MULTI-DRAFT ON-POLICY DISTILLATION FOR DIFFUSION DRAFTERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding is used to reduce the inference cost of language models. One-step diffusion drafters propose a block of multiple tokens in parallel, reducing the sequential cost of speculative decoding. On-policy distillation trains these drafters using verifier feedback on their own proposals. However, parallel drafting makes this feedback difficult to use: the drafter predicts each token without observing the preceding tokens sampled within the same block, while the verifier conditions its prediction on those tokens. Different sampled prefixes therefore give different verifier targets for the same drafter prediction. We show that distilling against these targets separately can suppress tokens that are plausible after some prefixes but unlikely after others, harming draft acceptance. We explain the effects of this mismatch and introduce Multi-Draft On-Policy Distillation (MDraft), a new parallel drafter distillation method which samples several draft blocks from the same context and aggregates the verifier's full-vocabulary conditionals across their prefixes. We use all the drafts to collect information from the verifier, allowing support from alternative prefixes to influence the training target. The resulting algorithm improves the acceptance length of a parallel drafter beyond the level of pre-training, and alleviates the effects of the distributional mismatch. MDraft changes only training, retaining the drafter's architecture and inference procedure, allowing the resulting drafter to be used as-is in place of the pretrained checkpoint. Experiments on a suite of mathematical, coding, and conversational tasks show longer accepted drafts than prior distillation methods.

1 INTRODUCTION

Generating text with a large language model requires a sequence of expensive forward passes, one for each output token. Speculative decoding reduces this cost by allowing a smaller drafter model to propose several tokens and verifying them together with the target model (Leviathan et al., 2023; Chen et al., 2023). The verification procedure preserves the LLM's output distribution, while its efficiency depends on how many proposals are accepted relative to the cost of producing them. One-step diffusion drafters offer an attractive trade-off: they generate an entire block of tokens in parallel, avoiding the sequential forward passes required by an autoregressive drafter (Christopher et al., 2025; Chen et al., 2026).

Training such drafters typically begins with supervision from target-generated continuations (Chen et al., 2026; Li et al., 2025b). Works in the LLM domain show that on-policy training helps further improve models by fine-tuning them on self-generated data (Gu et al., 2024; Agarwal et al., 2024), allowing the student model to be trained on the states that it actually encounters in inference. Recent works in parallel drafter training adopt this philosophy by generating proposal blocks with the drafter model itself and evaluating the verifier LLM on these drafts. Draft-OPD (Lei et al., 2026) shows that training on drafter-generated blocks can improve drafter performance even when off-policy training saturates.

Parallel drafters, however, carry a structural difference compared to autoregressive models: the tokens generated in different positions of the drafted block are only conditioned on the context, not on the previous drafted tokens, while the LLM, which is used to supply the targets for the training objective, processes this block autoregressively. Consequently, even plausible sampled prefixes can give conflicting verifier targets for the same drafter prediction: a token that is useful after one prefix

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

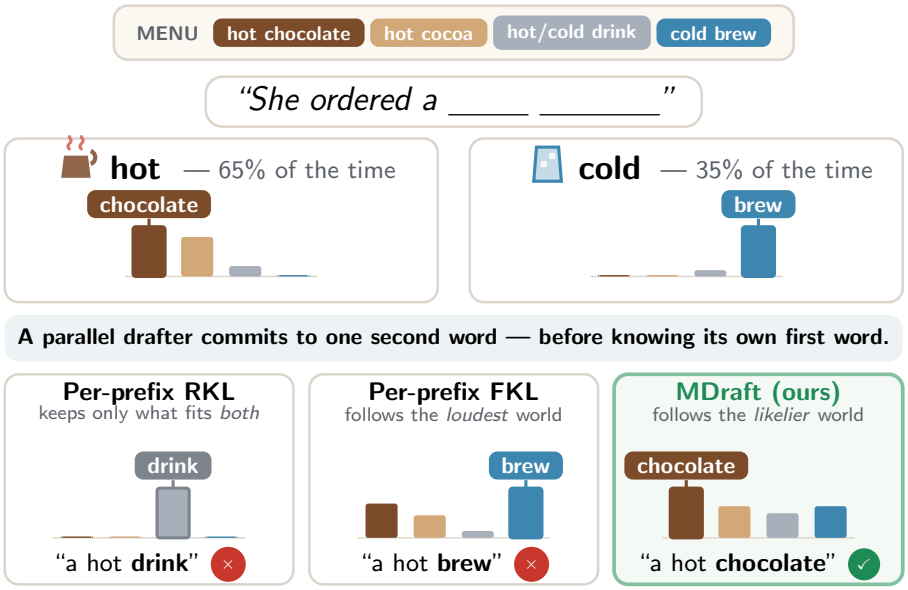

Figure 1: A parallel drafter predicts the second token without observing its sampled first token. The verifier's preferred continuation depends on that first token; different training objectives therefore yield different second-token choices. The verifier's greedy continuation is "hot chocolate." Bars are schematic; Table 2 gives the numerical distributions.

may receive little support after another. Distilling these targets separately can therefore suppress tokens that would be useful in other drafts.

In this work, we show that this problem has real consequences for parallel drafter training and design a new on-policy drafter distillation objective, MDraft, that aggregates verifier scores across multiple drafts to build a reliable target for the drafter distribution. We conduct an empirical evaluation by fine-tuning parallel drafters and evaluating their acceptance rates on multiple benchmarks.

Our main contributions are:

- **Identifying the target mismatch in parallel drafter distillation.** We show that parallel drafters introduce a unique challenge for on-policy distillation, and using verifier signals on drafter continuations as targets can favor tokens that are harmful to draft acceptance.
- **Developing Multi-Draft On-Policy Distillation.** We introduce Multi-Draft On-Policy Distillation (MDraft), which combines verifier distributions on multiple drafts into a single target to address the mismatch.
- **Improving on-policy distillation of parallel drafters.** MDraft achieves the best overall mean acceptance lengths among current on-policy distillation methods for parallel drafters, evaluated across a suite of mathematical, coding, and conversational benchmarks.

2 Background

In this section, we introduce the main concepts and notation used in our paper. In Section 2.1, we discuss the mechanics of LLM sampling, and why speculative decoding is used. We then introduce parallel drafters for speculative decoding. We then explain how they are trained in Section 2.2 and introduce on-policy training in Section 2.3. Appendix A provides a broader discussion of related work.

2.1 Language model sampling

An autoregressive LLM generates each token from a probability distribution conditioned on the prompt and all previously generated tokens. Temperature controls the sharpness of this distribution, with greedy decoding selecting the most probable token. Standard decoding requires a separate model forward pass for each new token, making generation sequential.

Speculative decoding. To reduce the number of expensive LLM passes, speculative decoding uses a cheaper drafter model to propose several tokens. In the simplest setup, the drafter is a smaller autoregressive language model that generates several tokens sequentially. The target LLM evaluates all these proposed tokens in one forward pass, using causal attention to compute the appropriate next-token distribution at each position. These tokens are either accepted or rejected, based on a special rule that ensures that the resulting sequence distribution matches that of the original LLM (Leviathan et al., 2023; Chen et al., 2023). Speedups arise when the cost of drafting and joint verification is lower than that of generating the committed tokens directly with the large model. For a fixed drafter and target architecture, the speedup becomes a function of the average *acceptance length*: the average number of proposed tokens that the verifier model accepts in one pass.

Parallel drafting. To further reduce the cost of speculative decoding, a *parallel drafter* can be used. A parallel drafter predicts multiple future tokens at once. This removes the sequential passes of an autoregressive drafter, making longer proposals inexpensive to generate. The verifier can then check the block using the same speculative decoding procedure.

Notation and conventions. When discussing parallel drafting, we will denote the length of the drafted block as L. The parallel drafter model is denoted q, and for the categorical distribution it induces over the ℓ-th token of the block we write $q_\ell(\cdot \mid c)$, where c is the context that includes the prompt and previously accepted tokens. The current draft is denoted as x. The LLM model is p, also called the verifier, and the distribution is autoregressive: $p_\ell(\cdot \mid c, x_{<\ell})$. The vocabulary is denoted $\mathcal{V}$.

2.2 Speculative drafter training

Autoregressive drafters can be trained explicitly to match the target LLM. These methods increase draft acceptance lengths by aligning the distributions of the drafter model with the target LLM. Token-level supervision minimizes cross-entropy, $\mathcal{L}_{\mathrm{CE}} = -\sum_\ell \log q(x_\ell \mid c, x_{<\ell})$, on reference continuations x. Distributional distillation instead matches the target LLM's full next-token predictions: for a training prefix $h = (c, x_{<\ell})$, a common objective is $\mathrm{KL}(p(\cdot \mid h) \| q(\cdot \mid h))$. DistillSpec studies how the choice of divergence and training sequences affects acceptance, including sequences generated by the drafter itself (Zhou et al., 2024).

Parallel drafter training. Similar loss functions can be used to train parallel drafters: DFlash is trained on responses generated by the verifier LLM with a cross-entropy loss: $\mathcal{L}_{\mathrm{DFlash}} = -\sum_{\ell=1}^{L} w_\ell \log q_\ell(x_\ell \mid c)$, where x_ℓ is the reference token generated by the LLM and the per-position weights are $w_\ell = \exp(-(\ell-1)/\gamma)$ for decay scale $\gamma > 0$. These weights emphasize the importance of early predictions because early rejection prevents later proposals from being used (Chen et al., 2026). DSpark tries to re-introduce dependency between tokens in the draft by augmenting the parallel backbone with a lightweight sequential head and using a confidence head to schedule the verification length (Cheng et al., 2026). It combines the same position-weighted cross-entropy used in DFlash with distribution matching and confidence supervision: $\mathcal{L}_{\mathrm{DSpark}} = 0.1\mathcal{L}_{\mathrm{DFlash}} + 0.9\mathcal{L}_{\mathrm{TV}} + \mathcal{L}_{\mathrm{conf}}$. Here, $\mathcal{L}_{\mathrm{TV}} = \sum_\ell w_\ell \|q_\ell - p_\ell\|_1$ is a surrogate for the expected token acceptance, with drafter q_ℓ and LLM p_ℓ evaluated on LLM-generated prefixes. The confidence loss is $\mathcal{L}_{\mathrm{conf}} = \sum_\ell w_\ell \,\mathrm{BCE}(s_\ell, a_\ell^\star)$, where s_ℓ is the predicted acceptance probability, $a_\ell^\star = 1 - \frac{1}{2}\|q_\ell - p_\ell\|_1$ is its soft target, and BCE is the binary cross-entropy loss.

2.3 On-policy training

Drafter training is usually performed by computing drafter distributions over a corpus of data generated by the verifier LLM. However, inspired by the success of on-policy knowledge distillation in LLMs (Agarwal et al., 2024), recent works develop methods for on-policy drafter training.

LLM on-policy distillation trains an autoregressive student LLM on its own continuations, using teacher feedback on sequences generated by the student. For a prompt c and previously generated tokens y, let $p(\cdot \mid c)$ and $q(\cdot \mid c)$ denote teacher and student sequence distributions, and p_ℓ, q_ℓ their next-token distributions given $(c, y_{<\ell})$. The fully on-policy variant of generalized knowledge distillation (GKD) uses $\mathcal{L}_{\mathrm{GKD}} = \mathbb{E}_{y\sim q}[|y|^{-1} \sum_{\ell=1}^{|y|} \mathrm{KL}(q_\ell \| p_\ell)]$, averaged over prompts (Agarwal et al., 2024). MiniLLM instead targets sequence-level reverse KL, $\mathcal{L}_{\mathrm{MiniLLM}} = \mathbb{E}_{y\sim q}[\log q(y \mid c) - \log p(y \mid c)]$, again averaged over prompts (Gu et al., 2024). Both approaches obtain supervision on student-generated states, an idea subsequently adapted to speculative drafters.

On-policy drafter training. A drafter cannot by itself generate full sequences. Hence, on-policy training algorithms for drafters use the following protocol: first, a corpus of prompts is selected, and responses are generated by the verifier LLM. Then, a random position in the response is sampled and a proposal is generated at that position using the drafter model. The verifier LLM's distribution is then computed on this drafted block, and a distribution matching loss is calculated.

Draft-OPD combines two losses: $\mathcal{L}_{\text{Draft-OPD}} = \frac{1}{2}(\mathcal{L}_{\text{acc}} + \mathcal{L}_{\text{rej}})$ (Lei et al., 2026). Writing $\mathcal{A}$ and $\mathcal{R}$ for accepted and rejected draft positions, $\mathcal{L}_{\text{acc}} = |\mathcal{A}|^{-1} \sum_{\ell \in \mathcal{A}} \text{KL}(p_\ell \| q_\ell)$ matches the verifier on accepted states, while $\mathcal{L}_{\text{rej}} = \sum_{\ell \in \mathcal{R}} u_\ell \text{KL}(q_\ell \| p_\ell)$ suppresses unsupported draft predictions. The authors argue that using a combination of forward-KL and reverse-KL for the accepted and rejected tokens is useful, exploiting the mode-covering and mode-seeking properties of these divergences. The weights u_ℓ decay exponentially with the position within the block. AdaFlash uses $\mathcal{L}_{\text{AdaFlash}} = \lambda \mathcal{L}_{\text{hard}} + (1 - \lambda)\mathcal{L}_{\text{RKL}}^{\text{clip}}$ (Qian et al., 2026). Here, $\mathcal{L}_{\text{hard}} = -L^{-1} \sum_\ell \log q_\ell(y_\ell^\star)$, with $y_\ell^\star = \arg\max_y p_\ell(y)$, encourages agreement with the verifier's preferred token. The distributional term is $\mathcal{L}_{\text{RKL}}^{\text{clip}} = L^{-1} \sum_{\ell,y} \min\{q_\ell(y) \log[q_\ell(y)/p_\ell(y)], \delta\}$: clipping each vocabulary-entry contribution at $\delta > 0$ limits outlier gradients. The hard-loss component encourages agreement with the verifier's argmax, while the reverse-KL component penalizes excess draft probability on tokens that receive little verifier support.

3 MDRAFT: MULTI-DRAFT ON-POLICY DISTILLATION

3.1 ON-POLICY OBJECTIVE FOR DRAFTING

A direct adaptation of LLM distillation objectives to parallel drafters is to sample a block from the current drafter, evaluate the verifier on that block, and minimize a token-level divergence such as forward or reverse KL. Writing $p_\ell = p_\ell(\cdot \mid c, x_{<\ell})$, the per-position reverse KL objective is $\mathbb{E}_{x \sim q(\cdot|c)}[\text{KL}(q_\ell \| p_\ell)]$.

However, the distributions being compared have different conditioning information. In autoregressive LLM distillation, both student and teacher predict the next token given the same generated prefix. A parallel drafter instead computes all L distributions $q_\ell(\cdot \mid c)$ in one forward pass and samples tokens at all positions independently, giving the factorized block distribution $q(x_{1:L} \mid c) = \prod_{i=1}^{L} q_i(x_i \mid c)$. Its sampled prefixes therefore follow $q_{<\ell}(x_{<\ell} \mid c) = \prod_{i<\ell} q_i(x_i \mid c)$. The prediction $q_\ell(\cdot \mid c)$ is shared across all such prefixes, whereas the verifier conditions on the particular realization $x_{<\ell}$. The drafter must therefore fit one distribution to several potentially incompatible verifier conditionals. Even individually plausible draft tokens can form an incoherent prefix, making the following tokens unlikely under autoregressive law. Therefore, the verifier's feedback is correct for that prefix, but need not describe which tokens are useful at that position across other possible drafts.

A per-prefix divergence penalizes both tokens that are completely unsuitable at a position and tokens that are useful there but incompatible with the particular prefix sampled in this draft. Because the parallel drafter cannot condition its prediction on that prefix, suppressing the latter tokens also reduces their probability in drafts where they would fit. On-policy sampling exposes this conflict directly.

Distribution mismatch in action. To illustrate the problem, consider the example in Figure 1, assuming every word is represented by a token. Take the sentence "She ordered a," followed by either *hot* or *cold*. In our illustrative example, *hot* is more likely, and the verifier then prefers *chocolate* while also supporting *cocoa*. After *cold*, it strongly prefers the word *brew*. The generic word *drink* has support after either prefix but is preferred after neither. A parallel drafter predicts both positions together: its second-word distribution cannot change after the first word is sampled. Thus, *chocolate* can receive a strong negative training signal after a sampled *cold*, even though it is the preferred continuation after *hot*. Simultaneously, *brew* will receive a strong positive signal, since it is the preferred token after *cold*, even though the verifier prefers *hot* as the first continuation token.

Per-token divergences, such as commonly used forward- and reverse-KL, fall into these traps: reverse KL strongly discounts tokens with low probability under one of the continuations, preferring the "safety" token *drink*, and the forward KL will favor *brew* in the second position, even though the verifier prefers *hot* in the first. Figure 1 illustrates these choices. We provide the exact calculation in Table 2 and the distribution-level explanation in Section 4.4.

3.2 THE MDRAFT OBJECTIVE

Verifying tokens across multiple drafts. A single draft only tells us how the verifier evaluates a token after one particular prefix. To learn whether a low score is specific to that prefix, MDraft samples K independent draft blocks from the same context and evaluates all of them with the verifier. At each position, these blocks provide different sampled prefixes under which the verifier can assess the same candidate token. For example, a token penalized after a draft containing *cold* may receive strong support after another containing *hot*. We collect this evidence by averaging the verifier's token probabilities across the K drafts, then use it to inform each draft's training target.

Prefix-averaged token distribution. Concretely, one drafter forward pass on context c produces the position-wise distributions used to sample $x_{1:L}^{(k)} \sim q(\cdot \mid c)$ independently for $k = 1, \ldots, K$. Evaluating the verifier on draft k gives $p_\ell^{(k)}(y) = p(y \mid c, x_{<\ell}^{(k)})$, the probability of candidate token y at position ℓ given the k-th draft's prefix. We average these distributions over the K prefixes as $\widehat{p}_\ell(y) = K^{-1} \sum_{k=1}^{K} p_\ell^{(k)}(y)$.

The average measures how much support a token receives across the sampled prefixes. A token with low probability under one prefix can get support from others, raising the average. Conversely, if every sampled prefix assigns it a low probability of at most ε, its average probability is also at most ε. *Thus, averaging can preserve useful alternatives without raising the scores of tokens that are implausible across all sampled drafts*.

Distillation target. We use the average $\widehat{p}_\ell$ to bring this evidence from other drafts into each proposal's distillation target. The average captures support across prefixes, while the individual conditional $p_\ell^{(k)}$ retains information about compatibility with the sampled prefix. We blend the two so that support from alternative continuations can temper a low score under one prefix without discarding prefix-specific information entirely. The coefficient $\alpha \in [0, 1]$ controls their balance through interpolation in log-probability space, making the target distribution:

$$\widetilde{r}_\ell^{(k)}(y) = p_\ell^{(k)}(y)^{1-\alpha} \widehat{p}_\ell(y)^{\alpha}, \qquad r_\ell^{(k)}(y) = \frac{\widetilde{r}_\ell^{(k)}(y)}{Z_{\ell,k}}, \quad Z_{\ell,k} = \sum_{z \in \mathcal{V}} \widetilde{r}_\ell^{(k)}(z). \tag{1}$$

Here $Z_{\ell,k}$ is the normalizing constant. At $\alpha = 0$, $r_\ell^{(k)} = p_\ell^{(k)}$, recovering the naive per-prefix target; at $\alpha = 1$, every proposal shares $r_\ell^{(k)} = \widehat{p}_\ell$.

We fit the drafter to this target using reverse KL. For a context c, the training objective is

$$\mathcal{L}_{\mathrm{MDraft}}(c) = \frac{1}{L \cdot K} \sum_{k=1}^{K} \sum_{\ell=1}^{L} w_{\ell,k} \, \mathrm{KL}(q_\ell \| r_\ell^{(k)}). \tag{2}$$

We minimize its batch average, holding sampled prefixes, targets, and nonnegative position weights fixed during differentiation. Appendix D discusses weighting options $w_{\ell,k}$.

MDraft retains the standard per-token reverse-KL distillation loss and changes the target it matches. Each target blends the verifier distribution for one draft with the average across multiple drafts, incorporating support from alternative prefixes into one distillation objective.

Revisiting the two-branch example. We can observe the benefit of MDraft's objective on the example from Figure 1. Let the verifier and first-position drafter both assign probability 0.65 to *hot* and 0.35 to *cold*. Per-prefix RKL favors *drink*, which has support on both branches, while FKL favors *brew* because its average probability exceeds that of *chocolate*. At $\alpha = 0.9$, MDraft preserves *chocolate*'s support from the more likely *hot* branch while discounting *brew* for being implausible on that branch. It therefore recovers the verifier's greedy second token and also achieves higher second-token acceptance at $T = 1$ than either KL baseline. Table 2 in Appendix B.1 provides the exact setup, optimal distributions, and acceptance values.

3.3 TRAINING ALGORITHM AND COMPUTATIONAL COST

We present the training procedure of MDraft in algorithm 1. We use one drafter forward pass per context and K verifier evaluations, which can be batched into one pass of the verifier model. Draft

blocks are sampled at training temperature 1; verifier targets use full-support probabilities. Evaluation temperature is specified independently.

Algorithm 1 MDraft: one training step

Input: drafter q, frozen verifier p, a batch of contexts; block length L, proposal count K, blending coefficient α, learning rate η

1: **for** each context c in the batch **do**
2: Run the drafter once to predict distributions for L next tokens.
3: Sample K independent draft blocks from these distributions at temperature 1.
4: Run the verifier on each block to score every position given its sampled prefix.
5: Average the K verifier distributions at each position to obtain $\widehat{p}_\ell$.
6: Blend each verifier distribution with $\widehat{p}_\ell$ to form the target $r_\ell^{(k)}$ (equation 1).
7: Compute position weights for each draft (Appendix D).
8: Compute the weighted distillation loss (equation 2).
9: **end for**
10: Average the losses over the batch and update the drafter at learning rate η.

Output: Updated drafter q.

Acceptance-aware position weights. D-PACE (Wu et al., 2026) suggests using position weights that decay at a rate tied to acceptance probability. An advantage of on-policy training is that verifying the drafter's own proposals provides direct acceptance feedback: we can compute the exact conditional acceptance probability of each sampled token under the verification rule. Let $a_{m,k} = \min\{1, p_m^{(k)}(x_m^{(k)})/q_m(x_m^{(k)} \mid c)\}$ denote the probability of accepting the sampled token at position m in proposal k, conditional on all preceding tokens being accepted. This motivates detached survival weights $w_{\ell,k} \propto \prod_{m<\ell} a_{m,k}$, normalized to sum to L within each proposal. This provides a decreasing position weight that is grounded in real acceptance probability, rather than an arbitrary schedule or approximation. We use survival weighting for MDraft and isolate its effect in Table 3. Appendix D gives more details.

4 Experiments

We first evaluate MDraft across benchmarks and draft positions, then study the number of proposals used during training. We conclude with a theoretical analysis of how target blending changes the distillation objective (Section 4.4).

Metrics and evaluation. We report mean *acceptance length* τ: the number of accepted draft tokens plus one correction (bonus) token per verifier pass, averaged first within each prompt and then across prompts. Higher values mean that each verifier pass produces more output tokens. All methods use the DFlash architecture, and the inference algorithm is unchanged. With the draft block length and inference configuration held fixed within each comparison, accepted draft length captures the efficiency gains from improved drafting; we therefore do not report speedup separately. We evaluate mathematics on GSM8K (Cobbe et al., 2021), MATH500 (Lightman et al., 2024), and AIME25 (Mathematical Association of America, 2025); coding on MBPP (Austin et al., 2021) and HumanEval (Chen et al., 2021); and conversational generation on MT-Bench (Zheng et al., 2023). The reported average across benchmarks is an unweighted mean of the six benchmark-level measurements. All methods within a comparison use the same evaluation prompts and decoding settings.

4.1 Main results

Models and methods. We fine-tune a pretrained DFlash drafter with Qwen3-4B (Yang et al., 2025) as the verifier model. We implement the AdaFlash and Draft-OPD losses, per-prefix RKL and FKL, and MDraft. All fine-tuned variants start from the same checkpoint and use the same architecture and block length, with no method-specific inference changes. Table 1 reports acceptance lengths at evaluation temperatures $T \in \{0, 0.6, 1\}$.

Training data and optimization. We sample prompts from Nemotron-Post-Training-Dataset-v2 (Nathawani et al., 2025) and sample 4 continuations from the verifier model for each prompt. We then truncate the sequence to a maximum length of 1024 tokens. We choose a blend of prompts from the

math, code and chat sections of the dataset. Specifically, our set contains 667,192 teacher-generated sequences (61.52% math, 19.24% code, and 19.24% chat). We sample one draft start position per sequence per epoch and train with AdamW for three epochs, using a global batch size of 32 and a peak learning rate 10^{-5}. MDraft uses $\alpha = 0.9$ and $K = 8$.

Evaluation. We evaluate the drafter on six benchmarks at different sampling temperatures $T \in \{0, 0.6, 1\}$ with thinking mode disabled and a 2,048-token generation cap. For every fine-tuned method, we evaluate the final checkpoint. Appendix E gives more details.

Table 1: Distilling DFlash with a Qwen3-4B teacher. Each cell reports mean acceptance length τ, including the bonus token (higher is better). Mean is the unweighted average across benchmarks. Bold and underlining mark the best and second-best results within each temperature and column.

	MATH			CODE		CHAT	
Method	GSM8K	MATH500	AIME25	MBPP	HumanEval	MT-Bench	Mean
Temperature $T = 0$							
DFlash	6.469	8.023	7.379	6.023	6.581	4.119	6.432
AdaFlash	<u>6.793</u>	**8.339**	**7.709**	<u>6.141</u>	<u>6.713</u>	**4.329**	<u>6.671</u>
Draft-OPD	6.604	7.844	7.113	5.882	6.438	4.198	6.347
RKL	6.688	7.899	7.108	5.940	6.513	4.247	6.399
MDraft	**6.907**	<u>8.322</u>	<u>7.640</u>	**6.187**	**6.740**	<u>4.317</u>	**6.685**
Temperature $T = 0.6$							
DFlash	6.168	7.365	6.044	5.599	6.214	3.964	5.892
AdaFlash	6.487	<u>7.747</u>	6.413	<u>5.792</u>	<u>6.469</u>	4.101	<u>6.168</u>
Draft-OPD	6.397	7.446	<u>6.448</u>	5.616	6.273	4.065	6.041
RKL	<u>6.518</u>	7.564	6.385	5.711	6.384	<u>4.109</u>	6.112
MDraft	**6.739**	**7.905**	**6.480**	**6.001**	**6.593**	**4.205**	**6.321**
Temperature $T = 1$							
DFlash	5.655	6.489	4.953	5.145	5.684	3.720	5.274
AdaFlash	5.988	6.914	5.264	5.285	5.980	3.843	5.546
Draft-OPD	6.067	6.854	5.214	5.343	5.916	3.919	5.552
RKL	<u>6.300</u>	<u>7.041</u>	<u>5.374</u>	<u>5.541</u>	<u>6.039</u>	<u>4.062</u>	<u>5.726</u>
MDraft	**6.424**	**7.258**	**5.649**	**5.672**	**6.330**	**4.087**	**5.903**

Results. MDraft achieves the best mean acceptance length across all temperature settings. At $T = 0.6$ and $T = 1$, MDraft achieves the highest acceptance length on all six benchmarks in Table 1. Its mean acceptance lengths are 6.321 and 5.903, compared with 6.168 and 5.726 for the strongest competing method at each temperature, respectively. The gains span mathematics, coding, and conversational generation.

Comparison with standard distillation objectives. We compare MDraft with per-prefix reverse KL (RKL) and forward KL (FKL). Table 1 includes the RKL baseline; Appendix E.1 reports FKL post-training dynamics. In our post-training setting, FKL fails to improve the pretrained DFlash checkpoint.

Controlling for additional verifier evaluations. To separate the effect of target blending from using more verifier evaluations, we also train per-prefix RKL with $K = 1$ and $K = 8$ ($\alpha = 0$). Increasing the number of drafts does not give consistent improvement when the aggregation is not used, and both controls remain below MDraft (Figure 4).

4.2 WHERE DO ACCEPTANCE GAINS ARISE?

We use saved drafter checkpoints to examine how acceptance gains vary across the 15 drafted positions. Figure 2 compares MDraft, AdaFlash and the RKL control, using DFlash as the reference and plotting the change in token acceptance rate conditioned on reaching each position.

MDraft improves conditional acceptance throughout the block. RKL yields larger gains under stochastic decoding than under greedy decoding, consistent with better distributional overlap without equally strong recovery of the verifier's argmax. AdaFlash shows the opposite trade-off relative to RKL: larger gains at $T = 0$ but smaller gains at $T = 1$. This pattern is consistent with its hard-target term favoring argmax agreement over matching the probabilities of alternative tokens. **MDraft improves acceptance in both greedy and stochastic decoding.** The gains extend through the middle and later positions, where verifier targets depend on more sampled draft tokens. Section 4.4 examines how target aggregation can account for this behavior.

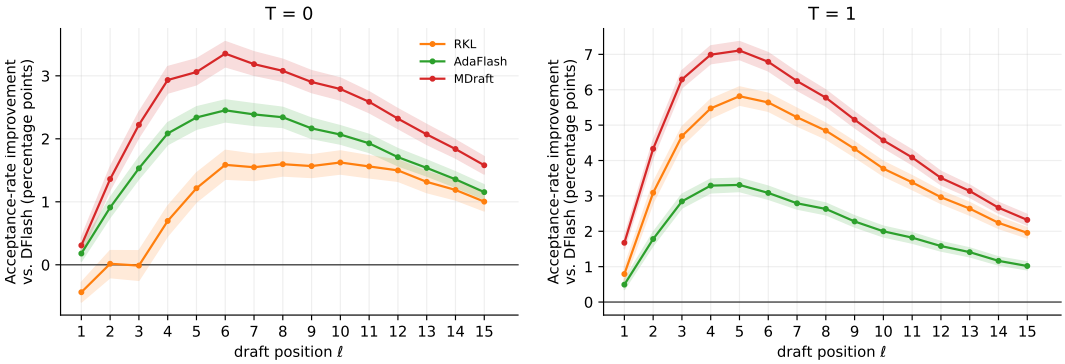

Figure 2: Position-wise conditional acceptance gains over DFlash, in percentage points, at $T = 0$ (left) and $T = 1$ (right). Red: MDraft; green: AdaFlash; orange: RKL. The evaluation uses 1,656 prompts from GSM8K, MBPP and MT-Bench. Shaded bands denote 95% confidence intervals. Positions index the 15 drafted tokens. MDraft's gains extend through the middle and later positions, with a larger advantage over AdaFlash at $T = 1$.

4.3 Effects of proposal count

To establish how additional drafts improve the performance of MDraft, we use Qwen3-4B and the same pretrained DFlash checkpoint and training corpus as in the main experiment. We keep position weighting and all other training settings fixed across runs and vary the number of used drafts K.

At $\alpha = 0.9$, we vary $K \in \{1, 2, 4, 8\}$ and track mean acceptance length throughout training (Figure 3). All curves begin at the common pretrained checkpoint. The $K = 1$ control reduces to per-prefix reverse KL. This comparison measures how additional proposals affect learning dynamics at a fixed number of training states.

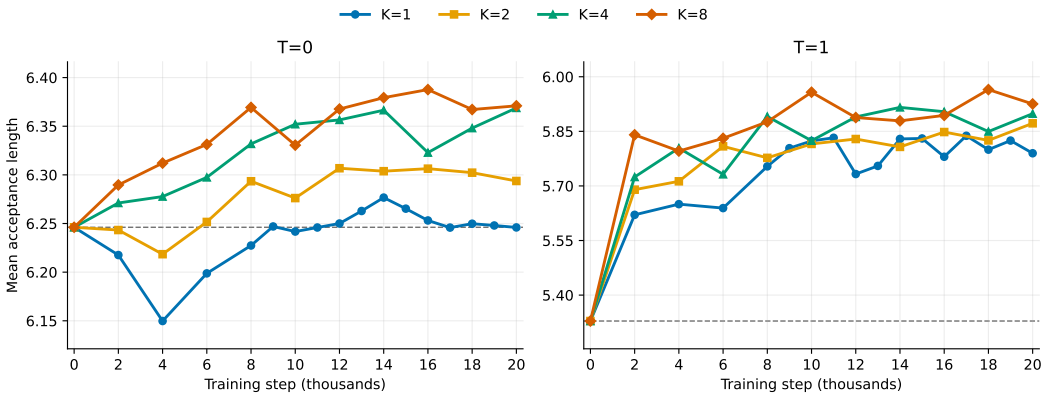

Figure 3: We fix $\alpha = 0.9$ and train DFlash for 20,000 iterations with varying numbers of parallel drafts $K = 1, 2, 4, 8$. We evaluate acceptance length on a subset of GSM8K and MBPP.

Using multiple proposals yields a clear improvement over $K = 1$, which finishes close to the initial checkpoint at $T = 0$. At $T = 1$, all settings improve acceptance, with the larger proposal counts reaching higher values.

4.4 How target blending changes the optimum

The experiments show that using an aggregated teacher in the distillation target improves drafting. To understand why, we ask which token distribution each objective would learn if it could be optimized exactly. We isolate one context and position, hold the draft-prefix distribution fixed, and use uniform position weights. We first analyze the population objective using the exact average verifier distribution, then account for the finite number of proposals used in practice. Write $p_x(y)$ for the verifier's probability of token y after a sampled prefix x, and $\bar{p}(y) = \mathbb{E}_x[p_x(y)]$ for its average.

What ordinary KL learns. The parallel drafter must fit a single distribution q to all of these prefix-dependent targets. In the optimum, forward KL averages their probabilities, while reverse KL averages their log probabilities (Genest & Zidek, 1986):

$$q^\star_{\mathrm{FKL}}(y) = \bar{p}(y), \tag{3}$$
$$q^\star_{\mathrm{RKL}}(y) \propto \exp(\mathbb{E}_x[\log p_x(y)])\,. \tag{4}$$

The first solution preserves a token's average support, but can favor a confident minority branch, as with *brew* in Table 2. The second strongly discounts a token if some prefixes give it very low probability. It can therefore favor a generic token such as *drink* over a token that is preferred on one branch but incompatible with another.

What MDraft changes. To quantify this discount, define $J(y) = \log \bar{p}(y) - \mathbb{E}_x[\log p_x(y)] \geq 0$. It is zero when the verifier assigns the same probability to y under every prefix, and grows when substantial average support coexists with very low probabilities under some prefixes. MDraft's reverse-KL loss differs from the following expression only by a constant independent of q:

$$\mathcal{L}_{\mathrm{MDraft}}(q) = \mathrm{KL}(q\|\bar{p}) + (1-\alpha)\sum_y q(y)J(y) + C \tag{5}$$

The first term encourages the drafter to match average verifier support. The second charges a penalty for placing probability on tokens whose support varies across prefixes. Ordinary per-prefix reverse KL includes this penalty at full strength ($\alpha = 0$). Blending with the average reduces its strength to $1-\alpha$, so incompatibility with one prefix has less influence on the shared drafter distribution.

Consequence for token selection. Minimizing this loss gives

$$q^\star_\alpha(y) \propto \bar{p}(y)\exp\big(-(1-\alpha)J(y)\big). \tag{6}$$

Each token's average support is multiplied by a discount for prefix disagreement. At $\alpha = 0$, this recovers the reverse-KL solution; at $\alpha = 1$, the discount disappears and the solution equals the forward-KL optimum. An intermediate value can preserve useful branch-specific tokens while still penalizing inconsistent support. In the example illustrated in Figure 1, it prevents *chocolate* from being suppressed in favor of *drink*, while discounting *brew* enough to recover the verifier's greedy choice.

Finite K also retains a disagreement penalty. The analysis above uses the exact average $\bar{p}$, which we cannot compute and therefore estimate during training with the sample average $\widehat{p}$. Although $\widehat{p}$ is unbiased, the loss uses its logarithm: concavity gives $\mathbb{E}[\log \widehat{p}(y)] \leq \log \bar{p}(y)$. Define this gap, which will depend on the sample size K, as $J_K(y) = \log \bar{p}(y) - \mathbb{E}[\log \widehat{p}(y)]$. In the expected training objective, the disagreement penalty becomes $(1-\alpha)J(y) + \alpha J_K(y)$ (Appendix C). Thus, finite K leaves a residual penalty even at $\alpha = 1$, qualitatively resembling a smaller blending coefficient. As K grows, J_K vanishes. At $\alpha = 1$, the limiting loss is reverse KL to $\bar{p}$: it shares the forward-KL optimum, but generally not its gradient away from that optimum. The implicit blending induced by finite K depends on the data and can differ across tokens. The explicit α provides a common, continuous control over the contribution of the average target, without requiring a reduction in the sampling budget. The parameters are complementary: increasing K improves estimation of the average, while α controls how strongly it enters the target. At finite K, however, a single α cannot generally compensate for the token-dependent bias $J_K(y)$. MDraft thus connects per-prefix RKL at $\alpha = 0$ for any K to the FKL optimum at $\alpha = 1$ as $K \to \infty$, with a residual disagreement penalty between these limits. The post-training comparisons with RKL (Table 1) and FKL (Appendix E.1) support the usefulness of this intermediate regime.

5 Conclusion

In this work, we identify a unique challenge that affects on-policy training of parallel drafters, explain the mathematical mechanisms by which it arises and its practical consequences. Based on this analysis, we introduce MDraft: a new method for parallel drafter distillation that uses multiple drafts drawn from the same context to construct a reliable target distribution using the verifier model. We conduct experiments that demonstrate that our method improves the drafter beyond off-policy training and establish the role of our method in improving drafter acceptance lengths on several diverse benchmarks.

6 AI USE STATEMENT

We used generative AI tools to help draft, edit, and format this manuscript, including conversion to the ICLR 2027 LaTeX style. Additionally, we used AI with search functionality to assist in literature search and discovery. We affirm that we have reviewed and verified the work we are submitting.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. 2024. URL https://arxiv.org/abs/2306.13649.

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for medusa decoding, 2024. URL https://arxiv.org/abs/2402.05109.

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. 2025. URL https://arxiv.org/abs/2503.09573.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

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, 2024. URL https://arxiv.org/abs/2401.10774.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding, 2026. URL https://arxiv.org/abs/2602.06036.

Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, Qinyu Chen, Chengqi Deng, Jiping Yu, Damai Dai, Zhengyan Zhang, Yixuan Wei, Yixuan Tan, Wenkai Yang, Runxin Xu, Yu Wu, Zhean Xu, Xuanyu Wang, Muyang Chen, Rui Tian, Xiao Bi, Zhewen Hao, Shaoyuan Chen, Huanqi Cao, Wentao Zhang, Anyi Xu, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation, 2026. URL https://arxiv.org/abs/2607.05147.

Jacob K Christopher, Brian R Bartoldson, Tal Ben-Nun, Michael Cardei, Bhavya Kailkhura, and Ferdinando Fioretto. Speculative diffusion decoding: Accelerating language generation through diffusion, 2025. URL https://arxiv.org/abs/2408.05636.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Christian Genest and James V. Zidek. Combining Probability Distributions: A Critique and an Annotated Bibliography. *Statistical Science*, 1(1):114 – 135, 1986. doi: 10.1214/ss/1177013825. URL https://doi.org/10.1214/ss/1177013825.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*, 2018. URL https://arxiv.org/abs/1711.02281.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2306.08543.

Lanxiang Hu, Zhaoxiang Feng, Yulun Wu, Haoran Yuan, Yujie Zhao, Yu-Yang Qian, Bojun Wang, Peng Zhao, Daxin Jiang, Yibo Zhu, Tajana Rosing, and Hao Zhang. Jetspec: Breaking the scaling ceiling of speculative decoding with parallel tree drafting, 2026. URL https://arxiv.org/abs/2606.18394.

Shijing Hu, Jingyang Li, Xingyu Xie, Zhihui Lu, Kim-Chuan Toh, and Pan Zhou. Griffin: Effective token alignment for faster speculative decoding. 2025. URL https://arxiv.org/abs/2502.11018.

Jianuo Huang, Yaojie Zhang, Qituan Zhang, Hao Lin, Hanlin Xu, and Linfeng Zhang. Domino: Decoupling causal modeling from autoregressive drafting in speculative decoding, 2026. URL https://arxiv.org/abs/2605.29707.

Amirmohammad Karimi, Chao Gao, and Negar Hassanpour. Dblast: Dependent block drafting for stochastic speculative decoding, 2026. URL https://arxiv.org/abs/2608.05448.

Jongwoo Ko, Sungnyun Kim, Tianyi Chen, and Se-Young Yun. DistiLLM: Towards streamlined distillation for large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 24872–24895, 2024. URL https://proceedings.mlr.press/v235/ko24c.html.

Haodi Lei, Yafu Li, Haoran Zhang, Shunkai Zhang, Qianjia Cheng, Xiaoye Qu, Ganqu Cui, Bowen Zhou, Ning Ding, Yun Luo, and Yu Cheng. Draft-opd: On-policy distillation for speculative draft models, 2026. URL https://arxiv.org/abs/2605.29343.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. 2023. URL https://arxiv.org/abs/2211.17192.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. 2024. URL https://arxiv.org/abs/2406.16858.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. 2025a. URL https://arxiv.org/abs/2401.15077.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test, 2025b. URL https://arxiv.org/abs/2503.01840.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.20050.

Alexander Lin, Jeremy Wohlwend, Howard Chen, and Tao Lei. Autoregressive knowledge distillation through imitation learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 6121–6133, 2020. doi: 10.18653/v1/2020.emnlp-main.494. URL https://aclanthology.org/2020.emnlp-main.494/.

Fuliang Liu, Xue Li, Ketai Zhao, Yinxi Gao, Ziyan Zhou, Zhonghui Zhang, Zhibin Wang, Wanchun Dou, Sheng Zhong, and Chen Tian. DART: Diffusion-inspired speculative decoding for fast LLM inference, 2026. URL https://arxiv.org/abs/2601.19278.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding, 2023. URL https://arxiv.org/abs/2310.07177.

Mathematical Association of America. American invitational mathematics examination (AIME), 2025. URL https://maa.org/maa-invitational-competitions/.

Dhruv Nathawani, Shuoyang Ding, Vitaly Lavrukhin, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Boris Ginsburg, and Jane Polak Scowcroft. Nemotron-Post-Training-Dataset-v2. NVIDIA, version 2.0, August 2025. URL https://huggingface.co/datasets/nvidia/Nemotron-Post-Training-Dataset-v2.

Yu-Yang Qian, Hao-Cong Wu, Chen Chen, Jiacheng Sun, Zhenhua Dong, Peng Zhao, and Zhi-Hua Zhou. Adaflash: Adaptive speculative decoding via on-policy distilled diffusion drafters, 2026. URL https://arxiv.org/abs/2607.19223.

Peer Rheinboldt, Frédéric Berdoz, and Roger Wattenhofer. Treeflash: Parallel ar-approximation for faster speculative decoding, 2026. URL https://arxiv.org/abs/2606.03819.

Liran Ringel and Yaniv Romano. Accelerating speculative decoding with block diffusion draft trees, 2026. URL https://arxiv.org/abs/2604.12989.

Matan Rusanovsky, Yoav Miron, Roy Uziel, Omer Belhasin, Hao Guo, Ran Zilberstein, Maor Ashkenazi, and Michael Elad. Lilicorr: Lightweight likelihood correlation of parallel drafts for speculative decoding, 2026. URL https://arxiv.org/abs/2608.20530.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. 2024. URL https://arxiv.org/abs/2406.07524.

Alexander Samarin, Sergei Krutikov, Anton Shevtsov, Sergei Skvortsov, Filipp Fisin, and Alexander Golubev. Lk losses: Direct acceptance rate optimization for speculative decoding, 2026. URL https://arxiv.org/abs/2602.23881.

Jameson Sandler, Jacob K. Christopher, Thomas Hartvigsen, and Ferdinando Fioretto. Specdiff-2: Scaling diffusion drafter alignment for faster speculative decoding, 2025. URL https://arxiv.org/abs/2511.00606.

Zheng Wang, Davis Wertheimer, Yu Chin Fabian Lim, Mudhakar Srivatsa, Raghu K. Ganti, Minjia Zhang, and Naigang Wang. xPress: Parallel refinement for diffusion drafters in speculative decoding, 2026a. URL https://arxiv.org/abs/2608.02438.

Zheng Wang, Zhifan Ye, Qi Cheng, Yonggan Fu, Ziyan Wang, Feng Zhu, Haozhe Zhao, Jan Kautz, Pavlo Molchanov, Humphrey Shi, and Minjia Zhang. PRESTO: Prefix-aligned tree drafting for diffusion speculative decoding, 2026b. URL https://arxiv.org/abs/2607.22634.

Zhuoyu Wang, Junnan Huang, and Xinyu Chen. Taps: Target-aware prefix tree selection for diffusion-drafted speculative decoding, 2026c. URL https://arxiv.org/abs/2606.00487.

Lexington Whalen, Yuki Ito, and Ryo Sakamoto. Teaching diffusion to speculate left-to-right, 2026. URL https://arxiv.org/abs/2606.11552.

Tianyu Wu, Yu Yao, Zhenting Qi, Han Zheng, Zhuohan Wang, Haoran Ma, Lawrence Liao, Himabindu Lakkaraju, Ju Li, and Yilun Du. D-pace: Dynamic position-aware cross-entropy for parallel speculative drafting, 2026. URL https://arxiv.org/abs/2605.18810.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025.

Tianjian Yang and Meng Li. Spec-auf: Accept-until-fail training under train-inference misalignment for masked block drafters, 2026. URL https://arxiv.org/abs/2607.01893.

Liyuan Zhang, Jiarui Zhang, Jinwei Yao, Ran Yan, Yuchen Yang, Jiahao Zhang, Tongkai Yang, Yi Wu, and Binhang Yuan. D^2SD: Accelerating speculative decoding with dual diffusion draft models, 2026. URL https://arxiv.org/abs/2606.04446.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2306.05685.

Chunting Zhou, Graham Neubig, and Jiatao Gu. Understanding knowledge distillation in non-autoregressive machine translation. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1911.02727.

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. 2024. URL https://arxiv.org/abs/2310.08461.

Xiandong Zou, Jianshu Li, Jing Huang, and Pan Zhou. Variational speculative decoding: Rethinking draft training from token likelihood to sequence acceptance, 2026. URL https://arxiv.org/abs/2602.05774.

A Related Work

Speculative decoding and drafter architectures. Work following the original speculative decoding algorithms (Leviathan et al., 2023; Chen et al., 2023) improves the trade-off between proposal quality and drafting cost. Medusa predicts future tokens with multiple heads, while Hydra introduces dependencies between those heads (Cai et al., 2024; Ankner et al., 2024). EAGLE drafts using target-model features; EAGLE-2 improves candidate-tree construction, and EAGLE-3 develops the training procedure and feature representation further (Li et al., 2025a; 2024; 2025b).

Parallel and diffusion drafting. Masked and block diffusion language models provide a framework for predicting several tokens together (Sahoo et al., 2024; Arriola et al., 2025). Speculative Diffusion Decoding applies diffusion models to drafting, and DFlash uses target hidden states to produce a draft block in one step (Christopher et al., 2025; Chen et al., 2026). DART also predicts future logits in a single pass from target hidden states, using position-weighted forward KL for training and consistency-aware tree pruning at inference (Liu et al., 2026). Parallel drafting reduces sequential computation but makes dependencies between proposed tokens harder to represent. DSpark combines semi-autoregressive generation with confidence-scheduled verification, while Domino and TreeFlash introduce mechanisms for causal modeling within parallel drafting (Cheng et al., 2026; Huang et al., 2026; Rheinboldt et al., 2026). DBLAST and LiLiCorr address dependencies between proposed tokens to improve their joint consistency (Karimi et al., 2026; Rusanovsky et al., 2026). xPress addresses the same lack of within-block dependencies through a lightweight causal refiner (Wang et al., 2026a). PRESTO identifies the mismatch between prefix-blind diffusion marginals and prefix-dependent verification, and uses prefix-aligned scoring for tree search (Wang et al., 2026b). MDraft instead changes the distillation targets while retaining the drafter architecture and inference procedure. Related inference methods explore block-diffusion draft trees, target-aware prefix selection, and extensions of parallel drafting (Ringel & Romano, 2026; Wang et al., 2026c; Zhang et al., 2026).

Acceptance-aware training. DistillSpec studies how knowledge distillation and the choice of divergence affect speculative decoding (Zhou et al., 2024). More recent objectives account explicitly for the fact that verification retains a prefix rather than independently accepting positions. DFlash applies a fixed positional decay, while GRIFFIN, left-to-right diffusion training, D-PACE, and Spec-AUF modify the weighting or support of token-level supervision to emphasize errors that limit the accepted prefix (Chen et al., 2026; Hu et al., 2025; Whalen et al., 2026; Wu et al., 2026; Yang & Li, 2026). Other approaches change the objective itself. LK Losses uses negative log acceptance and a scheduled KL–TV combination, and Variational Speculative Decoding formulates training in terms of sequence acceptance (Samarin et al., 2026; Zou et al., 2026). SpecDiff-2 optimizes a streak-based objective on verifier-generated trajectories (Sandler et al., 2025).

On-policy distillation. ImitKD addresses exposure bias by querying the teacher on student-generated prefixes (Lin et al., 2020). MiniLLM and generalized knowledge distillation study learning from

student-generated sequences and the role of divergence choice in language-model compression (Gu et al., 2024; Agarwal et al., 2024). DistiLLM studies skew KL objectives and adaptive off-policy reuse of student-generated outputs (Ko et al., 2024); its teacher–student distribution mixing differs from our aggregation of teacher predictions across sampled prefixes. Online Speculative Decoding adapts draft models to observed queries through knowledge distillation (Liu et al., 2023), addressing deployment-time distribution shifts. For speculative draft models, Draft-OPD constructs supervision from verification outcomes, combining forward KL on accepted positions with reverse KL on rejected positions (Lei et al., 2026). AdaFlash performs on-policy adaptation during deployment using a hard verifier target together with clipped reverse KL (Qian et al., 2026). Our geometric-pooling analysis isolates the unmodified reverse-KL term; it does not characterize the full clipped or verification-dependent losses of these methods. JetSpec's comparison of forward and reverse KL for parallel draft heads also highlights the importance of objective choice in this setting (Hu et al., 2026).

Multimodality in parallel generation. Independent output positions also create difficulties in representing multiple coherent continuations in non-autoregressive machine translation (Gu et al., 2018). Distillation can reduce the complexity of the training distribution and make it easier for a parallel model to fit (Zhou et al., 2020). MDraft addresses the related issue of conflicting prefix-dependent supervision during on-policy drafter training, while the verifier preserves the final output distribution.

B Analysis of Target Blending

Consider reverse KL with fixed, proposal-independent weights. Fix a context c and position ℓ, and suppress them in the notation: write $q = q_\ell(\cdot \mid c)$, $p^{(k)} = p_\ell^{(k)}$, $r^{(k)} = r_\ell^{(k)}$, $\widehat{p} = \widehat{p}_\ell$, and $Z_k = Z_{\ell,k}$. The normalized-target loss is related to a weighted combination of reverse-KL losses by

$$R_{\alpha,K}(q) = (1-\alpha)\frac{1}{K}\sum_{k=1}^{K} \mathrm{KL}(q\|p^{(k)}) + \alpha\,\mathrm{KL}(q\|\widehat{p}),$$

$$\frac{1}{K}\sum_{k=1}^{K} \mathrm{KL}(q\|r^{(k)}) = R_{\alpha,K}(q) + \frac{1}{K}\sum_{k=1}^{K} \log Z_k. \tag{7}$$

The normalization term is detached, so these objectives have identical drafter gradients. We analyze the equivalent objective $R_{\alpha,K}$. Hold the draft-prefix law $q_{<\ell}$ fixed. For a prefix $x \sim q_{<\ell}$, write $p_x(y) = p(y \mid c, x)$ and $\bar{p}(y) = \mathbb{E}_x[p_x(y)]$; thus $p^{(k)} = p_{x_{<\ell}^{(k)}}$. Assuming positive probabilities, define

$$J(y) = \log \bar{p}(y) - \mathbb{E}_x[\log p_x(y)] \geq 0. \tag{8}$$

Adding and subtracting $\log \bar{p}(y)$ inside the reverse-KL objective gives

$$\mathbb{E}_x\big[\mathrm{KL}(q\|p_x)\big] = \mathrm{KL}(q\|\bar{p}) + \sum_y q(y)J(y). \tag{9}$$

Thus, per-prefix reverse KL combines matching the marginal target with a penalty determined by how much each token's verifier probability varies across prefixes. The penalty is zero for a token whose probability is identical under all prefixes. It can be large for a token that is strongly supported under some prefixes but assigned near-zero probability under others. For a fixed prefix law, the drafter can lower this term by shifting probability away from tokens with large $J(y)$, even when they have substantial marginal support.

In the population limit, substituting the decomposition above into the two-term loss yields equation 5. Minimizing over a normalized probability vector gives equation 6.

B.1 Two-branch example

For the example in Figure 1, fix the first-position drafter to match the verifier: $q_1(\text{hot}) = 0.65$ and $q_1(\text{cold}) = 0.35$. Table 2 gives the two verifier conditionals and the optimal shared second-position distribution q under each objective. We use uniform position weights and the exact branch average (the population limit for MDraft). The solutions follow from equation 3, equation 4, and equation 6.

At $T = 0$, both models choose tokens greedily, so the first token is *hot* and A_0 is one exactly when the drafter's second-token argmax is *chocolate*. At $T = 1$, the first token is always accepted because its draft and verifier distributions agree. The expected second-token acceptance is therefore

$$A_1(q) = 0.65 \sum_y \min\{q(y), p(y \mid \text{hot})\} + 0.35 \sum_y \min\{q(y), p(y \mid \text{cold})\}.$$

MDraft at $\alpha = 0.9$ selects *chocolate* greedily and improves this overlap relative to both KL optima. The example demonstrates that an intermediate pooling objective can improve acceptance under both greedy and stochastic decoding.

Table 2: Numerical two-branch example for Figure 1, with $\delta = 10^{-9}$ and fixed branch probabilities $0.65/0.35$. Top: normalized verifier conditionals. Bottom: optimal second-position draft probabilities, rounded to three decimals, and second-token acceptance A_T at evaluation temperature T. Bold probabilities mark each drafter's greedy choice; bold acceptance values mark the best of the three objectives.

Verifier conditional	*chocolate*	*cocoa*	*drink*	*brew*		
$p(\cdot \mid \text{hot})$, 65%	0.42	0.31	$0.27 - \delta$	δ		
$p(\cdot \mid \text{cold})$, 35%	0.02	0.01	0.10	0.87		
Objective	*chocolate*	*cocoa*	*drink*	*brew*	A_0	A_1
Per-prefix RKL	0.338	0.217	**0.445**	0.000	0	0.582
Per-prefix FKL	0.280	0.205	0.210	**0.305**	0	0.604
MDraft ($\alpha = 0.9$)	**0.350**	0.253	0.278	0.119	**1**	**0.655**

C The Role of the Number of Proposals

Using the fixed-context, fixed-position notation of Appendix B, the empirical average $\widehat{p}(y) = K^{-1} \sum_{k=1}^{K} p^{(k)}(y)$ is an unbiased estimator of $\bar{p}(y)$, but its logarithm is not. Define $J_K(y) = \log \bar{p}(y) - \mathbb{E}[\log \widehat{p}(y)]$, where the expectation is over the K independent proposals. For the reverse-KL surrogate in equation 7,

$$\mathbb{E}[R_{\alpha,K}(q)] = \mathrm{KL}(q \| \bar{p}) + \sum_y q(y) \big[(1-\alpha) J(y) + \alpha J_K(y)\big]. \tag{10}$$

By concavity of the logarithm, $0 \le J_K(y) \le J(y)$. With $K = 1$, $J_K = J$, and the reverse-KL branch reduces to per-prefix reverse KL for every α. More generally, $K = 1$ makes $r^{(1)} = p^{(1)}$, so α has no effect for any choice of divergence. As K increases, the residual penalty is nonincreasing and converges to zero for a fixed finite vocabulary and block length with positive verifier probabilities. Consequently, K controls both training cost and how closely the empirical objective approaches the population interpolation. At the first block position, all proposals share the same verifier conditional, so marginalization has no effect. The same holds at any position where the sampled conditionals agree. These target-construction properties hold for other divergences as well; the disagreement decomposition above is specific to reverse KL.

Divergence choices. MDraft fits the drafter distribution q to each normalized tilted target $r^{(k)}$ by minimizing a divergence between them. Our implementation supports forward and reverse-KL, as well as Jensen-Shannon divergence. In the paper and the experiments, we always use reverse KL as the divergence in MDraft. We leave the analysis of the effects of these divergences to future work.

D Position Weighting

The weights $w_{\ell,k}$ in equation 2 control the contribution of position $\ell \in \{1, \dots, L\}$ in proposal k. We distinguish three choices below. All weights are held fixed when differentiating the loss; none changes the unweighted average $\widehat{p}_\ell$ used to construct the distillation targets.

Uniform weights. Setting $w_{\ell,k} = 1$ gives every position in every proposal equal weight. The objective is then a simple average of the position-wise divergences, regardless of how likely verification is to reach each position.

DFlash weights. DFlash (Chen et al., 2026) uses a fixed exponential decay with position. In our notation, this schedule is

$$w_{\ell,k} = \exp\left(-\frac{\ell-1}{\gamma}\right), \qquad \gamma > 0. \tag{11}$$

The first position has weight one, and later positions receive progressively less weight. Smaller γ concentrates supervision near the start of the block; as $\gamma \to \infty$, the weights approach uniform weighting. The schedule is shared across contexts and proposals: it favors earlier positions without using their actual acceptance probabilities.

Survival weights. Instead of prescribing a fixed decay, survival weighting adapts to each proposal. For a sampled token $x_m^{(k)}$, define its conditional acceptance probability under speculative rejection sampling as

$$a_{m,k} = \min\left\{1, \frac{p_m^{(k)}(x_m^{(k)})}{q_m(x_m^{(k)} \mid c)}\right\}. \tag{12}$$

Here $p_m^{(k)} = p(\cdot \mid c, x_{<m}^{(k)})$, and acceptance is conditional on all preceding draft tokens having been accepted. Thus, the probability that verification reaches position ℓ in this sampled proposal is

$$\rho_{\ell,k} = \prod_{m=1}^{\ell-1} a_{m,k}, \qquad w_{\ell,k} = \mathrm{sg}\left[\frac{L\rho_{\ell,k}}{\sum_{j=1}^{L}\rho_{j,k}}\right]. \tag{13}$$

The empty product is one, and sg denotes stop-gradient. Normalization gives $\sum_{\ell=1}^{L} w_{\ell,k} = L$, preserving the total weight of uniform supervision. The product includes only preceding positions: a token is still supervised if verification reaches it, even when that token itself is likely to be rejected. Positions after a likely rejection receive less weight. With fully greedy decoding, exact survival instead reduces to an indicator that all preceding draft tokens match the verifier's greedy choices.

Position-weight comparison. Table 3 compares uniform and survival weighting for AdaFlash and MDraft in the Qwen3-4B main-experiment setup. MDraft uses $\alpha = 0.9$, $K = 8$, reverse KL; AdaFlash retains its hard-target and clipped reverse-KL terms. Survival weights are normalized to preserve the uniform loss scale. MDraft consistently performs better with survival weights compared to the uniform-weight setup.

Table 3: Uniform versus survival weights: unweighted six-benchmark mean acceptance length τ. Means are reported to three decimal places.

Method	Weights	$T = 0$	$T = 0.6$	$T = 1$
AdaFlash	Uniform	6.671	6.168	5.546
AdaFlash	Survival	6.691	6.219	5.568
MDraft	Uniform	6.568	6.260	5.836
MDraft	Survival	6.685	6.321	5.903

E Main Experiment Details

Data and sampled states. We construct the corpus from the Nemotron-Post-Training-Dataset-v2 dataset. Qwen3-4B generates four independent completions per source prompt with thinking disabled, temperature 0.7, top-$p = 0.8$, top-$k = 20$, and an 8,192-token completion cap. We retain 410,448 math sequences and 128,372 code sequences in complete four-completion groups, plus 128,372 sampled chat completion rows; chat prompts need not retain all four attempts. Prompt and completion together are truncated to 1,024 tokens. Filtering sequences too short to provide a full block leaves 665,156 usable sequences. Each epoch samples one anchor uniformly among response positions with room for all 15 subsequent draft tokens. Contexts come from this fixed corpus, while training proposals are generated by the current drafter; teacher continuations are not refreshed during training.

Model and optimization. The initial checkpoint, `z-lab/Qwen3-4B-DFlash-b16`, has five non-causal layers and a 16-slot layout: one committed anchor and 15 drafted positions ($L = 15$ in our notation). Only drafter parameters are updated. We use fp32 trainable parameters with bf16 forward computation, AdamW with learning rate 10^{-5}, betas $(0.9, 0.95)$, epsilon 10^{-8}, zero weight decay, and gradient-norm clipping at 1.0. The schedule warms up for 3,118 steps and then decays cosinusoidally to zero at step 62,358. Four data-parallel workers each process eight anchors per step,

without gradient accumulation; $K = 8$ therefore gives 256 sampled blocks but still 32 independent anchors. Completed reference runs use four H100 80 GB GPUs.

Objectives and position weights. AdaFlash samples one greedy block and uses $0.8\mathcal{L}_{\text{hard}} + 0.2\mathcal{L}_{\text{RKL}}$, with reverse KL evaluated at temperature 1 and its vocabulary-entry contributions capped above at 0.01. Its survival variant applies the same weights to both terms. Our Draft-OPD adaptation uses corpus-sampled anchors, forward KL on the accepted prefix, and reverse KL on the rejected suffix, with equal stream coefficients, separate normalization of nonempty streams, and a rejected-position decay of 0.8^{j-1}. We reproduce the published system's live rollouts and verification-error replay; however, across our experiments we do not find a detectable gain over using random anchors, but live rollouts significantly increase the computation and time complexity of the method. We hence use random anchors for our implementation of Draft-OPD. MDraft uses $\alpha = 0.9$, $K = 8$, and temperature-1 proposals; the RKL-only control uses $K = 1$ and uniform weights. The reported MDraft runs use RKL as the divergence in Appendix C; the main table uses survival weights, while Table 3 is reserved for the uniform-versus-survival comparison.

Checkpoint selection and evaluation. For the main comparison, we use the final checkpoint at step 62,358 for every fine-tuned method, corresponding to three epochs of training. DFlash is evaluated at the unchanged pretrained checkpoint. Final evaluation covers GSM8K (1,319 prompts), MATH500 (500), AIME25 (30), sanitized MBPP (257), HumanEval (164), and MT-Bench (80, first turn only). Both models decode greedily at $T = 0$; at $T = 0.6$ and $T = 1$, proposal and verifier temperatures match and verification uses exact speculative rejection sampling. Thinking is disabled, the generation cap is 2,048 new tokens, and the evaluation seed is 0. Acceptance length counts committed tokens including the correction/bonus token, lies between 1 and 16, and is averaged within prompts before averaging over each benchmark.

Multiple-draft reverse-KL control. We train two controls with $\alpha = 0$ and $K \in \{1, 8\}$, using Qwen3-4B, the same initial DFlash checkpoint, and the training corpus from the main experiment. At $\alpha = 0$, each draft is trained against its own verifier conditional, so increasing K adds verifier evaluations without blending targets. We evaluate mean acceptance length on a GSM8K subset at temperatures $T = 0$ and $T = 1$. Figure 4 shows similar learning curves for the two controls. At step 14,000, $K = 1$ and $K = 8$ reach 6.63 and 6.69 at $T = 0$, and 6.247 and 6.29 at $T = 1$, respectively. MDraft with $K = 8$ reaches 6.81 and 6.37 at the same step. Using more drafts with an unchanged RKL target does not recover the gains from target blending.

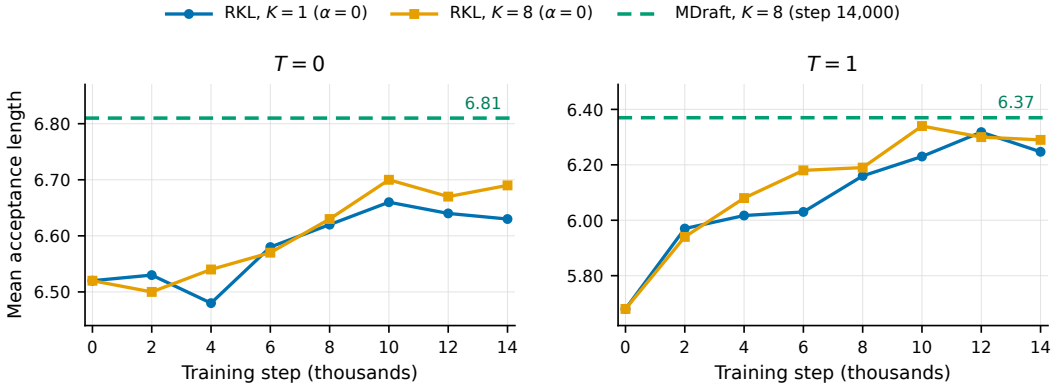

Figure 4: Reverse-KL controls with $\alpha = 0$ and $K = 1$ or $K = 8$, evaluated on a GSM8K subset at $T = 0$ (left) and $T = 1$ (right). Curves include the initial checkpoint and evaluations every 2,000 training steps. Dashed horizontal lines mark MDraft's acceptance length with $K = 8$ at step 14,000; they are endpoint references, not MDraft training trajectories.

E.1 Forward-KL baseline

We examine per-prefix forward KL as an additional baseline for post-training. In our experiments, it fails to improve acceptance length when fine-tuning the pretrained DFlash checkpoint. Figure 5 shows the post-training dynamics on a GSM8K subset at $T = 0$ and $T = 1$: FKL remains below the starting checkpoint throughout the measured updates, whereas RKL and MDraft improve acceptance length.

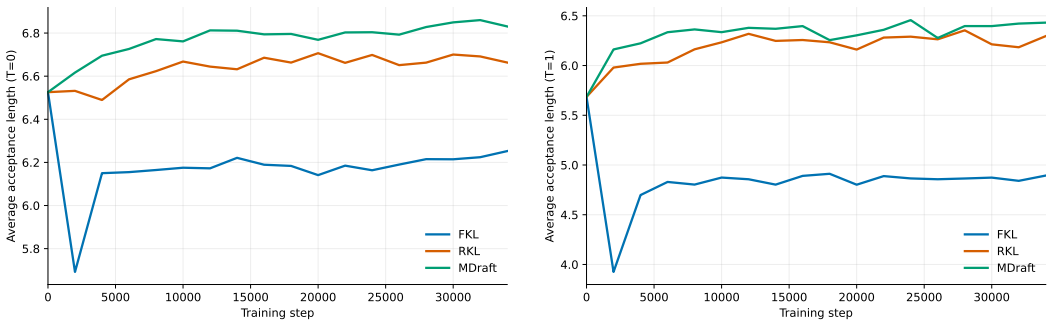

Figure 5: Post-training acceptance length on a GSM8K subset at $T = 0$ (left) and $T = 1$ (right) over 34,000 training steps. Starting from the same pretrained checkpoint, FKL initially reduces acceptance length and only partially recovers, remaining below its initial value. RKL improves over initialization, while MDraft achieves higher acceptance length throughout the subsequent evaluations.

F Parallel Drafting and Verification

Let c denote the context preceding a draft block, $\mathcal{V}$ the vocabulary, and L the number of newly proposed tokens. We write p for the target model, also called the verifier, and q for the drafter. In the one-step drafting setting, the drafter receives a fully masked block after the context and produces L categorical distributions in a single forward pass:

$$q(x_{1:L} \mid c) = \prod_{\ell=1}^{L} q_\ell(x_\ell \mid c). \tag{14}$$

Tokens are sampled independently across positions given c. The drafter's prediction q_ℓ for a set token position ℓ therefore does not depend on the sampled in-block prefix $x_{<\ell}$. The verifier evaluates $p_\ell = p(\cdot \mid c, x_{<\ell})$. All L verifier conditionals can be obtained in one causal forward pass over the proposed block.

Speculative sampling checks proposals from left to right (Leviathan et al., 2023; Chen et al., 2023). If the preceding tokens have been accepted, token x_ℓ is accepted with probability

$$a_\ell(x_{1:\ell}) = \min\left\{1, \frac{p_\ell(x_\ell)}{q_\ell(x_\ell \mid c)}\right\}. \tag{15}$$

At the first rejection, a replacement token is sampled from the normalized positive part of $p_\ell - q_\ell$, and the remaining proposals are discarded. If every proposal is accepted, the verifier supplies a bonus token. This correction procedure preserves the verifier's sampling distribution; the sampling-correctness argument below establishes this for the factorized drafter used here.

For a fixed prefix, averaging over the proposed token gives the acceptance probability

$$a(q,p) = \sum_{y\in\mathcal{V}} \min\{q(y), p(y)\} = 1 - \mathrm{TV}(q,p), \qquad \mathrm{TV}(q,p) = \frac{1}{2}\sum_{y\in\mathcal{V}} |q(y) - p(y)|. \tag{16}$$

Block acceptance also depends on whether earlier proposals survive. In particular, the expected number of accepted draft tokens is

$$\mathbb{E}[N_{\mathrm{acc}} \mid c] = \sum_{j=1}^{L} \mathbb{E}_{x_{1:L}\sim q(\cdot|c)}\left[\prod_{\ell=1}^{j} a_\ell(x_{1:\ell})\right]. \tag{17}$$

This count excludes the correction or bonus token. The product explains why reducing an error early in the block can be more valuable than reducing one near its end.

Sampling temperature. The probabilities in the acceptance rule must be those actually used to sample the target and proposal. For temperature $T > 0$, a model with logits $h(y)$ defines probabilities proportional to $\exp(h(y)/T)$; any additional sampling filters must also be reflected in those probabilities. We suppress temperature in the notation when it is clear from context. Under greedy decoding for both models, acceptance reduces to agreement with the verifier's argmax. If only the verifier is greedy and the drafter still samples, the mean acceptance is instead the draft probability assigned to that argmax.

Sampling correctness. We specialize the standard speculative-sampling argument (Leviathan et al., 2023; Chen et al., 2023) to the factorized proposal in equation 14. Fix the context c and an accepted prefix $x_{<\ell}$. Abbreviate $p(y) = p(y \mid c, x_{<\ell})$ and $q(y) = q_\ell(y \mid c)$. Because draft positions are sampled independently, conditioning on the acceptance of earlier positions leaves the current proposal distributed as q.

For $X \sim q$, accept with probability $a(X) = \min\{1, p(X)/q(X)\}$, evaluated only on the support of q. Let

$$A = \sum_y \min\{p(y), q(y)\}, \qquad p_{\mathrm{res}}(y) = \frac{[p(y) - q(y)]_+}{1 - A} \quad \text{if } A < 1, \tag{18}$$

where $[u]_+ = \max\{u, 0\}$. The residual is normalized because $\sum_y [p(y) - q(y)]_+ = 1 - A$. The joint probability of proposing and accepting token y is $q(y)a(y) = \min\{p(y), q(y)\}$. Therefore the emitted token Y, including correction after rejection, satisfies

$$\begin{aligned} \Pr(Y = y) &= \Pr(X = y, \text{accept}) + \Pr(\text{reject}) p_{\mathrm{res}}(y) \\ &= \min\{p(y), q(y)\} + [p(y) - q(y)]_+ = p(y). \end{aligned} \tag{19}$$

If $A = 1$, then $p = q$ and no correction is needed. Applying this identity successively to accepted positions, discarding unused proposals after a rejection, and sampling the bonus token from the verifier proves preservation of its autoregressive distribution. The argument uses the actual evaluation distributions, including any temperature or sampling filters, and does not depend on the training objective.