each block, long-run throughput is

$$R(\pi) = \frac{\mathbb{E}_b[a_b(i_b) + 1]}{\mathbb{E}_b[c(i_b)]}, \tag{4}$$

where $c(i) = i\,t_d + t_v$ is the cost of i drafting iterations at measured latency t_d each, followed by one target verification pass at latency t_v. The numerator includes the standard bonus token.

The ratio in Eq. 4 leads to a simple index rule (Dinkelbach, 1967). Let $\lambda^* = \max_\pi R(\pi)$. Then an optimal policy selects

$$i_b \in \arg\max_i \left[a_b(i) + 1 - \lambda^* c(i)\right]. \tag{5}$$

The multiplier λ^* acts as a global price for computation, converting drafting latency into the number of accepted tokens it must justify. ReDraft therefore needs no stopping threshold. It compares the predicted value of each candidate depth against its measured latency cost. At inference, we estimate λ^* from running throughput and measure t_d and t_v on the serving hardware at startup. Replacing a_b with the predicted curve $\hat{a}_b$ gives the deployed policy.

Only relative curve errors matter. The index depends on differences between candidate depths instead of on their absolute acceptance values. Define

$$\varepsilon_b = \max_{i,j} \left|[\hat{a}_b(i) - \hat{a}_b(j)] - [a_b(i) - a_b(j)]\right|.$$

If $\hat{\pi}$ applies the index rule to $\hat{a}_b$, then

$$\lambda^* - R(\hat{\pi}) \le \frac{\mathbb{E}_b[\varepsilon_b]}{\mathbb{E}_b[c(i_b)]}, \tag{6}$$

and adding a constant to all predicted acceptance values for a block leaves its selected depth unchanged. The policy therefore depends on the *relative shape* of the acceptance curve instead of its absolute calibration.

This distinction also tells us what to measure when evaluating the predictor. At our operating point, $\mathbb{E}[c] \approx 48$ ms and $\lambda^* \approx 118$ tokens/s. A pairwise error of 0.1 tokens therefore implies at most approximately 2 tokens/s of throughput loss under Eq. 6. Absolute MAE can be much larger without changing the selected depth when its error is shared across candidate depths. We therefore evaluate pairwise curve differences in §H, alongside absolute calibration.

4 EXPERIMENTAL SETUP

Training. We train recurrent drafters on 226K target-regenerated sequences from UltraChat, CodeAlpaca, and OpenMathInstruct-2. Each drafter is trained for 6 epochs on 8 data-center GPUs using the recurrent architecture and randomized-unrolling objective described in §3. Unless stated otherwise, ablations use the same world size and number of optimization steps, allowing us to isolate the effect of the ablated component. The released DFlash checkpoints use a different training corpus, and comparisons to their absolute acceptance or throughput therefore include this data difference. We describe the reproduction protocol (§D) and matched-data comparisons (§I).

Evaluation. We evaluate speculative decoding on GSM8K, MATH-500, HumanEval, MBPP, and MT-Bench at temperatures 0 and 1. We measure acceptance length τ, tokens per second, lossless speedup over vanilla autoregressive decoding, and realized mean drafter depth $\bar{r}$. Unless noted otherwise, headline results use 128 prompts and three prompt-subset seeds. We use the same target model, verifier, KV-cache implementation, and serving harness across policies. Fixed-depth drafters run open loop for their selected depth, matching the serving path used by ReDraft. We verify losslessness directly by checking token identities against the target model for each policy variant.

Depth allocation. Our primary policy, *plan-once*, predicts the full acceptance curve after the first recurrent iteration and commits to a depth for the block. This prediction happens once per block, and the drafter can then run open loop without intermediate readouts or synchronization. We compare plan-once against fixed depths $r \in \{1, \ldots, 8\}$, which define the tuned fixed-depth frontier, and against two adaptive alternatives. The first is a closed-loop value head that re-estimates the value of continuing after each iteration. The second is a policy-gradient controller trained from the same verifier labels. We also evaluate heuristic controllers based on KL convergence, model confidence, and acceptance feedback.

Oracle and verifier supervision. To measure the available headroom before introducing a learned policy, we evaluate two depth oracles. The fixed-depth oracle selects among separately trained drafters, while the iteration oracle selects among recurrent depths using the acceptance curve of a single trained drafter. For learned allocation, we harvest verifier labels for candidate depths and train the acceptance-curve predictor on these block-level outcomes. The predictor is evaluated only from the first-iteration state and therefore requires no additional verifier passes at deployment. Full label harvesting, policy specifications, and implementation details appear in §D.

5 RESULTS

5.1 TRAINING MAKES RECURRENT DEPTH USEFUL

We first ask whether a recurrent drafter can make use of additional computation at all. Simply sharing weights across layers is not enough. When we iterate the released fixed-5 DFlash drafter, acceptance drops from 4.47 to 3.29 tokens on the second pass. The model was trained to produce a good draft after one pass, and nothing in its training objective asks later passes to improve it. We therefore train the shared core with randomized unrolling and deep supervision. This changes the behavior substantially. On GSM8K, acceptance at r=1/2/3/4 grows from 3.05/3.63/3.64/3.68 after one epoch to 4.17/5.29/5.50/5.57 after five epochs (Figure 3). The gain from the second iteration grows from 0.6 to 1.1 tokens, showing that training does not merely preserve the one-pass draft. It teaches the recurrent core to extract useful computation from additional iterations.

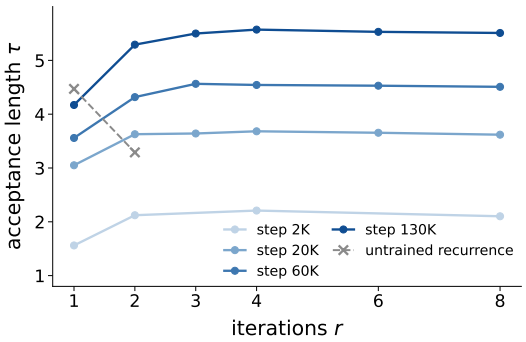

Figure 3: **Training makes additional recurrent computation useful.** Acceptance length versus iteration r at four training checkpoints (GSM8K, temperature 0). The untrained recurrence degrades with iteration, while training makes later iterations increasingly useful and the curve saturates near r=4.

We can now vary computation depth at inference time. At the final checkpoint, acceptance grows from 3.99 at r=1 to 5.34 at r=4, with most of the gain arriving by the third iteration. Throughput peaks earlier, at r=2–3, because each additional iteration eventually costs more than the accepted tokens it recovers. At r=3, the recurrent drafter reaches a 3.88× vanilla speedup. The released fixed-5 DFlash drafter reaches 4.57×, but uses more than twice the parameters and a different training corpus with 4× the data. We therefore get more than a cheaper fixed-depth drafter: computation depth is now a runtime variable, the drafting analogue of input-dependent test-time compute allocation (Snell et al., 2025; Muennighoff et al., 2025).

We next ask whether blocks actually have different computational needs. They do. A *depth* oracle chooses among four separately trained fixed-depth drafters and gains 1.02 accepted tokens over the best fixed depth across 2,297 GSM8K blocks. A single recurrent drafter captures much of the same opportunity: its *iteration* oracle gains another 0.43 tokens over the best fixed r while reducing mean depth from 4 to 2.0.

These oracles reveal why a fixed depth leaves throughput on the table. Some blocks continue to benefit from more drafting, while others reach their useful depth early. A fixed-depth drafter must compromise between them. The heterogeneity is not even confined to blocks. Figure 1b shows that token positions within the same block can stop benefiting from additional computation at different iterations. The opportunity is therefore not to find a better global depth. It is to decide, block by block, when another iteration is worth its cost.

5.2 How much of the oracle can we recover?

We now ask whether we can recover this oracle without evaluating each depth at inference time. We train a small predictor after the first iteration to estimate the block's acceptance curve, then use the predicted curve in our renewal-reward index to choose depth. Figure 4 shows the resulting trade-off on GSM8K. The fixed-depth runs trace the tuned frontier, while the plan-once policy reaches nearly the same peak without selecting a depth for this task.

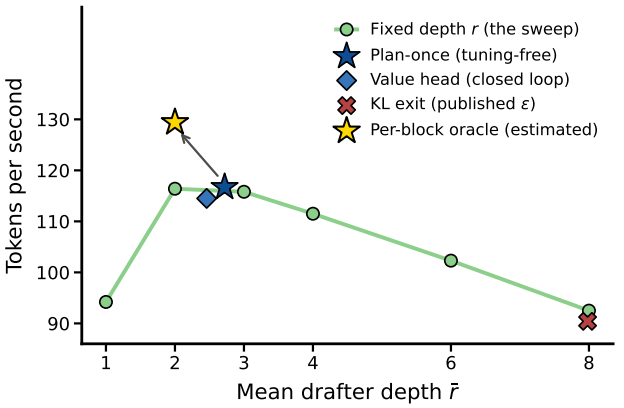

Figure 4: **Adaptive depth recovers most of the fixed-depth frontier.** Throughput versus mean drafter depth on GSM8K with Qwen3-4B. Fixed depths trace the tuned frontier, while plan-once reaches a similar peak without per-task tuning. The published convergence exit runs to maximum depth, and the per-block oracle marks the remaining headroom.

We find the same behavior across benchmarks. On the final Qwen3-4B drafter, the plan-once policy comes within 3% of the best fixed depth on four of five benchmarks and slightly exceeds it on GSM8K, reaching 3.85× versus 3.84× (Table 1). Three-seed replication gives 113.8±3.0 versus 114.3±2.3 tokens/s on GSM8K, with a similarly small difference on HumanEval. MT-Bench is the main outlier, where the policy falls 4.3% below the replicated fixed-depth result. The full benchmark grid appears in Figure 8 (§J).

We can also see what the policy has learned. The same predictor and index rule serve all tasks, yet the resulting mean depth ranges from 2.4 to 2.8. Mathematics concentrates computation near r=3, code favors shallower drafts, and open-ended dialogue uses a broader range of depths (Figure 5, with a token-level trace in §G). Thus, the policy is not learning another globally preferred depth. It uses the block state to decide where another iteration is useful.

We then test whether this decision must be made repeatedly during drafting. A closed-loop value head makes a new decision after each iteration, but these intermediate decisions add 16% to per-block latency. Our plan-once predictor adds only 1.8% while reaching 2–3% higher throughput at matched acceptance (Table 2). Predicting the full curve once therefore captures most of the benefit without repeatedly interrupting drafting.

Table 1: **Halting policies on the final Qwen3-4B drafter** at the definitive protocol (200 prompts for GSM8K, MATH-500, and MBPP, all 164 HumanEval problems, all 80 MT-Bench prompts, temperature 0, fresh vanilla baselines). Cells give speedup over vanilla decoding with acceptance length τ in parentheses. Temperature-1 results, the Qwen3-8B grid, and threshold sweeps appear in §J.

Policy	GSM8K	MATH-500	HumanEval	MBPP	MT-Bench
Best fixed r (per task)	3.84 (5.06)	3.96 (5.50)	2.61 (3.72)	2.68 (3.56)	2.07 (2.52)
Plan-once policy	**3.85** (5.31)	3.89 (5.42)	2.60 (3.67)	2.62 (3.61)	1.91 (2.53)
Closed-loop value head	3.78 (5.26)	3.74 (5.39)	2.52 (3.65)	2.57 (3.53)	1.86 (2.51)
KL exit, published ϵ	2.98 (5.35)	2.92 (5.51)	2.03 (3.75)	2.02 (3.66)	1.49 (2.53)

Finally, we examine where the remaining gap comes from. On held-out harvested blocks, the Qwen3-4B predictor achieves Kendall $\tau = 0.74$ between predicted and true depth orderings, compared with $\tau = 0.60$ for Qwen3-8B. Our regret bound depends on errors in the differences between candidate depths (§H), and a predictor can therefore systematically bias the entire curve and still choose the right depth if it preserves which iterations are worth their cost.

5.3 Convergence does not predict drafting value

We next test whether standard adaptive-computation signals can recover the same opportunity. They do not. KL convergence, hidden-state convergence, confidence thresholds, and acceptance-feedback control all run to maximum depth under their published or natural default settings, giving up 20–27% throughput across both target models and temperatures. We trace this failure to a mismatch in what the signals measure. They tell us how the drafter is changing, without telling us whether another iteration will increase target acceptance. Claim 1 formalizes this distinction. On 107K held-out blocks, even with thresholds tuned on the evaluation set, the best confidence rule leaves 0.57 tokens of mean utility regret and the best state-convergence rule collapses onto the fixed depth, while the predictor reaches 0.32 (§H). The acceptance curve retains information that these internal signals discard (Appendix A).

We then ask whether threshold tuning can recover the missing signal. On GSM8K, increasing the KL threshold by $400\times$ is enough to reach the fixed-depth frontier. But the threshold does not transfer: its best setting spans the full tested range across ten model–benchmark pairs, changes between model scales on the same task, and shifts again after training with $4\times$ more data (Table 5, §J). The threshold therefore encodes a depth choice indirectly, and its mapping to useful depth changes with the drafter and task. The feedback controller exhibits the same limitation, remaining 10% below the frontier even after tuning.

We also train a policy-gradient baseline on the same verifier labels under the earlier 96-prompt protocol. It reaches similar wall-clock throughput to the plan-once predictor, but its offline allocation achieves only the best-fixed reward (0.147 versus 0.145 for the predictor and 0.170 for the oracle). The difference is the supervision itself: verifier harvesting labels each candidate depth, while policy-gradient training observes only the reward of the final allocation.

5.4 The same allocation principle transfers

We next test whether this allocation principle survives changes in model scale and family. On Qwen3-8B, throughput peaks at $r{=}2$ with a $3.61\times$ speedup, while the published KL exit again runs to maximum depth and loses 27%. The plan-once predictor remains within 4–5% of the best fixed depth across the five benchmarks under three-seed replication. Its depth ordering does weaken, however: Kendall τ falls from 0.74 at 4B to 0.60 at 8B. Scaling therefore preserves the useful depth axis, while prediction becomes less precise about the value of each depth.

We find the same pattern across model families. On Llama-3.1-8B-Instruct with block size 10, acceptance again rises with recurrent depth before saturating, and throughput peaks at $r{=}2$ with a $2.79\times$ lossless speedup. A single predictor comes within 3.4–5.4% of the tuned frontier across five benchmarks, while the published convergence exit loses 34%. We can also improve transfer by

expanding the verifier labels: adding a 300-prompt MATH-500 harvest reduces the gap from 7.6% to 5.4%. The depth axis transfers across models, while the mapping from block states to acceptance shifts with the task distribution.

Finally, we increase the training data to test whether the same structure survives a stronger drafter. With the full 400K-sequence corpus, the recurrent model reaches a $4.40\times$ GSM8K speedup, matching the released five-layer DFlash drafter at $4.39\times$ while using 40% of its parameters and half as many sample-epochs. A five-distinct-layer drafter trained with the same data and recipe trails the recurrent model across all five benchmarks under seven-to-ten-seed replication, isolating the effect of recurrent depth under matched training conditions. A retrain on the released corpus source performs below the open-corpus model, pointing to data mixture and coverage as additional factors in drafter quality. Full replication results appear in §I.

5.5 What determines the quality of adaptive depth?

Our ablations separate two requirements for adaptive depth. We first need to make additional computation useful. We then need to learn where it is useful. Randomized unrolling and deep supervision determine the first. Removing deep supervision costs 1.4 accepted tokens at $r{=}1$. Training only at $r{=}4$ lowers acceptance at each tested depth, including $r{=}4$, while a 1-layer core is insufficient and a 3-layer core provides no gain at matched optimization steps, supporting the $K{=}2$ design. An acceptance-weighted loss variant (Samarin et al., 2026) also lowers drafter quality under our recipe, but its predictor allocates depth comparably. The scheduling mechanism therefore does not depend strongly on the particular loss used to train the drafter.

We next isolate the predictor. Supervising it on curve differences, the quantity that enters Eq. 6, performs comparably to absolute-curve supervision at four matched seeds, within 1% on three of five benchmarks and within 4% on the remaining two. The predictor does not need to reconstruct the acceptance curve exactly. It needs to preserve the relative value of competing depths.

Temperature gives us a direct test of this requirement. A head harvested at temperature 0 and served at temperature 1 widens the parity gap to about 3%. Temperature-matched labels, using expected accepted length under rejection sampling, reduce the GSM8K gap to 1.9%, with similar improvements on Qwen3-8B and Llama. On Llama, reducing the label budget to two thirds also hurts performance, showing that label coverage remains important once the temperature mismatch is removed.

Finally, we test whether calibration remains valid as the drafter changes. Reusing a head for 100K training steps increases its acceptance-curve error from 1.04 to 1.25 tokens. Per-position freezing, in contrast, leaves acceptance and throughput unchanged at block size 16 (§E). Across these experiments, we find a consistent division of labor: recurrent training creates a useful depth axis, verifier supervision identifies where to use it, and calibration determines how much of the oracle headroom the deployed policy recovers.

6 Conclusion

Speculative decoding offers an unusual setting for adaptive computation: verification preserves losslessness while directly revealing the value of additional drafting. ReDraft turns this signal into a practical depth policy. A recurrent drafter makes depth a useful computation axis, the verifier supplies the acceptance curve for each block, and a throughput-based index converts that curve into a depth choice using measured system costs. Across target models and model families, a single acceptance-curve predictor recovers most of the fixed-depth frontier without per-task tuning, with only 1.8% scheduling overhead. More broadly, our results show that the value of additional drafting computation varies across blocks and can be learned from the verification signal already present in speculative decoding.

Limitations. Our evaluation focuses on single-request serving. The straggler analysis in §F shows that depth-sorted batching recovers much of the single-request benefit, while fully continuous serving remains open. At Qwen3-8B, the predictor still trails the fixed-depth frontier by 3–5%, with difference calibration accounting for much of the gap. We also fix the block length, leaving a second computation budget for future adaptive policies to control. Finally, our experiments disable thinking modes, leaving verifier-supervised depth allocation for reasoning-enabled drafters untested.

AI USE STATEMENT

In this work, we used generative AI tools for designing or providing feedback on research methodology or experiments, implementing methods, and interpreting results. We have not used generative AI tools for generating synthetic data sets, developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, or proposing or refining hypotheses. Assisting with translation, cleaning and reformatting datasets, and supporting qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for editing the paper to improve readability and for identifying relevant literature. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness, AI-suggested interpretations were checked against outputs, and AI-identified references were verified against their source records. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We will release the full implementation upon acceptance. The release will include the recurrent drafter and acceptance-curve predictor, training recipes, verifier-label harvesting, policy implementations, and benchmark drivers. All implementation details needed to reproduce the experiments are specified in §3 and §D, including the recurrent unrolling distribution, supervision weights, candidate depths, and label-harvesting procedure. We report the measured hardware costs used by the stopping rule and provide the protocol used to obtain them. We also verify losslessness by checking token-identical greedy outputs against the target model across policy variants and validate the implementation against two released drafter checkpoints. The release will also include the open prompt sources and the per-block evaluation records used to produce the reported analyses.

REFERENCES

Talor Abramovich, Maor Ashkenazi, Izzy Putterman, Benjamin Chislett, Tiyasa Mitra, Bita Darvish Rouhani, Ran Zilberstein, and Yonatan Geifman. SPEED-bench: A unified and diverse benchmark for speculative decoding. In *International Conference on Machine Learning (ICML)*, 2026.

Zihao An, Huajun Bai, Ziqiong Liu, Dong Li, and Emad Barsoum. PARD: Accelerating LLM inference with low-cost PARallel draft model adaptation. *arXiv preprint arXiv:2504.18583*, 2025.

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations (ICLR)*, 2025.

Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali Thabet, and Jonas Kohler. Judge decoding: Faster speculative sampling requires going beyond model alignment. *arXiv preprint arXiv:2501.19309*, 2025.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *International Conference on Machine Learning (ICML)*, 2024.

Seongjin Cha, Gyuwan Kim, Dongsu Han, Tao Yang, and Insu Han. KnapSpec: Self-speculative decoding via adaptive layer selection as a knapsack problem. In *International Conference on Machine Learning (ICML)*, 2026.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *International Conference on Machine Learning (ICML)*, 2026.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable, robust, and hardware-aware speculative decoding. *arXiv preprint arXiv:2402.12374*, 2024.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations (ICLR)*, 2019.

Werner Dinkelbach. On nonlinear fractional programming. *Management Science*, 13(7):492–498, 1967.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *International Conference on Learning Representations (ICLR)*, 2020.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Jonas Geiping, Xinyu Yang, and Guinan Su. Efficient parallel samplers for recurrent-depth models and their connection to diffusion language models. In *International Conference on Machine Learning (ICML)*, 2026.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Yunlong Hou, Fengzhuo Zhang, Cunxiao Du, Xuan Zhang, Jiachun Pan, Tianyu Pang, Chao Du, Vincent Y. F. Tan, and Zhuoran Yang. BanditSpec: Adaptive speculative decoding via bandit algorithms. In *International Conference on Machine Learning (ICML)*, 2025.

Lanxiang Hu, Zhaoxiang Feng, Yulun Wu, Haoran Yuan, Yujie Zhao, Yu-Yang Qian, Bojun Wang, Peng Zhao, Daxin Jiang, Yibo Zhu, Tajana Rosing, and Hao Zhang. JetSpec: Breaking the scaling ceiling of speculative decoding with parallel tree drafting. *arXiv preprint arXiv:2606.18394*, 2026a.

Xinyi Hu, Yuhao Shen, Baolin Zhang, Hengxin Zhang, Jun Dai, Shuang Ge, Lei Chen, Yue Li, and Mingcheng Wan. ECHO: Elastic speculative decoding with sparse gating for high-concurrency scenarios. *arXiv preprint arXiv:2604.09603*, 2026b.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning (ICML)*, 2023.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *International Conference on Machine Learning (ICML)*, 2024.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*, 2025.

Aofan Liu, Jingxiang Meng, Fangxin Liu, and Yongbiao Chen. CURE: Local uncertainty repair for block-parallel speculative decoding. *arXiv preprint arXiv:2608.00531*, 2026a.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding via double early exiting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Fuliang Liu, Xue Li, Ketai Zhao, Yinxi Gao, Ziyan Zhou, Zhonghui Zhang, Zhibin Wang, Wanchun Dou, Sheng Zhong, and Chen Tian. DART: Diffusion-inspired speculative decoding for fast LLM inference. *arXiv preprint arXiv:2601.19278*, 2026b.

Hongyi Liu, Jiaji Huang, Zhen Jia, Youngsuk Park, and Yu-Xiang Wang. Not-a-bandit: Provably no-regret drafter selection in speculative decoding for LLMs. In *International Conference on Learning Representations (ICLR)*, 2026c.

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. PEARL: Parallel speculative decoding with adaptive draft length. In *International Conference on Learning Representations (ICLR)*, 2025.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating generative large language model serving with tree-based speculative inference and verification. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

Ibraheem Muhammad Moosa, Suhas Lohit, Ye Wang, Moitreya Chatterjee, and Wenpeng Yin. Understanding dynamic compute allocation in recurrent transformers. *arXiv preprint arXiv:2602.08864*, 2026.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

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

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.

Alexander Samarin, Sergei Krutikov, Anton Shevtsov, Sergei Skvortsov, Filipp Fisin, and Alexander Golubev. LK losses: Direct acceptance rate optimization for speculative decoding. In *International Conference on Machine Learning (ICML)*, 2026.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Weijie Shi, Qiang Xu, Fan Deng, Yaguang Wu, Jiarun Liu, Yehong Xu, Hao Chen, Jia Zhu, Jiajie Xu, Xiangjun Huang, Jian Yang, and Xiaofang Zhou. SpecBlock: Block-iterative speculative decoding with dynamic tree drafting. *arXiv preprint arXiv:2605.07243*, 2026.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *International Conference on Learning Representations (ICLR)*, 2025.

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. *arXiv preprint arXiv:2603.01914*, 2026.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dLLM: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.

Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. SWIFT: On-the-fly self-speculative decoding for LLM inference acceleration. In *International Conference on Learning Representations (ICLR)*, 2025.

Zilin Xiao, Hongming Zhang, Tao Ge, Siru Ouyang, Vicente Ordonez, and Dong Yu. ParallelSpec: Parallel drafter for efficient speculative decoding. *arXiv preprint arXiv:2410.05589*, 2024.

Jiebin Zhang, Zhenghan Yu, Liang Wang, Nan Yang, Eugene J. Yu, Zheng Li, Yifan Song, Dawei Zhu, Xingxing Zhang, Furu Wei, and Sujian Li. Learning to draft: Adaptive speculative decoding with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026.

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning harmonized representations for speculative sampling. In *International Conference on Learning Representations (ICLR)*, 2025.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. DistillSpec: Improving speculative decoding via knowledge distillation. In *International Conference on Learning Representations (ICLR)*, 2024.

Xiandong Zou, Jianshu Li, Jing Huang, and Pan Zhou. Variational speculative decoding: Rethinking draft training from token likelihood to sequence acceptance. *arXiv preprint arXiv:2602.05774*, 2026.

A Renewal-reward objective and index rule

Each speculative block defines one renewal cycle: the drafter spends computation to propose a block, the target verifies it, and the accepted tokens determine the next cycle. For a policy π, let i_b be the depth selected for block b. The cycle produces $a_b(i_b) + 1$ tokens and incurs cost

$$c(i_b) = i_b t_d + t_v + t_o,$$

where t_d, t_v, and t_o denote drafting, verification, and policy-overhead costs. The renewal-reward theorem therefore gives the long-run token rate in Eq. 4. The overhead t_o reduces the achieved throughput but is independent of depth and therefore does not affect which depth maximizes throughput. This is why the main text omits it from the allocation rule.

We can now ask how the global throughput objective determines the depth of an individual block. Define

$$F(\lambda) = \max_{\pi} \mathbb{E}_b \left[a_b(i_\pi) + 1 - \lambda c(i_\pi) \right].$$

Because $c(i) > 0$, there is a unique λ^* satisfying

$$F(\lambda^*) = 0, \qquad \lambda^* = \max_{\pi} R(\pi).$$

At λ^*, the global problem separates across blocks:

$$i_b \in \arg\max_i \left[a_b(i) + 1 - \lambda^* c(i) \right].$$

The optimal policy therefore has a simple interpretation: each block receives additional drafting computation only when the resulting acceptance gain is worth its latency at the global decoding rate. The multiplier λ^* acts as a common price for computation, while the block-specific acceptance curve determines how much that computation is worth.

This also clarifies why the policy should not optimize the per-block ratio $(a_b(i)+1)/c(i)$. That ratio treats each block as an independent decoding problem. The index instead evaluates a depth against the throughput rate of the entire process. Blocks with different acceptance curves can therefore receive different amounts of computation even when they incur the same drafting cost.

In our operating regime, the two objectives happen to make nearly identical choices because acceptance changes little beyond the knee. One additional iteration changes the index objective by $\Delta a - \lambda^* t_d$, with $\lambda^* t_d \approx 0.25$ tokens and observed $|\Delta a| \leq 0.1$ beyond the knee. The agreement is a property of this operating regime, and the two objectives remain distinct in general.

The index also gives a useful lens for understanding the depth policies we compare. Fixed depth discards block-level variation in the acceptance curve. Convergence-based policies use the drafter's internal dynamics as a proxy for target acceptance. Feedback controllers use aggregate history instead of the current block's predicted value. ReDraft retains the signal needed by the index: the expected acceptance gain of each candidate depth for the block being decoded.

B REGRET UNDER ACCEPTANCE-CURVE ERROR

The index rule also tells us what the acceptance-curve predictor needs to get right. For a block b, define

$$u_b(i) = a_b(i) + 1 - \lambda^* c(i), \qquad \hat{u}_b(i) = \hat{a}_b(i) + 1 - \lambda^* c(i).$$

Let i^* maximize $u_b(i)$ and $\hat{\imath}$ maximize $\hat{u}_b(i)$. We measure prediction error in terms of pairwise differences,

$$\varepsilon_b = \max_{i,j} \left| [\hat{a}_b(i) - \hat{a}_b(j)] - [a_b(i) - a_b(j)] \right|.$$

By optimality of $\hat{\imath}$ under the predicted objective,

$$u_b(i^*) - u_b(\hat{\imath}) = [u_b(i^*) - \hat{u}_b(i^*)] + [\hat{u}_b(i^*) - \hat{u}_b(\hat{\imath})] + [\hat{u}_b(\hat{\imath}) - u_b(\hat{\imath})] \le \varepsilon_b.$$

Taking expectations and using $F(\lambda^*) = 0$ gives

$$\lambda^* - R(\hat{\pi}) \le \frac{\mathbb{E}_b[\varepsilon_b]}{\mathbb{E}_b[c(i_{\hat{\pi}})]},$$

which is Eq. 6.

The bound exposes an important invariance of the policy. Suppose

$$\hat{a}_b(i) = a_b(i) + \beta_b$$

for some block-dependent constant β_b. Each candidate depth then receives the same shift, leaving the selected depth unchanged. The policy therefore does not require an absolutely calibrated acceptance curve. It requires the relative values of candidate depths.

This distinction changes how we evaluate the predictor. At our operating point, $\mathbb{E}[c] \approx 48$ ms and $\lambda^* \approx 118$ tokens/s. A pairwise error of 0.1 tokens therefore implies at most approximately 2 tokens/s of throughput loss under Eq. 6. By contrast, the 1.44-token absolute MAE gives a much looser bound because common prediction bias cancels from the scheduling decision. The relevant failure mode is therefore not simply poor absolute calibration: a shift between harvesting and serving can change the relative ordering of candidate depths. We test this directly under checkpoint and temperature mismatch in §5.5.

C CONVERGENCE-BASED HALTING UNDER CONTRACTION

Convergence-based halting uses a different signal: how much the drafter changes between iterations. Let $p_b^{(i)}$ denote the drafter's token distribution for block b after iteration i, and define

$$\mathrm{KL}_b(i) = \mathrm{KL}\left(p_b^{(i)} \| p_b^{(i+1)}\right).$$

Claim 1 (Convergence halting under contraction)**.** *Suppose*

$$\mathrm{KL}_b(i+1) \le \rho_b \, \mathrm{KL}_b(i), \qquad \rho_b \in (0,1).$$

Then

$$\mathrm{KL}_b(i) \le \rho_b^{i-1} \mathrm{KL}_b(1),$$

and a rule that halts at the first i satisfying $\mathrm{KL}_b(i) < \epsilon$ stops by approximately

$$1 + \frac{\log(\mathrm{KL}_b(1)/\epsilon)}{\log(1/\rho_b)},$$

up to integer rounding and the strict inequality. The resulting stopping depth is determined by the drafter's convergence dynamics and carries no direct information about target acceptance. If the divergence remains above ϵ at the maximum available depth, the rule reaches that depth.

The claim makes the limitation of convergence-based halting explicit. Two blocks with the same $(\mathrm{KL}_b(1), \rho_b)$ receive the same depth even when additional drafting has different value for the target. Convergence can therefore support adaptive depth only when its dynamics happen to correlate with target acceptance. In our harvested trajectories, the divergence decays approximately geometrically and remains above the published threshold on most blocks, explaining why the convergence policy typically runs to its maximum depth. Raising the threshold moves the crossing earlier and changes the resulting effective depth, as shown in Table 5.

D EXPERIMENTAL PROTOCOL

Hardware constants. On our serving hardware at context length 1024, the median latency over 200 warm calls is 2.02 ms per recurrent core iteration and 41.5 ms per 16-token target verification. We evaluate sensitivity to these measurements by replaying GSM8K traces over a $\pm 20\%$ grid for both t_d and t_v. The optimal depth ordering is unchanged in eight of nine settings. In the remaining setting, selected depths differ only between configurations whose throughputs differ by 0.2%. Both quantities are medians of per-call timings already produced by the serving loop and can therefore be maintained as running estimates.

Reproduction gates. The released Qwen3-4B checkpoint reaches an acceptance length of 6.30 on GSM8K under the upstream harness, compared with the published 6.54. Loading its five layers into our recurrent implementation gives 6.73 at $r{=}1$. For Qwen3-8B, the corresponding values are 6.34 and 6.49. We also run a token-identity test against the target's greedy output for each policy variant, including adversarial per-position freezing, to verify losslessness directly.

Parameter and memory accounting. The recurrent drafter contains 248M parameters, compared with 537M for the matched five-distinct-layer drafter, or 46% as many parameters. Its transformer core contains two layers versus five, giving the 40% figure reported in the main text. The remaining parameters are shared components, including KV-injection projections and input/output layers. In bf16, the drafter occupies approximately 0.5 GB. The acceptance-curve predictor adds 5M parameters, or approximately 10 MB, together with one pooled state and one probability vector per block at inference. The complete accounting appears in Table 2.

Table 2: **Per-block policy overhead** at context length 1024 on the serving hardware. Values are medians over 200 warm calls. Planning once adds 1.8% to block latency, while closed-loop decisions add 16%.

Component	Time (ms)
Core iteration	2.02
LM-head readout	0.41
Softmax and feature pooling	0.07
Acceptance-curve predictor	0.21
Decision synchronization	0.03
Fixed $r{=}3$ block (open loop)	6.48
Plan-once block at $\bar{r}{=}2.7$	6.60
Closed-loop block at $\bar{r}{=}2.7$	7.50
KL exit block at $\bar{r}{=}8$	20.3

Ablation protocol. An initial ablation comparison used the main 8-GPU run against 2-GPU variants at matched sequence counts, which changed the optimizer steps and effective batch size by $4\times$. We therefore reran all reported ablations at a matched optimization budget of 2 GPUs and 64K optimizer steps, including a retrained baseline under the same regime.

Table 3: **Training ablations at matched optimization budget** (world size 2, 64K steps, GSM8K acceptance by iteration count).

Variant	$r{=}1$	$r{=}2$	$r{=}4$	$r{=}8$
$K{=}2$, deep supervision, unrolling	**3.86**	**4.84**	**5.06**	**5.03**
without deep supervision	2.42	3.55	3.86	3.87
fixed $r{=}4$ training	2.82	3.95	4.38	4.35
$K{=}1$ core	2.39	3.31	3.68	3.63
$K{=}3$ core	4.12	4.51	4.61	4.47
acceptance-weighted loss	—	3.34	3.55	—

Policy specifications. The KL-convergence exit uses the published and tuned thresholds (Geiping et al., 2025), with successive-iterate divergence averaged over active block positions. We also

evaluate hidden-state convergence, confidence thresholds at $0.6/0.7/0.8$ (Liu et al., 2024), and an acceptance-feedback EMA controller at default and tuned targets. The policy-gradient baseline trains a depth policy with REINFORCE on throughput reward using the same frozen trunk, features, and label budget as the acceptance-curve predictor. We evaluate the predictor with both the index rule and the per-block reward-to-cost ratio.

Acceptance-curve predictor. After the first recurrent iteration, the predictor receives the block-mean drafter hidden state, sorted per-position maximum probabilities, and a learned 32-dimensional iteration embedding. A LayerNorm followed by a two-layer SiLU MLP of width $\max(d/4, 128)$ produces logits over accepted lengths for candidate depths $i \in \{1, 2, 3, 4, 6, 8\}$. The hidden-state dimension is 2560 for Qwen3-4B and 4096 for Qwen3-8B and Llama-3.1-8B-Instruct. The predictor contains approximately 5M parameters and is trained for 8 epochs with learning rate 10^{-3} using soft-label cross-entropy and a 10% held-out split.

Label harvesting. For each block, we evaluate candidate depths $\{1, 2, 3, 4, 6, 8\}$ with a batched target forward pass over an expanded KV cache, obtaining accepted lengths for all candidates. We place the grid densely where acceptance curves still change and sparsely where they flatten. To quantify the cost of skipping depths 5 and 7, we re-harvested GSM8K labels on the full grid $\{1, \ldots, 8\}$. Across $21{,}510$ blocks, the full-grid oracle gains only 0.014 accepted tokens per block over the candidate-set oracle, corresponding to at most 0.3 tokens/s at our operating point, and selects depth 5 or 7 on just 4.5% of blocks. The candidate grid therefore captures essentially all of the available allocation benefit.

We store pooled states in float32 because Qwen3 activations can reach approximately 6×10^4, exceeding the float16 range. A full harvest takes under one GPU-hour, while predictor training takes minutes on one GPU. This makes relabeling inexpensive relative to the drafter training run and practical whenever the predictor's calibration drifts. In our experiments, we observe two such triggers: checkpoint drift and sampling-temperature mismatch (§5.5). Prompt-distribution shift alone does not require relabeling in the five reported benchmarks: a single predictor harvested on GSM8K and held-out training-corpus prompts serves all five benchmarks.

For temperature-matched harvesting, we replace the greedy accepted length with its expected value under rejection sampling,

$$\mathbb{E}[a] = \sum_{k} \prod_{j \le k} \min\left(1, \frac{p_j}{q_j}\right),$$

using the same target forward pass.

E PER-POSITION FREEZING

The block is not the only possible unit of adaptive computation. We therefore ask whether the policy can save additional computation by stopping individual positions once their predictions stabilize. We freeze a position when its top token remains unchanged across recurrent iterations; frozen positions remain in the KV cache, preserving the target distribution.

At block size 16, this finer-grained policy changes acceptance by less than 0.02 tokens and throughput by less than 1.5% across both proxy and survival-head variants. The block-level policy already removes most of the computation before individual positions become worth freezing.

We next measure the remaining headroom with a per-position oracle. The oracle stops each position at the cheapest depth that produces its accepted token, gaining at most 0.54 accepted tokens per block beyond the per-block oracle, with a mean in-prefix depth of 1.3. This is an optimistic upper bound because it assumes that independently chosen position depths compose without interaction. Together with the iteration oracle's $+0.43$ tokens over the best fixed depth, the result suggests that most of the available adaptive gain comes from choosing depth at the block level. We therefore use block-level allocation throughout the paper.

F BATCHED SERVING UNDER DEPTH STRAGGLERS

Adaptive depth introduces a serving effect that does not appear at batch size 1: blocks in the same batch may require different numbers of recurrent iterations. We quantify this effect by replaying the observed per-block $(r, \text{accepted})$ traces with measured serving costs. One recurrent iteration costs 2.20–2.26 ms and target verification costs 42.7–43.9 ms for batch sizes from 1 to 16. Each replayed batch drafts to the deepest selected depth before performing one target verification.

The resulting straggler penalty grows with batch size. At B=16, synchronized padding reduces plan-once throughput by 6.7% relative to the best fixed depth on GSM8K and by 13.5% on MT-Bench. Plan-once makes its depth decision before drafting begins, however, allowing us to sort blocks by predicted depth and group similar workloads together. Sorting reduces the gap to 0.1% on GSM8K and 1.6% on MT-Bench.

This advantage comes from when the decision is made. A closed-loop policy cannot group requests by final depth because that depth is unknown until drafting has progressed. The same effect appears at 8B: synchronized padding costs 12–17% at B=16, while depth sorting reduces the GSM8K gap to 1.8%, close to the 1.9% gap at batch size 1. Plan-once therefore exposes useful workload structure before computation begins, turning its early depth decision into a serving advantage in addition to its reduction in average drafting computation.

The early decision also simplifies engine integration. Each block has a fixed iteration count once planning completes, allowing one precompiled graph per candidate depth. The decision synchronization in Table 2 is the only host-side branch. By contrast, exit rules that inspect a signal after each iteration require device-to-host synchronization inside the drafting loop and prevent whole-block graph capture. Fully continuous batching with mixed prefill, as well as evaluation with standardized engine-aware suites (Abramovich et al., 2026), remains outside our evaluation.

Table 4: **Depth-sorted batching removes most of the straggler penalty.** Aggregate tokens per second from replayed per-block traces using measured serving costs on Qwen3-4B. Synchronized batches draft to the deepest selected depth, and sorted batches group blocks by predicted depth. Fixed-depth policies are unaffected by construction.

	Policy	B=1	B=2	B=4	B=8	B=16
GSM8K	Best fixed r	107.1	212.1	425.8	872.7	1699.3
	Plan-once (padded)	106.6	206.9	409.9	824.7	1585.4
	Plan-once (sorted)	106.6	211.5	425.9	870.1	1700.1
MT-Bench	Best fixed r	52.3	104.0	209.4	426.8	833.1
	Plan-once (padded)	51.3	98.6	191.7	381.6	720.9
	Plan-once (sorted)	51.3	101.8	204.6	420.6	819.9

G DEPTH ALLOCATION AND A POLICY TRACE

Figure 5 shows that the plan-once policy does not collapse to a single global depth. It selects deeper depths more often on mathematics, shifts toward shallower depths on code, and spreads its allocation more broadly on open-ended dialogue. The same predictor therefore adapts its computation to the local block distribution without changing its policy.

Figure 6 shows the same behavior within a single GSM8K generation. The selected depth changes from block to block, with intermediate depths used for arithmetic-heavy blocks and shallower depths for simpler connective text. The trace makes explicit what the aggregate distributions capture: Re-Draft allocates drafting computation locally, one speculative block at a time.

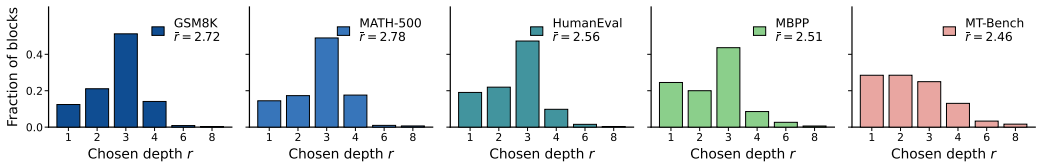

Figure 5: **Depth allocation across tasks.** Distribution of selected depth for the plan-once policy on Qwen3-4B at temperature 0, with the per-task mean $\bar{r}$ shown in each legend. The predictor allocates more computation to mathematics and a broader range of depths to open-ended dialogue.

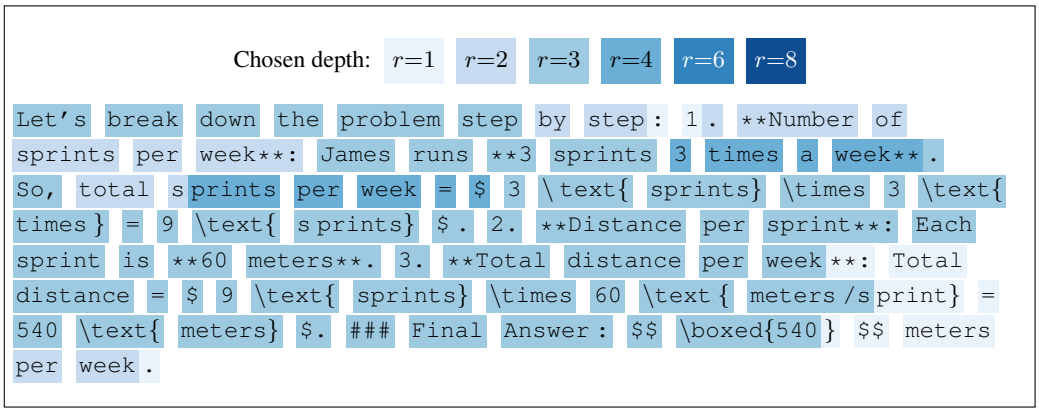

Figure 6: **A single policy trace.** Accepted text from one GSM8K generation, shaded by the depth selected for each block. Depth changes within the solution, with more computation allocated to blocks that benefit from additional refinement.

H Difference-calibration diagnostics

The regret bound in Eq. 6 depends on differences between candidate depths instead of their absolute predicted acceptance. We therefore evaluate the depth ordering induced by

$$u(i) = a(i) + 1 - \lambda c(i)$$

on held-out blocks with verifier labels for all candidate depths.

On Qwen3-4B, the predictor achieves Kendall $\tau = 0.74$ over 7,025 blocks and selects the true best depth on 69% of them. When it selects another depth, the mean utility loss is 0.89 tokens. Its mean absolute acceptance error is 1.14 tokens on this evaluation set, compared with a mean pairwise difference error of 0.52 tokens. The 1.44-token value reported in §3.2 is measured on the broader predictor validation split. At 8B, Kendall τ falls to 0.60, top-one agreement to 53%, and difference error rises to 0.87 tokens. The weaker depth ordering coincides with the larger throughput gap at 8B.

Signal comparison. We next ask whether simpler internal signals can recover the same depth ordering. On 107,104 held-out Qwen3-4B blocks with labels for the full candidate set, we tune each signal's threshold directly on the evaluation set to give it the most favorable comparison. Per-position confidence achieves a mean utility regret of 0.57 tokens at its best threshold. Pooled-state convergence reaches 0.47 tokens, and its optimal threshold selects effectively one fixed depth across all blocks, reproducing the best-fixed policy. The acceptance-curve predictor reaches 0.32 tokens without threshold tuning. Even with evaluation-set threshold selection, the internal signals do not recover the depth ordering available from verifier supervision.

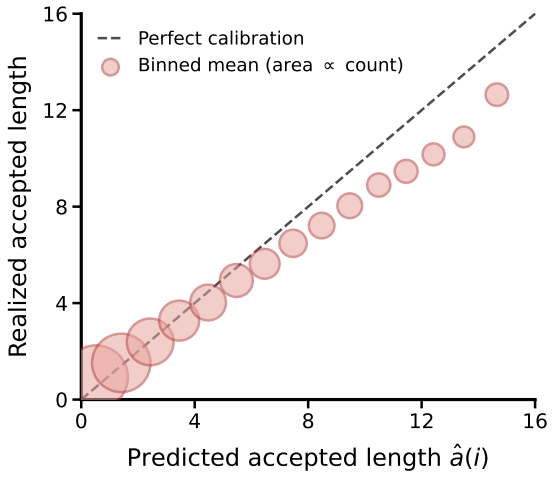

Figure 7: **Predicted versus realized acceptance.** Absolute prediction error can contain a block-dependent bias, while the pairwise differences that determine depth ordering remain substantially better calibrated.

I Replication at 8B, across model families, and at 4× data

Qwen3-8B. We first ask whether recurrent depth remains useful as the target model grows. We repeat the full pipeline on Qwen3-8B, regenerating the training data with the larger target and training the same 2-layer recurrent core. On GSM8K, acceptance increases from 3.87 at r=1 to 5.02 at r=4, while throughput peaks at r=2 at 3.61×. The published KL exit again reaches the maximum depth and loses 27% against the tuned fixed frontier. Across three seeds, the predictor remains within 4–5% of the best fixed depth on all five benchmarks. Doubling the label budget does not materially reduce this gap. The recurrent depth axis therefore persists at 8B, while the larger predictor gap points to weaker depth ordering.

Llama-3.1-8B-Instruct. We next test whether the same behavior transfers across model families. Using the released block size of 10, the Llama-3.1-8B-Instruct drafter shows the same useful depth axis: GSM8K acceptance rises from 3.35 at r=1 to 4.07 at r=4, while throughput peaks at r=2 at 2.79×. The predictor remains within 3.4–5.4% of the tuned fixed frontier across five benchmarks at the definitive protocol, with mean depths near 2.3 compared with fixed depths of 2–3. Across six prompt-subset seeds per benchmark, the per-benchmark mean gaps range from 3.0% to 5.5%.

Label coverage explains part of this variation. Training on GSM8K labels alone leaves a 7.6% gap on MATH-500, while adding labels from 300 MATH-500 prompts reduces the gap to 5.4% without changing GSM8K. The convergence exit again reaches maximum depth at its published threshold and loses 34% against the tuned frontier.

Data scale and architecture. Finally, we ask whether the recurrent advantage persists when the drafter is trained on the full released data scale. Retraining the 4B recurrent drafter on the full 400K-sequence corpus raises GSM8K acceptance at depth 3 from 5.27 to 6.02 and speedup from 3.88× to 4.40×, matching the released 5-layer drafter's 4.39× under our protocol. The recurrent model uses 40% as many distinct transformer-layer parameters and half as many sample-epochs.

The adaptive policy also persists at this scale: the default convergence exit reaches maximum depth, while a newly trained predictor remains within 4% of the tuned frontier. Under the same data and training recipe, the 5-distinct-layer drafter trails the recurrent core by 1.5–6.7% in mean throughput across the five benchmarks under seven-to-ten-seed replication, with lower acceptance on four benchmarks. The largest gap occurs on MATH-500, while the remaining benchmarks have overlapping seed-level deviations. Finally, retraining on the released corpus source yields 3.69× on GSM8K, compared with 4.40× from the open-corpus mixture. These comparisons separate the effect of data scale from that of corpus composition.

J FULL PER-BENCHMARK TABLES

We report the complete policy grids at temperatures 0 and 1 for both Qwen targets, followed by the multi-seed replication results. These tables provide the per-benchmark values underlying the aggregate comparisons in the main text. Figure 8 summarizes the temperature-0 comparison, while Table 5 gives the convergence-threshold sweep behind §5.3.

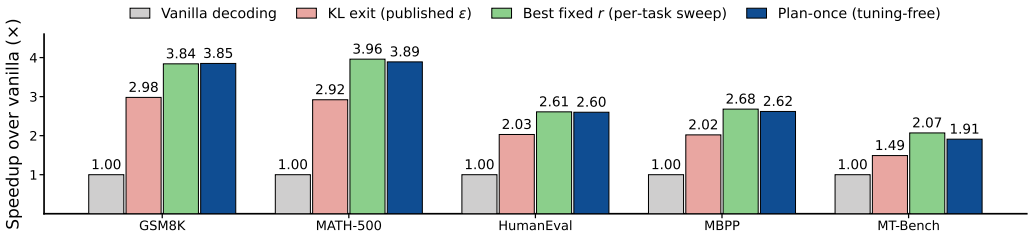

Figure 8: **Policy throughput relative to the fixed-depth frontier.** Speedup over vanilla decoding for Qwen3-4B at the definitive protocol. The fixed-depth reference is selected independently for each benchmark, while the plan-once policy uses a single learned depth predictor.

Table 5: **KL-exit throughput across convergence thresholds.** Tokens per second for the KL exit over the tested threshold grid at temperature 0. The published threshold is $\epsilon = 5\times10^{-4}$. Bold entries denote the highest throughput within each row.

		$\epsilon = 5\times10^{-4}$ (published)	$\epsilon = 0.05$	$\epsilon = 0.2$	$\epsilon = 0.5$	$\epsilon = 1.0$
4B	GSM8K	88.3	108.3	**115.1**	111.3	115.0
	MATH-500	87.6	111.7	114.7	118.8	**119.6**
	HumanEval	62.0	74.0	76.0	**78.7**	77.7
	MBPP	60.5	76.1	79.9	**81.4**	81.0
	MT-Bench	46.8	56.5	58.2	**58.6**	58.0
8B	GSM8K	79.0	95.3	97.7	99.5	**102.8**
	MATH-500	80.3	98.4	101.2	102.9	**106.3**
	HumanEval	—	69.8	72.2	**74.2**	68.8
	MBPP	52.2	66.0	67.2	69.3	**69.5**
	MT-Bench	39.6	51.9	53.0	**53.6**	53.5

Table 6 gives the complete temperature-0 grid on Qwen3-4B. The published convergence exit reaches $\bar{r}$=8.0 in each available column, while plan-once operates near $\bar{r}$=2.6 and remains within 7.5% of the per-task fixed-depth frontier.

Table 6: **Full policy grid, Qwen3-4B, temperature** 0**.** Tokens per second, with acceptance length and, for adaptive policies, realized mean depth in parentheses. Best throughput per column in bold.

Policy	GSM8K	MATH-500	HumanEval	MBPP	MT-Bench
Fixed r=2	116.4 (5.06)	117.3 (5.17)	79.7 (3.54)	**80.5** (3.56)	**60.5** (2.52)
Fixed r=3	115.8 (5.38)	**120.6** (5.50)	**80.2** (3.72)	79.6 (3.69)	58.7 (2.57)
Fixed r=4	111.5 (5.44)	116.4 (5.55)	78.5 (3.78)	76.9 (3.71)	56.3 (2.57)
KL exit (published ϵ)	90.4 (5.35, $\bar{r}$=8.0)	89.1 (5.51, $\bar{r}$=8.0)	62.1 (3.75, $\bar{r}$=8.0)	60.8 (3.66, $\bar{r}$=8.0)	43.7 (2.53, $\bar{r}$=8.0)
Plan-once (ours)	**116.7** (5.31, $\bar{r}$=2.7)	118.7 (5.42, $\bar{r}$=2.8)	79.8 (3.67, $\bar{r}$=2.6)	78.8 (3.61, $\bar{r}$=2.5)	55.9 (2.53, $\bar{r}$=2.5)
Value head (closed loop)	114.5 (5.26, $\bar{r}$=2.5)	114.0 (5.39, $\bar{r}$=2.8)	77.1 (3.65, $\bar{r}$=2.5)	77.2 (3.53, $\bar{r}$=2.1)	54.5 (2.51, $\bar{r}$=2.3)

Table 7 gives the corresponding temperature-1 grid. The same harvest–serve mismatch appears across benchmarks, while temperature-matched harvesting recovers most of the gap discussed in §5.5.

Table 7: **Full policy grid, Qwen3-4B, temperature** 1**.** Tokens per second, with acceptance length and, for adaptive policies, realized mean depth in parentheses. Best throughput per column in bold.

Policy	GSM8K	MATH-500	HumanEval	MBPP
Fixed r=2	90.4 (4.15)	**98.3** (4.17)	**65.5** (3.01)	**66.8** (2.98)
Fixed r=3	**92.3** (4.33)	97.6 (4.28)	65.1 (3.14)	66.1 (3.05)
KL exit (published ϵ)	70.0 (4.21, $\bar{r}$=8.0)	73.1 (4.16, $\bar{r}$=8.0)	49.7 (3.11, $\bar{r}$=8.0)	50.0 (2.99, $\bar{r}$=8.0)
Plan-once (ours)	89.6 (4.20, $\bar{r}$=2.7)	95.4 (4.25, $\bar{r}$=2.7)	64.5 (3.10, $\bar{r}$=2.5)	65.1 (2.98, $\bar{r}$=2.5)

Table 8 gives the complete temperature-0 grid on Qwen3-8B. The predictor remains substantially shallower than the published convergence exit while staying within 6.8% of the per-task fixed-depth frontier.

Table 8: **Full policy grid, Qwen3-8B, temperature** 0**.** Tokens per second, with acceptance length and, for adaptive policies, realized mean depth in parentheses. Best throughput per column in bold.

Policy	GSM8K	MATH-500	HumanEval	MBPP	MT-Bench
Fixed r=2	101.9 (4.76)	106.4 (4.75)	72.5 (3.31)	**74.3** (3.21)	**53.2** (2.37)
Fixed r=3	**103.9** (4.98)	**107.8** (4.97)	**72.6** (3.43)	73.0 (3.29)	51.3 (2.42)
Fixed r=4	96.5 (5.03)	104.6 (5.02)	71.1 (3.46)	69.3 (3.31)	48.7 (2.41)
KL exit (published ϵ)	75.6 (4.99, $\bar{r}$=8.0)	80.3 (4.96, $\bar{r}$=8.0)	53.9 (3.43, $\bar{r}$=8.0)	53.5 (3.26, $\bar{r}$=8.0)	39.3 (2.38, $\bar{r}$=8.0)
Plan-once (ours)	96.8 (4.76, $\bar{r}$=2.4)	103.0 (4.77, $\bar{r}$=2.5)	69.2 (3.30, $\bar{r}$=2.5)	69.5 (3.18, $\bar{r}$=2.5)	51.2 (2.35, $\bar{r}$=2.5)
Value head (closed loop)	94.2 (4.85, $\bar{r}$=2.7)	99.4 (4.89, $\bar{r}$=3.1)	67.0 (3.31, $\bar{r}$=2.6)	67.5 (3.17, $\bar{r}$=2.6)	48.9 (2.37, $\bar{r}$=3.0)

Table 9 gives the corresponding temperature-1 grid. The harvest–serve mismatch persists at 8B, consistent with the Qwen3-4B results.

Table 9: **Full policy grid, Qwen3-8B, temperature** 1**.** Tokens per second, with acceptance length and, for adaptive policies, realized mean depth in parentheses. Best throughput per column in bold.

Policy	GSM8K	MATH-500	HumanEval	MBPP
Fixed r=2	**86.0** (3.90)	**83.4** (3.75)	**56.5** (2.77)	**58.7** (2.74)
Fixed r=3	84.7 (4.00)	83.1 (3.90)	55.4 (2.83)	57.3 (2.76)
KL exit (published ϵ)	63.5 (3.91, $\bar{r}$=8.0)	61.4 (3.82, $\bar{r}$=8.0)	—	—
Plan-once (ours)	82.1 (3.88, $\bar{r}$=2.4)	80.6 (3.80, $\bar{r}$=2.5)	54.3 (2.76, $\bar{r}$=2.6)	54.9 (2.65, $\bar{r}$=2.5)

Table 10 reports the seed-level variability behind these comparisons. At temperature 0, plan-once and fixed-depth throughput are close across the replicated rows, while the larger 8B gaps track the weaker depth ordering measured in §H. The temperature-1 rows quantify the harvest–serve mismatch, and the matched-harvest rows show how much of it is recovered by temperature-aware labels.

Table 10: **Multi-seed replication of the policy comparison.** Tokens per second, reported as mean $\pm$ standard deviation over disjoint prompt subsets. The Seeds column gives the number of subsets used for each row.

Model	Benchmark	Seeds	Plan-once	Fixed r=2	Fixed r=3
Qwen3-4B	GSM8K	86	116.4 ± 3.0	116.5 ± 3.0	118.0 ± 3.0
	GSM8K (T=1)	9	93.3 ± 2.8	94.8 ± 2.8	93.8 ± 2.3
	GSM8K (T=1, matched harvest)	15	92.9 ± 1.8	94.6 ± 2.1	93.9 ± 2.0
	HumanEval	63	78.3 ± 1.8	78.4 ± 1.8	78.7 ± 1.7
	HumanEval (T=1)	10	65.8 ± 1.3	66.4 ± 1.1	66.0 ± 1.2
	HumanEval (T=1, matched harvest)	8	65.0 ± 1.5	66.4 ± 1.7	66.2 ± 2.1
	MATH-500	73	118.1 ± 2.7	119.0 ± 2.8	120.3 ± 2.6
	MATH-500 (T=1)	12	92.5 ± 1.7	94.7 ± 2.1	94.5 ± 2.3
	MATH-500 (T=1, matched harvest)	14	92.1 ± 2.6	93.8 ± 2.8	93.9 ± 3.6
	MBPP	56	79.1 ± 1.7	81.1 ± 2.0	80.2 ± 1.9
	MBPP (T=1)	10	66.0 ± 1.4	68.0 ± 1.6	66.4 ± 1.3
	MBPP (T=1, matched harvest)	9	66.6 ± 1.7	67.7 ± 1.2	66.1 ± 1.6
	MT-Bench	56	57.8 ± 1.1	59.3 ± 1.1	57.8 ± 1.1
	MT-Bench (T=1)	8	49.4 ± 1.2	51.1 ± 1.1	49.9 ± 0.5
	MT-Bench (T=1, matched harvest)	9	49.3 ± 1.1	50.6 ± 1.0	49.4 ± 1.0
Qwen3-8B	GSM8K	81	101.6 ± 2.7	105.2 ± 2.8	105.8 ± 2.6
	GSM8K (T=1)	8	82.3 ± 3.3	86.6 ± 2.6	85.3 ± 2.3
	HumanEval	56	68.8 ± 1.8	71.4 ± 1.7	70.7 ± 1.7
	HumanEval (T=1)	5	56.6 ± 0.8	59.2 ± 1.0	57.8 ± 1.0
	MATH-500	80	103.9 ± 2.4	107.3 ± 2.5	107.4 ± 2.5
	MATH-500 (T=1)	10	82.8 ± 1.7	85.4 ± 1.6	83.9 ± 2.5
	MBPP	40	69.0 ± 1.6	72.3 ± 1.9	70.9 ± 1.7
	MBPP (T=1)	7	56.6 ± 1.1	59.2 ± 1.4	57.7 ± 1.3
	MT-Bench	38	51.6 ± 1.5	53.7 ± 1.5	52.5 ± 1.5
	MT-Bench (T=1)	8	44.4 ± 0.9	46.8 ± 0.9	45.0 ± 1.2