RELAYSPEC: ADAPTING FROZEN SPECULATIVE DRAFTERS WITH LINEAR MAPS

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates autoregressive LLM inference by using a lightweight drafter to propose tokens that the target verifies in parallel. Recent high-performance drafters improve proposal quality by conditioning on intermediate features from the target model. This coupling, however, makes the drafter difficult to reuse when the target changes. LoRA adaptation shifts these features, and replacing the target can even change their dimensionality. We introduce RelaySpec, a method that adapts a pretrained drafter to a new target by learning one linear map per target feature stream while keeping the drafter's transformer frozen. On Qwen3-4B with GSM8K, KiCad and NanoCoder LoRAs, RelaySpec improves DFlash throughput by 53% on average over the unadapted drafter. When transferring a Qwen3-4B drafter to Qwen3-8B with in-domain fitting data, the adapted drafter reaches 94–100% of native-drafter throughput across math, code, and chat. The reverse transfer, from Qwen3-8B to Qwen3-4B, reaches 88–97%. A single fit on mixed data reaches 99.4% of native throughput on Spec-Bench. In vLLM serving, three LoRA-fine-tuned targets share a single frozen drafter, achieving 77% higher aggregate throughput than the unadapted drafter at 32 concurrent clients. RelaySpec also generalizes to EAGLE-3, improving throughput by up to 105% over the unadapted drafter across three LoRA-fine-tuned targets. It also outperforms SD^2 by 3.6–10.2% even when SD^2 also fine-tunes the drafter, while training 19$\times$ fewer parameters, and outperforms OmniDraft by 1.9–52.4%. RelaySpec adapts an existing drafter with thousands of target-generated trajectories instead of training a new one from scratch, which for DFlash takes roughly 800,000 examples.

1 INTRODUCTION

The latency and cost of generating outputs matter more as large language models become more capable and are deployed across more tasks. Autoregressive decoding generates one token at a time from a transformer decoder. This serial computation creates a performance bottleneck and can leave modern GPUs underutilized (Cai et al., 2024).

Speculative decoding accelerates a language model, called the *target*, using a smaller model or prediction module, called the *drafter*. The drafter proposes several candidate tokens, and the target checks them in parallel. Verification keeps an accepted prefix and replaces the first rejected proposal with a correction token. Appropriate acceptance and correction rules preserve the target's output distribution (Leviathan et al., 2023; Chen et al., 2023). Speedup depends primarily on the length of the accepted prefix in each proposed block, together with the time spent drafting and verifying that block. See Xia et al. (2024) for a broader survey.

Recent high-performance drafters condition on the target's own hidden states. EAGLE drafts autoregressively in feature space, starting from the target's hidden states just before the language-model head (Li et al., 2024b), whereas EAGLE-3 fuses low-, middle- and high-level features taken from three target layers (Li et al., 2025). DFlash reads hidden states from five fixed target layers, combines them through a fusion projection and RMSNorm, and supplies the fused context as keys and values to every drafter layer, predicting a whole block of tokens in one block-diffusion pass (Chen et al., 2026). In both families, the drafter's input interface is trained against one target's features.

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.

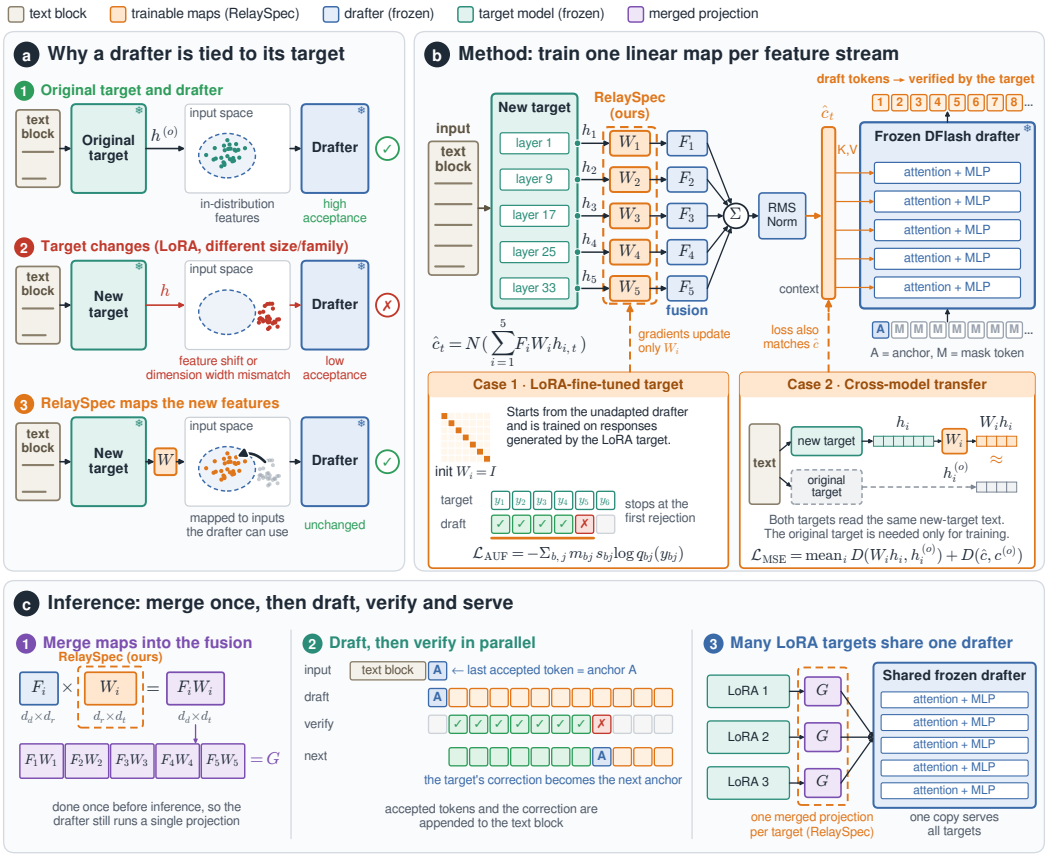

Figure 1: RelaySpec overview. (a) A feature-conditioned drafter trained for one target degrades when the target is fine-tuned and fails when it is replaced. (b) RelaySpec trains only the linear maps W_i in front of the frozen fusion blocks F_i and merges them into the fusion projection for inference. For the Qwen3-8B target with the Qwen3-4B drafter, the new-target, original-target and drafter widths are $d_t = 4096$ and $d_r = d_d = 2560$. Details: Appendices B.1 and I.1.

This dependence degrades the drafter when a LoRA adapter (Hu et al., 2022) is added to the target model and can make it incompatible when the target model itself changes. The hidden states change, sometimes in width, and so do the continuations the drafter must predict. Retraining target-specific drafters from scratch increases training and maintenance costs. Can we instead reuse an existing drafter by adapting only its target-feature inputs for cases where no native drafter for the new target exists?

We introduce RelaySpec, which learns one linear map per target feature stream before the drafter's original fusion projection (Figure 1). Only these maps are trained, and the target and drafter remain frozen. Before inference, we multiply each map into its corresponding fusion block, so the drafter still runs a single projection with no separate mapping operation.

RelaySpec uses training signals that match the change in the feature interface. For a LoRA-fine-tuned target, identity-initialized maps preserve the existing interface and receive token supervision. For cross-model transfer, the maps are trained to reconstruct the original target's features and fused context. Figure 2 shows the resulting throughput. Our contributions are as follows.

- **Input-only adaptation.** We adapt a feature-conditioned drafter by training only per-stream linear maps, which merge into its fusion projection.
- **LoRA and cross-model transfer.** RelaySpec speeds up LoRA-fine-tuned targets by up to 2.09× over the unadapted drafter, ahead of SD^2 (Berdoz et al., 2026), and reaches 94–

100% of native throughput on Qwen3-8B with in-domain fitting data from a few thousand trajectories.

- **Generalization.** The maps work with DFlash and EAGLE-3 drafters and on two GPU architectures.

2 RELATED WORK

RelaySpec builds upon feature-conditioned drafters like DFlash (Chen et al., 2026) and EAGLE-3 (Li et al., 2025). When the target model changes, the usual remedy is to retrain or distill the drafter (Zhou et al., 2024), or to update it online from serving traffic (Liu et al., 2024). Domain-specific drafter training can also raise acceptance on its own (Hong et al., 2025). SD^2 (Berdoz et al., 2026) and OmniDraft (Ramakrishnan et al., 2025) adapt a pretrained drafter, but they still update its weights. PARD (An et al., 2026a) and the target-independent mode of PARD-2 (An et al., 2026b) avoid per-target training with drafters that read no target features, whereas Osprey (Bie et al., 2026) pretrains a target-agnostic drafter and still adapts it to each target. Text-level bridges such as universal assisted generation (Korat et al., 2024), TokenTiming (Xiao et al., 2026) and the algorithms of Timor et al. (2025) allow drafting across different vocabularies. RelaySpec instead keeps the drafter frozen and trains only its input maps (Table 1).

These maps build on evidence that a single linear layer can connect independently trained networks (Bansal et al., 2021). Linear maps can also transfer features between language models (Chen et al., 2025) and align their concept representations (Huang et al., 2025). RepSpec adds linear structure while training a new drafter and merges it for inference (Huo et al., 2026), whereas RelaySpec merges its maps into an existing frozen drafter. Because this drafter stays shared, RelaySpec also fits multi-LoRA serving systems such as S-LoRA (Sheng et al., 2024) and Punica (Chen et al., 2024).

Table 1: Methods that adapt a drafter to a new target. The "Shown to support" columns report what each paper demonstrates.

Method	Trained for a new target	Drafter frozen	Shown to support a different-size target	Shown to support a different-family target
SD^2 (Berdoz et al., 2026)	Steering projections + whole drafter	No (a)	Yes (b)	No
PARD-2 (An et al., 2026b)	Drafter + feature projection, none in target-independent mode (c)	No (c)	Yes (c)	No
Osprey (Bie et al., 2026)	Feature projection, vocabulary, whole drafter	No	Yes	Yes
OmniDraft (Ramakrishnan et al., 2025)	Whole drafter or LoRA adapters, online	No	Yes	Yes
RelaySpec	**Linear input maps only**	**Yes**	**Yes**	**Yes (d)**

(a) The main method fine-tunes the drafter, and a frozen variant is a weaker ablation. We evaluate both (Section 4.6). (b) SD^2 does not adapt a drafter's existing feature input, so a target with a different input width is incompatible. (c) PARD-2 has two modes. The target-dependent mode trains a drafter with a feature projection for each target, and the target-independent mode trains nothing per target, reads no target features and serves Qwen3-8B/14B/32B unchanged (the "Yes"). (d) Via a text-level token bridge.

3 METHOD

RelaySpec adapts an existing drafter to a new target by training only linear maps on the target features the drafter receives, and the target and drafter weights stay frozen. We call the model that the drafter was originally trained for the *original target*, and the model we want to accelerate the *new target*. Results use one of two references. When the new target is the original target with a LoRA adapter, the reference is the *unadapted drafter*, the original drafter used as is. When the new target is a different model, the reference is that model's *native drafter*, which was originally trained for it.

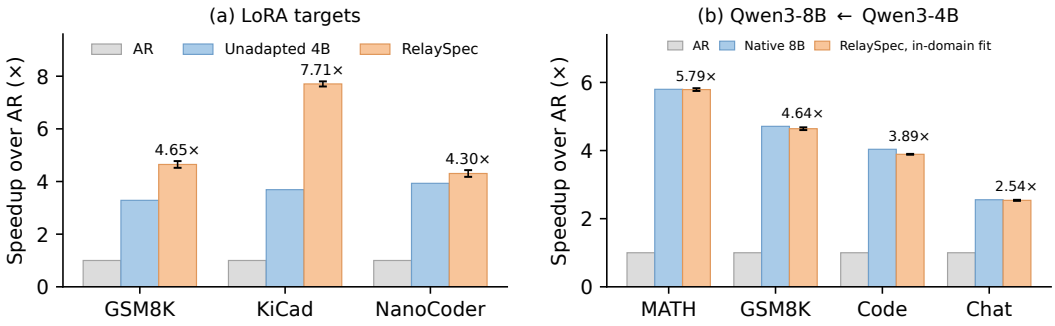

Figure 2: Single-request speedups on 128 prompts per workload. RelaySpec bars average three fits (error bars show the std. dev). Details: Tables 2 and 3 and Appendix C.

3.1 DFLASH AND THE FEATURE MAPS

DFlash begins each block with the last token already committed by the target, called the *anchor* (Chen et al., 2026). The drafter receives this known token followed by 15 masked positions, DFlash's default, and predicts the next 15 tokens in one pass. The anchor itself is not a proposal. The target checks the proposals in parallel.

DFlash also reads hidden states from five fixed target layers. Its original fusion projection combines those states, applies RMSNorm, and supplies the context as keys and values to every drafter layer (Chen et al., 2026). The anchor and masked positions provide queries, and only target states before the anchor are visible. Let K be the number of target layers the drafter reads ($K = 5$ for DFlash), and we call the output of each such layer a feature stream. At token position t, $h_{i,t}^{(o)} \in \mathbb{R}^{d_r}$ and $h_{i,t} \in \mathbb{R}^{d_t}$ are the hidden-state vectors of stream i in the original and new targets, whose hidden widths are d_r and d_t. The frozen fusion block $F_i \in \mathbb{R}^{d_d \times d_r}$, the columns of the fusion projection that read stream i, projects to drafter width d_d. A bias-free map $W_i \in \mathbb{R}^{d_r \times d_t}$ gives

$$\begin{aligned} c_t^{(o)} &= N\left(\sum_{i=1}^{K} F_i h_{i,t}^{(o)}\right), \\ \widehat{c}_t &= N\left(\sum_{i=1}^{K} F_i W_i h_{i,t}\right), \end{aligned} \tag{1}$$

where N is the frozen RMSNorm, $c_t^{(o)}$ is the original context vector, and $\widehat{c}_t$ is the context vector with our linear maps applied. The same W_i is applied at every token position. Only the W_i are trained, and everything else stays frozen.

Before inference, we multiply each map into its fusion block and concatenate the products to obtain

$$\begin{aligned} G &= [F_1 W_1 \ \cdots \ F_K W_K] \in \mathbb{R}^{d_d \times K d_t}, \\ \widehat{c}_t &= N(G[h_{1,t}; \ldots; h_{K,t}]). \end{aligned} \tag{2}$$

Thus one projection computes the mapped context, exactly as in the original drafter (Figure 1, with alternatives compared in Section 4.7). When the new target's width differs, the merged projection simply reads the new width, at the cost of a slightly wider projection.

3.2 TRAINING THE MAPS

LoRA-fine-tuned targets. A target LoRA leaves feature width unchanged. Each square map starts at $W_i = I$, preserving the unadapted drafter's initial computation with that target. Mapping the LoRA target's features back onto the base model's would steer the drafter toward the base model's continuations, so we use token supervision instead. We save responses generated by the LoRA target and sample anchors within them (Appendix I.4). The following saved tokens provide up to 15 proposal labels, and the anchor receives no loss. Verification stops at the first wrong proposal and

later positions can never be accepted, so we supervise the correct prefix and the first wrong proposal, following the accept-until-fail (AUF) loss (Yang & Li, 2026).

For block b and proposal position j, let q_{bj} be the drafter's token distribution and y_{bj} the saved target token at that position, which serves as its label. An anchor near the end of a response has fewer than 15 following tokens, so $m_{bj} \in \{0, 1\}$ marks whether the label exists. The variable u_{bj} is 1 if the prediction is correct or the label is missing, and s_{bj} keeps a position only if every earlier prediction in the block was correct.

$$u_{bj} = \mathbf{1}[\arg\max_v q_{bj}(v) = y_{bj} \text{ or } m_{bj} = 0], \quad s_{bj} = \text{stopgrad}\left(\prod_{k<j} u_{bk}\right),$$
$$\mathcal{L}_{\text{AUF}} = -\frac{\sum_{b,j} m_{bj} s_{bj} \log q_{bj}(y_{bj})}{\sum_{b,j} m_{bj} s_{bj}}. \tag{3}$$

Gradients pass through the frozen drafter to the maps.

Cross-model transfer. When the new target is a different model, its hidden states can differ in width and content. We initialize the maps with independent Xavier-uniform values (Glorot & Bengio, 2010). We generate responses with the new target, then run both targets on the saved text to collect paired features. Because both models read the same new-target text, the original target's features describe it in the form the drafter was trained to use. The original target supplies these references during fitting but is absent during adapted inference. For targets with a different tokenizer, paired features are aligned at shared text prefixes (Appendix I.1), and proposals pass through a standard text-level bridge (Korat et al., 2024; Timor et al., 2025) at each block boundary.

Let B be the number of examples in a batch and S_b a randomly chosen 25% of the prompt and response token positions in example b. Hidden-state norms vary widely across positions, so we compare features with the normalized squared error D. The loss matches both individual features and the fused normalized context.

$$D(u, v) = \frac{\|u - v\|_2^2}{\|v\|_2^2 + 10^{-6}}, \tag{4}$$

$$\mathcal{L}_{\text{MSE}} = \frac{1}{B} \sum_{b=1}^{B} \frac{1}{|S_b|} \sum_{t \in S_b} \left[\frac{1}{K} \sum_{i=1}^{K} D(W_i h_{i,bt}, h_{i,bt}^{(o)}) + D(\widehat{c}_{bt}, c_{bt}^{(o)}) \right]. \tag{5}$$

The two terms have equal weight, padding is excluded, and both targets and the drafter stay frozen. Unlike AUF, reconstruction requires no backward pass through the drafter transformer. We optimize the full objective iteratively. Section 4.7 shows that a closed-form solution of its feature term alone performs nearly as well and compares token supervision and reconstruction on both LoRA-fine-tuned targets and cross-model transfer. Appendix H.1 compares Xavier initialization with a rectangular-identity initialization.

3.3 Extension to EAGLE-3

EAGLE-3 reads three target feature streams and drafts autoregressively (Li et al., 2025). We use three maps before its existing feature projection, retain its drafter, and merge the maps as in Equation 2. For LoRA targets, the same AUF loss (Equation 3) covers up to eight token positions. Cross-model transfer uses the reconstruction loss of Equation 5.

4 Experiments

4.1 Experimental setup

Models. We use the released DFlash drafters for Qwen3-4B, Qwen3-8B and Llama3.1-8B (Chen et al., 2026; Yang et al., 2025; Grattafiori et al., 2024). For cross-model transfer, we write new target ← original target. For example, Qwen3-8B ← Qwen3-4B is the Qwen3-4B drafter adapted to Qwen3-8B. Appendix B gives model revisions, feature layers and widths.

Table 2: LoRA-fine-tuned targets, single-request setting. Speedup is relative to autoregressive decoding (AR), and the row below RelaySpec is relative to the unadapted DFlash drafter (std. dev at most 0.06). Details: Appendix C.1.

	Qwen3-4B + LoRA						Qwen3-8B + LoRA					
Method	GSM8K		KiCad		NanoCoder		GSM8K		CodeAlpaca		*Avg.*	
Temperature = 0	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
DFlash	3.28×	4.47	3.69×	4.82	3.93×	5.11	3.40×	4.76	4.11×	5.62	3.68×	4.96
RelaySpec	**4.65**×	**5.98**	**7.71**×	**10.07**	**4.30**×	**5.53**	**4.63**×	**6.34**	**4.31**×	**5.63**	**5.12**×	**6.71**
vs. DFlash	*1.41*×		*2.09*×		*1.09*×		*1.36*×		*1.05*×		*1.40*×	
Temperature = 1	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
DFlash	3.11×	4.12	2.76×	3.91	2.84×	3.48	2.74×	3.67	3.24×	**4.38**	2.94×	3.91
RelaySpec	**4.04**×	**5.19**	**4.74**×	**6.51**	**3.00**×	**3.59**	**3.48**×	**4.62**	**3.36**×	4.37	**3.73**×	**4.86**
vs. DFlash	*1.30*×		*1.72*×		*1.06*×		*1.27*×		*1.04*×		*1.28*×	

Table 3: Cross-model transfer with in-domain fitting data, single-request setting (std. dev of ×native at most 0.05). Details: Appendix C.2.

	Same dataset				Other dataset					
Fitting data	Chat		CodeAlpaca		NuminaMath				CodeAlpaca	
Evaluation	Chat		CodeAlpaca		MATH		GSM8K		LiveCodeBench	
Temperature = 0	×native	τ	×native	τ	×native	τ	×native	τ	×native	τ
Qwen3-8B ← Qwen3-4B	0.993	3.23	0.936	4.90	**0.999**	**7.99**	0.985	6.23	0.964	5.33
Qwen3-4B ← Qwen3-8B	0.903	3.19	0.905	4.92	**0.965**	**7.66**	0.960	5.98	0.882	5.50
Llama3.1-8B ← Qwen3-4B	0.665	2.72	0.761	3.68	0.946	**4.63**	0.710	3.22	**0.950**	4.49
Temperature = 1	×native	τ	×native	τ	×native	τ	×native	τ	×native	τ
Qwen3-8B ← Qwen3-4B	**1.037**	2.66	0.999	4.20	1.023	**5.93**	1.032	5.42	1.013	4.06
Qwen3-4B ← Qwen3-8B	**0.966**	2.54	0.907	3.97	0.922	**5.62**	0.882	4.82	0.828	3.61
Llama3.1-8B ← Qwen3-4B	0.703	1.89	0.841	**3.32**	0.922	1.72	0.768	2.02	**0.931**	2.85

Training. For LoRA targets, we train the maps for one epoch on 4,096 rollouts generated by the target, with AUF loss (Equation 3) using up to 32 anchors per rollout. For cross-model transfer, we train them for three epochs on 16,384 trajectories that the new target generates, with the reconstruction loss (Equation 5). We chose 32 AUF anchors per rollout and 25% of positions for reconstruction by trading throughput against fitting time. Using 128 anchors adds 4.3–6.7% throughput but fits 2.5–3.5× slower (Appendix I.4), and supervising 50% or 100% of positions changes throughput by -3.7 to $+2.2$% (Appendix I.3). We train on same-domain, same-dataset; same-domain, different-dataset; and cross-domain data (to measure how far one fit carries beyond its domain); see Tables 3 and 4. Every main configuration is fit on three training subsets (different data-selection seeds, same optimization seed). Appendices C.1 and I.1 give the exact recipes and data splits.

Evaluation. Unless stated otherwise, we use a *single-request setting*, with BF16 Transformers inference (Wolf et al., 2020) on one L40S GPU and 128 held-out prompts per workload with a 2,048-token output cap. We test greedy decoding (temperature 0) and temperature-1 sampling (top-p 1). We evaluate each fit with 3 GPU timing passes and report the overall mean (Appendix C).

Metrics. Throughput is averaged over requests. The acceptance length τ is the average number of tokens emitted per verification step, including the token contributed by the target. Serving throughput is all returned tokens divided by wall time, HM is the harmonic mean over workloads, and ×native is throughput relative to the native drafter. Appendix B.1 gives the timing details.

4.2 Main results

RelaySpec speeds up the released drafter on all five LoRA-fine-tuned targets (Table 2), from 1.05× on Qwen3-8B + CodeAlpaca (Chaudhary, 2023) to 2.09× on Qwen3-4B + KiCad (AbijahKaj,

Table 4: Cross-model transfer with cross-domain fitting data (std. dev of ×native at most 0.04). Details: Appendices C.1 and C.2.

Fitting data	NuminaMath					
Evaluation	LiveCodeBench		Chat		CodeAlpaca	
Temperature = 0	×native	τ	×native	τ	×native	τ
Qwen3-8B ← Qwen3-4B	0.839	**4.76**	**0.897**	2.89	0.777	3.99
Qwen3-4B ← Qwen3-8B	0.757	**4.90**	**0.839**	2.83	0.726	3.85
Llama3.1-8B ← Qwen3-4B	**0.653**	**3.05**	0.527	2.06	0.512	2.38
Temperature = 1	×native	τ	×native	τ	×native	τ
Qwen3-8B ← Qwen3-4B	0.878	**3.66**	**0.932**	2.41	0.830	3.42
Qwen3-4B ← Qwen3-8B	0.679	**3.24**	**0.874**	2.27	0.719	3.16
Llama3.1-8B ← Qwen3-4B	**0.684**	**2.19**	0.596	1.58	0.587	**2.19**

Table 5: Overall Spec-Bench speedup over AR for Qwen3-8B ← Qwen3-4B. Details: Appendix B.1.

Method	Speedup
Native drafter	**2.51**×
RelaySpec	2.49×
vs. native DFlash	*0.994*×

2026), and increases τ almost everywhere. The gains hold under temperature-1 sampling (1.04–1.72×), even though AUF trains on greedy targets. They are smallest on code, likely because the released drafter was already trained on CodeAlpaca (Chen et al., 2026). In cross-model transfer, maps fitted on NuminaMath (Li et al., 2024a) bring both Qwen drafters close to native speed on MATH (Hendrycks et al., 2021) and GSM8K (Cobbe et al., 2021) (Table 3, and Appendix A.2 for new MATH prompts), but fall behind on code (LiveCodeBench (Jain et al., 2025)) and chat (UltraFeedback (Cui et al., 2024), Table 4), which Section 4.3 traces to the fitting data. A transferred Llama3.1-8B drafter also gives Llama3.2-3B (Meta AI, 2024), which has no DFlash drafter, a 1.61–2.35× speedup over autoregressive decoding (Appendix B.2). On Spec-Bench (Xia et al., 2024), whose 480 prompts span six task types, one set of maps fitted on mixed math, code, and chat data reaches 99.4% of native throughput (Table 5).

4.3 How far does a fit carry across domains?

Maps fitted only on math data keep most of the throughput of in-domain fits on code and chat (Tables 3 and 4). For Qwen3-8B ← Qwen3-4B they reach 83.9% and 89.7% of native throughput on LiveCodeBench and Chat. Because the maps never see code or chat, this suggests that they learn a largely domain-general correspondence between the two models' hidden-state spaces, which in-domain data then refines. Across model families the retention is lower (67–79% of the in-domain fits and 51–65% of native throughput). The tokenizer bridge takes only 2.0–4.1% of request time (Appendix A.3), so most of this gap comes from lower acceptance.

4.4 How much target-specific supervision is needed?

Figure 3 varies the number of training examples and anchors. With only 16 training examples, the maps raise LoRA-target throughput by up to 12.3% over the unadapted drafter, and 1,024 NuminaMath trajectories bring cross-model transfer to 97.0% of native throughput on MATH.

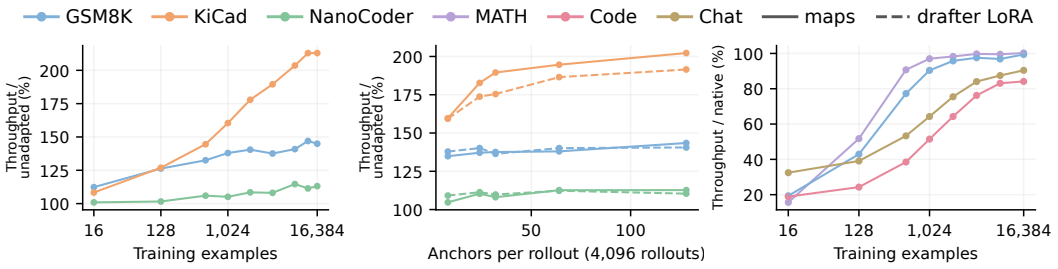

Figure 3: Supervision scaling (single-request). The left and middle panels vary training examples and anchors for the LoRA-fine-tuned Qwen3-4B targets. Dashed lines train LoRA on the drafter instead of maps. The right panel shows cross-model transfer Qwen3-8B ← Qwen3-4B with NuminaMath fits and 25% of positions supervised. Details: Appendices I.3 and I.4.

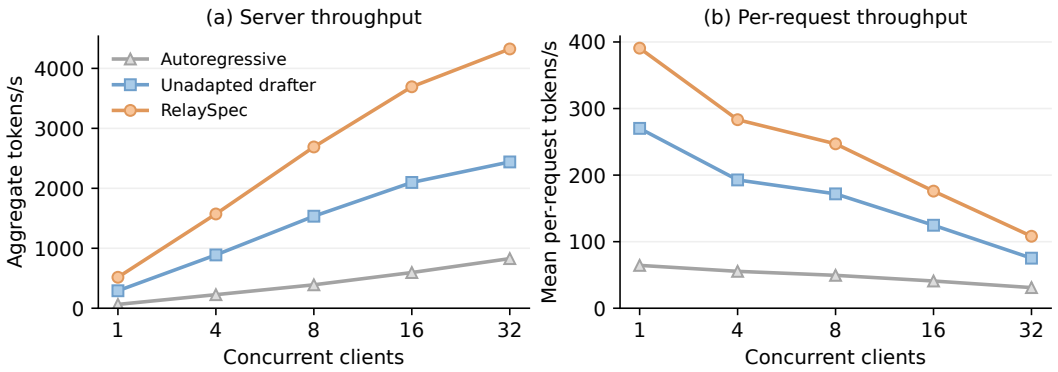

Figure 4: vLLM serving of three LoRA-fine-tuned Qwen3-4B targets that share one frozen DFlash drafter (greedy, one L40S, 384 requests per point). Details: Appendix E.1.

Table 6: RelaySpec versus SD^2 and OmniDraft on LoRA-fine-tuned Qwen3-4B targets (single-request setting, fresh prompts). Values are speedups over the unadapted DFlash drafter, mean $\pm$ std. dev over three seeds. Details: Appendix A.1.

Method	Params (M)	GSM8K	KiCad	NanoCoder	HM
RelaySpec (five maps)	32.8	**1.339**$\pm$**.012**	**2.124**$\pm$**.041**	**1.092**$\pm$**.009**	**1.406**
SD^2 (steering + drafter fine-tuning)	626.6	1.293$\pm$.018	1.927$\pm$.029	1.041$\pm$.029	1.332
SD^2, frozen drafter (steering only)	44.6	1.218$\pm$.018	1.758$\pm$.047	0.991$\pm$.027	1.250
OmniDraft (drafter LoRA)	9.2	1.314$\pm$.035	1.393$\pm$.015	1.031$\pm$.020	1.225

4.5 What does adaptation cost?

With the chosen recipes, a fit takes 4.6 minutes on average on two L40S GPUs for LoRA targets (0.15 GPU-hours) and 18.0 minutes for cross-model transfer (0.60 GPU-hours) excluding rollout generation and feature extraction (Appendix F).

4.6 Comparison with SD^2 and OmniDraft

RelaySpec is faster than both on every task (Table 6), with an HM of 1.406 against 1.332 for SD^2 and 1.225 for OmniDraft.

4.7 Architecture and objective ablations

Architecture ablation. We compare the five per-stream linear maps with an additive weight update of the fused projection, rank-128 LoRA on the drafter transformer, one joint map over all streams, and rank-128 maps without an identity path. Averaged over two training subsets, the five maps have the highest HM (Appendix I.1).

A nonlinear map. Adding a nonlinear branch to each map, an MLP $W_i h + B_i \,\mathrm{SiLU}(A_i h)$ (Elfwing et al., 2018) with bottleneck width 256, gives no consistent benefit. It trails the linear maps on five of seven workloads and cannot be merged into the fusion projection (Appendix D.1).

Which loss trains the maps. For LoRA targets, AUF has the highest HM (1.376 against 1.353 for decaying cross-entropy (Chen et al., 2026) and 1.052 for reconstruction), while for cross-model transfer reconstruction is far better (92.1% against 60.4% of native throughput for AUF, Appendix I.2).

A closed-form fit. The feature term of Equation 5 is a least-squares problem per stream with solution $W_i = C_i A_i^{\dagger}$. On the in-domain features, it keeps 97.0–99.6% of the iterative maps' throughput (Appendix C.2).

4.8 Generalization to a second drafter family

With three input maps on a frozen EAGLE-3 drafter, throughput on the LoRA-fine-tuned Qwen3-4B targets rises 1.47–2.05× over the unadapted drafter (Table 13 in Appendix D.2), and the Qwen3-4B EAGLE-3 drafter transferred to Qwen3-8B matches or beats the native one (103–108% of native throughput, Table 14).

4.9 Serving many targets from one drafter

In vLLM (Kwon et al., 2023), all LoRA variants share one frozen DFlash drafter, with each request routed to its map. The gains persist under batched inference (Figure 4). At 32 concurrent clients, aggregate throughput is 4,324 tokens/s, 1.77× that of the unadapted drafter and 5.2× that of AR.

4.10 A second GPU architecture

On a DGX Spark (NVIDIA GB10, Blackwell), the same Qwen3-4B LoRA-target maps speed up the unadapted drafter by 1.40×, 2.07× and 1.11× on GSM8K, KiCad and NanoCoder, matching the L40S results (1.41×, 2.09× and 1.09×, see Appendix A.4).

5 Analysis

Which loss to use. A new model, such as Qwen3-8B in place of Qwen3-4B, mostly generates the same text as the original target, but its hidden states look different. Reconstruction targets this mismatch by mapping the new hidden states onto those the drafter was trained on (Appendix G). A LoRA fine-tune instead was trained to generate different text, and mapping its hidden states back would make the drafter predict the original model's text. It therefore needs token supervision (AUF), as Section 4.7 confirms.

Why a few thousand examples are enough. For a new model, learning the maps is essentially a linear regression problem, and solving it directly by least squares gives 97.0–99.6% of the trained maps' throughput. Regression needs data in proportion to its number of inputs, not the model's size, and each example provides about 330 positions to learn from. On MATH, throughput stops improving at about 80–160 positions per input dimension (Figure 3 and Appendix G). On Code and Chat it keeps improving, so their gap is a result of fitting on math only, and in-domain fitting data largely closes that gap (Table 3).

The learned correction is low-rank. For the GSM8K and NanoCoder LoRA targets, keeping only 256 of the 2,560 directions of each correction $W_i - I$ preserves 93–98% of its gain in acceptance length. The maps thus learn a far simpler change than their size allows (Appendix H).

6 Limitations

RelaySpec assumes that a trained drafter already exists for a related model. Our experiments cover targets of up to 8B parameters and LoRA fine-tuning only, not full fine-tuning, and transfer across model families remains the hardest case.

7 Conclusion

RelaySpec shows that a feature-conditioned drafter need not be retrained when its target changes. Training only merged linear input maps speeds up LoRA targets by 1.40× on average, and a Qwen3-4B drafter transferred to Qwen3-8B reaches 99.4% of native throughput on Spec-Bench. Each adaptation takes at most about 5 L40S GPU-hours, a small fraction of the compute needed to train a native drafter (Chen et al., 2026).

AI USE STATEMENT

We used generative AI tools (LLM coding and writing assistants) to implement experiment and evaluation code, to process logs into tables and figures, and to edit the manuscript for readability. We did not use them to propose the method, formulate hypotheses or design the experiments. The authors reviewed all AI-assisted code and text, and take full responsibility for the content.

ETHICS STATEMENT

This work adheres to the ICLR Code of Ethics. It uses only publicly released models, adapters and datasets under their licenses and involves no human subjects or personal data. Speculative decoding accelerates inference while preserving the target model's output distribution, so RelaySpec inherits, and does not add to, the risks of the models it accelerates.

REPRODUCIBILITY STATEMENT

All model revisions, training recipes and evaluation protocols needed to reproduce our results are given in the appendix.

REFERENCES

AbijahKaj. KiCad netlist SFT dataset. Hugging Face dataset `AbijahKaj/kicad-netlist-sft-dataset`, 2026. URL `https://huggingface.co/datasets/AbijahKaj/kicad-netlist-sft-dataset`.

Zihao An, Huajun Bai, Ziqiong Liu, Dong Li, and Emad Barsoum. PARD: Accelerating LLM inference with low-cost PARallel draft model adaptation. In *International Conference on Learning Representations*, 2026a. URL `https://openreview.net/forum?id=XbOyv7iVGL`.

Zihao An, Taichi Liu, Ziqiong Liu, Dong Li, Ruofeng Liu, and Emad Barsoum. PARD-2: Target-aligned parallel draft model for dual-mode speculative decoding. *arXiv preprint arXiv:2605.08632*, 2026b. URL `https://arxiv.org/abs/2605.08632`.

Yamini Bansal, Preetum Nakkiran, and Boaz Barak. Revisiting model stitching to compare neural representations. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL `https://proceedings.neurips.cc/paper/2021/hash/01ded4259d101feb739b06c399e9cd9c-Abstract.html`.

Frédéric Berdoz, Peer Rheinboldt, and Roger Wattenhofer. Steering pretrained drafters during speculative decoding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 30067–30075, 2026. doi: 10.1609/aaai.v40i36.40255. URL `https://ojs.aaai.org/index.php/AAAI/article/view/40255`.

Fengxiang Bie, Yuqing Jian, Yifan Yu, Zhongzhu Zhou, Zelei Shao, Ben Athiwaratkun, Shuaiwen Leon Song, Chenfeng Xu, Xiaoxia Wu, and Tianyi Zhang. Osprey: Target-agnostic pretraining makes stronger drafters in speculative decoding. *arXiv preprint arXiv:2609.09338*, 2026. URL `https://arxiv.org/abs/2609.09338`. Author-reported acceptance at EMNLP 2026.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 5209–5235. PMLR, 2024. URL `https://proceedings.mlr.press/v235/cai24b.html`.

Rui Cen, QiangQiang Hu, Hong Huang, Hong Liu, Song Liu, Xin Luo, Lin Niu, Yifan Tan, Decheng Wu, Linchuan Xie, Rubing Yang, Guanghua Yu, and Jianchen Zhu. AngelSlim: A more accessible, comprehensive, and efficient toolkit for large model compression. *arXiv preprint arXiv:2602.21233*, 2026. URL `https://arxiv.org/abs/2602.21233`.

Sahil Chaudhary. Code Alpaca: An instruction-following LLaMA model for code generation. GitHub repository, https://github.com/sahil280114/codealpaca, 2023.

Alan Chen, Jack Merullo, Alessandro Stolfo, and Ellie Pavlick. Transferring linear features across language models with model stitching. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.nips.cc/paper_files/paper/2025/hash/4569a868e7aa891248832ec08445d071-Abstract-Conference.html.

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. URL https://arxiv.org/abs/2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2602.06036.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant LoRA serving. In *Proceedings of Machine Learning and Systems*, volume 6, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/054de805fcceb78a201f5e9d53c85908-Abstract-Conference.html.

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. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, Zhiyuan Liu, and Maosong Sun. UltraFeedback: Boosting language models with scaled AI feedback. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 9722–9744. PMLR, 2024. URL https://proceedings.mlr.press/v235/cui24e.html.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018. URL https://arxiv.org/abs/1702.03118.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256. PMLR, 2010. URL https://proceedings.mlr.press/v9/glorot10a.html.

Aaron Grattafiori et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Fenglu Hong, Ravi Raju, Jonathan Lingjie Li, Bo Li, Urmish Thakker, Avinash Ravichandran, Swayambhoo Jain, and Changran Hu. Training domain draft models for speculative decoding: Best practices and insights. *arXiv preprint arXiv:2503.07807*, 2025. URL https://arxiv.org/abs/2503.07807. SCOPE Workshop at ICLR 2025.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2106.09685.

Youcheng Huang, Chen Huang, Duanyu Feng, Wenqiang Lei, and Jiancheng Lv. Cross-model transferability among large language models on the Platonic representations of concepts. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3686–3704. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.185. URL https://aclanthology.org/2025.acl-long.185/.

Feiye Huo, Jianchao Tan, Jiahao Liu, Zixu Jiang, Jiacheng Li, Jingang Wang, Xunliang Cai, and Shengli Sun. RepSpec: Structural re-parameterized draft model training for speculative decoding. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/d70ea003729b440b89a2f958a5554c1f-Abstract-Conference.html.

Naman Jain, King Han, 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*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/94074dd5a072d28ff75a76dabed43767-Abstract-Conference.html.

Daniel Korat, Oren Pereg, Moshe Berchansky, Jonathan Mamou, Joao Gante, Lewis Tunstall, Nadav Timor, and Moshe Wasserblat. Universal assisted generation: Faster decoding with any assistant model. Hugging Face blog, 2024. URL https://huggingface.co/blog/universal_assisted_generation.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023. URL https://doi.org/10.1145/3600006.3613165.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 19274–19286. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. NuminaMath. Hugging Face dataset, https://huggingface.co/datasets/AI-MO/NuminaMath-CoT, 2024a.

Shenggui Li, Chao Wang, Yikai Zhu, Yubo Wang, Fan Yin, Shuai Shi, Yefei Chen, Xiaomin Dong, Qiaoling Chen, Jin Pan, Ji Li, Laixin Xie, Yineng Zhang, Lei Yu, Yonggang Wen, Ivor Tsang, and Tianwei Zhang. SpecForge: A flexible and efficient open-source training framework for speculative decoding. *arXiv preprint arXiv:2603.18567*, 2026. URL https://arxiv.org/abs/2603.18567.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 28935–28948. PMLR, 2024b. URL https://proceedings.mlr.press/v235/li24bt.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/c7b5a35ea98b62512a869c19ea7b03cb-Abstract-Conference.html.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 31131–31146. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24y.html.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1711.05101.

Meta AI. Llama 3.2: Revolutionizing edge AI and vision with open, customizable models. Meta AI blog, 25 September 2024, 2024. URL https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

Ramchalam Kinattinkara Ramakrishnan, Zhaocong Yuan, Shaojie Zhuo, Chen Feng, Yicheng Lin, Chenzheng Su, and Xiaopeng Zhang. OmniDraft: A cross-vocabulary, online adaptive drafter for on-device speculative decoding. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/3c2fe1417eed1c6ff9acf169617981ea-Paper-Conference.pdf.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. SLoRA: Scalable serving of thousands of LoRA adapters. In *Proceedings of Machine Learning and Systems*, volume 6, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/906419cd502575b617cc489a1a696a67-Abstract-Conference.html.

Nadav Timor, Jonathan Mamou, Daniel Korat, Moshe Berchansky, Gaurav Jain, Oren Pereg, Moshe Wasserblat, and David Harel. Accelerating LLM inference with lossless speculative decoding algorithms for heterogeneous vocabularies. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 59598–59620, 2025. URL https://proceedings.mlr.press/v267/timor25a.html.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, 2020. URL https://aclanthology.org/2020.emnlp-demos.6/.

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. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 7655–7671, 2024. doi: 10.18653/v1/2024.findings-acl.456. URL https://aclanthology.org/2024.findings-acl.456/.

Sibo Xiao, Jinyuan Fu, Zhongle Xie, and Lidan Shou. TokenTiming: A dynamic alignment method for universal speculative decoding model pairs. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 42798–42812. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1983. URL https://aclanthology.org/2026.acl-long.1983/.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

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

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*,

2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/8766fbc68e1ed1cdef712ce273e0a363-Abstract-Conference.html.

A Additional results

A.1 Comparison with SD^2 and OmniDraft

We port both methods to the Qwen3-4B DFlash drafter. SD^2 (Berdoz et al., 2026) adds a steering vector computed from the target features to every drafter MLP. Its main method also fine-tunes all 582.0M drafter parameters, while its ablation keeps the drafter frozen. OmniDraft (Ramakrishnan et al., 2025) trains a rank-32 LoRA on the drafter with a reverse-KL loss. All methods use the same 4,096 rollouts per task, 512 updates and three seeds, and are evaluated on 128 held-out prompts per task that appear nowhere else in our experiments. The held-out loss of SD^2 with drafter fine-tuning stops improving after about 160 updates, and more updates would not help it. RelaySpec is 3.6–10.2% faster than SD^2 (HM 1.406 against 1.332) and 9.9–20.8% faster than its frozen-drafter ablation, while training 19$\times$ fewer parameters than SD^2. It is also faster than OmniDraft on every task (HM 1.225, and 2.124 against 1.393 on KiCad).

A.2 Held-out prompt cohort

The evaluation prompts of Tables 3 and 4 were also used during method development. To check that they do not favor RelaySpec, we evaluate the three math fits of Qwen3-8B $\leftarrow$ Qwen3-4B on 128 new MATH and 128 new Chat prompts that appear nowhere in our data. The fits reach 1.003–1.007 of native throughput on MATH and on average 0.920 of native throughput on Chat, in line with the original prompt sets (0.999 and 0.897).

A.3 Tokenizer-bridge cost

Llama3.1-8B and Qwen3-4B use different tokenizers. At every draft block, the bridge (Korat et al., 2024; Timor et al., 2025) decodes the committed text and re-encodes it with the drafter's tokenizer, then converts the drafter's proposals into target tokens. Timing every tokenizer call shows that the bridge takes 2.0–4.1% of request time; removing it would raise throughput by at most 1.04$\times$ (Table 7). The remaining gap to the native drafter therefore comes from lower acceptance length (Appendix C.1).

Table 7: Tokenizer-bridge cost for Llama3.1-8B $\leftarrow$ Qwen3-4B over the three fits. f is the bridge share of request time, and $1/(1-f)$ bounds the speedup from removing it.

Workload	Mean output tokens	Tokenizer ms/token	Share of request time (%)	Bound $1/(1-f)$
MATH	652	0.258	3.26–3.45	1.036
GSM8K	216	0.213	1.98–2.01	1.021
Code	400	0.488	4.04–4.08	1.043
Chat	507	0.544	3.15–3.21	1.033

A.4 Other hardware

On a DGX Spark (NVIDIA GB10, Blackwell), the gain over the unadapted drafter matches the L40S (Table 8). Speedups over AR are larger because single-request decoding is more bandwidth-bound on this device.

B Models and evaluation protocol

B.1 Models, datasets and timing

We call the cross-model pairs T1 (Qwen3-8B $\leftarrow$ Qwen3-4B), T2 (Qwen3-4B $\leftarrow$ Qwen3-8B), T3 (Llama3.2-3B $\leftarrow$ Llama3.1-8B) and T4 (Llama3.1-8B $\leftarrow$ Qwen3-4B). Each has five $d_r \times d_t$ maps

Table 8: Qwen3-4B LoRA targets on a DGX Spark (NVIDIA GB10, Blackwell, 128 GB unified memory) with the same checkpoints, prompts, 2,048-token cap and evaluator as Table 2. RelaySpec averages the three fits (± std. dev).

		Speedup over AR				RelaySpec ×unadapted
Workload	AR tok/s	Unadapted	RelaySpec	L40S RelaySpec	GB10	L40S
GSM8K	20.10	3.96	5.55	4.65	1.399±.008	1.41±.04
KiCad	19.36	4.06	8.39	7.71	2.066±.004	2.09±.03
NanoCoder	20.13	4.68	5.21	4.30	1.113±.002	1.09±.03

with 52.4M parameters in total (62.9M for T3). The hidden widths are 2,560 for Qwen3-4B, 4,096 for Qwen3-8B and Llama3.1-8B, and 3,072 for Llama3.2-3B. Qwen3-4B and Qwen3-8B use feature layers $[1, 9, 17, 25, 33]$, Llama3.1-8B $[1, 8, 15, 22, 29]$ and Llama3.2-3B $[1, 7, 13, 19, 25]$. LoRA-target maps are square, with 32.8M parameters for Qwen3-4B and 83.9M for Qwen3-8B, and the task LoRA is merged into the target for single-request evaluation. We use Qwen3-4B at revision `1cfa9a7`, Qwen3-8B at `b968826`, their DFlash drafters at `b74e3a3` and `9b41424`, and the Unsloth Llama-3.1-8B- and Llama-3.2-3B-Instruct models (Grattafiori et al., 2024; Meta AI, 2024) at `4699cc7` and `006f5dc`.

Datasets. The LoRA-target workloads use GSM8K (Cobbe et al., 2021), the KiCad netlist dataset (AbijahKaj, 2026) and CodeAlpaca (Chaudhary, 2023) (the `HuggingFaceH4/CodeAlpaca_20K` release), which provides the NanoCoder and Qwen3-8B CodeAlpaca prompts. Cross-model maps are fit on NuminaMath (Li et al., 2024a), CodeAlpaca or UltraFeedback (Cui et al., 2024) prompts and evaluated on MATH (Hendrycks et al., 2021), GSM8K, CodeAlpaca, LiveCodeBench (Jain et al., 2025) and UltraFeedback chat prompts. Spec-Bench (Xia et al., 2024) adds 480 prompts from six task types, and its mixed fit uses 16,384 trajectories, a third each from NuminaMath, CodeAlpaca and UltraFeedback chat, with the unchanged cross-model recipe.

Timing. Single-request runs use PyTorch (Paszke et al., 2019) 2.13, Transformers 5.16.1 and DFlash 0.1.0 with BF16 and SDPA, block size 16 (AR uses the block-size-1 path) and Qwen thinking disabled. The timer covers the whole generation call, including prefill, and closes with a CUDA synchronization. Setup and warmup are excluded. Throughput is the mean over requests of output tokens per second, and τ is the number of tokens emitted per verification step, including the target's correction or bonus token.

B.2 Llama3.2-3B transfer without a native control

Llama3.2-3B has no DFlash drafter, so Table 9 compares the transferred Llama3.1-8B drafter only with AR.

Table 9: Llama3.2-3B ← Llama3.1-8B (T3). Each row has 128 prompts.

Workload	AR (tokens/s)	RelaySpec (tokens/s)	RelaySpec/AR	τ
MATH	61.90	145.43	2.35×	4.780
GSM8K	64.45	140.60	2.18×	4.159
Code	63.09	118.57	1.88×	3.440
Chat	64.72	104.43	1.61×	2.974

C Repeated fits, sampled decoding, and in-domain fitting

C.1 Training-subset repeats and sampling

Each main configuration is fit on three training subsets of the same size, drawn with different selection seeds from the same pool after excluding held-out prompts. The subsets overlap but are never identical. Each target generates its own responses (capped at 4,096 tokens). Initialization, minibatch

order and anchor or position sampling keep their original seeds, so the spread across fits measures sensitivity to the fitting data. Tables 2–4 weight the three fits equally, and the std. dev in their captions is taken over the three per-fit ratios. The median pass-to-pass variation of throughput is 0.5%. Across all cross-model pairs and workloads, including the tokenizer-bridged pair, the throughput ratio to the native drafter divided by the τ ratio lies between 0.97 and 1.05. Throughput differences therefore come from acceptance length. Some early greedy timings of T2 also included building the prompt tensor on the GPU, and using only the later timings changes its ratios by at most 0.006.

Sampling. Temperature-1 runs use top-p 1 and top-k 0, reset the sampling seed for each prompt, and use the upstream DFlash rejection sampler. For Llama3.1-8B ← Qwen3-4B, the retokenized proposals form a deterministic target-vocabulary proposal that the target accepts with the point-mass rule (Korat et al., 2024; Timor et al., 2025), which preserves the target's sampling distribution. AUF fits are not retrained for sampling, and the temperature-1 AR baseline is a separate run with the same prompts and seeds.

C.2 In-domain fitting and the closed-form baseline

In-domain fits use 16,384 CodeAlpaca or UltraFeedback ShareGPT training prompts, disjoint from the 128 evaluation prompts, with the cross-model recipe and three subsets per domain. Table 3's LiveCodeBench column evaluates the CodeAlpaca fits, and Table 4's CodeAlpaca column the NuminaMath fits, to separate the fitting domain from the evaluation prompts.

The closed form solves the feature term of Eq. 5 exactly for each stream, giving $W_i = C_i A_i^{\dagger}$ with $A_i = \sum_{b,t} a_{bti} h_{i,bt} h_{i,bt}^{\top}$ and $C_i = \sum_{b,t} a_{bti} h_{i,bt}^{(o)} h_{i,bt}^{\top}$, where the weights $a_{bti} = 1/(|S_b|(\|h_{i,bt}^{(o)}\|_2^2 + 10^{-6}))$ reproduce the normalization of Eq. 4 and the sums run over the same selected positions as the iterative fit. We accumulate A_i and C_i in FP64 and merge and export the maps exactly as for iterative fits. The closed form keeps 97.0–99.6% of the iterative maps' throughput (Table 10), and its fused-context error is in Table 17.

Table 10: Closed-form feature-only maps against the iterative in-domain fits of Table 3, fitted on the same cohort.

Pair	Workload	Iterative ×native	Closed-form ×native	Closed form, % of iterative
T1	CodeAlpaca	0.936	0.933	99.6
T1	Chat	0.993	0.988	99.0
T2	CodeAlpaca	0.905	0.888	97.0
T2	Chat	0.903	0.897	98.6
T4	CodeAlpaca	0.761	0.748	98.3
T4	Chat	0.665	0.645	97.2

Table 11 lists the speedups over AR that Figure 2(b) plots.

Table 11: Speedups over AR behind Figure 2(b), Qwen3-8B ← Qwen3-4B with in-domain fitting data. The Code bars are LiveCodeBench with the CodeAlpaca fits. RelaySpec averages three fits (± std. dev).

Workload	Fitting data	AR (tokens/s)	Native drafter	RelaySpec
MATH	NuminaMath	37.84	5.80×	5.79×±0.04
GSM8K	NuminaMath	37.76	4.71×	4.64×±0.04
LiveCodeBench	CodeAlpaca	37.50	4.04×	3.89×±0.01
Chat	Chat	35.09	2.56×	2.54×±0.02

D NONLINEAR MAPS AND EAGLE-3

D.1 NONLINEAR MAPS

A nonlinear branch, $W_i h + B_i \,\mathrm{SiLU}(A_i h)$ (Elfwing et al., 2018) with bottleneck 256 and zero-initialized B_i, gives no consistent gain. It trails the linear maps on five of seven workloads (Table 12) and cannot be merged into the fusion projection.

Table 12: Linear maps versus maps with a nonlinear branch, same data, loss and schedule (single fits with their own timing measurement, so throughput differs slightly from Table 25, while τ matches). Throughput ratio is to the unadapted drafter for the LoRA targets and to the native drafter for Qwen3-8B ← Qwen3-4B.

		Throughput ratio		τ	
Setting	Workload	Linear	MLP	Linear	MLP
LoRA targets	GSM8K	**1.395**	1.381	**5.99**	5.95
	KiCad	1.855	**1.910**	11.26	**11.41**
	NanoCoder	**1.139**	1.126	**5.55**	5.52
Qwen3-8B ← Qwen3-4B	MATH	**1.002**	0.982	**7.98**	7.98
	GSM8K	**0.994**	0.981	6.24	**6.26**
	Code	0.842	**0.847**	4.74	**4.78**
	Chat	**0.904**	0.902	2.89	**2.90**

D.2 EAGLE-3 RESULTS

The EAGLE-3 drafters are AngelSlim's released checkpoints (Cen et al., 2026), `AngelSlim/Qwen3-4B_eagle3` at revision `fd331e59` and `AngelSlim/Qwen3-8B_eagle3` at `9629dfce`. For the LoRA targets, we train three identity-initialized 2560×2560 maps on EAGLE-3's feature layers $[2, 18, 33]$ with AUF over up to eight recurrent predictions (Table 13). For transfer to Qwen3-8B, three rectangular maps on the Qwen3-4B EAGLE-3 drafter are trained with the reconstruction recipe, and Table 14 compares them with the native Qwen3-8B EAGLE-3 drafter.

Table 13: EAGLE-3 with three input maps, on 128 prompts per workload. The reference is the unadapted drafter, and RelaySpec throughput averages its timing passes.

		Throughput (tokens/s)			τ	
Setting	Workload	Reference	RelaySpec	Ratio ×ref	Reference	RelaySpec
LoRA Qwen3-4B	GSM8K	71.19	119.18	1.67	3.347	5.695
	KiCad	73.82	151.54	2.05	3.794	7.363
	NanoCoder	81.77	119.86	1.47	3.852	5.668

Table 14: EAGLE-3 transfer from the Qwen3-4B drafter to Qwen3-8B (T1). The reference is the native Qwen3-8B EAGLE-3 drafter.

Setting	Workload	Reference (tokens/s)	RelaySpec (tokens/s)	Ratio
T1	MATH	72.18	77.99	1.08
T1	GSM8K	75.58	80.59	1.07
T1	Code	63.88	65.53	1.03
T1	Chat	61.82	63.41	1.03

E CONCURRENT SERVING

E.1 SERVING CONFIGURATION

Figure 4 uses one vLLM 0.28.0 engine (Kwon et al., 2023) on one L40S with BF16, CUDA graphs and 15 DFlash proposals per block. It serves the Qwen3-4B target with its GSM8K, KiCad and NanoCoder LoRAs and a single frozen drafter, and each request selects its target LoRA and the matching set of five maps. Each point has 384 requests (128 per target) with response caps of 2,048 tokens (8,192 for KiCad). Aggregate throughput divides all returned tokens by the wall time until the last request finishes.

F ADAPTATION COST AND OUTPUT LENGTH

Table 15 gives the end-to-end cost of two adaptations on L40S GPUs, from generating the target's trajectories to fitting the maps, excluding the pretrained drafter's own training. Table 16 gives the map fitting time of every configuration in Tables 2–4 on two L40S GPUs. On average, a fit takes 4.6 minutes for a LoRA target and 18.0 minutes for cross-model transfer.

Table 15: Adaptation cost of RelaySpec in L40S GPU-hours, measured end to end over target-generated trajectories, paired feature capture, and fitting on two GPUs.

	Data		GPU-hours			
Adaptation	Trajectories	Epochs	Generation	Features	Training	Total
RelaySpec, Qwen3-8B ← Qwen3-4B	16,384	3	3.36	1.13	0.55	5.04
RelaySpec, Qwen3-4B + KiCad LoRA	4,096	1	1.99	0.63	0.27	2.89

Table 16: Map fitting time on two L40S GPUs for every configuration of Tables 2–4, averaged over the fits on the second and third training subsets of Appendix C.1. Times exclude rollout generation and feature extraction.

	Qwen3-4B + LoRA			Qwen3-8B + LoRA		
LoRA targets	GSM8K	KiCad	NanoCoder	GSM8K	CodeAlpaca	*Avg.*
Updates	512	512	512	512	512	
Minutes	3.00	8.04	3.01	4.40	4.34	4.56
GPU-hours	0.100	0.268	0.100	0.147	0.145	0.152

Cross-model transfer	Qwen3-8B ← Qwen3-4B	Qwen3-4B ← Qwen3-8B	Llama3.1-8B ← Qwen3-4B	*Avg.*
Updates	6,144	6,144	6,144	
Minutes	16.38	18.73	18.80	17.97
GPU-hours	0.546	0.624	0.627	0.599

Output length and KiCad. KiCad responses are much longer than those of the other LoRA workloads (mean 5,040 against 153–218 tokens), but long outputs do not inflate its gain by amortizing fixed per-request costs. For the same fit, the gain over the unadapted drafter is 2.151× at the 2,048-token cap of Table 2 and 1.895× at an 8,192-token cap, and under both caps it closely tracks the gain in acceptance length.

G THEORETICAL OBSERVATIONS

Two sources of lost acceptance. A mapped drafter can accept fewer proposals than the original drafter did with the original target for two reasons. Either its input context differs from the one the original target would supply (interface mismatch), or the new target prefers different continuations

(distribution mismatch). Reconstruction reduces only the first, and the following bound makes this precise.

Proposition 1. If the drafter's proposal distribution is L-Lipschitz in total variation with respect to its contexts, then at every proposal position the acceptance probability under speculative sampling changes by at most $L\,\|c_t^{(o)}\|\sqrt{D(\widehat{c}_t, c_t^{(o)})}$ when the drafter reads the mapped context $\widehat{c}_t$ instead of the original context $c_t^{(o)}$.

Proof. Let p be the new target's next-token distribution and q_c the drafter's proposal distribution given contexts c. Speculative sampling accepts with probability $a(q) = \sum_x \min(p(x), q(x)) = 1 - \mathrm{TV}(p, q)$ (Leviathan et al., 2023; Chen et al., 2023), so by the triangle inequality $|a(q_{\widehat{c}}) - a(q_{c^{(o)}})| \le \mathrm{TV}(q_{\widehat{c}}, q_{c^{(o)}}) \le L\|\widehat{c} - c^{(o)}\|$, and $\|\widehat{c}_t - c_t^{(o)}\| = \|c_t^{(o)}\|\sqrt{D(\widehat{c}_t, c_t^{(o)})}$ by Equation 4, ignoring its 10^{-6} term. □

Under greedy decoding the same distance controls acceptance whenever the induced logit change stays below the drafter's top-1 margin. The bound therefore identifies the quantity that reconstruction should minimize, the context term of Equation 5. It does not involve distribution mismatch, which only token supervision can reduce.

Interface residual and transfer quality. Table 17 lists the closed-form map's normalized context error on its own fitting features against the iterative same-dataset fit's throughput relative to native (Table 3). Over these six settings the Pearson correlation is -0.91 (Spearman -0.83), and the cross-family pair has the largest residuals.

Table 17: Closed-form context error and same-dataset throughput relative to native.

Pair	Dataset	Context error	×native
Qwen3-8B ← Qwen3-4B	CodeAlpaca	0.167	0.936
Qwen3-8B ← Qwen3-4B	Chat	0.197	0.993
Qwen3-4B ← Qwen3-8B	CodeAlpaca	0.192	0.905
Qwen3-4B ← Qwen3-8B	Chat	0.231	0.903
Llama3.1-8B ← Qwen3-4B	CodeAlpaca	0.257	0.761
Llama3.1-8B ← Qwen3-4B	Chat	0.302	0.665

Fitting the maps is a linear regression. The feature term of Equation 5 asks each map W_i to predict the original target's features $h_{i,t}^{(o)}$ from the new target's features $h_{i,t}$ at the selected positions. It is therefore an ordinary least-squares regression with d_t inputs and d_r outputs per stream, with the closed-form solution of Appendix C.2. That solution keeps 97.0–99.6% of the throughput of the iteratively trained maps (Table 10), and a nonlinear map gives no consistent gain (Table 12). The change of interface between two models is therefore close to linear.

Supervised positions per input dimension. The estimation error of a least-squares fit falls with the number of training positions n per input dimension, n/d_t, rather than with the size of the model. A NuminaMath trajectory of the Qwen3-8B ← Qwen3-4B cohort averages 1,315 tokens, and supervising 25% of positions gives 329.5 supervised positions per trajectory. The fits of Figure 3, with 16 to 16,384 trajectories, therefore span $n/d_t \approx 1.3$ to 1,318 (Table 18). On MATH, throughput rises steeply up to $n/d_t \approx 82$ (97.0% of native) and then flattens (98.3% at 165 and 99.7% at 330). Code and Chat keep improving up to $n/d_t \approx 1{,}318$ and stay below native (84.2% and 90.4%). Their remaining gap comes from fitting on math only rather than from too little data, and in-domain fitting data closes most of it (Table 3).

H Analysis probes and initialization

Low-rank corrections. For the Qwen3-4B GSM8K and NanoCoder LoRA targets, we replace each learned correction $W_i - I$ by its best rank-r approximation from the truncated singular-value decomposition, keep the identity, and evaluate the drafter again. Table 19 reports acceptance length, which under greedy decoding does not depend on timing. A single direction per map already re-

Table 18: Cross-model supervision scaling for Qwen3-8B ← Qwen3-4B (the right panel of Figure 3). Every fit supervises 25% of positions and trains for three epochs. Here n/d_t is the number of supervised positions per input dimension ($d_t = 4{,}096$). Throughput is a percentage of the native Qwen3-8B drafter's.

Trajectories	n/d_t	MATH	GSM8K	Code	Chat
16	1.3	15.65	19.37	18.89	32.47
128	10	51.75	42.93	24.28	39.08
512	41	90.73	77.26	38.50	53.32
1,024	82	97.02	90.41	51.47	64.27
2,048	165	98.28	95.76	64.27	75.49
4,096	330	99.74	97.56	76.23	84.06
8,192	659	99.53	96.86	83.09	87.49
16,384	1,318	100.22	99.40	84.16	90.43

covers about 29% of the gain over the unadapted drafter, and 256 of the 2,560 directions recover 93–98%. The useful correction therefore lies in a small subspace of the feature space.

Table 19: Best rank-r approximation of each learned correction $W_i - I$ (truncated singular-value decomposition, identity kept) for the Qwen3-4B LoRA targets. τ is the acceptance length on 128 prompts per task, and "% of gain" is the share of the full correction's gain over the unadapted drafter that the truncated correction keeps. Each map has 2,560 directions.

	GSM8K		NanoCoder	
Rank r per map	τ	% of gain	τ	% of gain
Unadapted (0)	4.472	0	5.111	0
1	4.913	29	5.239	29
2	5.072	40	5.259	34
64	5.931	96	–	–
128	5.928	96	5.492	86
256	5.964	98	5.521	93
Full (2,560)	5.988	100	5.552	100

H.1 Rectangular-identity initialization

We refit the Qwen3-8B ← Qwen3-4B maps with each objective from a rectangular-identity initialization, which copies the first 2,560 of the 4,096 input coordinates, instead of Xavier initialization. Throughput changes by -5% to $+25\%$ (Table 20), and reconstruction keeps the highest τ on every workload under both initializations.

I Recipes and ablation results

I.1 Recipes and placement ablation

LoRA targets. The targets use the public LoRAs `witcheer/qwen3-4b-gsm8k-grpo`, `AbijahKaj/qwen3-4b-kicad-netlist` and `usernamebetter/nanocoder-v1` on Qwen3-4B, and `sumitdotml/lora-and-friends` and `lovepon/Qwen3-8B-code_alpaca-lora` on Qwen3-8B. Each fit uses 4,096 rollouts that the LoRA target generates for training prompts disjoint from the 128 evaluation prompts. For each rollout we sample up to 32 response anchors. Each block, built as in SpecForge (Li et al., 2026), holds the anchor and 15 masked positions, sees only the context before the anchor, and is trained with Eq. 3, with gradients flowing through the frozen drafter into the maps. One epoch is 512 updates at global batch 8 with AdamW (Loshchilov & Hutter, 2019) (learning rate 10^{-4}, no weight decay, 5% warmup, cosine decay, gradient clipping 1), and we evaluate the final checkpoint.

Table 20: Qwen3-8B ← Qwen3-4B (T1) fits from Xavier and from rectangular-identity initialization, for reconstruction with 25% of positions supervised, AUF and decaying cross-entropy.

Workload	Objective	Xavier TPS	Identity TPS	TPS change	Xavier τ	Identity τ	τ change
MATH	Reconstruction	218.75	213.70	−2.31%	7.973	7.983	+0.14%
MATH	AUF	194.22	206.16	+6.15%	6.892	7.326	+6.29%
MATH	Decaying CE	194.69	203.74	+4.65%	6.947	7.348	+5.77%
GSM8K	AUF	162.05	175.35	+8.21%	5.677	6.166	+8.60%
GSM8K	Decaying CE	163.47	169.72	+3.82%	5.693	6.073	+6.68%
GSM8K	Reconstruction	172.35	166.70	−3.28%	6.204	6.217	+0.21%
Code	Reconstruction	125.98	119.49	−5.15%	4.690	4.569	−2.58%
Code	Decaying CE	61.56	76.02	+23.49%	2.449	3.042	+24.20%
Code	AUF	58.37	73.10	+25.24%	2.346	2.931	+24.94%
Chat	Reconstruction	78.31	76.13	−2.78%	2.802	2.747	−1.94%
Chat	Decaying CE	51.31	61.11	+19.09%	1.812	2.183	+20.52%
Chat	AUF	49.93	60.77	+21.73%	1.767	2.163	+22.41%

Cross-model transfer. Fits use 16,384 new-target trajectories with responses capped at 4,096 tokens, three epochs at global batch 8 (6,144 updates), Xavier-initialized maps and learning rate 10^{-3}. The 25% of prompt and response positions supervised in each trajectory are fixed across epochs. For Llama3.1-8B ← Qwen3-4B, paired features are taken at positions where the two tokenizations end on a shared text prefix, and proposals pass through the tokenizer bridge at inference.

Placement variants. The fused-projection update starts from $\Delta F = 0$, the joint map from the identity on the concatenated streams, drafter-transformer LoRA uses rank 128 on the attention and MLP projections, and rank-128 maps are $W_i = B_i A_i$ without an identity path. Table 21 compares these placements on the Qwen3-4B LoRA targets, averaged over fits on two training subsets, and Table 22 (Appendix I.2) compares the training objectives.

Table 21: Placement ablation on LoRA-fine-tuned Qwen3-4B targets, single-request setting. Values are throughput gains over the unadapted drafter, averaged over fits on two training subsets, with both values in brackets. All trained objects share data, AUF loss, learning rate and 512 updates.

Trained object	Params (M)	GSM8K (%)	KiCad (%)	NanoCoder (%)	HM
Five per-stream maps	32.8	**+43.6 [+42,+45]**	**+107.5 [+106,+109]**	+11.3 [+11,+12]	**1.445**
Fused projection update	32.8	+37.4 [+36,+39]	+36.7 [+36,+37]	+9.2 [+9,+10]	1.263
Drafter-transformer LoRA	36.7	+43.0 [+40,+46]	+88.3 [+88,+89]	**+12.9 [+11,+15]**	1.418
Joint map over all streams	163.8	+26.1 [+24,+28]	+100.0 [+96,+104]	−5.6 [−6,−5]	1.275
Rank-128 whole maps	3.3	+10.3 [+10,+11]	+50.6 [+49,+52]	−25.8 [−26,−26]	1.028

I.2 OBJECTIVE COMPARISON

Table 22: Objective comparison for the five maps, covering AUF, decaying cross-entropy (DFlash's training loss (Chen et al., 2026), which weights proposal position j by $\exp(-(j-1)/7)$ instead of stopping at the first error), and reconstruction with half (MSE50) or all (MSE100) eligible positions supervised. Values are throughput ratios to the unadapted drafter (LoRA) or native drafter (cross-transfer), single fits. Cross-transfer uses 8,192 trajectories, and each row uses its objective's full recipe, not only a different loss. Best is bold and second best underlined.

LoRA-target objective	GSM8K	KiCad	NanoCoder	HM
Accept-until-fail	**1.376**	<u>1.895</u>	**1.081**	**1.376**
Decaying cross-entropy	<u>1.368</u>	**1.935**	<u>1.031</u>	<u>1.353</u>
Reconstruction, MSE50	1.011	1.191	0.977	1.052

Cross-transfer objective	MATH	GSM8K	Code	Chat	HM
Reconstruction, MSE100	<u>0.998</u>	**0.990**	<u>0.828</u>	**0.891**	**0.921**
Reconstruction, MSE50	**0.999**	<u>0.975</u>	**0.832**	<u>0.888</u>	<u>0.919</u>
Decaying cross-entropy	0.886	0.919	0.406	0.573	0.623
Accept-until-fail	0.884	0.911	0.385	0.558	0.604

I.3 CROSS-MODEL SUPERVISION SCALING

Table 18 (Appendix G) gives the cross-model points of Figure 3 (right). Table 23 varies the fraction of supervised positions. Since no fraction is consistently better, we use 25%.

Table 23: Position-coverage sweep for Qwen3-8B ← Qwen3-4B, with throughput as a percentage of the native drafter.

Trajectories	Selected positions	MATH	GSM8K	Code	Chat
4,096	25%	99.74	97.56	76.23	84.06
4,096	50%	97.64	94.22	75.22	80.96
4,096	100%	99.71	97.97	77.85	85.37
8,192	25%	99.53	96.86	83.09	87.49
8,192	50%	99.86	97.53	83.22	88.82
8,192	100%	99.79	99.01	82.83	89.14
16,384	25%	100.22	99.40	84.16	90.43
16,384	50%	99.89	98.48	84.77	90.53

I.4 LORA-TARGET SUPERVISION SCALING

Tables 24 and 25 give the LoRA-target points of Figure 3 (left and middle), covering 16 to 2,048 training examples at 32 anchors and 4,096 to 16,384 examples or 8 to 128 anchors. Each point is an independent fit evaluated at its final checkpoint.

Table 24: Small-data LoRA-target scaling of the five maps (AUF, one epoch, 32 anchors). Gain is over the unadapted drafter.

Workload	Examples	Fit time (s)	TPS	Gain over unadapted	τ	Mean output tokens
GSM8K	16	1.6	142.83	+12.35%	4.91	152.76
GSM8K	128	5.9	160.54	+26.28%	5.36	152.76
GSM8K	512	19.9	168.46	+32.51%	5.66	152.76
GSM8K	1,024	39.0	175.36	+37.94%	5.72	152.76
GSM8K	2,048	78.2	178.58	+40.47%	5.87	152.76
KiCad	16	2.3	176.91	+8.31%	6.42	5040.14
KiCad	128	11.3	207.30	+26.91%	7.55	5040.14

Continued

Workload	Examples	Fit time (s)	TPS	Gain over unadapted	τ	Mean output tokens
KiCad	512	37.7	236.12	+44.56%	8.58	5040.14
KiCad	1,024	85.2	262.02	+60.42%	9.27	5040.14
KiCad	2,048	163.7	290.60	+77.91%	10.27	5040.14
NanoCoder	16	1.8	151.60	+0.93%	5.12	217.69
NanoCoder	128	6.1	152.57	+1.58%	5.17	217.69
NanoCoder	512	20.4	159.15	+5.95%	5.30	217.69
NanoCoder	1,024	40.4	157.87	+5.10%	5.37	217.69
NanoCoder	2,048	80.2	162.89	+8.44%	5.44	217.69

Table 25: Training-example and anchor sweeps of the five maps behind Figure 3 (AUF). The 4,096-example, 32-anchor points are the first of the three fits averaged in Table 2.

Workload	Examples	Anchors	TPS	×unadapted	τ	Fit time (s)
GSM8K	4096	8	171.43	1.3485	5.797	70.20
GSM8K	4096	24	174.33	1.3713	5.924	121.09
GSM8K	4096	32	174.90	1.3758	5.992	159.76
GSM8K	4096	64	175.44	1.3800	6.044	293.66
GSM8K	4096	128	182.39	1.4347	6.112	548.78
GSM8K	8192	32	179.11	1.4088	6.065	309.52
GSM8K	12288	32	186.85	1.4698	6.134	466.97
GSM8K	16384	32	184.28	1.4495	6.198	618.88
KiCad	4096	8	260.77	1.5965	10.006	178.18
KiCad	4096	24	298.47	1.8273	11.137	260.22
KiCad	4096	32	309.56	1.8952	11.259	319.25
KiCad	4096	64	317.91	1.9463	11.717	448.98
KiCad	4096	128	330.35	2.0224	11.930	788.16
KiCad	8192	32	332.56	2.0360	12.096	662.57
KiCad	12288	32	347.75	2.1290	12.419	1034.44
KiCad	16384	32	347.85	2.1296	12.594	1426.56
NanoCoder	4096	8	157.26	1.0470	5.331	69.94
NanoCoder	4096	24	165.76	1.1036	5.517	125.28
NanoCoder	4096	32	162.32	1.0807	5.549	165.37
NanoCoder	4096	64	169.21	1.1265	5.587	298.66
NanoCoder	4096	128	169.28	1.1270	5.625	586.43
NanoCoder	8192	32	172.25	1.1467	5.605	327.32
NanoCoder	12288	32	167.33	1.1140	5.607	490.16
NanoCoder	16384	32	169.88	1.1310	5.672	656.52

Table 26 gives the drafter-transformer LoRA points behind the dashed lines of Figure 3 (middle).

Table 26: Drafter-transformer LoRA (rank 128) instead of maps, the dashed lines in the middle panel of Figure 3. Values are throughput ratios to the unadapted drafter for 4,096 rollouts and the given number of anchors per rollout (AUF).

Anchors per rollout	8	24	32	64	128
GSM8K	1.379	1.402	1.364	1.401	1.406
KiCad	1.593	1.738	1.755	1.865	1.915
NanoCoder	1.091	1.113	1.098	1.123	1.104