GROWMTP: CAN RL GROW ITS OWN DRAFT HEAD?

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) post-training drives the frontier capabilities of large language models, with its wall-clock dominated by autoregressive rollout generation. Speculative decoding is an established remedy for this bottleneck, but existing draft heads must be pretrained or warmed up before RL, introducing substantial training cost outside the RL run to be accelerated. We observe that RL training itself provides both conditions required for online draft-head training: its rollout distribution is far narrower than that of pretraining, and its verification step continuously produces supervision signals aligned with this distribution. Building on these observations, we propose *GrowMTP*, which uses this supervision to train a draft head from scratch entirely within the RL loop, with all head updates detached from the policy backbone. On Qwen3-4B (no draft head), MiMo-7B-SFT (weak head), and Qwen3.5-4B-Base (strong head), *GrowMTP* achieves rollout speedups of $2.13\times$, $1.93\times$, and $1.36\times$, and end-to-end speedups of $1.60\times$, $1.41\times$, and $1.20\times$, respectively. *GrowMTP* therefore serves existing RL training frameworks as a modular component, particularly offering a from-scratch acceleration path for models without pretrained draft heads. Website: **https://growmtp-sub.github.io/**.

1 INTRODUCTION

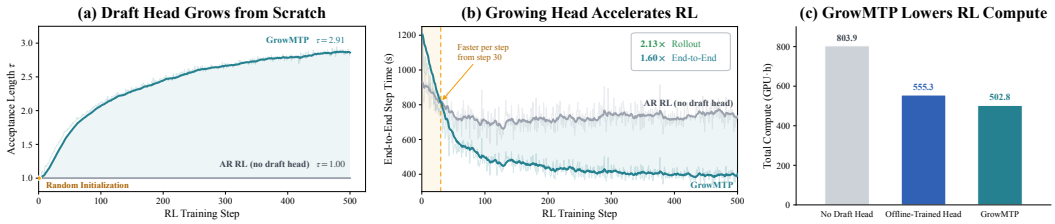


Figure 1: **An RL run can grow its own draft head from scratch and reduce its training compute.** **(a)** The acceptance length grows from 1.00 to 2.91. **(b)** This yields $2.13\times$ rollout and $1.60\times$ end-to-end speedups, including head updates. **(c)** *GrowMTP* uses less total compute than matched offline-trained and no-head baselines. Qwen3-4B, 500 RL steps on DAPO-Math-17K.

Reinforcement learning (RL) post-training now drives the frontier capabilities of large language models (Jaech et al., 2024; Guo et al., 2025). The RL stage accounts for a substantial share of the training budget (Khatri et al., 2026): OpenAI o3 was trained with $10\times$ the RL compute of o1 (Singh et al., 2025). Within this budget, the dominant cost is rollout generation: the policy decodes tens of thousands of tokens per response token by token, typically 70–90% of each training step (Chen et al., 2026). The bottleneck is thus not the RL algorithm itself but autoregressive decoding.

This bottleneck is not unique to RL: it has long been central to model serving, where speculative decoding is the established solution (Leviathan et al., 2023; Chen et al., 2023). A cheap draft proposes several tokens, the target model verifies them in a single parallel forward pass, and rejection sampling preserves the output distribution exactly. Modern drafts are lightweight auxiliary (Ankner et al., 2024; Li et al., 2024a;b) or native multi-token-prediction (MTP) heads (Cao et al., 2026; Xiaomi et al., 2025), which reuse the backbone's hidden states at negligible cost and are now standard components of frontier models. Porting this machinery to RL rollouts is therefore a natural step.

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.

However, existing approaches train draft heads before RL (Cai et al., 2024). An EAGLE-style head for a 70B target model takes roughly 5,000 offline GPU hours (Li et al., 2026b; Hu et al., 2026), while native MTP heads are trained throughout pretraining, 14.8T tokens for DeepSeek-V3 (Liu et al., 2024). Improvement is no different: drafting depth depends on training, and raising acceptance at larger K typically requires another offline stage. Drafting capability thus enters RL as a prerequisite, and a model without a head has no route to acceleration. In effect, the established remedy for the dominant cost of RL is itself a further training cost, incurred before RL benefits at all.

We argue that this tension stems from a twofold mismatch between how draft heads are trained and what an RL process both requires and provides. Draft heads brought into RL are pretrained for broad deployment across diverse tasks. What RL requires is much narrower: rollouts concentrate on a particular task distribution, and the head only needs to fit it. Meanwhile, existing pipelines source draft-head supervision entirely outside RL. Yet during speculative rollouts, the target model verifies every draft, producing predictive distributions and acceptance decisions that can supervise the draft head but are otherwise discarded. An RL process can thus grow its own draft head from the rollouts it already requires and accelerate subsequent rollouts, without an upfront training bill.

Although RL provides the distribution and the supervision, training a multi-step draft head online presents three challenges. First, draft-head training must remain consistent with rollout-time drafting. Each proposal is conditioned on the preceding drafts, so its verification signal must be applied under the same draft-conditioned state. Second, learning across drafting depths is inherently dependent. The k-th draft token counts only if the preceding $k-1$ are accepted, so early rejections limit gains from deeper predictions. Training should therefore reflect this dependence when assigning supervision across drafting depths. Finally, not every verification signal corresponds to an executed prefix. Once the target rejects a draft token, deeper drafts remain conditioned on it and no longer follow the executed sequence, so supervising all positions equally also trains the head on discarded suffixes.

To meet these demands, we introduce *GrowMTP*, which trains the draft head online from rollout verification signals. To preserve consistency with rollout-time drafting, *GrowMTP* records the draft trajectory and reconstructs its draft-conditioned states, applying each signal under the state that produced it. To encode the dependence across drafting depths, we design a depth-coupled acceptance (DCA) loss based on an acceptance-chain surrogate. We apply a logarithmic transformation to give cycles with smaller chain values greater relative gradient weight. Because this objective can still assign nonzero gradients beyond the first rejection, *GrowMTP* gates the loss there and excludes subsequent positions. All draft-head updates are detached from the policy backbone, without directly interfering with policy learning. Since the first draft position always provides valid supervision, training can begin from random initialization without an offline stage or a depth schedule.

To evaluate from-scratch training, we train a random $K{=}5$ head on Qwen3-4B (Yang et al., 2025) for 500 math RL steps. Its acceptance length grows from the autoregressive floor to 2.91, accelerating rollouts by $2.13\times$ and end-to-end steps by $1.60\times$, with early per-step speedups. Over 500 steps, *GrowMTP* reduces total GPU-hours by 9.4% versus an offline-trained head at comparable acceptance. *GrowMTP* also accelerates models with pretrained heads: on Qwen3.5-4B-Base (Qwen, 2026), online updates accelerate rollouts by $1.36\times$ over a frozen head; on MiMo-7B-SFT (Xiaomi et al., 2025), extending the head from $K{=}1$ to $K{=}5$ raises acceptance from 1.65 to 4.0 and accelerates rollouts by $1.93\times$. Heads trained on Math and Code rollouts achieve higher in-domain acceptance. Thus, in-RL draft-head training does not replace MTP pretraining but suffices for the current RL run.

In summary, our contributions are as follows:

- **Revisiting MTP for RL acceleration.** We show that an RL run only requires a draft head adapted to its rollout distribution and continuously produces matching verification supervision, allowing such a head to be trained during RL rather than pretrained.
- **GrowMTP.** We propose *GrowMTP*, which trains an MTP head online using valid verification supervision under rollout-consistent draft states, extends learning along the accepted draft chain, and retains the original RL objective.
- **From-scratch acceleration.** On the headless Qwen3-4B, *GrowMTP* grows a random $K{=}5$ head for a $2.13\times$ rollout and $1.60\times$ end-to-end speedup, head-training cost included.
- **Modular RL acceleration.** *GrowMTP* extends speculative acceleration to models without pretrained MTP heads, which grow one during RL instead.

2 PRELIMINARIES

2.1 REINFORCEMENT LEARNING FOR LARGE LANGUAGE MODELS

We consider Group Relative Policy Optimization (GRPO)-style RL post-training (Zheng et al., 2025), in which the policy π_θ, the language model itself, samples a group of G responses for each prompt x and receives a scalar reward per response. Each response y is generated autoregressively: the policy emits one token per forward pass, $y_t \sim \pi_\theta(\cdot \mid x, y_{<t})$. A training step thus comprises two phases: *rollout*, which generates the responses, and *update*, which converts the rewards into advantages and adjusts the policy. The cost of a step is dominated by rollout, because of this token-by-token decoding, and this work accordingly targets the rollout bottleneck while retaining the original RL objective.

2.2 SPECULATIVE DECODING AND MULTI-TOKEN PREDICTION

Speculative decoding generates tokens in *draft-then-verify* cycles. We use a recurrent MTP head (Xu et al., 2026) h_ϕ as the draft head, reusing backbone hidden states to propose K tokens per cycle; K is the *drafting depth*. Drafting is itself autoregressive: the k-th draft token d_k is sampled from a head distribution $q_k(\cdot \mid x, y_{\le t}, d_{<k})$, conditioned on the committed context and on the draft tokens before it. The target model then verifies all K candidates in a single forward pass, producing its own distribution p_k at every draft position, and accepts d_k by *rejection sampling*, with probability

$$\min\left(1,\ \frac{p_k(d_k)}{q_k(d_k)}\right). \tag{1}$$

At the first rejection, a replacement token is resampled from a corrected distribution, and if all K candidates are accepted, a bonus token is drawn from the target's next-position distribution p_{K+1}, so the committed sequence remains distributed exactly as if the target had decoded autoregressively (Xia et al., 2024; Miao et al., 2024). The tokens committed in one cycle, the accepted prefix plus this final token, always number at least one: even a draft rejected at the first position commits the resampled token, regardless of the quality of the draft. We call the average number of tokens committed per cycle the *acceptance length* $\tau \in [1, K+1]$: rollout throughput grows in proportion to τ, up to the drafting overhead, with $\tau = 1$ recovering plain autoregressive decoding.

2.3 TRAINING DRAFT HEADS FROM RL ROLLOUTS

We observe that a draft head's ability to accelerate a given RL process depends on its acceptance on that process's rollout distribution, rather than its general drafting capability. At the same time, every speculative rollout produces target distributions and acceptance decisions on this distribution, verification supervision for exactly the proposals generated. The first draft position always provides supervision, so head training can begin from random initialization using the rollout records that RL already produces for policy optimization, without a separate dataset or an offline head-training stage.

However, using this supervision effectively for multi-step online training presents three challenges. First, each verification signal corresponds to a proposal generated under a particular draft-conditioned state, so training must apply the signal under the same state. Second, position k increases the acceptance length only if the preceding $k-1$ drafts are accepted, so training should reflect this dependence across depths. Finally, once the target rejects a draft token, subsequent drafts remain conditioned on it and no longer follow the executed sequence; supervising all positions equally also trains the head on discarded suffixes that were drafted but never committed to the executed response.

3 *GrowMTP*: GROWING DRAFT HEADS WITHIN RL

GrowMTP uses verification supervision from each rollout to train the draft head for subsequent rollouts, forming a training-and-acceleration loop within RL. Figure 2 illustrates both phases: rollout records the realized draft trajectory and verification signals, while update trains the head from detached backbone states alongside the policy update. Using these records, *GrowMTP* reconstructs the draft-conditioned state under which each signal was produced, optimizes an acceptance-chain surrogate with the DCA loss, and excludes positions beyond the first rejection. The head loss updates only draft-head parameters, while the backbone optimizes the original RL objective. **Appendix F** presents the full training algorithm, and **Appendix E** describes its implementation.

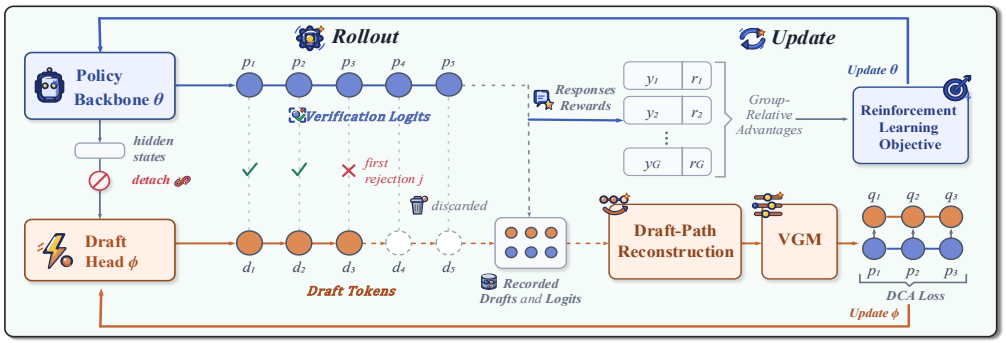


Figure 2: **The *GrowMTP* pipeline.** Rollout performs speculative decoding as usual and records the drafts, verification logits, and acceptance decisions of each cycle. Update trains the draft head on these records through detached hidden states, alongside the policy update.

3.1 Training Pipeline and Draft-Path Reconstruction

Training Pipeline. *GrowMTP* preserves the two phases of the existing RL training step: rollout performs speculative decoding as usual, and update continues the original policy optimization. During each draft-then-verify cycle, rollout records the backbone hidden states at the start of the cycle, the drafts generated by the head, the target verification logits, and the acceptance decision at each position. These quantities are already produced by drafting and verification, so collecting the training supervision requires no additional target-model forward pass. During update, the original RL objective continues to update the policy, while these rollout records separately train the draft head. The recorded backbone hidden states enter the head detached, so the head loss updates only the head parameters ϕ and sends no gradient to the backbone parameters θ. Figure 2 illustrates this pipeline.

Draft-Path Reconstruction. A natural use of these records treats the final response as an ordinary token sequence and trains the head on it by a right shift, which is teacher forcing (Gloeckle et al., 2024), constructed in full in **Appendix G**. Figure 3 contrasts the two training paths. Consider draft-then-verify cycle r: its committed context $\mathbf{c}_r$ is the prompt plus all tokens committed earlier, and its draft path $\mathbf{d}_r = [d_{r,1}, \ldots, d_{r,K}]$ its K draft tokens. During verification, the target model z_θ conditions its logits on $\mathbf{c}_r$ and the preceding drafts, yielding the target distribution at position k:

$$p_{r,k} = \mathrm{softmax}\big(z_\theta(\mathbf{c}_r, d_{r,1}, \ldots, d_{r,k-1})\big). \tag{2}$$

Teacher forcing conditions position k on the committed token rather than the head's own draft: the two are identical only when that draft was accepted. If position j is the first rejection, the target replaces $d_{r,j}$ with $\tilde{d}_{r,j} \neq d_{r,j}$, and the tokens that follow, written $\tilde{d}_{r,j+1}, \ldots$ and committed by subsequent cycles, continue from this replacement, so the paths diverge as

$$\mathbf{d}_r^{\mathrm{draft}} = [d_{r,<j},\, d_{r,j},\, d_{r,j+1}, \ldots], \qquad \mathbf{d}_r^{\mathrm{commit}} = [d_{r,<j},\, \tilde{d}_{r,j},\, \tilde{d}_{r,j+1}, \ldots]. \tag{3}$$

The head consumes at each position the hidden state and token embedding of the position before it, so the committed path supplies a different pair at every position past j, along with different cached keys and values: teacher forcing fits the head at a state that drafting reaches only when the whole prefix is accepted. The training state at position k agrees with the drafting state if and only if none of the preceding $k-1$ positions was rejected, and this condition tightens monotonically in k. The bias of teacher forcing therefore grows with draft depth, and drafting depth is what *GrowMTP* extends during RL. We discuss this train–inference inconsistency in detail in **Appendix G**.

The final response neither marks cycle boundaries nor retains the drafts the target replaced, so both must be recorded during rollout. *GrowMTP* therefore applies *Draft-Path Reconstruction*, which starts from the recorded cycle context and feeds the recorded drafts back in their original order:

$$s_{r,1} = \mathbf{c}_r, \qquad s_{r,k+1} = [s_{r,k}, d_{r,k}], \qquad q_{r,k} = h_\phi(\cdot \mid s_{r,k}). \tag{4}$$

This recursion gives $s_{r,k} = [\mathbf{c}_r, d_{r,1}, \ldots, d_{r,k-1}]$, pairing every $p_{r,k}$ with a head distribution under the same cycle context and realized draft prefix. We provide the implementation details of the reconstruction in **Appendix E**. The only remaining asymmetry lies between successive RL steps: the head learns from logits of the current rollout but drafts for a policy one update newer, and **Appendix H** bounds the resulting discrepancy and shows it remains small in our runs.

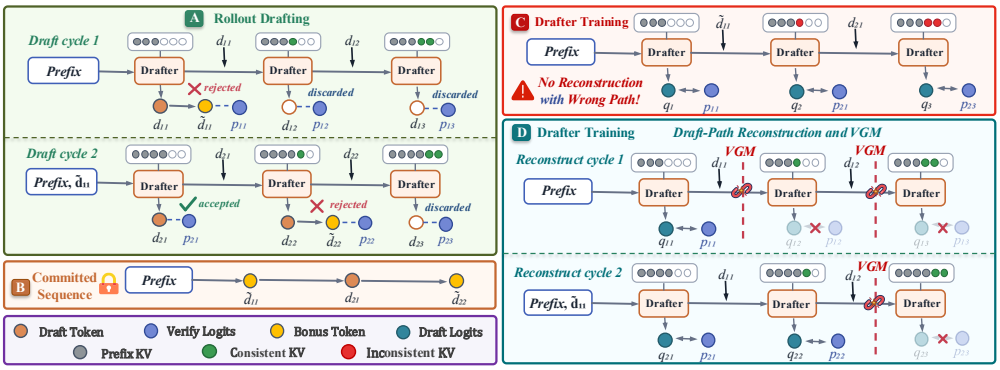


Figure 3: **Rollout cycles and the two training paths.** Teacher forcing (C) trains on the committed path. Reconstruction (D) feeds the recorded drafts back, and VGM truncates at the first rejection.

3.2 The Depth-Coupled Acceptance Loss

To target acceleration, we construct a differentiable surrogate for the expected accepted draft length from the reconstructed pairs. The building block is the acceptance probability at a single position: writing p_k and q_k for the paired target and head distributions at position k within a cycle, rejection sampling accepts a token sampled from the head with probability averaged over possible draft tokens:

$$\alpha_k \;=\; \sum_{d} \min\big(p_k(d),\, q_k(d)\big) \;=\; 1 - \mathrm{TV}(p_k,\, q_k), \tag{5}$$

where TV is the total variation distance. Acceptance is chain-dependent: the l-th token contributes only when positions 1 through l are all accepted. Chaining these local rates yields a differentiable surrogate for the accepted length, whose negation is the TV loss (Zhou et al., 2024; Li et al., 2026a)

$$\mathcal{L}_{\mathrm{TV}} \;=\; -\sum_{l=1}^{K}\prod_{i=1}^{l}\alpha_i. \tag{6}$$

To give cycles with smaller chain values greater relative gradient weight, we apply a logarithmic transformation to the acceptance-chain surrogate to obtain the DCA training objective (**Appendix I**):

$$\mathcal{L}_{\mathrm{DCA}} \;=\; -\log\big(-\mathcal{L}_{\mathrm{TV}}\big) \;=\; -\log\sum_{l=1}^{K}\exp\Big(\sum_{i=1}^{l}\log\alpha_i\Big), \tag{7}$$

minimized with respect to the head alone: the gradient flows through each q_k, and the recorded p_k stay constant. The logarithm preserves the gradient direction on each fixed cycle but changes the relative cycle weights in the batch, so the two batch objectives can have different stationary points.

3.3 Learning up to the First Rejection

After the first rejection, later draft positions remain conditioned on discarded tokens and no longer follow the executed sequence. Verification still produces target distributions at each of these draft positions, so treating every position equally also assigns full training weight to the discarded suffix.

However, differentiating DCA with respect to α_k factors the gradient weight at position k into the product of the preceding acceptance probabilities and a bounded tail term,

$$-\frac{\partial\mathcal{L}_{\mathrm{DCA}}}{\partial\alpha_k} \;=\; \Big(\prod_{i<k}\alpha_i\Big)\frac{D_k}{-\mathcal{L}_{\mathrm{TV}}}, \qquad D_k \;=\; \sum_{l=k}^{K}\prod_{i=k+1}^{l}\alpha_i \;\in\; [1,\, K-k+1], \tag{8}$$

where beyond the leading product, its denominator shared across positions, the weights differ at most by a factor of K, with the full derivation given in **Appendix I**. The leading product weights supervision at position k by the acceptance overlaps of preceding drafts.

This soft weighting does not enforce the realized rejection boundary: positions after the first rejection can still receive nonzero gradients. Since verification already marks the first-rejection position j of

each cycle, with $j = K+1$ when every draft is accepted, we introduce *verify-gated masking* (VGM), which truncates the chain sum at and including the first rejected position of the recorded cycle,

$$\mathcal{L}_{\text{DCA}}^{\text{VGM}} \;=\; -\log \sum_{l=1}^{\min(j,K)} \prod_{i=1}^{l} \alpha_i, \tag{9}$$

removing supervision beyond the first rejection. The chain sum includes the first rejected position, if any, and always includes the first draft position. Supervision is therefore available from the first cycle, allowing training to begin even with a randomly initialized head. The resulting supervision is also self-paced: its depth follows the realized accepted prefix, while gradient weights reflect the acceptance overlaps of preceding drafts. This requires no shallow-depth warm-up or schedule for K.

Each update minimizes the mean of Eq. 9 over cycles from the rollout policy and draft head, holding recorded tokens, target distributions, and j fixed. Gradients flow through reconstructed head distributions, excluding sampling and rejection decisions. This fixed-record objective is an acceptance surrogate; unbiasedness for deployment-time expected accepted length is not assumed.

4 Experiments

4.1 Experimental Setup

Models. Qwen3-4B (Yang et al., 2025) has no pretrained draft head: we attach a single randomly initialized EAGLE-style MTP layer and apply it recurrently at $K{=}5$. Qwen3.5-4B-Base (Qwen, 2026) has a native MTP head ($K{=}3$), isolating the online training strategy without cold-start interference. MiMo-7B-SFT (Xiaomi et al., 2025) has an MTP head ($K{=}1$), isolating how online training extends drafting depth from shallow capability. The three heads take the same form, detailed in **Appendix B**. **Appendix D.1** discusses which models *GrowMTP* can accelerate and the signal is model-agnostic.

Datasets. We evaluate *GrowMTP* in two representative domains: mathematical reasoning and code reasoning. For RL training, we use DAPO-Math-17K (Yu et al., 2026) and TACO-Verified (Li et al., 2023), two established datasets. Inference evaluation uses AMC23 (Zhang & Math-AI, 2023), AIME24, and AIME25 (Zhang & Math-AI, 2024; 2025) for mathematical reasoning, and LiveCodeBench v6 (Jain et al., 2025) with its AtCoder and LeetCode slices for code reasoning, detailed in **Appendix B**. *GrowMTP* thus requires no data constructed for draft-head training.

Metrics. For the training evaluation, we measure rollout time, end-to-end step time with head updates included, and the acceptance length τ, with speedups over the same run with the head frozen. For the Inference evaluation, we report policy quality, Mean@16 for mathematical reasoning (Team et al., 2025) and Pass@4 for code reasoning (Li et al., 2026c), to assess the effect of draft-head training on policy quality, and the held-out τ, with full protocols in **Appendix B.4**.

Implementation Details. We implement *GrowMTP* on verl (Sheng et al., 2025), with rollouts served by SGLang (Zheng et al., 2024) and the draft head in its speculative decoding path. Each RL step samples 64 prompts with 8 responses each, capped at 8192 tokens (16384 on Qwen3.5-4B-Base), and Qwen3-4B runs in thinking mode. Rollouts and evaluation use rejection sampling at temperature 1. Scaling K introduces no new parameters, the same head applied recurrently, and the head is updated once per RL step from that step's recorded signals. Full configurations are given in **Appendix B**.

4.2 RQ1: Can an RL Run Grow Its Own Draft Head from Scratch?

***GrowMTP* vs. Baseline.** We train Qwen3-4B for 500 RL steps in each domain with a randomly initialized $K{=}5$ head. As shown in Table 1 and **Appendix C.1**, the acceptance length τ grows from the autoregressive floor of 1.00 to 2.91 on mathematical reasoning and 2.64 on code reasoning, and the growth begins immediately, as the first draft position supplies valid supervision even under random initialization. This growth converts into rollout speedups of $2.13\times$ and $1.93\times$, and with head-update cost included, end-to-end step speedups of $1.60\times$ and $1.56\times$. Held-out benchmark scores are comparable. The acceleration is not free early on, as the random head accepts little while already paying update cost, but per-step time falls below the baseline within the first 30 steps in both domains and subsequent steps remain faster. **Appendix D.2** projects this speedup at longer rollouts and more steps. *GrowMTP* can thus train an MTP head from scratch during RL and use it to accelerate the run itself. **Appendix D.3** shows that the grown head specializes to that run.

Table 1: **Training evaluation on Qwen3-4B (500 RL steps, K=5).** A draft head grown from scratch delivers end-to-end step speedups of 1.60× and 1.56× over autoregressive decoding (K=0).

		Math Reasoning RL					Code Reasoning RL				
Method	K	Rollout (s)	Speedup	Step (s)	Speedup	τ	Rollout (s)	Speedup	Step (s)	Speedup	τ
AR RL	0	523.86	1.00×	723.51	1.00×	1.00	771.22	1.00×	1028.39	1.00×	1.00
GrowMTP	**5**	**245.95**	**2.13×**	**452.53**	**1.60×**	**2.91**	**398.65**	**1.93×**	**659.39**	**1.56×**	**2.64**

Online vs. Offline. We further compare against the established offline route, generating rollouts from the same initial checkpoint on the same dataset, training an identical head with the same objective, and then running RL with that fully trained head frozen throughout. As shown in Table 2, both routes reach nearly identical serving acceptance, 2.92 offline and 2.91 online. The offline route, however, totals 555.3 against 502.8 GPU·h for *GrowMTP*, a 9.4% difference over the 500-step run. The gap arises at generation: offline training must produce its trajectories before any head exists, whereas *GrowMTP* learns from rollouts that RL must generate anyway. A separate offline stage is thus unnecessary for accelerating the run: *GrowMTP* obtains an equally capable head at lower cost from the RL process itself.

Table 2: ***GrowMTP* avoids 161.6 GPU·h offline head-training stage.** Costs in GPU·h; Qwen3-4B, 500 RL steps on DAPO-Math-17K, K=5.

Method	Offline (h)	RL (h)	Total (h)	τ
AR RL (K = 0)	0	803.9	803.9	1.00
Offline (K = 5)	161.6	393.7	555.3	2.92
GrowMTP (K = 5)	0	502.8	**502.8**	2.91

4.3 RQ2: How Should the Draft Head Be Trained Online?

Training Strategy. On Qwen3.5-4B-Base, we train for 200 RL steps in each domain under four strategies: frozen (Chandiramani et al., 2026), joint CE (Wang et al., 2026), detached CE (Zeng et al., 2025), and detached DCA, the last being *GrowMTP*. Frozen is the prevailing practice in current RL pipelines, and the two CE arms transplant the established recipe online, differing only in whether gradients reach the backbone. As shown in Table 3 and **Appendix C.2**, joint training reaches the highest τ, 3.99 and 3.42, yet its held-out quality is zero in both domains: τ can rise as the policy that produces it degrades. The detached CE arm avoids the observed collapse and accelerates the run: τ reaches 3.18 and 3.06, and rollouts speed up by 1.29× and 1.17×. Under the same detachment at K=3, DCA leads CE on every training-side column and every held-out τ, with end-to-end step speedups of 1.20× and 1.12× including head-update cost and held-out quality on par with the frozen head. The draft head should thus be trained online, but with gradients detached from the policy and an objective aligned with acceptance: *GrowMTP* further improves a pretrained head.

Table 3: **Training evaluation on Qwen3.5-4B-Base (200 RL steps, K=3).** Both detached arms accelerate. DCA leads among quality-preserving strategies. Joint CE speeds up code but scores zero.

		Math Reasoning RL					Code Reasoning RL				
Training Mode	Loss	Rollout (s)	Speedup	Step (s)	Speedup	τ	Rollout (s)	Speedup	Step (s)	Speedup	τ
Frozen	–	357.62	1.00×	554.42	1.00×	2.65	500.09	1.00×	778.54	1.00×	2.59
Joint	CE	372.87	0.96×	643.76	0.86×	3.99	258.34	1.94×	411.88	1.89×	3.42
Detached	CE	277.14	1.29×	507.91	1.09×	3.18	426.88	1.17×	719.40	1.08×	3.06
Detached	**DCA (*GrowMTP*)**	**263.86**	**1.36×**	**461.56**	**1.20×**	**3.22**	**408.89**	**1.22×**	**694.00**	**1.12×**	**3.11**

Drafting Depth. We further evaluate how far *GrowMTP* can extend the drafting depth, and which depth best serves RL acceleration. We train on MiMo-7B-SFT at K=3, 5, and 7 for 200 RL steps in each domain, against the shipped head frozen at K=1. As shown in Table 4 and **Appendix C.3**, online training converts nominal depth into realized acceptance: at K=5, τ climbs to 4.04 and 3.72, beyond the structural ceiling of two for single-step drafting, rollouts accelerate by 1.93× and 1.72×, and end-to-end steps by 1.41× and 1.35×. Greater depth, however, does not translate monotonically into end-to-end gains: at K=7, τ rises further to 4.38 and 4.00, yet step speedups fall to 1.30× on math and 1.31× on code, as both drafting and verification costs per cycle grow with depth while the marginal acceptance gain diminishes (Chen et al., 2024). Held-out scores are comparable across depths. The drafting depth should thus follow end-to-end acceleration rather than acceptance alone, with K=5 best in this setting: *GrowMTP* also proves effective for a single-step head.

Table 4: **Training evaluation across drafting depths on MiMo-7B-SFT (200 RL steps).** τ rises monotonically with depth, while end-to-end step speedups peak at $K{=}5$, the highlighted depth.

		Math Reasoning RL					Code Reasoning RL				
Method	K	Rollout (s)	Speedup	Step (s)	Speedup	τ	Rollout (s)	Speedup	Step (s)	Speedup	τ
Frozen	1	318.66	1.00×	523.02	1.00×	1.65	441.81	1.00×	707.42	1.00×	1.59
GrowMTP	3	176.26	1.81×	378.88	1.38×	3.25	264.69	1.67×	528.64	1.34×	3.11
GrowMTP	5	164.84	1.93×	**371.02**	**1.41×**	4.04	256.86	1.72×	**525.31**	**1.35×**	3.72
GrowMTP	7	171.24	1.86×	402.20	1.30×	4.38	252.03	1.75×	542.02	1.31×	4.00

4.4 RQ3: How Does Each Component Contribute to Multi-Step Adaptation?

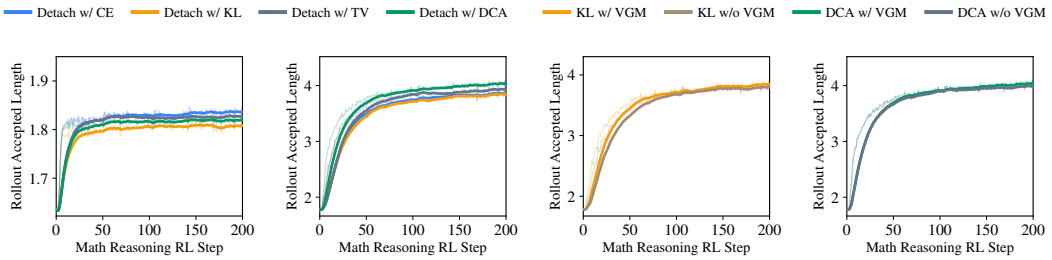
Following the MiMo depth study, we isolate component effects using the same pretrained single-step head initialization. RQ1 separately establishes the complete *GrowMTP*'s from-scratch capability.

Draft-Path Reconstruction. On MiMo-7B-SFT at $K{=}5$, we replace the reconstructed draft path with the committed response, teacher forcing under an unchanged objective (Zhang et al., 2025). Teacher forcing supervises every position at every depth rather than one chain per cycle, and the two agree only at the first draft position. As shown in Table 5 and **Appendix C.4**, the added positions raise cost without raising aggregate acceptance length: acceptance ties within 0.03 while the step speedup falls from 1.41× to 1.28×. Resolved by depth, however, the discrepancy does not vanish: acceptance is indistinguishable at shallow positions, while the deepest retain a gap that narrows without closing. Aggregate τ absorbs that residue, since the deepest positions contribute the smallest chain terms. Thus, despite similar aggregate τ at $K{=}5$, teacher forcing retains a deeper-position gap and yields lower end-to-end efficiency than draft-path reconstruction. **Appendix G** analyzes this gap in detail.

Table 5: **Ablation of Draft-Path Reconstruction on MiMo-7B-SFT (200 RL steps, $K{=}5$).** Acceptance ties within 0.03, and reconstruction lifts the end-to-end step speedup from 1.28× to 1.41×.

	Math Reasoning RL					Math Reasoning Eval					
	DAPO-Math-17K					AMC23		AIME24		AIME25	
Method	Rollout (s)	Speedup	Step (s)	Speedup	τ	Mean@16	τ	Mean@16	τ	Mean@16	τ
Teacher forcing	169.05	1.89×	408.09	1.28×	4.02	91.56	3.93	54.37	3.82	43.96	3.79
GrowMTP	**164.84**	**1.93×**	**371.02**	**1.41×**	**4.04**	91.25	**3.95**	56.87	**3.84**	46.25	**3.82**

DCA Loss. Retaining the setup of the depth study, we train in mathematical reasoning and replace only the objective: CE, KL (Zhou et al., 2024), and TV (Li et al., 2026a). As shown in Figure 4(a,b) and **Appendix C.4**, at $K{=}1$ the four objectives differ by less than 0.03 in τ, as the chain reduces to a single acceptance rate leaving no depth structure. At $K{=}5$, DCA leads consistently: τ reaches 4.04 against 3.86 for CE, 3.85 for KL, and 3.94 for TV, and the end-to-end step speedup 1.41× against 1.36× to 1.37×, head-update cost included. Relative to CE and KL, the advantage widens with depth: DCA is built directly on the acceptance chain, whereas per-position imitation ignores the chain dependence. TV and DCA use the same acceptance-chain surrogate; the logarithm changes their relative cycle weights in the batch average, thus favoring cycles with smaller chain values.



(a) DCA ablation **(K=1)**. (b) DCA ablation **(K=5)**. (c) VGM ablation **(KL)**. (d) VGM ablation **(DCA)**.

Figure 4: **DCA and VGM support *GrowMTP* from complementary sides (MiMo-7B-SFT).** (a,b) The τ gap of DCA over CE, KL, and TV opens with depth. (c,d) Removing VGM degrades τ.

Verify-Gated Masking. We remove VGM under both the KL and DCA objectives. As shown in Figure 4(c,d) and **Appendix C.4**, both degrade: the step speedup of KL falls from $1.37\times$ to $1.28\times$, showing an end-to-end efficiency benefit from masking post-rejection positions. DCA degrades less, from $1.41\times$ to $1.35\times$. Its soft weighting can retain post-rejection gradients; VGM enforces the rejection boundary and provides an additional efficiency gain. Together, the components improve multi-step adaptation: reconstruction lowers training cost at comparable acceptance length, DCA improves acceptance, and VGM further improves acceptance and end-to-end efficiency.

5 Related Work

5.1 Speculative Decoding

The draft model has evolved from independent small models (Miao et al., 2024) to lightweight draft heads that reuse the backbone's hidden states (Cai et al., 2024; Li et al., 2024a), now common components of frontier models (Liu et al., 2024; Qwen, 2026; Xiaomi et al., 2025). Draft heads can be trained jointly with the backbone or through offline distillation from target-generated data (Gloeckle et al., 2024; Li et al., 2026b; Hu et al., 2026), or adapted online to the served query distribution (Liu et al., 2023). HASS (Zhang et al., 2025) and EAGLE-3 (Li et al., 2026b) seek to align draft-head training states with those encountered during inference. After SFT warm-up, Draft-OPD (Lei et al., 2026) replays draft paths in a separate online distillation stage and treats accepted and rejected positions differently. *GrowMTP* shares this alignment motivation, reconstructing training states from draft paths and matching verification distributions recorded during RL. For acceptance-oriented training, Bebop (Li et al., 2026a) introduces a multi-step TV-chain objective. DCA applies a logarithmic transformation to this objective to adjust the relative gradient weights of cycles and combines it with VGM, which retains supervision up to and including the first rejected position.

5.2 Accelerating Reinforcement Learning for LLMs

Recent work accelerates the rollout stage that dominates RL wall-clock. One line integrates high-throughput inference engines into the RL training loop to reduce rollout cost (Hu et al., 2025; Shen et al., 2024). For example, VeRL (Sheng et al., 2025) adopts a hybrid controller architecture that decouples RL control flow from distributed computation. Another line optimizes rollout scheduling to reduce tail waiting (Noukhovitch et al., 2025). For example, AReaL (Fu et al., 2026) uses asynchronous execution to reduce synchronization stalls. Speculative decoding increases the number of tokens committed per verification step: RhymeRL (He et al., 2025) constructs drafts from historical rollouts, while ReSpec (Chen et al., 2026) updates draft models with RL supervision and includes an offline warm-up stage. *GrowMTP* supports random initialization, integrating drafting, verification, and head updates into the same RL run without a separate offline training stage. We measure end-to-end speedups over the full run, including training cost, and evaluate it on pretrained heads.

6 Conclusion

In this paper, we propose *GrowMTP*, a from-scratch online draft-head training method for accelerating the rollout stage of RL training. We observe that RL training itself provides both conditions required for online draft-head training: its rollout distribution is far narrower than that of pretraining, and its verification step continuously produces supervision signals aligned with this distribution. Building on these observations, *GrowMTP* trains the draft head online within the RL loop using this signal directly, enabling a randomly initialized head to grow entirely within RL without any offline pretraining or warm-up. Comprehensive evaluations on Qwen3-4B (no draft head), MiMo-7B-SFT (weak head), and Qwen3.5-4B-Base (strong head) show that *GrowMTP* achieves rollout speedups of $2.13\times$, $1.93\times$, and $1.36\times$, and end-to-end speedups of $1.60\times$, $1.41\times$, and $1.20\times$, respectively, with comparable held-out benchmark scores in the reported runs. These results demonstrate that speculative decoding for RL acceleration no longer requires a separate offline training stage. *GrowMTP* therefore can be integrated into existing RL training frameworks as a modular acceleration component, particularly offering a from-scratch acceleration path for models without pretrained MTP heads.

AI USE STATEMENT

AI tools were used to assist with language editing. The authors reviewed, verified, and approved the final content and take full responsibility for the accuracy, originality, and conclusions of this paper.

ETHICS STATEMENT

This work studies the computational efficiency of RL post-training using publicly available models and benchmarks. Reducing the compute required for training can improve access to research, but does not address biases or harmful outputs of the underlying models. Applications should retain appropriate safety evaluations and follow the applicable model and dataset licenses in deployment.

REPRODUCIBILITY STATEMENT

The code repository is available at `[https://anonymous.4open.science/r/GrowMTP]`. **Appendix B** specifies the models, datasets, training hyperparameters, hardware, and evaluation protocols. **Appendix E** details the recording of verification signals, draft-path reconstruction, and parallel training implementation, while **Appendix F** provides the complete training algorithm. Additional results and ablations appear in **Appendix C**. **Appendices G–I** present the technical analyses, including the underlying assumptions, complete proofs, and detailed derivations.

REFERENCES

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for medusa decoding. *arXiv preprint arXiv:2402.05109*, 2024.

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. *arXiv preprint arXiv:2401.10774*, 2024.

Ruisheng Cao, Mouxiang Chen, Jiawei Chen, Zeyu Cui, Yunlong Feng, Binyuan Hui, Yuheng Jing, Kaixin Li, Mingze Li, Junyang Lin, et al. Qwen3-coder-next technical report. *arXiv preprint arXiv:2603.00729*, 2026.

Aakshita Chandiramani, Aaron Blakeman, Abdullahi Olaoye, Abhibha Gupta, Abhilash Somasamudramath, Abhinav Khattar, Adeola Adesoba, Adi Renduchintala, Adil Asif, Aditya Agrawal, et al. Nemotron 3 super: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2604.12374*, 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.

Qiaoling Chen, Zijun Liu, Peng Sun, Shenggui Li, Guoteng Wang, Ziming Liu, Yonggang Wen, Siyuan Feng, and Tianwei Zhang. Respec: Towards optimizing speculative decoding in reinforcement learning systems. *Proceedings of Machine Learning and Systems*, 8:367–379, 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.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, et al. Areal: A large-scale asynchronous reinforcement learning system for language reasoning. *Advances in Neural Information Processing Systems*, 38:36256–36282, 2026.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Jingkai He, Tianjian Li, Erhu Feng, Dong Du, Qian Liu, Tao Liu, Yubin Xia, and Haibo Chen. History rhymes: Accelerating llm reinforcement learning with rhymerl. *arXiv preprint arXiv:2508.18588*, 2025.

Jian Hu, Xibin Wu, Wei Shen, Jason Klein Liu, Weixun Wang, Songlin Jiang, Haoran Wang, Hao Chen, Bin Chen, Wenkai Fang, et al. Openrlhf: A ray-based easy-to-use, scalable and high-performance rlhf framework. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 656–666, 2025.

Shijing Hu, Jingyang Li, Zhihui Lu, and Pan Zhou. Bridging draft policy misalignment: Group tree optimization for speculative decoding. In *International Conference on Learning Representations*, volume 2026, pp. 112507–112524, 2026.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Devvrit Khatri, Lovish Madaan, Rishabh Tiwari, Rachit Bansal, Venkata Sai Surya Subramanyam Duvvuri, Manzil Zaheer, Inderjit Dhillon, David Brandfonbrener, and Rishabh Agarwal. The art of scaling reinforcement learning compute for llms. In *International Conference on Learning Representations*, volume 2026, pp. 72438–72467, 2026.

Haodi Lei, Yafu Li, Haoran Zhang, Shunkai Zhang, Qianjia Cheng, Xiaoye Qu, Ganqu Cui, Bowen Zhou, Ning Ding, Yun Luo, et al. Draft-opd: On-policy distillation for speculative draft models. *arXiv preprint arXiv:2605.29343*, 2026.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International conference on machine learning*, pp. 19274–19286. PMLR, 2023.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023.

Yucheng Li, Huiqiang Jiang, Yang Xu, Jianxin Yang, Yi Zhang, Yizhong Cao, Yuhao Shen, Fan Zhou, Rui Men, Jianwei Zhang, et al. Breaking entropy bounds: Accelerating rl training via mtp with rejection sampling. *arXiv preprint arXiv:2606.12370*, 2026a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 7421–7432, 2024b.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *Advances in Neural Information Processing Systems*, 38:136737–136756, 2026b.

Zongqian Li, Tengchao Lv, Shaohan Huang, Yixuan Su, Qinzheng Sun, Qiufeng Yin, Ying Xin, Scarlett Li, Lei Cui, Nigel Collier, et al. Scaling data difficulty: Improving coding models via reinforcement learning on fresh and challenging problems. *arXiv preprint arXiv:2603.07779*, 2026c.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. *arXiv preprint arXiv:2310.07177*, 2023.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 932–949, 2024.

Michael Noukhovitch, Shengyi Huang, Sophie Xhonneux, Arian Hosseini, Rishabh Agarwal, and Aaron Courville. Asynchronous rlhf: Faster and more efficient off-policy rl for language models. In *International Conference on Learning Representations*, volume 2025, pp. 4003–4029, 2025.

Qwen. Qwen3.5: Towards native multimodal agents, February 2026. URL `https://qwen.ai/blog?id=qwen3.5`.

Gerald Shen, Zhilin Wang, Olivier Delalleau, Jiaqi Zeng, Yi Dong, Daniel Egert, Shengyang Sun, Jimmy Zhang, Sahil Jain, Ali Taghibakhshi, et al. Nemo-aligner: Scalable toolkit for efficient model alignment. *arXiv preprint arXiv:2405.01481*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, C Du, C Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms, 2025. *URL https://arxiv. org/abs/2501.12599*, 118, 2025.

Zili Wang, Jiajun Chai, Lin Chen, Xiaohan Wang, Shiming Xiang, and Guojun Yin. Joint training of multi-token prediction in reinforcement learning via optimal coefficient calibration. *arXiv preprint arXiv:2605.28184*, 2026.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 7655–7671, 2024.

LLM Xiaomi, Bingquan Xia, Bowen Shen, Dawei Zhu, Di Zhang, Gang Wang, Hailin Zhang, Huaqiu Liu, Jiebao Xiao, Jinhao Dong, et al. Mimo: Unlocking the reasoning potential of language model–from pretraining to posttraining. *arXiv preprint arXiv:2505.07608*, 2025.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning harmonized representations for speculative sampling. In *International Conference on Learning Representations*, volume 2025, pp. 35367–35388, 2025.

Yifan Zhang and Team Math-AI. American mathematics competitions (amc) 2023, 2023.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2024, 2024.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025.

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

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

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*, volume 2024, pp. 32011–32050, 2024.

Appendix

Contents

A Notation and Definitions **15**

B Experimental Setup and Implementation **16**
B.1 Models and Draft Heads 16
B.2 RL Training Configuration 16
B.3 Draft-Head Training Configuration 17
B.4 Evaluation Protocol 17

C Additional Results **18**
C.1 Additional Results for Growing from Scratch (RQ1) 18
C.2 Additional Results for Training Strategy (RQ2) . . . 19
C.3 Additional Results for Drafting Depth (RQ2) 20
C.4 Additional Results for Component Ablations (RQ3) 20

D Discussion **22**
D.1 Which Models Can *GrowMTP* Accelerate? 22
D.2 Does the Acceleration Grow with RL Scale? 22
D.3 Does *GrowMTP* Replace Pretrained Draft Heads? . 22

E *GrowMTP* Implementation Details **23**
E.1 Draft-Path Reconstruction 23
E.2 Implementing Draft-Path Reconstruction in verl . . 23
E.3 Parallelizing *GrowMTP* Training 24

F *GrowMTP* Training Algorithm **25**

G Train–Inference Inconsistency under Teacher Forcing **26**

H One-Step Lag Bound under Rejection Sampling **28**

I Optimization Analysis of Training Objective **30**

J Limitations **31**

A NOTATION AND DEFINITIONS

Table 6 summarizes the notation used throughout the paper.

Table 6: Notation used throughout the paper.

Symbol	Definition
	RL Training Notation
π_θ	The policy, the language model itself.
θ	Backbone (policy) parameters.
x	Prompt.
$y,\ y_t$	A sampled response and its token at position t.
G	Number of responses sampled per prompt.
t	RL step index in Appendix H; elsewhere the token position of y_t.
$p^{(t)},\ p^{(t+1)}$	Target distributions before and after one policy update.
$\rho(y)$	Importance ratio $p^{(t+1)}(y \mid s) \,/\, p^{(t)}(y \mid s)$ between successive policies.
$\bar{\varepsilon}_t$	Uniform bound on the per-update target shift.
	Speculative Decoding Notation
K	Drafting depth: the number of draft tokens per cycle.
r	Draft-then-verify cycle index.
$d_{r,k}$	The k-th draft token of cycle r.
$\tilde{d}_{r,j}$	The replacement token resampled at the first rejection.
$j,\ j_r$	First-rejection position of a cycle, with $j = K{+}1$ when every draft is accepted.
$p_{r,k}$	Target distribution at draft position k of cycle r.
$q_{r,k}$	Head distribution at draft position k of cycle r.
α_k	Acceptance probability at position k: $\alpha_k = 1 - \mathrm{TV}(p_k, q_k)$.
$\tilde{\alpha}(s),\ \alpha(s)$	Acceptance rate against the old and the new target at state s.
τ	Acceptance length: the average number of tokens committed per cycle, $\tau \in [1, K{+}1]$.
TV	Total variation distance.
	Model and State Notation
z_θ	Target-model logit function.
h_ϕ	The MTP draft head, mapping a conditioning state to a distribution.
ϕ	Draft-head parameters.
$\mathbf{c}_r$	Committed context of cycle r: the prompt and all tokens committed before the cycle.
$\mathbf{d}_r$	Draft path of cycle r: $[d_{r,1}, \ldots, d_{r,K}]$.
$s_{r,k}$	Reconstructed state $[\mathbf{c}_r, d_{r,1}, \ldots, d_{r,k-1}]$.
	Training Objective Notation
$\mathcal{L}_{\mathrm{TV}}$	TV-chain loss: $-\sum_{l=1}^{K} \prod_{i=1}^{l} \alpha_i$, a differentiable surrogate for $1-\tau$.
$\mathcal{L}_{\mathrm{DCA}}$	DCA loss: $-\log(-\mathcal{L}_{\mathrm{TV}})$.
$\mathcal{L}_{\mathrm{DCA}}^{\mathrm{VGM}}$	Verify-gated DCA loss: the chain sum truncated at $\min(j, K)$.
M	The chain quantity $-\mathcal{L}_{\mathrm{TV}}$ in Appendix I.
D_k	Tail term of the gradient weight: $\sum_{l=k}^{K} \prod_{i=k+1}^{l} \alpha_i \in [1, K{-}k{+}1]$.
g_r	Gradient of $\mathcal{L}_{\mathrm{DCA}}$ on cycle r in Appendix I.

B Experimental Setup and Implementation

B.1 Models and Draft Heads

Baseline Models. We evaluate *GrowMTP* on three models: Qwen3-4B, Qwen3.5-4B-Base, and MiMo-7B-SFT. The three differ in the draft head they ship: Qwen3-4B has never shipped one, while Qwen3.5-4B-Base and MiMo-7B-SFT each ship a native head. The drafting depth of such a head is not fixed by its architecture, so we adopt the depth recommended by each model's release: $K{=}3$ for Qwen3.5-4B-Base and $K{=}1$ for MiMo-7B-SFT. Apart from this shipped state, the draft heads of the three models take the same single-layer recurrent MTP form throughout this paper, detailed below.

MTP Architecture. All three draft heads take the EAGLE-style single-layer recurrent MTP form. The module contains a single decoder layer. It drafts one position at a time, consuming the hidden state and the token embedding of the position before it: the two are normalized by separate RMSNorms, concatenated, and projected back to the hidden dimension by a linear layer without bias before entering the decoder layer. It maintains a KV cache of its own, which grows by one entry with every position it drafts. The head contains neither an embedding nor an output projection and takes both from the backbone. The heads of the three models therefore run through the same drafting and training implementation, and the family difference is confined to the type of the decoder layer, one each from Qwen3, Qwen3.5, and Qwen2. The native head of MiMo concatenates the two inputs in the opposite order, and weight loading swaps the two halves of that linear layer accordingly.

From-Scratch Initialization. The draft head of Qwen3-4B is built in this same form, identical to the two native heads down to the parameter names. Randomization covers this newly built module alone: every linear layer within it is drawn from $\mathcal{N}(0,\, 0.02)$, the value of the backbone's own `initializer_range`, and every RMSNorm weight is set to one. The head is materialized once into a checkpoint under seed 0, and the training side and the SGLang drafting side load the same weights from it, so no divergence between two independent initializations arises. The embedding and the output head are inherited from the backbone rather than randomized, so training from scratch refers to the drafting capability and not to every parameter that participates in drafting.

B.2 RL Training Configuration

Policy Optimization. Policy optimization uses GRPO with one modification: the negative-advantage branch carries a weight of 0.5, and every other term follows the standard form, dual clipping and token-level averaging among them. The objective retains a KL regularizer at coefficient 0.01, whose reference policy is the initial checkpoint and stays fixed throughout training. The policy is updated by AdamW in verl, at a constant learning rate of 1×10^{-6} and with a clipping range of 0.2. All three models, both domains, and every ablation arm share this configuration, so the differences between groups arise only from the variable isolated in each comparison with policy optimization held fixed.

Training Data. Mathematical reasoning uses DAPO-Math-17K and code reasoning uses TACO-Verified. The reward comes from matching the reference answer in mathematical reasoning and from matching the standard output of the generated program in code reasoning. Both datasets are used as released, without difficulty filtering or subsampling, with head supervision drawn from RL rollouts.

Rollout and Batching. All runs execute on a single node of $8\times$H800 GPUs, with the model sharded across the eight devices. Each step samples 64 prompts and generates 8 responses per prompt, and the policy update uses a mini-batch of 64. Prompts are truncated to 1024 tokens, and responses are capped at 8192 tokens on Qwen3-4B and MiMo-7B-SFT and at 16384 tokens on Qwen3.5-4B-Base. Rollouts are served by SGLang at temperature 1.0 and top-p 1.0, with draft tokens accepted by rejection sampling. Qwen3-4B runs in thinking mode, and the other two models are not thinking models. All runs fix the data order with seed 1, keeping the data order consistent across methods.

B.3 Draft-Head Training Configuration

Draft Head Training. In every *GrowMTP* run the head is trained under the DCA objective with detached gradients and with VGM enabled throughout. The ablation arms each change one of these: the objective becomes CE, KL, or the TV-chain objective of Appendix I, the gradients become joint or frozen, or VGM is removed. The drafting depth is $K{=}5$ on Qwen3-4B, $K{=}3$ on Qwen3.5-4B-Base, and $K{=}3$, 5, and 7 on MiMo-7B-SFT, whose frozen baseline runs at $K{=}1$. The head is updated once per RL step, from the verification signals recorded in that step's own rollouts. Training includes no freeze phase, and the head is updated from the first step to the last using online RL supervision.

Optimizer and Learning Rate. A shared AdamW optimizer updates the head and backbone in separate parameter groups with independent learning rates. Head learning rates are 3×10^{-4} for random initialization and 1×10^{-4} for the two pretrained heads. Both head rates warm up over 10 steps, then follow a cosine decay to one tenth of their peak; the policy learning rate stays constant.

B.4 Evaluation Protocol

Training Metrics. On the training side we report two times, the rollout time and the end-to-end step time, the latter including the cost of head updates. Both are per-step means over the whole of training. The denominator of a speedup is the run with the head frozen under the same script, the same data, and the same hyperparameters, and on Qwen3-4B that denominator carries no head and reduces to autoregressive decoding. The training-side τ is instead averaged over the final 10 steps: the times are costs incurred throughout, whereas τ is the capability of the head once it has converged in the run.

Inference Evaluation Benchmarks. Mathematical reasoning is evaluated on AMC23, AIME24, and AIME25, with 40, 30, and 30 questions respectively. Code reasoning is evaluated on LiveCodeBench v6, whose 1055 questions include the AtCoder and LeetCode platform slices at 602 and 444 questions. Judging uses the same string-matching verifier as training in mathematical reasoning and the official LiveCodeBench framework in code reasoning, with a timeout of 6 seconds for each separate test case.

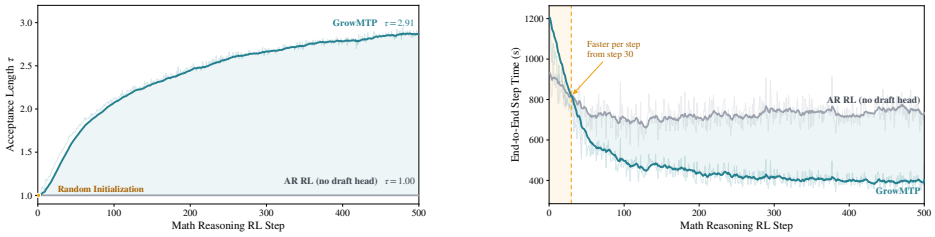
Evaluation Configuration. SGLang generates responses at temperature 1.0 and top-p 0.7, capped at 16384 tokens. Speculative decoding uses rejection sampling, as in training. Evaluation uses each run's trained depth, each frozen baseline's shipped depth, and no draft head for the AR baseline.

Evaluation Metrics. Policy quality is reported as Mean@16 in mathematical reasoning, the mean accuracy over 16 samples. In code reasoning it is reported as Pass@4, the fraction of questions on which at least one of 4 attempts passes every test. Drafting capability is the held-out τ, one plus the ratio of accepted tokens to verifications, measured on the same generations used to assess quality.

C Additional Results

C.1 Additional Results for Growing from Scratch (RQ1)

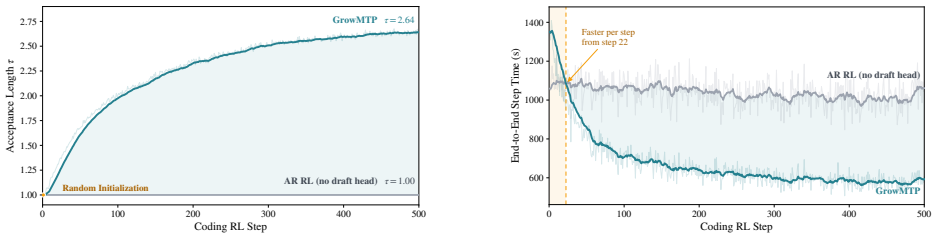
Math Reasoning RL Training. Figure 5 shows the mathematical reasoning run: the acceptance length τ and the end-to-end step time over the 500 RL steps. From random initialization at the autoregressive floor, τ rises steepest over the first hundred steps and is still climbing at step 500. Step time opens above the no-head baseline, the early overhead visible on the curve, and crosses below it at step 30: every later step runs faster, and the widening gap tracks the growing τ throughout the run.



(a) Acceptance length τ. (b) End-to-end step time.

Figure 5: **Math reasoning (Qwen3-4B): a draft head grows from scratch within RL and accelerates the run itself.** τ rises to 2.91, and step time drops below the no-head baseline within 30 steps.

Code Reasoning RL Training. Figure 6 shows the training trajectories of the code reasoning run: τ and the end-to-end step time over the 500 RL steps. The trajectories mirror the mathematical reasoning run of Figure 5: τ leaves the autoregressive floor immediately and climbs to 2.64 over the 500 steps, and step time falls below the no-head baseline within 22 steps. The from-scratch growth pattern is thus not specific to a single training domain: both domains support online head training.



(a) Acceptance length τ. (b) End-to-end step time.

Figure 6: **Code reasoning (Qwen3-4B): the from-scratch growth pattern of Figure 5 holds.** τ rises to 2.64 over 500 RL steps, and step time drops below the no-head baseline within 22 steps.

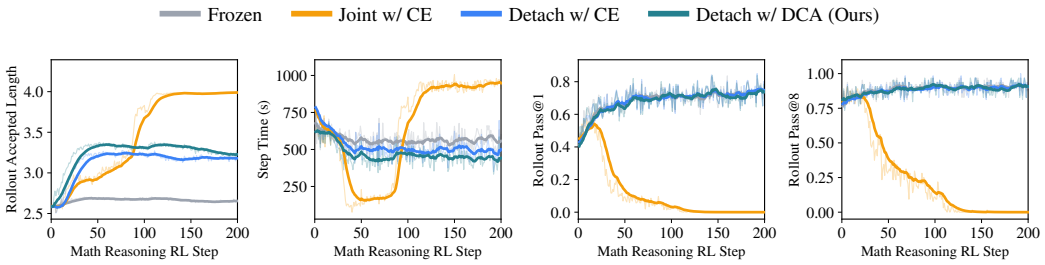
Inference Evaluation. Table 7 reports the held-out evaluation of the two runs, pairing policy quality with τ on each benchmark. Policy quality stays on par with the autoregressive baseline on all six benchmarks, with Mean@16 and Pass@4 differing by at most 2.29 points. The trained heads reach τ of 2.30–2.34 on the mathematical benchmarks and 2.11–2.27 on the code benchmarks: the drafting capability learned on training rollouts carries over to held-out evaluation prompts of the same domain.

Table 7: **Inference evaluation on Qwen3-4B (500 RL steps, K=5).** The trained draft head reaches τ above 2.1 on every benchmark in both domains, while policy quality (Mean@16 for math, Pass@4 for code) remains on par with the K=0 baseline, autoregressive decoding without any draft head.

		Math Reasoning Eval						**Code Reasoning Eval**					
		AMC23		AIME24		AIME25		AtCoder		LeetCode		LiveCodeBench	
Method	K	Mean@16	τ	Mean@16	τ	Mean@16	τ	Pass@4	τ	Pass@4	τ	Pass@4	τ
AR RL	0	62.81	1.00	20.42	1.00	18.75	1.00	42.19	1.00	42.79	1.00	42.37	1.00
GrowMTP	5	64.69	**2.33**	20.42	**2.34**	21.04	**2.30**	43.36	**2.27**	43.24	**2.11**	43.51	**2.22**

C.2 Additional Results for Training Strategy (RQ2)

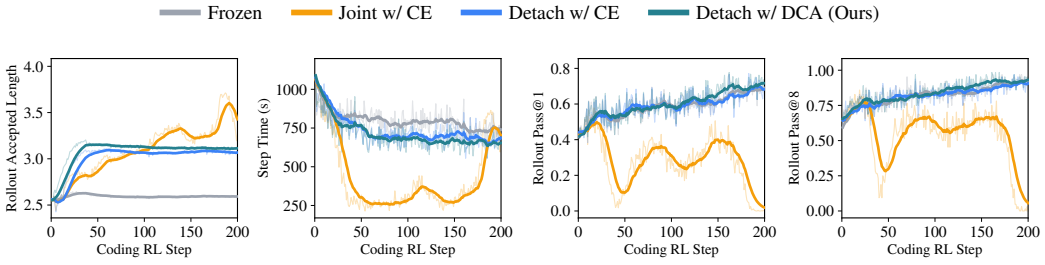
Math Reasoning RL. Figure 7 shows the four training strategies on mathematical reasoning: τ, the end-to-end step time, and held-out quality (Pass@1 and Pass@8) over the 200 RL steps. Both detached arms raise τ above the frozen head while their quality curves stay level throughout. Joint CE traces the opposite pattern: its τ climbs to the highest of the four while its Pass@1 and Pass@8 collapse to zero, the covariation behind reading a rising τ as no evidence of a healthy RL training run.



(a) Acceptance length. (b) End-to-end step time. (c) Pass@1. (d) Pass@8.

Figure 7: **Math reasoning (Qwen3.5-4B-Base): four training strategies.** Both detached arms raise τ above the frozen head and DCA leads, while joint CE reaches the highest τ at zero held-out quality.

Code Reasoning RL. Figure 8 repeats the comparison on code reasoning. Among the quality-preserving strategies the trajectories mirror the math domain: both detached arms accelerate the run with level quality curves, and DCA leads. Joint CE, however, reaches the lowest step time of the four here after slowing the run on math, at zero quality in both domains: the speed of a collapsed policy flips sign across domains and cannot establish whether RL accelerates with preserved policy quality.



(a) Acceptance length. (b) End-to-end step time. (c) Pass@1. (d) Pass@8.

Figure 8: **Code reasoning (Qwen3.5-4B-Base): the same comparison.** The quality-preserving ranking mirrors the math domain, while joint CE reaches the largest speedups at zero held-out quality.

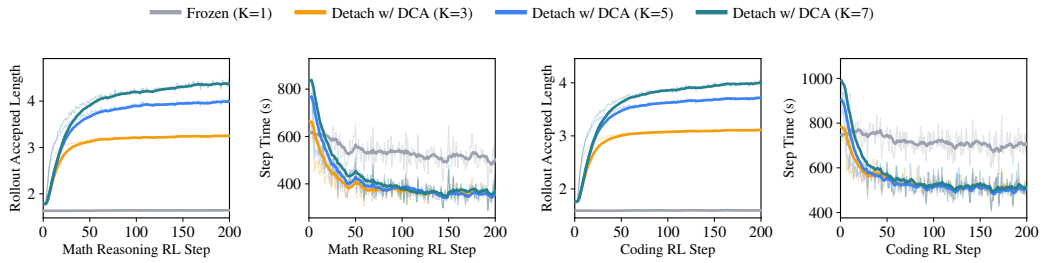
Inference Evaluation. Table 8 reports the held-out evaluation of the four strategies, pairing policy quality with τ on each benchmark. Among the quality-preserving strategies, DCA reaches τ of 3.22–3.31 against 3.10–3.19 for detached CE, leading on all six benchmarks. Joint CE scores zero on every benchmark in both domains, and its τ of 3.98–3.99 is measured on this collapsed policy: these high drafting numbers are excluded from the comparison among the quality-preserving strategies.

Table 8: **Inference evaluation on Qwen3.5-4B-Base (200 RL steps, K=3).** Among quality-preserving strategies, DCA reaches the highest τ on every benchmark. Joint CE collapses to zero.

		Math Reasoning Eval						**Code Reasoning Eval**					
		AMC23		AIME24		AIME25		AtCoder		LeetCode		LiveCodeBench	
Training Mode	Loss	Mean@16	τ	Mean@16	τ	Mean@16	τ	Pass@4	τ	Pass@4	τ	Pass@4	τ
Frozen	–	80.47	2.61	55.00	2.57	46.67	2.57	63.29	2.52	72.52	2.52	67.39	2.52
Joint	CE	0.00	3.99	0.00	3.99	0.00	3.99	0.00	3.98	0.00	3.98	0.00	3.98
Detached	CE	84.69	3.19	57.92	3.10	52.50	3.10	61.63	3.15	71.40	3.13	65.97	3.14
Detached	DCA (*GrowMTP*)	85.78	**3.31**	59.58	**3.23**	51.46	**3.22**	63.29	**3.22**	70.50	**3.22**	66.54	**3.22**

C.3 Additional Results for Drafting Depth (RQ2)

RL Training. Figure 9 shows the drafting depth sweep on MiMo-7B-SFT: τ and the end-to-end step time at $K{=}3$, 5, and 7 against the frozen $K{=}1$ head, in both domains. The τ trajectories stay separated by depth throughout training, a deeper head drafting more accepted tokens at every step, and none of the three curves has saturated by step 200. Deeper heads initially cost more per step, but as acceptance grows, the three step-time curves converge far below the frozen baseline: the depth ranking visible on the τ axis does not carry to the time axis, which determines the preferred depth.



(a) τ **(Math)**. (b) Step time **(Math)**. (c) τ **(Code)**. (d) Step time **(Code)**.

Figure 9: **Drafting depth (MiMo-7B-SFT):** $K{=}3, 5, 7$ **against the frozen** $K{=}1$ **head.** Acceptance grows beyond the single-step ceiling, and end-to-end gains peak at $K{=}5$ in both training domains.

Inference Evaluation. Table 9 reports the held-out evaluation across the four depths, pairing policy quality with τ on each benchmark. Every trained depth reaches τ of at least 3.02, increasing with K on each of the six columns, while policy quality stays on par with the frozen head. The τ of $K{=}7$ is the highest in every column, yet the highlighted depth is $K{=}5$: held-out acceptance does not decide the selection, which follows the end-to-end speedups reported in Table 4, including head-update cost.

Table 9: **Inference evaluation across drafting depths on MiMo-7B-SFT (200 RL steps).** Online training lifts τ above 3.0 at every depth, monotonically in K, while policy quality stays on par with the frozen head. The highlighted row follows end-to-end acceleration, not the highest measured τ.

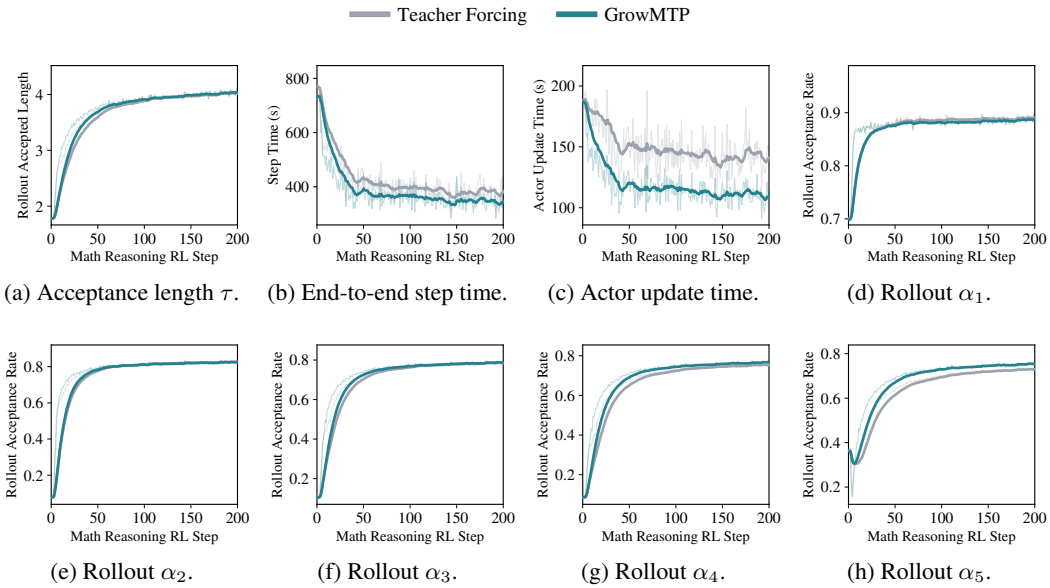
		Math Reasoning Eval						**Code Reasoning Eval**					
		AMC23		AIME24		AIME25		AtCoder		LeetCode		LiveCodeBench	
Method	K	Mean@16	τ	Mean@16	τ	Mean@16	τ	Pass@4	τ	Pass@4	τ	Pass@4	τ
Frozen	1	90.47	1.67	51.88	1.64	43.13	1.64	69.93	1.61	80.18	1.61	74.50	1.61
GrowMTP	3	90.47	3.20	53.33	3.15	42.29	3.14	70.10	3.02	78.38	3.03	73.84	3.03
GrowMTP	5	91.25	3.95	56.87	3.84	46.25	3.82	71.59	3.59	78.60	3.59	74.69	3.59
GrowMTP	7	89.69	4.30	56.04	4.11	43.33	4.08	69.93	3.83	76.35	3.85	72.80	3.84

C.4 Additional Results for Component Ablations (RQ3)

Ablation of Draft-Path Reconstruction. Figure 10 traces both arms: τ, the two timing curves, and rollout acceptance at every draft position. Teacher forcing pays roughly 30 extra seconds per actor update, and its bias sits at depth: α_1 is indistinguishable between the arms, while the α_5 gap narrows without closing. This is the train–inference inconsistency of Appendix G observed in the results.

Ablation of Training Objective. Table 10 reports the four objectives at both depths, training and held-out sides. The $K{=}1$ rows stay within 0.03 on every τ column, and at $K{=}5$ DCA holds the highest τ in every column, extending its training-side lead to all three held-out evaluation benchmarks.

Ablation of Verify-Gated Masking. Table 11 reports the mask switch under both objectives. Under KL the slowdown is visible already at rollout, while under DCA it falls almost entirely outside rollout, and held-out τ is never higher with the mask removed on any of the three held-out benchmarks.



(a) Acceptance length τ. (b) End-to-end step time. (c) Actor update time. (d) Rollout α_1.

(e) Rollout α_2. (f) Rollout α_3. (g) Rollout α_4. (h) Rollout α_5.

Figure 10: **Teacher forcing costs more per step and leaves a residual gap at the deepest draft positions.** MiMo-7B-SFT, 200 RL steps, K=5. (a–c) *GrowMTP* reaches the same acceptance length at a lower actor update cost. (d–h) Rollout acceptance α_k separates further with each added depth.

Table 10: **Ablation of Objective on MiMo-7B-SFT (200 RL steps, $K \in \{1, 5\}$, mathematical reasoning).** At K=1, the four objectives yield similar acceptance lengths. At K=5, DCA leads on training acceptance, end-to-end step speedup, and held-out τ on each of the three benchmarks.

		Math Reasoning RL					**Math Reasoning Eval**					
		DAPO-Math-17K					AMC23		AIME24		AIME25	
Loss	K	Rollout (s)	Speedup	Step (s)	Speedup	τ	Mean@16	τ	Mean@16	τ	Mean@16	τ
CE	1	274.47	1.16×	507.99	1.03×	1.84	90.16	1.88	51.67	1.87	40.42	1.87
KL	1	288.58	1.10×	532.91	0.98×	1.81	90.31	1.86	54.37	1.85	44.58	1.84
TV	1	288.30	1.11×	530.29	0.99×	1.83	89.22	1.88	56.04	1.86	43.13	1.86
DCA (*GrowMTP*)	1	280.58	1.14×	517.00	1.01×	1.82	89.53	1.87	53.96	1.85	36.88	1.86
CE	5	175.70	1.81×	384.22	1.36×	3.86	89.84	3.81	54.17	3.67	43.54	3.66
KL	5	174.98	1.82×	382.41	1.37×	3.85	89.69	3.67	53.75	3.53	42.29	3.51
TV	5	174.00	1.83×	382.69	1.37×	3.94	89.53	3.80	55.00	3.67	40.83	3.66
DCA (*GrowMTP*)	5	**164.84**	**1.93**×	**371.02**	**1.41**×	**4.04**	91.25	**3.95**	56.87	**3.84**	46.25	**3.82**

Table 11: **Ablation of Verify-Gated Masking on MiMo-7B-SFT (200 RL steps, K=5, mathematical reasoning).** Removing VGM slows end-to-end training steps under both objectives, and KL degrades more than DCA: the step speedup falls from 1.37× to 1.28× and from 1.41× to 1.35×.

		Math Reasoning RL					**Math Reasoning Eval**					
		DAPO-Math-17K					AMC23		AIME24		AIME25	
Loss	VGM	Rollout (s)	Speedup	Step (s)	Speedup	τ	Mean@16	τ	Mean@16	τ	Mean@16	τ
KL	✗	186.40	1.71×	408.65	1.28×	3.80	88.59	3.66	54.17	3.49	41.04	3.48
KL	✓	174.98	1.82×	382.41	1.37×	3.85	89.69	3.67	53.75	3.53	42.29	3.51
DCA	✗	165.97	1.92×	387.27	1.35×	4.00	89.22	3.94	52.92	3.81	41.88	3.79
DCA (*GrowMTP*)	✓	164.84	1.93×	**371.02**	**1.41**×	4.04	91.25	3.95	56.87	3.84	46.25	3.82

D DISCUSSION

D.1 WHICH MODELS CAN *GrowMTP* ACCELERATE?

In this section, we discuss the applicability of *GrowMTP*. The experiments of Section 4 evaluate Qwen3-4B, MiMo-7B-SFT, and Qwen3.5-4B-Base, achieving end-to-end step speedups of $1.60\times$, $1.41\times$, and $1.20\times$ on mathematical reasoning: *GrowMTP* is effective regardless of initial head capability. For models with pretrained heads, the RL rollout distribution is more concentrated than the broad pretraining data, leaving room for domain-specific adaptation. Beyond head capability, *GrowMTP*'s acceptance-based training objective is model-agnostic: it depends on draft–target token-distribution overlap and the first-rejection index under rejection sampling. *GrowMTP* can therefore serve as a modular acceleration component in RL training frameworks that employ speculative decoding, particularly for base models without MTP heads, which accelerate without extra pretraining.

D.2 DOES THE ACCELERATION GROW WITH RL SCALE?

The $1.60\times$ of the Qwen3-4B mathematical reasoning run is measured at 500 steps and an 8K rollout length, a small RL scale. We extrapolate from it along rollout length and training steps: the former increases the proportion of time spent on rollouts, and the latter amortizes the head's growth-phase overhead. For this projection, we assume that rollout time scales linearly with length while the remaining per-step cost stays constant. Beyond 500 steps we take the step time measured at the end of training, 398.7 s, of which 205.7 s is rollout. We further assume that acceptance length grows neither with rollout length nor with training steps. The end-to-end speedup then grows monotonically, reaching $1.91\times$ at 16K with 1000 steps and $2.17\times$ at 32K with 2000 steps. These values are illustrative projections under the stated assumptions. Since the learning curve has not saturated at 500 steps, further head improvement could increase the realized speedup. Under these assumptions, the ceiling is $2.55\times$, the final measured RL rollout speedup.

Table 12: **The projected speedup of *GrowMTP* grows monotonically with RL scale.** End-to-end step speedup over the AR baseline, extrapolated from the measured Qwen3-4B run on DAPO-Math-17K ($K=5$, the 500-step 8K entry).

	Rollout Length		
Math Reasoning RL Steps	8K	16K	32K
500	$1.60\times$	$1.79\times$	$1.93\times$
1000	$1.70\times$	$1.91\times$	$2.08\times$
2000	$1.76\times$	$1.99\times$	$2.17\times$

D.3 DOES *GrowMTP* REPLACE PRETRAINED DRAFT HEADS?

We freeze the two heads grown from scratch over 500 RL steps, one in each domain, and measure τ on the held-out benchmarks of both. Within the training domain, τ falls from the training rollouts to the held-out benchmarks, 2.91 to 2.32 for the math-trained head and 2.64 to 2.20 for the code-trained head. Crossing domains is far costlier: the math-trained head falls to 1.45 on code, and the code-trained head to 1.75 on math. The head receives supervision only from the rollout distribution it must accelerate, and τ falls as evaluation departs from it. This specialization is a direct consequence of the narrow-distribution premise: RL acceleration faces a sufficiently concentrated distribution for from-scratch online training to suffice, whereas general serving must cover unforeseen deployment distributions and still relies on pretrained MTP heads with capability learned across broader tasks.

Table 13: **Each head specializes to the domain it was trained on.** Acceptance length τ on Qwen3-4B ($K=5$), with Rollout on each head's own training rollouts and shaded blocks in-domain.

		Math Reasoning Eval τ				**Code Reasoning Eval τ**			
Training Domain	Rollout	AMC23	AIME24	AIME25	Avg.	AtCoder	LeetCode	LiveCodeBench	Avg.
Math	2.91	2.33	2.34	2.30	2.32	1.50	1.40	1.46	1.45
Code	2.64	1.80	1.79	1.68	1.75	2.27	2.11	2.22	2.20

E *GrowMTP* IMPLEMENTATION DETAILS

E.1 DRAFT-PATH RECONSTRUCTION

The Reconstructed Forward. Draft-path reconstruction removes the train–inference inconsistency of Section 3.1 by aligning the training-side computation with inference: the head is trained at the drafted states. The update phase reconstructs the chain of every recorded cycle: it starts from the backbone hidden recorded at the cycle's start, runs under the current head parameters inside the training graph, and is supervised by the verification signals recorded in the same cycle. Reconstruction is not redrafting: sampling would not reproduce the rollout's tokens, and with the recursion consuming a token at every position, one changed token changes every hidden state and every cache entry that follows. The draft tokens therefore turn from outputs into inputs: rollout records the K drafts of every cycle, and they are fed back to the recursion, with nothing sampled. The hidden states and cache entries along the chain need no recording, since the recursion recomputes them from the same inputs, and every reconstructed state conditions on the recorded prefix of its own draft-then-verify cycle.

The Recorded Supervision. The same obstacle appears on the supervision side: $p_{r,k}$ conditions on the drafts themselves (Eq. 2), the trajectory retains only the committed path (Eq. 3), and where the two paths diverge, a target distribution recomputed on the trajectory is no longer the one that verified the head. The remedy is again recording, not recomputation: verification is itself one target forward over the entire draft path, so it computes the distribution at every draft position, and rollout stores these beside the drafts, one per position. The two remedies interlock in the conditioning: the reconstructed $q_{r,k}$ of Eq. 4 and the recorded $p_{r,k}$ condition on the same recorded prefix $[\mathbf{c}_r, d_{r,1}, \ldots, d_{r,k-1}]$, and the pairing Section 3.1 defines at the token level becomes K concrete pairs per cycle in the training graph, consumed directly by the loss of Section 3.2. The reconstruction is self-contained: a chain reads only its own cycle's records and the shared prefix, and no chain uses another chain's states.

E.2 IMPLEMENTING DRAFT-PATH RECONSTRUCTION IN VERL

Recording in Rollout. The record of a cycle holds five items: the backbone hidden at the cycle's start, the K drafts, the verification distributions, the accepted length, and the absolute position of the start. The hidden is captured rather than recomputed: it is the very tensor the engine's drafting consumed, and the reconstruction therefore starts from a state numerically identical to drafting's. The verification distributions are stored as the target's top-64 log-probabilities in bf16: keeping the full distribution at every draft position would scale with the vocabulary, and the truncation fixes the record's width at 64. The acceptance decisions compress to one integer per cycle: the accepted positions form a prefix, so a single length, the further token counted, restores the decision at every position. The record carries the absolute position of the start, and the reconstruction puts each chain back where it sat in the sequence. Each sequence additionally stores the rollout-time target hidden states and token ids for the prompt and all committed tokens. These recorded inputs are used to prefill the MTP head and rebuild the shared prefix KV cache attended by each reconstructed draft chain.

The stored top-64 log-probabilities retain full-vocabulary normalization rather than being renormalized over the retained set. During head training, the draft distribution is normalized over the full vocabulary and gathered at the target's top-64 indices; the remaining mass of each distribution forms a residual bin. The resulting overlap upper-bounds the full-vocabulary overlap by at most the target mass outside the top-64, so the error is small when the retained target mass is high. We quantify its effect on three checkpoints from Qwen3-4B Math RL: random initialization, step 100, and step 500.

Against the full-vocabulary reference, mean absolute overlap errors range from 1.8×10^{-6} to 5.7×10^{-6}, and the mean per-batch cosine similarity between sparse and full-vocabulary head gradients is at least 0.99996 at each checkpoint. Thus, top-64 sparsification closely preserves overlap values and gradient directions. Rollout verification still uses the full distribution, so the approximation affects only the draft-head training objective without affecting rollout-time verification decisions.

Integration into the RL Loop. The head is a submodule of the FSDP-wrapped actor model, so the shared optimizer of Appendix B.3 is structure rather than wiring. Deployment rides the same structure: the weight synchronization that carries the updated actor into the drafting engine carries the head with it, and after every step the engine drafts with the head just trained. The recording sits on the same loop: signals are captured inside the engine's verification and delivered to the update phase with the rollout batch. The update phase therefore receives one record per cycle: a cycle commits τ tokens on average, so the records are fewer than the response's token positions by a factor of τ.

E.3 PARALLELIZING *GrowMTP* TRAINING

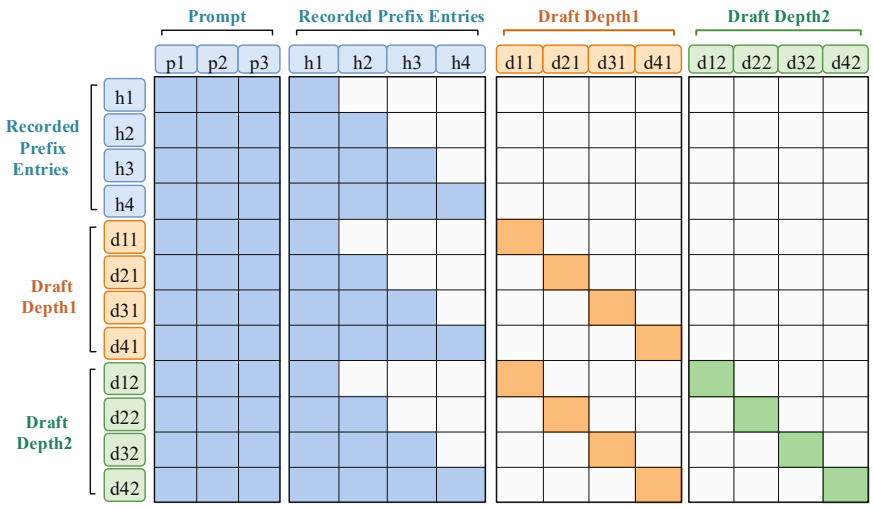


Figure 11: **The attention mask of one batched reconstruction forward.** Every chain attends over one shared cache: causal over the full committed-token prefix, diagonal over the draft entries of every depth. Each query sees exactly the context of chain-by-chain execution on recorded cycle inputs.

Serial Depth, Parallel Cycles. Executed chain by chain, the reconstruction of Appendix E.1 costs one forward per draft: for a response of three thousand cycles at $K = 5$, that is fifteen thousand single-token forwards in sequence. Each of these forwards is indexed by the pair (r, k), with the cycle index r locating the chain within the response and the depth k—the draft position within its cycle—locating the forward within the chain. Along k there is no parallelism to extract: Eq. 4 takes the state at depth $k-1$ as input at depth k, and the K forwards of one chain can only run in order. Along r no such dependence exists: chains share only recorded inputs, never each other's computed states, so the forwards of every cycle at a fixed depth admit a single batched execution. Serialization is therefore confined to depth: any schedule spends at least K forwards one after another, and the width available at each of them is the number of cycles in the response, each advanced independently.

Parallelism across Cycles. The reconstruction therefore runs as K batched forwards: the forward at depth k advances all chains by one draft, and the serial count of the response falls from fifteen thousand to K. In one batched forward, visibility no longer follows from packing order: each query must be restricted to the context chain-by-chain execution would give it, a restriction the attention mask of Figure 11 carries. All chains attend over one shared cache, with causal visibility over the full committed-token prefix and diagonal visibility over the draft entries, so no chain sees another's drafts. For each cycle, the prefilled cache is restricted to its committed prefix, while the cycle's own draft KV entries are recomputed by feeding back its recorded draft tokens in order. The mask is additive and needs no custom attention kernel: standard scaled dot-product attention consumes it directly. Positions are taken from the record rather than from the packing: every entry enters the rotary encoding at the position it held at rollout, the depth-k draft of cycle r sitting k places past its cycle's start. Merging changes the execution schedule while preserving each chain's computation.

Bounding the Memory. What merging does change is retention: a forward's activations stay resident until its backward runs, so cycles that advance together are held in memory together. At full width the resident state is the whole response—every chain at every depth, held for one backward—and at training lengths it exceeds the accelerator's memory. Bounding memory is bounding the span of one backward: the reconstruction proceeds in chunks of 1024 consecutive cycles, each chunk running its K forwards and its own backward and releasing its activations before the next begins. Chunking parallelizes nothing: it only sets how many cycles are resident at once. Across chunks the entries of the recorded hiddens accumulate, entering later chunks as constants, while a chunk's draft entries—entries the diagonal pattern shows to no later chain—are dropped at its end. Each chunk's loss is weighted by its share of the response's positions, and the gradients accumulated across chunks equal those of one backward over the response's loss: cross-chunk entries are constants, the chunks' graphs are disjoint, and gradients over disjoint graphs add. A response of three thousand cycles spans three chunks: fifteen batched forwards, where chain-by-chain reconstruction ran fifteen thousand.

F *GrowMTP* TRAINING ALGORITHM

Algorithm 1 states one *GrowMTP* training step: against the two-phase step of Section 2.1, the additions are the records $\{R_r\}$ of line 1 and the head update of lines 4–12, and no other line computes anything new. The record R_r of cycle r collects the five items of Appendix E.2; the update consumes four of them by name—the backbone hidden b_r at the cycle's start, the drafts $\mathbf{d}_r$, the verification distributions $p_{r,1:K}$, and the first rejection j_r, read from the accepted length—while the start position is consumed inside the reconstruction alone. The inner loop runs the recursion of Eq. 4 on recorded inputs: the chain starts from the hidden b_r, which enters the graph detached, and the recorded start position gives the chain its place in the sequence. SPECDECODE abbreviates the draft-then-verify generation of Section 2.2, inside which the records are captured cycle by cycle. SYNC abbreviates the weight synchronization of Appendix E.2, and all remaining symbols are those of Sections 2 and 3.

Algorithm 1 One *GrowMTP* training step.

Require: policy π_θ, draft head h_ϕ, a batch of prompts X
Ensure: θ and ϕ updated and synchronized into the drafting engine

// Rollout (drafting engine)
1: $Y, \{R_r\} \leftarrow \textsc{SpecDecode}(X;\, \pi_\theta,\, h_\phi)$ // Sec. 2.2; App. E.2
2: deliver Y, its rewards, and $\{R_r\}$ to the update phase
// Update (actor)
3: update π_θ by the RL objective on Y // Sec. 2.1
4: **for** each recorded cycle r **do** // batched across cycles, App. E.3
5: **for** $k = 1, \ldots, K$ **do** // Eq. 4; serial in k
6: $q_{r,k} \leftarrow h_\phi(\cdot \mid s_{r,k})$
7: $s_{r,k+1} \leftarrow [s_{r,k},\, d_{r,k}]$
8: **end for**
9: $\alpha_{r,k} \leftarrow 1 - \mathrm{TV}(p_{r,k},\, q_{r,k}),\, k = 1, \ldots, K$ // Eq. 5
10: $\mathcal{L}_r \leftarrow \mathcal{L}_{\mathrm{DCA}}^{\mathrm{VGM}}(\alpha_{r,1:K};\, j_r)$ // Eq. 9: chain sum truncated at $\min(j_r, K)$
11: **end for**
12: update h_ϕ by ∇_ϕ of the mean of $\mathcal{L}_r$ over the batch's cycles // the gradient reaches only ϕ
13: $\textsc{Sync}(\theta,\, \phi)$ // line 1 of the next step drafts with this ϕ

G TRAIN–INFERENCE INCONSISTENCY UNDER TEACHER FORCING

This appendix identifies the cycle-boundary and KV-cache mismatch between committed-response teacher forcing and rollout-time drafting, and traces its dependence on draft depth. The cause is one mechanism: inference restarts the head's recursion, chain and cache alike, only at cycle boundaries, while teacher forcing, consuming the flat response alone, makes every position a start and restarts nowhere. Draft-path reconstruction resolves the inconsistency by recomputing exactly the states inference runs, and no teacher-forcing mask can replace the reconstruction of the drafting states.

Training Under Teacher Forcing. Teacher forcing consumes only the final response $y_1, \ldots, y_T$, and none of the cycle structure recorded during the rollout. The head is a single module applied in a loop, consuming at depth k and position i the depth-$(k{-}1)$ output and the token embedding at the position before it, and attending the key-value entries held in the cache at earlier positions of the response,

$$g_i^{(k)} \;=\; f_\phi\big(g_{i-1}^{(k-1)},\, e(y_{i-1})\,;\, \mathcal{K}_i^{(k)}\big), \qquad k = 1, \ldots, K, \tag{10}$$

where e maps a token to its embedding, $g^{(0)}$ are the hidden states the target model produced during the rollout forward, and $g_i^{(k)}$ plays the role the sequences $s_{r,k}$ play in Eq. 4. The cache grows by one entry per position, written by the head's key-value projection c_ϕ from the same pair of head inputs,

$$\mathcal{K}_i^{(k)} \;=\; \mathcal{K}_{i-1}^{(k)} \cup \big\{c_\phi\big(g_{i-1}^{(k-1)},\, e(y_{i-1})\big)\big\}, \tag{11}$$

starting from the entries of the prompt in $\mathcal{K}_0^{(k)}$: the cache attended at depth k is thus made entirely of entries computed from depth-$(k{-}1)$ features. The K causal forwards yield at most $T \cdot K$ training pairs $\big(g_i^{(k)}, y_i\big)$, whose unrolled recursion traces each trained state back to the target's hidden features,

$$g_{i-k}^{(0)} \xrightarrow{f_\phi} g_{i-k+1}^{(1)} \xrightarrow{f_\phi} \cdots \xrightarrow{f_\phi} g_{i-1}^{(k-1)} \xrightarrow{f_\phi} g_i^{(k)}. \tag{12}$$

The chain starts at position $i - k$, from the target's own feature there: teacher forcing makes every position the start of a chain, and sets the start by depth alone rather than by recorded cycle boundaries.

Inference Under Speculative Decoding. At inference the head runs the same recursion and restarts it at one kind of position only: the end of the committed prefix, once per cycle. Cycle r begins drafting at context length m_r, and every position behind this boundary carries a committed token together with its target feature $g^{(0)}$. To draft position $m_r{+}k$, the head applies f_ϕ k times, and the first application consumes the target feature at the boundary before it uses its own recursive features,

$$\begin{aligned} \bar{g}_{r,1} &= f_\phi\big(g_{m_r}^{(0)},\, e(y_{m_r})\,;\, \bar{\mathcal{K}}_{r,1}\big), \\ \bar{g}_{r,k} &= f_\phi\big(\bar{g}_{r,k-1},\, e(d_{r,k-1})\,;\, \bar{\mathcal{K}}_{r,k}\big), \qquad k = 2, \ldots, K. \end{aligned} \tag{13}$$

The first line shows where the recursion begins again: $\bar{g}_{r,k}$, the state at which the head evaluates $h_\phi(\cdot \mid s_{r,k})$ in Eq. 4, is a chain of length k starting at m_r, and the only chain the cycle runs. The tokens this chain consumes are the head's own drafts, which follow the committed response up to the first rejection, the divergence Eq. 3 records. The restart resets the cache the recursion has written along with the chain's start: cycle r closes by committing up to the next boundary m_{r+1}, and the cache the last draft state read stands above the cache the next cycle's first drafting application reads,

$$\begin{aligned} \bar{\mathcal{K}}_{r,K} &= \big\{\ldots,\; c_\phi\big(g_{m_r}^{(0)},\, e(y_{m_r})\big),\; c_\phi\big(\bar{g}_{r,1},\, e(d_{r,1})\big),\; \ldots,\; c_\phi\big(\bar{g}_{r,K-1},\, e(d_{r,K-1})\big)\big\}, \\ \bar{\mathcal{K}}_{r+1,1} &= \big\{\ldots,\; c_\phi\big(g_{m_r}^{(0)},\, e(y_{m_r})\big),\; c_\phi\big(g_{m_r+1}^{(0)},\, e(y_{m_r+1})\big),\; \ldots,\; c_\phi\big(g_{m_{r+1}}^{(0)},\, e(y_{m_{r+1}})\big)\big\}. \end{aligned} \tag{14}$$

Over the accepted range the two rows hold the same tokens, the committed y_{m_r+l} being the draft $d_{r,l}$, so the restart replaces the feature alone: $\bar{g}_{r,l}$ becomes $g_{m_r+l}^{(0)}$. This composition then holds at every depth of every cycle: the entries behind the boundary are computed from the target's features, and the head's recursive features enter the cache only through the current chain's writes. The restart does not depend on rejection: a rollout that accepts every draft still closes its cycle after $K{+}1$ committed tokens, with $m_{r+1} = m_r{+}K{+}1$, and recursion restarts at the boundary to draft the next cycle.

Train–Inference Inconsistency. Take a position i that cycle r both drafted and committed: teacher forcing trains K states there, starts $i-1$ through $i-K$, while inference computes exactly one, reached at depth $k = i-m_r$ by the only chain the cycle runs from its start m_r. Inference never computes the states at the other starts: a mid-cycle position is never a restart, and the chain from an earlier boundary leaves the committed response before reaching i (Eq. 3). The remaining pair shares start and tokens alike, the drafts along the drafted chain being committed tokens here, so the two computation graphs can differ only through their caches. A single boundary entry already separates the two caches:

$$\begin{aligned} &\text{teacher forcing training:} && c_\phi\big(g_{m_r}^{(k-1)},\, e(y_{m_r})\big) \;\in\; \mathcal{K}_i^{(k)} && \text{(Eq. 11)},\\ &\text{speculative decoding inference:} && c_\phi\big(g_{m_r}^{(0)},\, e(y_{m_r})\big) \;\in\; \bar{\mathcal{K}}_{r,k} && \text{(Eq. 14)}. \end{aligned} \tag{15}$$

The same projection of the same token, the difference contracted to a single superscript, $k-1$ against 0: the entries agree only at $k = 1$, where both caches are computed from the target's features end to end, and the matching states across the whole response occur only at each cycle's first draft position:

$$g_i^{(k)} \;=\; \bar{g}_{r,\,i-m_r} \qquad\Longleftrightarrow\qquad k \;=\; i-m_r \;=\; 1. \tag{16}$$

That is one state per cycle, out of the $T \cdot K$ states teacher forcing trains, and the rest update the same ϕ, teaching it computations inference never runs. At depth one every trained state still has the drafted form, one application of f_ϕ to a target feature under a cache computed from the target's features: the two schemes differ only in which starts the rollout realizes. At depth k the trained chain spans k positions and crosses a cycle boundary whenever one falls among them, carrying head-computed features through a position at which inference restarts, and a longer chain straddles a boundary more often: the departure from the states inference runs grows along the depth axis. And this is the very axis a deeper draft must scale: the deeper the head drafts, the further from its inference states teacher forcing trains it, so deeper drafting increases the mismatch between training and inference states.

***GrowMTP* Addresses the Inconsistency.** *GrowMTP* trains the head at the states of Eq. 13 themselves: the update phase restarts the recursion of Eq. 4 at each recorded boundary (Algorithm 1, App. E.1), feeding the recorded drafts back in order, and its batched execution changes the schedule of this computation and nothing that is computed (App. E.3). Every separation drawn above closes: the chain starts only at the recorded boundary feature $g_{m_r}^{(0)}$, not at every position, the tokens it consumes are the recorded drafts themselves, and the cache rebuilt behind the boundary holds entries computed from the target's features, the head's recursion entering only through the chain's own writes—precisely the composition of the inference cache. In Eq. 15 the training-side entry now carries superscript 0, the very entry inference attends, and where Eq. 16 left teacher forcing one coincidence per cycle, all K states of the cycle coincide. No mask over the teacher-forcing forward substitutes for this: a mask selects within what teacher forcing supplies, yet the forward, restarting nowhere, computes no drafted state to keep, and the flat response it consumes carries no cycle for the gate of Eq. 9 to truncate—what produces the drafted states is reconstruction itself. The inconsistency this appendix constructs therefore does not arise under *GrowMTP*: training uses inference states at all depths.

H One-Step Lag Bound under Rejection Sampling

The training loop of Section 3.1 retains a single asymmetry between successive RL steps: the head that drafts at step $t{+}1$ was trained on verification signals recorded at step t, one policy update earlier. The head is therefore always trained against a target one update older than the one it drafts for, and this *one-step lag* may degrade the acceptance rate that training establishes. This appendix characterizes the sensitivity of the acceptance chain to this lag through the target shift in a single policy update.

Setup. Fix policy update $t \to t{+}1$ and conditioning state s, and consider these three distributions:

$$p^{(t)}(\cdot \mid s) = \mathrm{softmax}\big(z_{\theta_t}(s)\big), \quad p^{(t+1)}(\cdot \mid s) = \mathrm{softmax}\big(z_{\theta_{t+1}}(s)\big), \quad q(\cdot \mid s) = h_{\phi_{t+1}}(\cdot \mid s): \tag{17}$$

the old target, from which the head's training signals were recorded, the new target, against which the head's drafts are verified at step $t{+}1$, and the head distribution, which generates those drafts. Eq. 5 assigns the head two acceptance rates at the same conditioning state, one for each of the two targets:

$$\tilde{\alpha}(s) = 1 - \mathrm{TV}\big(p^{(t)}(\cdot \mid s),\, q(\cdot \mid s)\big), \qquad \alpha(s) = 1 - \mathrm{TV}\big(p^{(t+1)}(\cdot \mid s),\, q(\cdot \mid s)\big), \tag{18}$$

the training-time rate $\tilde{\alpha}$, taken against the target the head was trained on, and the deployment rate α, taken against the target that verifies its drafts. The cost of the lag is the difference between the two.

Single-Position Bound. By Eq. 18, the cost is a difference of two distances from q, and the triangle inequality bounds it from above by using the old target distribution as the intermediate distribution:

$$\tilde{\alpha}(s) - \alpha(s) = \mathrm{TV}\big(p^{(t+1)}(\cdot \mid s),\, q(\cdot \mid s)\big) - \mathrm{TV}\big(p^{(t)}(\cdot \mid s),\, q(\cdot \mid s)\big) \le \mathrm{TV}\big(p^{(t+1)}(\cdot \mid s),\, p^{(t)}(\cdot \mid s)\big), \tag{19}$$

which holds for every state s and uniformly in the head distribution q: the deployment rate falls below the training-time rate by at most the single-update displacement of the target. Since TV is a metric, the same argument with the roles of $p^{(t)}$ and $p^{(t+1)}$ exchanged bounds $\alpha(s) - \tilde{\alpha}(s)$ by the same quantity, so the bound controls both increases and decreases in acceptance at the same fixed state.

Per-Cycle Bound. For the per-cycle acceptance-chain value defined in Eq. 6, define the target shift as

$$\bar{\varepsilon}_t \;\triangleq\; \max_{k \le K} \mathbb{E}_{s_k}\Big[\mathrm{TV}\big(p^{(t+1)}(\cdot \mid s_k),\, p^{(t)}(\cdot \mid s_k)\big)\Big] \;=\; \frac{1}{2} \max_{k \le K} \mathbb{E}_{s_k}\, \mathbb{E}_{y \sim p^{(t)}} \big|\, \rho(y) - 1 \,\big|, \tag{20}$$

where $\mathbb{E}_{s_k}$ averages over the states conditioning draft position k in the cycles of the step-$t{+}1$ rollout and $\rho(y) = p^{(t+1)}(y \mid s_k) \,/\, p^{(t)}(y \mid s_k)$ is the token importance ratio between successive policies.

$$\mathrm{TV}(p, q) \;=\; \frac{1}{2} \sum_y p(y) \left| \frac{q(y)}{p(y)} - 1 \right|, \tag{21}$$

valid here since softmax distributions have full support. The first form is used in the derivation, and the second expresses $\bar{\varepsilon}_t$ through the importance ratio and underlies its estimation below. Fix a cycle of the step-$t{+}1$ rollout with conditioning states $s_1, \ldots, s_K$, and write $\alpha_i = \alpha(s_i)$, $\tilde{\alpha}_i = \tilde{\alpha}(s_i)$, and $\delta_i = \mathrm{TV}\big(p^{(t+1)}(\cdot \mid s_i),\, p^{(t)}(\cdot \mid s_i)\big)$. Since every factor lies in $[0, 1]$ and $\tilde{\alpha}_n - \alpha_n \le \delta_n$ by Eq. 19, the difference between products can be expanded as the following telescoping sum for each $l \le K$:

$$\prod_{i=1}^{l} \tilde{\alpha}_i \;-\; \prod_{i=1}^{l} \alpha_i \;=\; \sum_{n=1}^{l} \Big(\prod_{i<n} \tilde{\alpha}_i\Big)(\tilde{\alpha}_n - \alpha_n)\Big(\prod_{n<i\le l} \alpha_i\Big) \;\le\; \sum_{n=1}^{l} \delta_n. \tag{22}$$

Summing over $l = 1, \ldots, K$, taking expectations over the cycles of the rollout, and bounding each $\mathbb{E}[\delta_n]$ by $\bar{\varepsilon}_t$, the shifts are counted $\sum_{l=1}^{K} l = K(K+1)/2$ times across the chain terms, yielding

$$\mathbb{E}\left[\sum_{l=1}^{K} \prod_{i=1}^{l} \tilde{\alpha}_i\right] \;-\; \mathbb{E}\left[\sum_{l=1}^{K} \prod_{i=1}^{l} \alpha_i\right] \;\le\; \frac{K(K+1)}{2}\, \bar{\varepsilon}_t. \tag{23}$$

The second expectation is the expected acceptance-chain value of Eq. 6 at step $t{+}1$, and the first is the same quantity measured against the training-time target: it is the acceptance-chain quantity underlying the DCA loss, with both targets evaluated at the same draft states from the new rollout.

Magnitude of $\bar{\varepsilon}_t$. The shift is a property of the RL update alone, and its leading-order magnitude follows from two classical facts. First, by Eq. 17 the two targets are softmax outputs of the same network at neighboring parameter points, so their KL divergence admits the following second-order expansion in the parameter update $\Delta\theta = \theta_{t+1} - \theta_t$, evaluated at the same conditioning state s:

$$\mathrm{KL}\Big(p^{(t)}(\cdot \mid s) \,\Big\|\, p^{(t+1)}(\cdot \mid s)\Big) \;=\; \frac{1}{2}\,\Delta\theta^\top F(s)\,\Delta\theta \;+\; O\big(\|\Delta\theta\|^3\big), \tag{24}$$

where $F(s)$ is the Fisher information matrix of the target model at state s, the Hessian of the divergence at zero displacement. Second, Pinsker's inequality $\mathrm{TV} \le \sqrt{\mathrm{KL}/2}$, Jensen's inequality, and the maximization over draft positions in Eq. 20 convert Eq. 24 into this bound on target shift:

$$\bar{\varepsilon}_t \;\le\; \frac{1}{2}\sqrt{\Delta\theta^\top \bar{F}\,\Delta\theta \;+\; O\big(\|\Delta\theta\|^3\big)}, \tag{25}$$

where $\bar{F}$ is the Fisher matrix averaged over the conditioning states at the maximizing position: to leading order, the shift is at most half the length of the update measured in the Fisher metric, and vanishes linearly with the learning rate. The clipped surrogate, small learning rate, and gradient-norm clipping aim to limit policy movement but do not determine $\bar{\varepsilon}_t$, the per-update target-policy shift.

Measuring $\bar{\varepsilon}_t$. The ratio form of Eq. 20 makes the shift directly measurable: rollout tokens are sampled from $p^{(t)}$, so the sample mean of $\frac{1}{2}|\rho - 1|$ over a training batch estimates the inner expectation unbiasedly, at the cost of one additional scoring pass of the batch under the updated policy. In our runs, the measured per-token $\mathrm{KL}\big(p^{(t)} \,\|\, p_{\mathrm{ref}}\big)$ remains below 0.007 nats throughout training. This fixed-reference KL provides contextual evidence that the policy trajectory remains localized, although it does not directly estimate the consecutive-policy shift $\bar{\varepsilon}_t$ in the acceptance-chain bound of Eq. 23.

Conclusion. Eq. 23 shows that the acceptance-chain shift is controlled by the consecutive-policy shift $\bar{\varepsilon}_t$, with a worst-case bound of $\frac{K(K+1)}{2}\bar{\varepsilon}_t$. Because the head is refreshed after every RL step, the bound depends only on the current policy update rather than accumulated training drift; the sustained acceptance gains in our runs show that this bounded staleness does not prevent effective online adaptation. Under rejection sampling, Eq. 5 makes the acceptance rate 1-Lipschitz in the target distribution under total variation, with both the draft distribution and the conditioning state held fixed.

I Optimization Analysis of Training Objective

The most direct way to improve rollout efficiency is to train the MTP head with a differentiable surrogate for the acceptance length, the TV-chain objective $\mathcal{L}_{\mathrm{TV}}$ of Section 3.2, which prior work adopts. This surrogate is multiplicative in the per-position acceptance probabilities, so its value on a draft step measures how well the head already drafts there, and a direct design carries that value into the gradient. This appendix compares TV and DCA gradients (Eq. 7) for individual draft steps and batches under SGD and Adam, characterizing how the logarithm changes relative cycle weights.

Setup. The head enters either objective only through the acceptance probabilities of one draft-then-verify cycle, which compound into the per-cycle acceptance-chain surrogate specified in Eq. 6,

$$M \;=\; \sum_{l=1}^{K} \prod_{i=1}^{l} \alpha_i, \qquad \mathcal{L}_{\mathrm{TV}} \;\triangleq\; -M, \qquad \mathcal{L}_{\mathrm{DCA}} \;\triangleq\; -\log M, \tag{26}$$

where $\mathcal{L}_{\mathrm{TV}}$ differs by an affine constant from the normalized form $1 - M/K$ in which prior work writes it. Since softmax distributions have full support, $\alpha_k \in (0, 1]$ and $M \in (0, K]$, the lower end approached by a head whose drafts are rejected at the first position and the upper end attained by one whose drafts are always accepted. The verify-gated form of Eq. 9 replaces K by $\min(j, K)$ in the chain sum, which changes the value of M and the upper end of this range and leaves the statements below unaffected. Both objectives are built from the same acceptance probabilities of the same cycle, so the logarithm of the shared chain value accounts for every difference between the two objectives.

Gradient Analysis. To compare the gradients of the two objectives, differentiate each of them with respect to the individual acceptance probabilities. The recorded target distributions enter both objectives as constants, and the gradients below are taken with respect to the head parameters ϕ alone. The derivative of M at a single position follows from its multilinearity in the acceptance probabilities: α_k occurs once in every term whose index is at least k and in no other term, and factoring it out separates the product over all preceding positions from the sum of products over subsequent positions:

$$\frac{\partial M}{\partial \alpha_k} \;=\; \sum_{l=k}^{K} \prod_{i \le l,\, i \neq k} \alpha_i \;=\; \Big(\prod_{i<k} \alpha_i\Big) \sum_{l=k}^{K} \prod_{i=k+1}^{l} \alpha_i. \tag{27}$$

The remaining sum is the chain quantity of Eq. 26 formed over the positions after k, a sum of $K - k + 1$ terms, with the first term equal to one and each remaining term between zero and one:

$$D_k \;\triangleq\; \sum_{l=k}^{K} \prod_{i=k+1}^{l} \alpha_i \;\in\; [1,\, K - k + 1]. \tag{28}$$

Since $\mathcal{L}_{\mathrm{TV}} = -M$ and $\mathcal{L}_{\mathrm{DCA}} = -\log M$, the coefficients the two objectives place on position k are $\partial M/\partial \alpha_k$ and $M^{-1}\, \partial M/\partial \alpha_k$, respectively. The derivative of M yields the following coefficients:

$$-\frac{\partial \mathcal{L}_{\mathrm{TV}}}{\partial \alpha_k} \;=\; \Big(\prod_{i<k} \alpha_i\Big) D_k, \qquad -\frac{\partial \mathcal{L}_{\mathrm{DCA}}}{\partial \alpha_k} \;=\; \Big(\prod_{i<k} \alpha_i\Big) \frac{D_k}{M}, \tag{29}$$

where D_k/M matches the coefficient of Eq. 8, its denominator shared across positions and its numerator at most $K-k+1$, which bounds its variation across positions by a factor of K. Assembling the positions into the gradient with respect to ϕ places the common positive factor outside the sum,

$$\nabla_\phi \mathcal{L}_{\mathrm{DCA}} \;=\; -\frac{1}{M} \sum_{k=1}^{K} \frac{\partial M}{\partial \alpha_k}\, \nabla_\phi \alpha_k \;=\; -\frac{1}{M}\, \nabla_\phi M \;=\; \frac{1}{M}\, \nabla_\phi \mathcal{L}_{\mathrm{TV}}. \tag{30}$$

The factor is positive at every parameter value and carries no position index, so on this cycle the two objectives share their descent direction, their stationary points, their minimizers and the relative weights they place on the draft positions, the depth weighting of Section 3.3 among them. The logarithm only rescales that cycle's gradient magnitude by the factor $1/M$, common to all positions.

Optimization on a Draft Step. How much of the factor M^{-1} reaches the parameters depends on the optimizer. Under SGD with learning rate η, Eq. 30 gives a draft step with chain value M the update $-\eta\,\nabla_\phi \mathcal{L}_{\mathrm{DCA}}$ under $\mathcal{L}_{\mathrm{DCA}}$ and the update $-\eta M\,\nabla_\phi \mathcal{L}_{\mathrm{DCA}}$ under $\mathcal{L}_{\mathrm{TV}}$. The rate at which the shared direction acts on the parameters is therefore η under $\mathcal{L}_{\mathrm{DCA}}$ and $\eta M \in (0, \eta K]$ under $\mathcal{L}_{\mathrm{TV}}$, which makes the effective rate of $\mathcal{L}_{\mathrm{TV}}$ a function of the head's own acceptance on that draft step and leaves the effective rate of $\mathcal{L}_{\mathrm{DCA}}$ at the scheduled learning rate, without extra scaling by the value M.

Adam divides the first moment of the gradient history by the square root of the second moment, and a factor common to every gradient in that history scales the first moment and the square root of the second moment alike, so the factor cancels in the quotient up to the constant that stabilizes the denominator. Ignoring Adam's stabilizing constant, a positive constant scaling of the entire gradient history cancels in the moment ratio; factors varying across draft cycles generally do not cancel.

Optimization on a Batch. The optimizer does not receive the gradient of a single draft step: what reaches it is the batch average, in which M^{-1} is no longer a scalar but a weight carried by each draft step separately. Writing $g_r \triangleq \nabla_\phi[-\log M_r]$ for the gradient $\mathcal{L}_{\mathrm{DCA}}$ produces on draft step r, Eq. 30 makes the gradient of $\mathcal{L}_{\mathrm{TV}}$ on the same draft step $M_r g_r$, giving the following two batch gradients:

$$\nabla_\phi\, \mathbb{E}_r\big[\mathcal{L}_{\mathrm{TV}}\big] \;=\; \mathbb{E}_r\big[M_r\, g_r\big], \qquad \nabla_\phi\, \mathbb{E}_r\big[\mathcal{L}_{\mathrm{DCA}}\big] \;=\; \mathbb{E}_r\big[g_r\big], \tag{31}$$

where $\mathbb{E}_r$ averages over the recorded draft steps that make up the batch. Relative to the DCA cycle gradients g_r, the TV-chain objective assigns the explicit weights M_r; DCA averages g_r without these factors. The norms of g_r can still differ across cycles. Consequently, the stationary conditions $\mathbb{E}_r[M_r g_r] = 0$ and $\mathbb{E}_r[g_r] = 0$ need not coincide. The logarithm thus changes the batch objective, favoring smaller-M_r cycles in relative gradient weight without guaranteeing faster convergence.

Conclusion. The logarithm preserves the single-cycle gradient direction and relative depth weights while changing the batch objective through cycle reweighting. The experiments assess whether this change in relative cycle weights improves realized acceptance and reduces overall RL training time.

J LIMITATIONS

Run length and compute allocation. Net savings depend on the run length and the rollout share of total compute. In both from-scratch experiments, per-step acceleration begins within 30 steps, and the 500-step runs recover the initial cost with substantial net savings. Typical RL training spans hundreds or thousands of steps, providing ample time to amortize this brief initial training overhead.

Distribution specialization. The grown head is tailored to the rollout distribution of the current RL run. As shown in **Appendix D.3**, it retains higher acceptance within its training domain than across domains. Its reuse across unrelated tasks therefore requires separate evaluation, while broad deployment remains a distinct setting from the acceleration studied within a single ongoing RL run.

Evaluation scope. Our measurements cover 4B–7B models on mathematical and code reasoning with a single node of eight GPUs. Larger models, multi-node systems, and asynchronous RL may have different compute and communication costs, so their speedups require direct measurement. The projections in **Appendix D.2** illustrate scaling under assumptions, rather than measured performance.