LiLiCorr: Lightweight Likelihood Correlation of Parallel Drafts for Speculative Decoding

Anonymous authors
Paper under double-blind review

Abstract

Speculative decoding accelerates language-model inference by drafting future tokens that the target model verifies in parallel. A diffusion-style drafter, such as DFlash, drafts an entire block in one forward pass. It is trained on the per-position marginals rather than on the joint distribution over the block, so the tokens it emits are individually plausible yet jointly incoherent. We introduce LiLiCorr, a **Li**ghtweight **Li**kelihood-based model that **Corr**elates the per-position marginal distributions such a drafter produces. It keeps the top-K tokens at each position as *candidates* and processes them jointly, producing for each an *in* and an *out* vector. A pair at consecutive positions matches when the earlier candidate's *out* vector aligns, in cosine similarity, with the later one's *in* vector. Training makes the correct pairings score highest while pushing competing ones down, so coherent blocks outscore incoherent ones. The joint distribution over the block, exponential in its length, is never materialized. One lightweight network pass produces all the vectors, and the pairwise scores follow as batched matrix operations, leaving only a cheap greedy walk sequential. We co-train the DFlash drafter with LiLiCorr, so it learns to propose candidates that correlate into longer accepted sequences. Over the vanilla DFlash drafter it builds on, LiLiCorr accepts more and serves faster at every one of the 72 settings we test: nine benchmarks at two target sizes under greedy and temperature-one decoding, together with a throughput sweep over six concurrencies, two input lengths and three output-entropy tiers. It raises acceptance length by 7 to 19%, while its single-pass scoring head accounts for only about 3% of the per-block latency. Against three concurrently developed methods that also restore coherence at draft time, every system equally optimized on a common serving stack, LiLiCorr holds the highest throughput in 63 of those settings, ties within a measured noise floor in 6, and trails in only 3.

1 Introduction

Speculative decoding accelerates autoregressive language-model inference by letting a small draft model propose several future tokens that the target model verifies in parallel, keeping the longest correct prefix (Leviathan et al., 2023; Chen et al., 2023). Because decoding is memory-bandwidth bound, verifying a multitude of tokens together costs little more than generating just one token. The resulting speedup depends on two quantities: the *acceptance length*, the expected number of draft tokens accepted per step, and the cost of the drafter itself. A good drafter must raise the former without inflating the latter. Much effort therefore goes into raising acceptance length, with designs ranging from feature-autoregressive heads such as EAGLE (Li et al., 2024a;b; 2026) to multi-head predictors such as Medusa (Cai et al., 2024). A particularly promising approach predicts an entire block of future tokens in a single forward pass via a parallel diffusion-style drafter, such as DFlash (Chen et al., 2026), offering a long speculative window at very low cost.

In this work we build on DFlash, which predicts an entire block of future tokens in one forward pass. For each position in the block, henceforth a *slot*, it produces a distribution over the full vocabulary. Although DFlash attends bidirectionally across slots, so its representations can in principle correlate different positions, it is optimized with a per-position cross-entropy that supervises only the token

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.

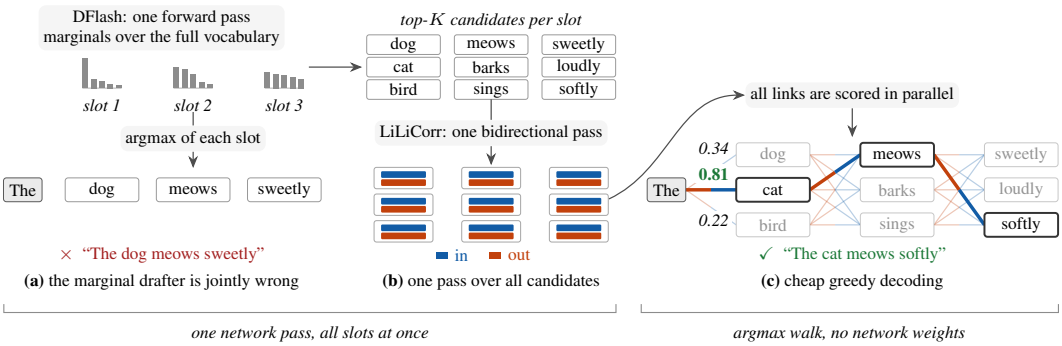


Figure 1: Overview of LiLiCorr. **(a)** DFlash produces a distribution logit over the vocabulary at each slot in one forward pass, and taking the most probable token of each slot independently yields a sequence that is locally probable but jointly incoherent. **(b)** LiLiCorr retains the top-K tokens of every slot and scores them jointly in a single pass, emitting an *in* and an *out* vector per candidate. **(c)** Adjacent candidates are coupled by the alignment of the earlier *out* vector with the later *in* vector, so each slot pair reduces to one matrix product and the whole block is scored in parallel. The block is then committed left to right by a greedy argmax operation that touches no network weights.

marginal at each slot. This leaves the *joint* distribution over the block largely unconstrained, so DFlash predicts each slot well yet can assemble the tokens into an incoherent block.

This marginals problem is well known in the discrete diffusion literature, and its origin is combinatorial: the joint distribution over a block of B positions, each drawn from a vocabulary of size V, has V^B possible realizations, so materializing it is intractable even for very short blocks. Diffusion language models therefore never model the joint explicitly, and both diffusion and speculative-decoding literature offer ways to recover coherence without it. Masked diffusion models such as LLaDA, MDLM, and Dream (Nie et al., 2026; Sahoo et al., 2024; Ye et al., 2025) correlate the sequence through iterative denoising, re-masking a subset of the predicted tokens and re-predicting them over several rounds. In a speculative-decoding setting, however, each such round is another full drafter pass, which is too expensive. Tree-based reranking, such as DDTree (Ringel & Romano, 2026), expands several candidate paths from the per-slot marginals and lets the target model select among them. This shifts the correlation work onto the target, whose passes speculative decoding is designed to save. The trade is especially unfavorable in the compute-bound, high-concurrency regime typical of real-world serving. In contrast, Domino (Huang et al., 2026) and DSpark (Cheng et al., 2026), two works concurrent with ours, act at draft time while avoiding extra target passes, but they do so through a sequence of auto-regressive neural-network passes, one on each draft slot. A third work developed concurrently with ours, DFlash2 (Inco AI, 2026), avoids the per-slot pass, scoring each adjacent pair of tokens from learned embeddings of their two identities. Compatibility is therefore a property of the token pair itself rather than of the draft it appears in: the same two tokens score alike in every context.

We introduce LiLiCorr, a lightweight likelihood-based model that recovers a coherent sequence from the per-slot marginals in a *single* network pass. Figure 1 sets out the method end to end. LiLiCorr takes the top-K candidate tokens at each slot and produces two vectors per candidate: *in* and *out*. These capture how candidates chain together: two adjacent candidates are compatible when the *out* vector of the first aligns, in cosine similarity, with the *in* vector of the second. Training aligns the *out* and *in* vectors of consecutive ground-truth tokens and separates the competing pairings. At inference, a cheap greedy walk commits the block left to right, at each position taking the candidate that best matches its committed predecessor. The novelty is that every vector is produced in a single pass over all candidates at all slots, so each pair is scored in the light of the candidates around it, at the slots before it and after it. Correlation then reduces to *matrix operations that are easy to optimize*, all candidate interactions computed in parallel, with only the greedy path selection left sequential. Approaches that correct one slot at a time instead run their network at every position.

LiLiCorr is trained jointly with DFlash rather than on top of a frozen drafter, so the drafter learns to propose candidate sets that LiLiCorr can turn into longer accepted sequences. Against a vanilla DFlash drafter trained on the same data, this raises acceptance length by 7 to 19% on every bench-

mark, for 8B and 4B target sizes and both greedy and temperature-1 decoding. We also compare against the three works above, Domino, DSpark and DFlash2, which likewise tune the drafter alongside their head, giving every system the same optimization effort on a common serving stack. LiLiCorr is the fastest system at almost every setting we measure, and most of the few exceptions sit inside the measurement noise. Against the two systems that correct one slot at a time the advantage is structural, since they spend a network pass per slot where LiLiCorr scores every candidate at once. Against DFlash2, which also drafts in one pass, the difference lies in the scoring rule. DFlash2 scores a pair from the identities of its two tokens, so the same pair scores alike in every draft, whereas every LiLiCorr score is formed from vectors that have seen every candidate at every slot. This structural difference yields longer accepted prefixes, on average at both target sizes and under both decoding regimes. The lead persists as concurrency rises.

Contributions.

(1) **A novel likelihood-based formulation of the marginals problem.** We cast the recovery of a coherent sequence from per-slot marginals as a learned scoring model over candidate paths. Rather than reason in the full vocabulary, the model attends over all candidates of all slots by assigning two low-dimensional vectors to each, which encode how neighbors are matched.

(2) **Correlation as efficient parallel matrix operations.** At decoding time, a single network pass precomputes all of the correlations, as the pairwise candidate interactions reduce to matrix operations that are easy to optimize. This is followed by a cheap greedy walk that remains sequential.

(3) **State-of-the-art results.** LiLiCorr sets a new state-of-the-art for draft-time correlation in speculative decoding, trained jointly with the drafter and given the same data, training budget, and optimization effort as the three concurrent works Huang et al. (2026); Cheng et al. (2026); Inco AI (2026). We evaluate on nine benchmarks, at two target sizes and under both greedy and temperature-one decoding, and on a throughput sweep over six serving concurrencies, two input lengths, and three output-entropy tiers. LiLiCorr holds the highest throughput of any system compared in 63 of these 72 settings, ties within a measured noise floor in 6 and trails in only 3.

2 Related Work

Speculative decoding and parallel drafters. Speculative decoding accelerates inference by proposing draft tokens and verifying them with the target model under a rule that preserves its distribution (Leviathan et al., 2023; Chen et al., 2023). Most work along these lines improves the drafter, from multi-head predictors such as Medusa (Cai et al., 2024) to feature-autoregressive heads such as the EAGLE family (Li et al., 2024a;b; 2026). A separate line of work drafts an entire block in one forward pass: parallel and diffusion-style block heads such as DFlash (Chen et al., 2026) and diffusion-based self-speculation (Fu et al., 2026) handle a long speculative window cheaply. D-PACE (Wu et al., 2026) improves DFlash with an acceptance-aware training objective. Their parallel prediction mirrors that of diffusion language models, such as LLaDA (Nie et al., 2026), MDLM (Sahoo et al., 2024), Dream (Ye et al., 2025), and Nemotron-Labs-Diffusion (Fu et al., 2026). These models exhibit a well-known marginals limitation: although attention is bidirectional, the block is factorized into independent per-position predictions, so the sampled tokens are not guaranteed to be jointly coherent. This is addressed via an iterative refinement, re-masking and re-predicting subsets of tokens so that later rounds condition on earlier ones. Parallel drafters inherit the same limitation, but in speculative decoding each refinement round adds a full drafter pass, which negates the latency gains that motivate parallel drafting. Recovering coherence without repeated drafter passes is the problem LiLiCorr addresses.

Recovering coherent drafts. Several recent methods recover a coherent sequence from the per-slot marginals that a parallel drafter such as DFlash produces. One direction expands these marginals into a draft tree of several continuations that the target verifies in a single pass. DDTree (Ringel & Romano, 2026) is training-free, ranking candidate prefixes by their marginal product, whereas Oda et al. (2026) first learn a small autoregressive adapter before building the tree. Both push the correlation work onto the target, spending the compute speculative decoding aims to save, and both rely on tree-structured attention, which is typically less efficient than standard causal attention. An alternative direction reintroduces the dependencies at draft time: Domino (Huang et al., 2026)

and DSpark (Cheng et al., 2026), two of the works developed concurrently with ours, condition each slot on the previously sampled tokens, which avoids extra target passes but runs a correction network once at every slot. A third work developed concurrently with ours, DFlash2 (Inco AI, 2026), correlates the block without such a per-slot pass. It learns two embedding tables over the vocabulary, one holding a vector for each token as a predecessor and one for each token as a successor, and scores a pair of neighboring tokens by the inner product of the two. The score is added to the drafter's logits, and the block is filled left to right, each slot committing the candidate that scores best against the token just committed. What a pair receives is therefore fixed by the identities of its two tokens, the same score in every draft. DFlash2 introduces a second and independent contribution, a grouped causal convolution inside each draft layer that lets neighboring slots exchange local information while the block stays parallel. The convolution acts on the drafter rather than on the correlation, so the two compose, and we ablate it on top of LiLiCorr in Appendix C. The marginals problem is not unique to speculative decoding: non-autoregressive translation meets it in the same form, and Sun et al. (2019) correlate a parallel decoder by factorizing adjacent-pair scores into per-token vectors over a top-K pool. What differs here is that speculative decoding credits only an accepted prefix, which is why LiLiCorr normalizes each slot locally and commits greedily rather than decoding the best-scoring path. LiLiCorr also acts at draft time and in a single pass: it produces all candidate representations at once, computes their pairwise correlations in parallel, and leaves only a cheap greedy argmax over the resulting scores to run sequentially. Its vectors are formed from every candidate in the block, so the score of a pair reflects the context of the entire draft rather than the two token identities alone.

3 METHOD

3.1 SETUP: PREFIX ACCEPTANCE AND THE MARGINAL DRAFTER

Let M be a frozen target language model. A parallel block drafter proposes, in a single forward pass, a block of B positions conditioned on the verified context, of which $B-1$ are future tokens. The first block position is the last token the target has committed, which serves as the *anchor*, slot 0. The remaining $B-1$ positions are the speculative *slots* $1, \ldots, B-1$. At each slot the drafter produces a distribution (logit) over the vocabulary, a per-slot *marginal*. Standard decoding resolves each slot independently, taking its most probable token. Once the draft is complete, the target verifies the whole block in a single pass and accepts a *prefix* of it, discarding everything from the first rejected position onward. With a greedy target a draft token is accepted when it coincides with the target's own choice. Under a sampling target, acceptance instead follows the standard rejection test, which preserves the target distribution exactly (Leviathan et al., 2023; Chen et al., 2023). In both regimes acceptance is prefix-structured, and it is this structure that LiLiCorr is built around.

3.2 A SCORING MODEL OVER THE CANDIDATE LATTICE

The drafter is supervised on per-slot marginals, so the tokens it selects are individually probable but not necessarily compatible with one another. Selecting tokens that are jointly compatible, and not only locally probable, can lengthen the accepted prefix, and doing so at draft time could avoid extra target passes. LiLiCorr addresses this goal by correlating the slots over a small pool of alternatives that the drafter's own marginal already places near the top. At each slot s it retains the drafter's top-K candidate tokens $c_{s,1}, \ldots, c_{s,K}$ rather than its single most probable one. Each candidate is a token, and we represent it by a vector we call its *node*, formed by summing four terms of equal width: the target's frozen input embedding of the token, the drafter's hidden state at that slot, a small network over a few scalar features of how confident the drafter is in that candidate, listed in Appendix A, and learned encodings of the slot and of the candidate's rank. The hidden state belongs to the slot and is shared by its K candidates, so what separates them is the token itself, its confidence, and its rank. Across the $B-1$ slots these nodes form a $(B-1) \times K$ lattice.

A small Transformer processes the $(B-1) \times K$ nodes jointly. Its self-attention lets every candidate exchange information with every other, both the competing candidates within a slot and the candidates at neighboring slots, so each is represented in the context of the whole block rather than on its own. Each attended node is then fused with a projection of the target's hidden state that produced the last verified token: the candidate's own state, that projection, and their elementwise product are concatenated and mapped back to the node width by a single layer. Though one vector common to

the whole block, it enters every candidate separately, so that the verifier's view of the context, and not the drafter's alone, shapes each of them. For each node LiLiCorr emits two vectors in $\mathbb{R}^{d_e}$, an *out* vector $\mathbf{o}$ and an *in* vector $\mathbf{i}$, each normalized to unit norm. The dimension d_e is given in Appendix A. The anchor, which precedes the first slot, is encoded far more thinly: the anchor projection itself, mapped by its own head to one further out vector. Indexing the anchor as slot 0, we write this vector $\mathbf{o}_0$. Adjacent slots s and $s+1$ are then coupled by a single *matching* term, the cosine similarity between the out and in vectors:

$$\phi^{\mathrm{p}}(s,k,k') = \sigma\,\cos(\mathbf{o}_{s,k},\,\mathbf{i}_{s+1,k'}) \quad \text{for} \quad s=0,\dots,B-2,\quad 1\le k\le K_s,\ 1\le k'\le K, \tag{1}$$

where $\cos(\cdot,\cdot)$ is the cosine similarity and σ is a fixed scaling constant, with $K_0=1$ at the anchor, which carries a single out vector, and $K_s=K$ at every speculative slot. Figure 1(b) shows the lattice and the pair of vectors each candidate carries.

The factorization is local, but the information behind it is not. Only adjacent slots are coupled by a score, yet the vectors that form that score depend on the entire lattice, so a pair at slots s and $s+1$ may be judged compatible on evidence lying outside them, among the candidates that precede s or follow $s+1$. We hypothesize that this is the more useful place to spend capacity: rather than enrich the coupling itself, we keep it pairwise and let the vectors it multiplies see the whole block. The pairwise form is what leaves the scores computable as batched matrix products.

Casting each score as a cosine is a deliberate stabilizer: it removes vector magnitude as an unconstrained degree of freedom. Without this bound we observe runaway score magnitudes that corrupt, and in practice destroy, the shared DFlash backbone under joint training. The cosine instead confines every learned factor to $[-\sigma,\sigma]$ while avoiding saturating nonlinearities whose gradients vanish. We use a fixed scale σ to separate the candidate scores to discriminate among them. Its value, and the architecture of the head, are given in Appendix A.

3.3 Training the reranker

Local normalization aligned with prefix acceptance. A natural formulation scores each full path by summing its matching terms across the block and selects the path that maximizes this whole-chain score. We argue that this is the wrong target under prefix acceptance. The global optimum may sacrifice an early slot to win a richer tail, yet under prefix acceptance that tail is never reached. Section 3.5 and Appendix F take up this comparison at decoding time. Optimizing the whole chain thus spends capacity on regions the metric never credits. We therefore score each slot *locally*, conditioned on the candidate committed just before it. Writing the per-slot *node score* of candidate k as $z_s[k]$, the chain rule factorizes the path distribution as

$$P_\theta(c_s \mid c_{s-1}) = \mathrm{softmax}_k\big(z_s[k]\big), \qquad z_s[k]=\phi^{\mathrm{p}}(s-1,\,c_{s-1},\,k), \qquad s=1,\dots,B-1, \tag{2}$$

where c_s is the candidate committed at slot s and c_0 is the anchor. Each conditional still depends on the previous pick through the matching term in $z_s[k]$, so locality avoids scoring the whole chain without reducing the model to a per-slot reranker.

Oracle-prefix supervision. We train LiLiCorr as a teacher-forced reranker over the drafter's marginals, so the supervision must reflect how it is served. Two conditions decide which slots carry a learning signal. The first is coverage: a slot is useful only when its ground-truth token is among the drafter's K candidates, since otherwise no reranking can recover it. The second follows from prefix acceptance. We condition each transition on the ground-truth predecessor g_{s-1}: a path that has already left the ground truth is discarded by the verifier, so the transitions leaving it are never exercised in serving. A slot is therefore supervised only when every earlier slot is covered as well. Together these define the *oracle prefix*, the longest leading run of slots that each contain the ground-truth token among their K candidates. Let $\mathcal{S}$ collect these slots across a training batch. We teacher-force the per-slot cross-entropy on them:

$$\mathcal{L}_{\mathrm{CE}} \;=\; -\frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}} \log P_\theta\big(c_s=g_s \mid c_{s-1}=g_{s-1}\big), \tag{3}$$

where g_s is the ground-truth token at slot s. The softmax of Eq. (2) runs over all K candidates at slot s, so each supervised transition raises the ground-truth successor and pushes its $K-1$ competitors down. A block whose very first slot already misses contributes zero loss. The effect resembles the per-slot decay used to train DFlash, though it arises from the coverage mask rather than from a designed weighting: supervision concentrates on earlier slots.

Target-weighted distractor penalty. The per-slot cross-entropy raises the ground-truth probability but is indifferent to how the residual mass is distributed among the remaining candidates. Under cross-entropy supervision alone, we observed that LiLiCorr's errors were typically candidates the target scored well-below the ground truth, a mismatch with the verifier rather than a genuine ambiguity in the target distribution. Since the target is itself the verifier, its logits provide a graded measure of candidate compatibility that cross-entropy ignores. Scoring against the target lets us direct the negative gradient at these high-scoring but target-incompatible competitors, widening the ground truth's margin over the candidates most likely to trigger an early verification failure.

Let q_k denote the target probability of candidate k, and g_s the ground-truth candidate at slot s. Both q_{g_s} and the q_k are indexed out of the teacher-forced target pass that already supplies the drafter's supervision. We define

$$\mathcal{L}_{\text{dist}} = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \Omega_s, \qquad \Omega_s = \sum_k P_\theta(c_s = k \mid c_{s-1} = g_{s-1}) \, \text{ReLU}(\log q_{g_s} - \log q_k) \,, \tag{4}$$

where Ω_s is the target-relative cost LiLiCorr expects to pay at slot s under its own distribution, which are averaged over the supervised slots in $\mathcal{L}_{\text{dist}}$. Candidates the target ranks below g_s are penalized in proportion to the gap, and those it ranks above are not penalized at all. When g_s is the target's top candidate, this term shares its gradient with a target cross-entropy over the candidate pool, and the rectifier extends it safely to the rare positions where g_s is not the target's top candidate, as detailed in Appendix E.

We keep it alongside the hard cross-entropy because the two supply complementary gradients. Cross-entropy makes LiLiCorr rank the ground truth first even when the target assigns it only modest probability, so it remains the selected token however soft the target is. The distractor penalty complements this by pushing the confusers the target disfavors further down. It is therefore cost-aware shaping of cross-entropy, not a replacement, as detailed in Appendix E.

3.4 Co-adapting the drafter and the reranker

We tune DFlash jointly with LiLiCorr, so that the drafter learns to expose candidate pools and hidden representations from which LiLiCorr can assemble longer accepted prefixes. Joint training must cross the top-K selection that forms the candidate pool, which is not differentiable. We keep the selected candidate indices fixed and retain gradients through their draft log-probabilities and the hidden states from which they were produced,[1] both of which are inputs to LiLiCorr. The acceptance-oriented loss therefore reaches the DFlash backbone without differentiating pool membership, and inference is unchanged.

The drafter keeps its own DFlash training loss, a per-slot block cross-entropy over the drafter marginals that weights earlier slots more heavily (Chen et al., 2026),

$$\mathcal{L}_{\text{DFlash}} = -\sum_{s=1}^{B-1} w_s \, \log p_s(g_s), \qquad w_s = \exp\left(-\frac{s-1}{\beta}\right), \tag{5}$$

where p_s is the drafter's marginal at slot s and β sets the decay. The drafter and LiLiCorr are trained jointly to minimize the drafter loss together with the two reranker terms,

$$\mathcal{L} \;=\; \mathcal{L}_{\text{DFlash}} \;+\; \gamma \, [\mathcal{L}_{\text{CE}} \;+\; \lambda \, \mathcal{L}_{\text{dist}}] \,. \tag{6}$$

Here γ sets the weight of the reranker objective as a whole and λ scales the distractor penalty within it. These weights and the decay β are given in Appendix A. Appendix D ablates the distractor penalty at a fixed objective weight: it raises acceptance length on all nine benchmarks, and by 1.5% on the average.

3.5 Single-pass decoding

Decoding has a parallel- and a sequential stage, and the network runs only once. In the parallel stage, a single LiLiCorr pass produces every outgoing and incoming vector, including the anchor out vector. The block has $B - 1$ adjacent slot pairs, and within each pair every candidate is matched

[1] The candidate token embeddings remain frozen.

against every candidate in the next slot. This gives K^2 comparisons for each of the $B-2$ interior pairs and K for the anchor pair, $(B-2)K^2+K$ matching terms in total. Evaluating them reduces to one matrix multiplication per slot pair, batched across the full block, which keeps the computation static in shape and well suited to kernel fusion and graph capture. In the sequential stage, the block is committed greedily from left to right: at slot s we select

$$\hat{c}_s \;=\; \arg\max_k \; z_s[k] \;=\; \arg\max_k \; \phi^{\mathrm{p}}(s-1,\, \hat{c}_{s-1},\, k), \tag{7}$$

conditioning each slot on the candidate just committed, a walk traced in Figure 1(c). This greedy rule is exactly the inference behavior the local objective in Eq. (2) was trained to optimize. An alternative is to commit the path that maximizes the sum of the matching terms. Appendix F recovers that path exactly with a dynamic program and finds it decodes worse than the greedy rule. Only the argmax walk is sequential, and it touches no network weights; the expensive computation is the single parallel pass. This is the key distinction from iterative-refinement drafters, which invoke a network once per position: LiLiCorr confines all learning to one pass and reduces the per-position work to an argmax over precomputed matching scores.

4 Experiments

4.1 Experimental setup

Models and evaluation. We use frozen Qwen3-8B and Qwen3-4B targets (Yang et al., 2025) with their reasoning mode disabled. Evaluation spans eight public datasets across three domains: math (GSM8K, MATH-500, AIME 2025), code (HumanEval, MBPP, LiveCodeBench), and chat (Alpaca, MT-Bench). We additionally report on SPEED-Bench (Abramovich et al., 2026), hereafter SPEED, whose qualitative split holds 880 prompts across eleven categories. We evaluate two decoding regimes: a deterministic one with a greedy target, and a sampling one at temperature 1 over the untruncated target distribution, verified by exact rejection sampling so that the target distribution is preserved and averaged over three seeds. We measure the acceptance length τ, computed block-weighted as the total number of accepted tokens divided by the number of verification blocks, together with output throughput and the resulting speedup over the target's own autoregressive decoding on the same hardware. Three ablations are deferred to Appendices D, F and C: the distractor penalty, the greedy decode rule, and DFlash2's convolution placed in the DFlash backbone that precedes LiLiCorr.

Baselines. We compare LiLiCorr against three concurrent works to ours, Domino Huang et al. (2026), DSpark Cheng et al. (2026) and DFlash2 Inco AI (2026), and against the DFlash drafter it builds on. All three tune the DFlash drafter as they train their own head, as LiLiCorr does. DFlash2's convolution acts on the drafter rather than on the correlation head, so the two compose, and we ablate it on top of LiLiCorr in Appendix C. At temperature 1 the verifier's rejection-sampling step reproduces the target distribution however the block was drafted, so a drafter is free either to commit each slot to its own argmax or to sample its proposal. Domino and DFlash only take the argmax, whereas LiLiCorr, DSpark and DFlash2 do both. Table 1 reports the sampled proposal where a system has one and the argmax otherwise, and Table 7 reports every system under the argmax. All systems are trained on the same corpus, for the same number of epochs and at the same global batch size and sequence length, under an identical configuration across systems and target sizes, so acceptance length and throughput are directly comparable.

Implementation details. We serve every system on the same deployment-representative SGLang backend, with block size 16 on a single H100 and concurrency one unless stated otherwise. Every speculative head is folded into the drafter and optimized with fused kernels and graph capture. Fully optimized, the LiLiCorr head costs 0.299 ms per block, against Domino's 0.611 and DSpark's 0.779, and accounts for about 3% of the per-block wall on Qwen3-8B and 4.5%

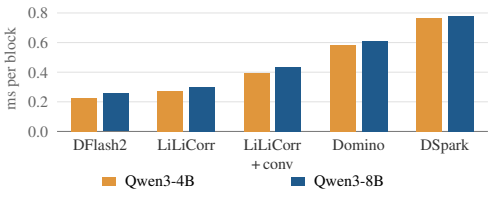


Figure 2: Per-block latency of each approach.

on the cheaper Qwen3-4B block (Figure 2). DFlash2's head and convolution together are slightly cheaper, at 0.256 ms. Table 3 presents these as shares of the block, and the per-stage decomposition behind them is in Appendix A.

Acceptance length is reproducible to the digits we print, but throughput is not: repeated runs of one system on one node at concurrency one differ by up to 0.94%. We therefore set the noise floor at 1%, reading a throughput difference of at most that as a tie and counting a win only where a system leads every other by more.

Training data. All systems are trained on roughly 1.4M examples drawn from the code, mathematics, STEM, and general-chat splits of the Nemotron Post-Training Dataset V2 (Nathawani et al., 2025), with the multilingual splits excluded and every assistant turn regenerated with the target model (Appendix A).

4.2 Main results

Table 1 reports speedup over autoregressive decoding on the nine benchmarks, for both target sizes and both decoding regimes; the acceptance lengths and absolute throughputs behind it are deferred to Appendix B. The four draft-time methods reach comparable acceptance lengths, so throughput separates them. Under greedy decoding LiLiCorr is fastest in twelve of the eighteen settings, ties in four and trails in two, both deficits being HumanEval against DFlash2. Against the DFlash drafter, which reranks nothing, LiLiCorr improves on both axes everywhere, raising acceptance length by 13 to 19% and throughput by 6 to 14%. Tables 8 and 9 resolve SPEED into its eleven categories, in all forty-four of which LiLiCorr serves faster than every system compared, and by its widest acceptance margin on multilingual. It is the only domain held out of training, which suggests a capacity to generalize out of distribution.

Sampling does not reorder the systems. At temperature one accepted prefixes shorten for every system, as expected once the target no longer commits to its own argmax. Among argmax proposals LiLiCorr is fastest at all eighteen settings, and among sampled ones at fifteen, tying at two and trailing at one. DSpark accepts longer prefixes than LiLiCorr at 8B, by 3.1% on the sampled average, and still serves slower: our cheaper head completes 7.2% more verification blocks per second, which more than offsets the shorter prefix (Table 5).

4.3 Scaling with concurrency

Deployed systems run batched, a regime in which decoding moves from memory-bandwidth bound towards compute bound and the cost of speculation is weighed differently. SPEED provides a second, throughput split for this setting, which holds the input length fixed, generates $1{,}024$ tokens per request and varies serving concurrency (Abramovich et al., 2026). We evaluate on its 1K and 2K input-length blocks, in each of their three output-entropy tiers. The tiers group prompts by how predictable their continuations are: coding and mathematics sit in the low-entropy tier and are easier to draft, roleplay and writing in the high-entropy tier and are harder. Every drafter is trained on sequences of $3{,}120$ tokens (Table 4), and these are the two blocks whose input and generation together stay within that length.

Table 2 covers concurrencies from $c=1$ to $c=32$. LiLiCorr attains both the highest throughput and the longest accepted prefix at every one of the thirty-six operating points, none of them inside the tie band, leading on acceptance by 1.1 to 5.7% over the strongest baseline in each cell. Its accepted prefix barely moves with the batch, by between -1.4% and $+0.5\%$ from $c=1$ to $c=32$. Speedups compress for all systems as the batch grows, from 2.1–$3.8\times$ at $c=1$ to 1.3–$1.8\times$ at $c=32$, and the ordering is preserved throughout. Acceptance lengths and absolute throughputs are given in Appendix G.

5 Conclusion

LiLiCorr correlates the per-slot marginals of a parallel block drafter in a single network pass. Assigning each candidate an *in* and an *out* vector reduces the correlation to batched matrix products, leaving only a greedy walk that touches no network weights. Trained jointly with the drafter, LiL-

Table 1: Speedup over autoregressive decoding, single H100, concurrency one. $T = 1$ entries are means over three seeds and the Average is unweighted. Fastest per row and target size in bold, any system within the 1% tie band underlined.

		AR (TPS)		LiLiCorr		DFlash2		Domino		DSpark		DFlash	
	Benchmark	8B	4B	8B	4B	8B	4B	8B	4B	8B	4B	8B	4B
Greedy, $T = 0$	GSM8K	149	234	**5.32**×	**5.00**×	5.15×	4.90×	4.94×	4.56×	4.95×	4.50×	4.66×	4.43×
	MATH-500	150	235	**6.57**×	6.13×	6.54×	**6.14**×	6.29×	5.78×	6.20×	5.55×	5.93×	5.60×
	AIME 2025	149	233	**6.02**×	5.89×	5.90×	**5.90**×	5.75×	5.53×	5.64×	5.30×	5.42×	5.37×
	HumanEval	146	226	4.02×	3.90×	**4.11**×	**3.99**×	3.89×	3.73×	3.99×	3.67×	3.79×	3.64×
	MBPP	150	235	**4.16**×	**3.91**×	4.08×	3.83×	3.93×	3.63×	3.99×	3.58×	3.72×	3.50×
	LiveCodeBench	148	218	**5.43**×	5.53×	5.35×	**5.58**×	5.12×	5.18×	5.18×	5.16×	4.94×	5.03×
	AlpacaEval	150	236	**2.70**×	**2.58**×	2.58×	2.48×	2.59×	2.44×	2.52×	2.35×	2.46×	2.39×
	MT-Bench	150	234	**2.90**×	**2.80**×	2.77×	2.69×	2.81×	2.62×	2.76×	2.55×	2.64×	2.58×
	SPEED	149	232	**3.61**×	**3.50**×	3.44×	3.40×	3.38×	3.25×	3.37×	3.16×	3.24×	3.22×
	Average	149	232	**4.53**×	**4.36**×	4.44×	4.32×	4.30×	4.08×	4.29×	3.98×	4.09×	3.97×
Sampling, $T = 1$	GSM8K	148	230	**4.87**×	**4.62**×	3.88×	3.45×	4.31×	4.10×	4.53×	4.09×	4.13×	4.01×
	MATH-500	148	228	5.01×	4.48×	4.20×	3.72×	4.40×	4.17×	**5.03**×	**4.52**×	4.30×	4.07×
	AIME 2025	147	228	3.99×	**3.93**×	3.38×	2.99×	3.43×	3.29×	**4.16**×	3.77×	3.47×	3.28×
	HumanEval	143	221	**4.51**×	**4.12**×	3.77×	3.32×	4.24×	3.89×	4.09×	3.80×	4.13×	3.97×
	MBPP	148	231	**3.73**×	**3.68**×	3.06×	2.74×	3.34×	3.29×	3.64×	3.27×	3.26×	3.24×
	LiveCodeBench	146	226	**4.28**×	**3.91**×	3.29×	2.88×	3.88×	3.56×	4.05×	3.50×	3.78×	3.51×
	AlpacaEval	148	229	**2.36**×	**2.30**×	1.90×	1.64×	2.08×	2.13×	2.24×	2.06×	2.05×	2.11×
	MT-Bench	148	230	**2.67**×	**2.59**×	2.13×	1.85×	2.39×	2.34×	2.51×	2.31×	2.29×	2.33×
	SPEED	147	228	**2.77**×	**2.81**×	2.23×	2.06×	2.41×	2.41×	2.63×	2.50×	2.36×	2.40×
	Average	147	228	**3.80**×	**3.60**×	3.10×	2.74×	3.39×	3.24×	3.65×	3.31×	3.31×	3.21×

Table 2: Speedup over autoregressive decoding as serving concurrency c grows, on the 1K and 2K blocks of the SPEED throughput split (Abramovich et al., 2026), Qwen3-8B, greedy. Tiers are low (L), mixed (M) and high (H) output entropy. Fastest per cell in bold, and no cell falls inside the tie band. Acceptance lengths are in Table 17.

		LiLiCorr			DFlash2			Domino			DSpark			DFlash		
	c	L	M	H	L	M	H	L	M	H	L	M	H	L	M	H
1K	1	**3.48**×	**3.55**×	**2.31**×	3.26×	3.39×	2.16×	3.13×	3.32×	2.17×	3.19×	3.31×	2.13×	3.11×	3.18×	2.13×
	2	**3.37**×	**3.51**×	**2.33**×	3.17×	3.36×	2.16×	3.05×	3.28×	2.20×	3.11×	3.25×	2.12×	3.05×	3.16×	2.12×
	4	**3.18**×	**3.31**×	**2.24**×	3.03×	3.21×	2.13×	2.88×	3.14×	2.11×	2.94×	3.11×	2.06×	2.88×	3.00×	2.08×
	8	**2.79**×	**2.95**×	**2.08**×	2.65×	2.81×	1.93×	2.52×	2.80×	1.93×	2.59×	2.75×	1.91×	2.54×	2.69×	1.93×
	16	**2.32**×	**2.44**×	**1.81**×	2.19×	2.39×	1.69×	2.16×	2.36×	1.73×	2.19×	2.33×	1.67×	2.15×	2.28×	1.67×
	32	**1.68**×	**1.81**×	**1.33**×	1.61×	1.76×	1.27×	1.60×	1.77×	1.30×	1.62×	1.76×	1.26×	1.56×	1.69×	1.25×
2K	1	**3.83**×	**3.68**×	**2.31**×	3.54×	3.52×	2.17×	3.39×	3.41×	2.17×	3.48×	3.41×	2.13×	3.37×	3.30×	2.13×
	2	**3.72**×	**3.60**×	**2.33**×	3.46×	3.42×	2.19×	3.32×	3.35×	2.19×	3.37×	3.32×	2.13×	3.29×	3.23×	2.17×
	4	**3.40**×	**3.32**×	**2.24**×	3.20×	3.20×	2.12×	3.09×	3.10×	2.11×	3.15×	3.10×	2.07×	3.06×	3.04×	2.10×
	8	**2.97**×	**2.90**×	**2.04**×	2.77×	2.78×	1.93×	2.67×	2.75×	1.93×	2.74×	2.69×	1.88×	2.66×	2.65×	1.91×
	16	**2.39**×	**2.39**×	**1.80**×	2.32×	2.32×	1.71×	2.24×	2.31×	1.73×	2.27×	2.25×	1.69×	2.22×	2.21×	1.67×
	32	**1.83**×	**1.82**×	**1.39**×	1.75×	1.78×	1.33×	1.73×	1.79×	1.36×	1.75×	1.77×	1.34×	1.70×	1.71×	1.30×

iCorr improves on the vanilla DFlash drafter on every benchmark we test, and a draft of comparable quality at a fraction of the cost of correcting one slot at a time makes it the faster system almost everywhere we measure.

One limitation is inherent to the design: LiLiCorr reranks only the top-K tokens the drafter proposes, so a slot whose correct token is absent from that pool cannot be recovered, and the achievable acceptance length is bounded by the drafter's coverage. A second follows from scoring the block through adjacent pairs alone. Each candidate is represented from the whole block, so evidence from distant slots reaches a pair score, but only after being compressed into that one number. Coupling distant slots explicitly is a natural next step, and whether it repays its cost remains open.

Reproducibility Statement

The method is specified in Section 3. The architecture of the scoring head, the objective, the configuration of every arm and the joint training contract are in Appendix A, including how each baseline is configured and trained and how the per-block head costs of Table 3 are isolated by subtraction. Section 4.1 gives the evaluation protocol: the targets, the nine benchmarks, both decoding regimes, the serving backend and the block size. All arms share one training corpus, one optimizer configuration, one epoch count and one sequence length (Table 4), and the corpus is built from a public dataset by the procedure of Section 4.1. Appendix B reports the acceptance lengths and absolute throughputs behind every speedup in the main text, together with the seed dispersion at temperature one. We will release the training and serving code on publication.

Ethics Statement

This work accelerates inference for existing language models and does not change what they generate: verification preserves the target's distribution exactly under greedy decoding and under sampling alike, so a served model carries the same capabilities and the same risks as the target it speculates for. Training uses a public post-training corpus whose assistant turns are regenerated with the target model, and involves no human subjects, private data or personally identifying information. The efficiency gains reduce the energy cost of serving, and we are aware of no harm specific to the method.

References

Talor Abramovich, Maor Ashkenazi, Izzy Putterman, Benjamin Chislett, Tiyasa Mitra, Bita Darvish Rouhani, Ran Zilberstein, and Yonatan Geifman. SPEED-bench: A unified and diverse benchmark for speculative decoding. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Rl2uQlCoQX.

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 *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=PEpbUobfJv.

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

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Oz335dV48X.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, et al. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv preprint arXiv:2607.05147*, 2026.

Yonggan Fu, Lexington Whalen, Abhinav Garg, Chengyue Wu, Maksim Khadkevich, Nicolai Oswald, Enze Xie, Daniel Egert, Sharath Turuvekere Sreenivas, Shizhe Diao, et al. Nemotron-labs-diffusion: A tri-mode language model unifying autoregressive, diffusion, and self-speculation decoding. *arXiv preprint arXiv:2607.05722*, 2026.

Jianuo Huang, Yaojie Zhang, Qituan Zhang, Hao Lin, Hanlin Xu, and Linfeng Zhang. Domino: Decoupling causal modeling from autoregressive drafting in speculative decoding. *arXiv preprint arXiv:2605.29707*, 2026.

Inco AI. DFlash 2: Keep Drafting Parallel, August 2026. URL https://inco.ai/blog/dflash2/.

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Forty-first International Conference on Machine Learning*, 2024a. URL https://openreview.net/forum?id=1NdN7eXyb4.

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

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

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

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *Advances in Neural Information Processing Systems*, 38:50608–50646, 2026.

Yuma Oda, Ryan Mathieu, Roman Knyazhitskiy, and Artur Chakhvadze. Trees from marginals: Autoregressive drafting with factorized priors. *arXiv preprint arXiv:2607.06763*, 2026.

Liran Ringel and Yaniv Romano. Accelerating speculative decoding with block diffusion draft trees. *arXiv preprint arXiv:2604.12989*, 2026.

Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

Zhiqing Sun, Zhuohan Li, Haoqing Wang, Di He, Zi Lin, and Zhihong Deng. Fast structured decoding for sequence models. *Advances in Neural Information Processing Systems*, 32, 2019.

Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

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. *arXiv preprint arXiv:2605.18810*, 2026.

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

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

Disclosure of AI Use

In this work, we used generative AI tools to *implement methods*: parts of the training and inference code were written with the assistance of AI coding assistants. All of it was reviewed and tested by human reviewers, both authors of this paper and colleagues outside the author list, before any result reported here was produced. Separately, the training corpus was built by regenerating the assistant turns of a public dataset with the target model itself (Section 4.1), which we disclose here as use of a generative model in dataset construction.

We did not use generative AI tools for the other subtasks that require disclosure. The method, its objective and the claims made about it, the hypotheses, the experimental design, and the interpretation of the results are the authors' own. No proofs, mathematical claims, thematic analysis, or translation are involved in this work.

Additionally, we used generative AI tools for several subtasks whose disclosure is recommended: drafting parts of the manuscript, editing it for readability and style, typesetting figures, and assembling tables from our measurement outputs. Every number in those tables was traced back to the measurement logs by the authors.

All AI-assisted work has been reviewed by humans, and code was tested in addition to being reviewed. We take responsibility for the final content of this paper, including its text, claims, and artifacts.

A Implementation and Training Details

Architecture. Each candidate enters the head as one vector of the head's hidden width, formed by summing four terms. The first is the target's input embedding of the candidate token, held frozen and projected down. The second is the drafter's hidden state at that slot, projected down and shared by the slot's K candidates. The third is a two-layer network over five scalars describing the drafter's confidence in the candidate, all read from its distribution at that slot: the log-probability and the probability under the full vocabulary, the log-probability gap to the slot's most likely token, the rank normalized to $[0, 1]$, and an indicator for being the slot's top choice. The fourth is a pair of learned tables, one indexed by slot and one by rank. The target state that every candidate is later fused with is formed separately: the target's hidden states that produced the last verified token, collapsed across five captured layers, normalized, and projected to the same width.

The head itself is a two-layer bidirectional Transformer over the $(B-1) \times K$ candidate lattice, attending over all of it at once, with hidden width 1024, eight attention heads, and a feed-forward width of 2048, using RMSNorm and SiLU activations. Beyond the slot table, position enters through two attention biases learned per head and added to the attention logits, one indexed by the signed slot distance between a pair and one marking pairs of candidates that occupy the same slot. Only after the Transformer is each candidate combined with that target state, by concatenating the candidate state, the target state, and their elementwise product, and projecting the result back down to the head's width through a single linear layer followed by a SiLU. This is the one concatenation in the head, everything else being a sum. Two linear heads then emit the out and in vectors of Section 3.2, each of dimension d_e and normalized to unit norm, and a third emits the anchor's single out vector. Table 4 presents the settings.

Candidate pool size. We fixed K at 8 after trying smaller and larger pools. At $K = 4$ fewer slots contain the ground-truth token, so the reranker has less to recover; at $K = 16$ coverage improves, but the added candidates are ones the drafter already ranks low and the wider lattice costs more per block. Both served slower than $K = 8$ in our runs.

Training corpus. For each conversation we keep the original system and user messages and regenerate every assistant turn with the target model, decoding greedily with reasoning disabled and a cap of $4{,}096$ completion tokens.

Objective. The DFlash cross-entropy of Eq. (5) is applied at every slot, whereas the LiLiCorr terms are evaluated only on the oracle prefix, the maximal run of leading slots in which the ground-truth token is present in the candidate pool. Slots beyond the first coverage miss contribute to none of the three terms.

Serving. We fuse top-K candidate extraction and vocabulary-wide logit normalization into a single kernel, compile the matching and selection step, and use wide CUDA graphs that fold candidate pooling and reranking into a single replay. The same level of optimization is applied to Domino's correction loop. Table 3 gives the resulting per-block cost of each head and is the source of the figures quoted in Section 4.1.

Isolating the head. Table 3 is obtained by subtraction. Every benchmark run under greedy decoding reports throughput and acceptance length, and their ratio is the wall-clock time of one verification block. The DFlash control is the same system with no reranking at all, and its block takes 8.87 ms on Qwen3-8B and 5.88 ms on Qwen3-4B, so subtracting it from another system's block time leaves

Table 3: Per-block cost of each speculative head, obtained by the subtraction described above. For DFlash2 and LiLiCorr + conv the figure covers the drafter-side convolution as well as the head.

	Qwen3-8B		Qwen3-4B	
System	Head (ms)	Share of block	Head (ms)	Share of block
DFlash2	0.256	2.8%	0.227	3.7%
LiLiCorr	0.299	3.3%	0.276	4.5%
LiLiCorr + conv	0.436	4.7%	0.396	6.3%
Domino	0.611	6.5%	0.586	9.1%
DSpark	0.779	8.1%	0.765	11.5%

what that system's head costs. All six run together in each allocation, so the choice of machine cancels in the subtraction. We take the median block time over eight of the nine benchmarks, excluding HumanEval, whose short outputs let prefill dominate, and subtract one median from the other. Two things follow from measuring a difference rather than a stage. The figure is net of the per-slot argmax the control performs instead, which is the cost a deployment actually pays for reranking. And it resolves about 0.03 ms, the spread across three independent reads of the convolution's per-block cost, with per-benchmark head costs varying by up to ± 0.1 ms around the median.

Precision. The drafter is trained with a full-precision master copy of its weights, and the optimizer moments are kept in full precision throughout. Two details follow from how that copy is handled. Some training runs stopped and resumed. On resume the drafter's weights are reloaded at the target model's precision, so the master copy is round-tripped through bf16 at that point, in resumed and uninterrupted runs alike. Where a resumed run re-covered steps it had already taken, the recomputed loss values matched the earlier pass to the precision logged. Separately, the drafter is initialized in full precision, but in the runs reported here that initialization was rounded to the target's precision before training began, so that arms differing only in precision started from numerically identical weights. The rounding moves the initialization only below bf16 resolution and leaves first-step loss and gradients unchanged.

Baselines. Domino is served in its widened configuration, trained at 512 anchors rather than its released 256. Its correction head is a recurrent cell over the drafter's hidden width followed by a vocabulary projection. DSpark is served in a width-matched configuration, and at temperature 1 under each of its two proposals, argmax and sampled. Both are run at block size 15, so that every arm drafts the same fifteen tokens and verifies the same sixteen slots. DFlash2 is likewise served under both, and at its released selector width rather than at ours: matching the two candidate pools would change its method rather than control a variable, since the two rerankers are different mechanisms at different budgets, so what we match across arms is the drafted width instead. Each baseline is composed from its own released branch rather than reimplemented from its published settings, and DFlash2 specifically is benchmarked through a third-party reimplementation trained under our contract, since its own sources do not disclose a training recipe. All systems are trained on the same regenerated corpus for the same six epochs, at the same global batch size and sequence length, with self-distillation disabled on every arm, and with drafter and target sharing a rotary base of 10^6.

B Additional Results

Acceptance length and absolute throughput. Table 1 reports speedup, which is the quantity the comparison turns on. Tables 5 and 6 give the two underlying measurements for each target, the block-weighted acceptance length and the output throughput in tokens per second, under the same protocol. They are the source of the acceptance-length statements of Section 4.2.

Per-category results on SPEED. Tables 8 and 9 break the SPEED aggregate of Section 4.2 into its eleven categories, under the same protocol. The per-benchmark conclusions carry over: LiLiCorr serves faster than every system compared in all forty-four cells. The multilingual lead noted in Section 4.2 is the clearest of any category on both axes, reaching 7.4% in acceptance length and

Table 4: Configuration of LiLiCorr and of the joint training run. Symbols follow Section 3. Every arm in the comparison is trained under this configuration, with four keys changed for the smaller target. The step count follows from the conversation count and the global batch size, $\lfloor N/64 \rfloor \times 6$, since batches are drawn over conversations rather than tokens and the last partial batch is dropped. The sequence length is the smallest common multiple of the arms' block sizes that reaches 3,072.

Symbol	Quantity	Value
B	Block size, of which drafted slots	16, of which 15
K	Candidates retained per slot	8
d_e	Dimension of the in and out vectors	1024
σ	Cosine scale	8.0
	Scoring Transformer layers	2
	Hidden width, heads, feed-forward width	1024, 8, 2048
	Parameters in the LiLiCorr head	36.76M (8B), 32.04M (4B)
β	Decay of the DFlash cross-entropy	7.0
λ	Weight of the distractor penalty	1.0
γ	Weight of the reranker objective, Eq. (6)	0.25
	Optimizer	AdamW
	Learning rate	6×10^{-4}
	Weight decay	0
	Schedule	linear, 4% warmup
	Gradient-norm clipping	1.0
	Global batch size, sequence length	64, 3120
	Anchor positions sampled per sequence	512
	Epochs	6
	Training conversations	1,397,048 (8B), 1,396,889 (4B)
	Optimizer steps	130,968 (8B), 130,956 (4B)
	Precision	bf16, fp32 master weights
	Hardware	8 nodes × 8 H100, DDP

Table 5: Acceptance length (τ, block-weighted) and throughput behind Table 1, Qwen3-8B. Autoregressive decoding does not speculate and so has no τ. The $T=1$ panel is the sampled proposal. Highest per row and temperature in bold, throughputs within the 1% tie band of the fastest underlined.

	Benchmark	AR	LiLiCorr	DFlash2	Domino	DSpark	DFlash
Greedy, $T=0$	GSM8K	149	**7.56 / 795**	7.25 / 769	7.23 / 738	7.38 / 740	6.34 / 696
	MATH-500	150	**9.06 / 987**	9.00 / 984	8.98 / 946	9.01 / 932	7.91 / 891
	AIME 2025	149	**8.16 / 899**	7.97 / 882	8.07 / 860	8.04 / 842	7.13 / 811
	HumanEval	146	7.08 / 588	7.08 / **600**	6.86 / 568	**7.16** / 582	6.16 / 553
	MBPP	150	5.85 / **626**	5.68 / 614	5.68 / 591	**5.89** / 601	5.03 / 560
	LiveCodeBench	148	7.75 / **804**	7.60 / 792	7.55 / 759	**7.78** / 768	6.81 / 733
	AlpacaEval	150	**3.66 / 405**	3.47 / 386	3.63 / 387	3.59 / 377	3.22 / 368
	MT-Bench	150	3.94 / **435**	3.75 / 415	3.95 / 421	**3.96** / 414	3.48 / 396
	SPEED	149	**4.92 / 538**	4.65 / 512	4.76 / 504	4.82 / 501	4.27 / 482
	Average	149	**6.44 / 675**	6.27 / 662	6.30 / 641	6.40 / 640	5.59 / 610
Sampling, $T=1$	GSM8K	148	6.86 / **719**	6.51 / 574	6.22 / 636	**6.87** / 669	5.58 / 609
	MATH-500	148	6.83 / 739	6.84 / 621	6.18 / 649	**7.43 / 743**	5.67 / 635
	AIME 2025	147	5.38 / 588	5.45 / 498	4.76 / 505	**6.06 / 613**	4.54 / 512
	HumanEval	143	**8.28 / 646**	7.83 / 540	7.90 / 606	7.85 / 586	7.09 / 591
	MBPP	148	5.18 / **552**	5.06 / 452	4.77 / 494	**5.46** / 538	4.36 / 482
	LiveCodeBench	146	6.02 / **622**	5.88 / 479	5.62 / 565	**6.16** / 590	5.13 / 551
	AlpacaEval	148	3.19 / **350**	3.07 / 282	2.90 / 309	**3.27** / 332	2.68 / 304
	MT-Bench	148	3.63 / **395**	3.44 / 315	3.35 / 354	**3.69** / 372	3.00 / 338
	SPEED	147	3.75 / **407**	3.62 / 329	3.37 / 355	**3.85** / 387	3.09 / 347
	Average	147	5.46 / **558**	5.30 / 454	5.01 / 497	**5.63** / 537	4.57 / 486

6.3% in throughput on Qwen3-4B under greedy decoding. Sample standard deviations across the

Table 6: As Table 5, Qwen3-4B.

	Benchmark	AR	LiLiCorr	DFlash2	Domino	DSpark	DFlash
Greedy, $T=0$	GSM8K	234	**7.50 / 1171**	7.27 / 1147	7.15 / 1068	7.24 / 1054	6.28 / 1038
	MATH-500	235	**8.88** / 1438	8.85 / **1441**	8.80 / 1356	8.70 / 1304	7.74 / 1314
	AIME 2025	233	**8.35** / 1375	8.32 / **1376**	8.25 / 1289	8.14 / 1237	7.29 / 1253
	HumanEval	226	7.22 / 882	**7.44 / 902**	7.07 / 842	7.26 / 829	6.37 / 822
	MBPP	235	**5.78 / 920**	5.63 / 902	5.63 / 854	5.69 / 842	4.91 / 823
	LiveCodeBench	218	**7.86** / 1202	7.76 / **1213**	7.59 / 1126	7.79 / 1123	6.94 / 1095
	AlpacaEval	236	**3.63 / 609**	3.47 / 587	3.62 / 577	3.59 / 556	3.22 / 565
	MT-Bench	234	**3.99 / 655**	3.80 / 630	3.93 / 614	3.93 / 598	3.51 / 604
	SPEED	232	**4.99 / 812**	4.81 / 789	4.86 / 753	4.86 / 733	4.38 / 746
	Average	232	**6.47 / 1007**	6.37 / 999	6.32 / 942	6.36 / 919	5.63 / 918
Sampling, $T=1$	GSM8K	230	**6.88 / 1063**	6.57 / 793	6.35 / 943	6.79 / 939	5.62 / 921
	MATH-500	228	6.89 / 1024	6.92 / 850	6.26 / 951	**7.32 / 1032**	5.68 / 928
	AIME 2025	228	5.52 / **897**	5.45 / 683	4.84 / 750	**5.95** / 860	4.41 / 748
	HumanEval	221	8.03 / **908**	**8.04** / 732	7.66 / 857	7.99 / 837	7.15 / 877
	MBPP	231	**5.40 / 849**	5.17 / 632	5.05 / 760	5.37 / 755	4.51 / 748
	LiveCodeBench	226	**5.88 / 885**	5.61 / 652	5.49 / 805	5.81 / 792	4.96 / 793
	AlpacaEval	229	**3.32 / 526**	3.11 / 374	3.13 / 487	**3.32** / 470	2.83 / 484
	MT-Bench	230	**3.67 / 596**	3.41 / 427	3.48 / 539	**3.67** / 531	3.15 / 537
	SPEED	228	**3.98 / 642**	3.77 / 471	3.62 / 549	3.97 / 572	3.33 / 548
	Average	228	5.51 / **821**	5.34 / 624	5.10 / 738	**5.58** / 754	4.63 / 732

Table 7: The argmax-proposal panel at $T = 1$, the other half of the measurement whose sampled panel is in Tables 5 and 6. Reported as in Table 5.

	Benchmark	AR	LiLiCorr	DFlash2	Domino	DSpark	DFlash
Qwen3-8B	GSM8K	148	**6.49 / 685**	6.25 / 549	6.22 / 636	6.32 / 640	5.58 / 609
	MATH-500	148	6.23 / **678**	**6.28** / 568	6.18 / 649	6.22 / 646	5.67 / 635
	AIME 2025	147	**4.86 / 533**	4.73 / 432	4.76 / 505	4.74 / 497	4.54 / 512
	HumanEval	143	**8.33 / 649**	8.13 / 547	7.90 / 606	8.02 / 611	7.09 / 591
	MBPP	148	**4.96 / 530**	4.85 / 431	4.77 / 494	4.92 / 503	4.36 / 482
	LiveCodeBench	146	**5.87 / 607**	5.77 / 499	5.62 / 565	5.82 / 558	5.13 / 551
	AlpacaEval	148	**2.93 / 323**	2.83 / 258	2.90 / 309	2.92 / 307	2.68 / 304
	MT-Bench	148	3.33 / **365**	3.23 / 294	**3.35** / 354	3.34 / 349	3.00 / 338
	SPEED	147	**3.43 / 373**	3.31 / 300	3.37 / 355	3.39 / 352	3.09 / 347
	Average	147	**5.16 / 527**	5.04 / 431	5.01 / 497	5.08 / 496	4.57 / 486
Qwen3-4B	GSM8K	230	**6.57 / 1021**	6.44 / 774	6.35 / 943	6.42 / 935	5.62 / 921
	MATH-500	228	**6.38 / 976**	6.34 / 775	6.26 / 951	6.31 / 929	5.68 / 928
	AIME 2025	228	4.81 / **783**	**4.87** / 608	4.84 / 750	4.75 / 722	4.41 / 748
	HumanEval	221	8.20 / **929**	**8.50** / 758	7.66 / 857	8.09 / 868	7.15 / 877
	MBPP	231	**5.23 / 823**	5.09 / 618	5.05 / 760	5.19 / 764	4.51 / 748
	LiveCodeBench	226	**5.61 / 861**	5.53 / 648	5.49 / 805	5.54 / 774	4.96 / 793
	AlpacaEval	229	**3.16 / 510**	3.02 / 376	3.13 / 487	3.10 / 451	2.83 / 484
	MT-Bench	230	3.46 / **562**	3.35 / 417	**3.48** / 539	3.45 / 524	3.15 / 537
	SPEED	228	**3.70 / 599**	3.60 / 443	3.62 / 549	3.66 / 553	3.33 / 548
	Average	228	**5.24 / 785**	5.20 / 602	5.10 / 738	5.17 / 724	4.63 / 732

three $T = 1$ seeds stay below 0.28 on acceptance length in every category and system, with one exception, roleplay on Qwen3-4B.

Seed variability at $T = 1$. Tables 10 and 11 give the mean and sample standard deviation over seeds 42, 43, and 44 that stand behind the $T = 1$ panels of Tables 5 and 6. Dispersion is small throughout: the standard deviation of τ stays below 0.25 on every benchmark and system, and below 0.16 on all but eight of the ninety cells. What scatter remains concentrates on the small splits,

Table 8: Per-category results on SPEED (Abramovich et al., 2026), Qwen3-8B, reported as in Table 5.

	Category	LiLiCorr	DFlash2	Domino	DSpark	DFlash
Greedy, $T=0$	Math	**7.01 / 772**	6.84 / 757	6.75 / 719	6.92 / 724	6.05 / 687
	Coding	**6.51 / 707**	6.09 / 665	6.14 / 645	6.30 / 650	5.43 / 609
	Multilingual	**5.54 / 607**	4.98 / 550	4.89 / 521	5.15 / 536	4.59 / 519
	STEM	**6.21 / 686**	5.89 / 654	6.03 / 646	6.10 / 641	5.42 / 619
	Humanities	**5.74 / 634**	5.43 / 603	5.67 / 605	5.67 / 595	5.03 / 572
	Reasoning	**5.09 / 553**	4.85 / 530	4.99 / 524	5.00 / 517	4.39 / 493
	RAG	**4.72 / 493**	4.45 / 470	4.52 / 460	4.57 / 456	4.02 / 438
	Writing	**3.74 / 413**	3.52 / 392	3.65 / 391	3.66 / 385	3.25 / 370
	QA	**3.43 / 378**	3.27 / 362	3.37 / 360	3.42 / 359	3.02 / 344
	Summarization	**3.25 / 345**	3.01 / 323	3.01 / 312	3.12 / 316	2.80 / 308
	Roleplay	**2.88 / 313**	2.74 / 300	2.84 / 300	2.85 / 295	2.63 / 296
Sampling, $T=1$	Math	4.64 / **508**	4.65 / 425	3.99 / 424	**4.99** / 504	3.77 / 426
	Coding	5.11 / **550**	4.85 / 438	4.56 / 476	**5.28** / 526	4.12 / 460
	Multilingual	**3.88 / 422**	3.56 / 326	3.56 / 375	3.62 / 365	3.27 / 367
	STEM	3.93 / **431**	3.88 / 356	3.45 / 366	**4.15** / 421	3.17 / 360
	Humanities	3.68 / **402**	3.53 / 323	3.25 / 344	**3.83** / 387	3.01 / 340
	Reasoning	4.26 / **459**	4.11 / 371	3.88 / 406	**4.32** / 431	3.48 / 388
	RAG	4.23 / **441**	4.02 / 353	3.80 / 386	**4.25** / 411	3.57 / 387
	Writing	3.12 / **341**	3.02 / 275	2.81 / 298	**3.19** / 323	2.57 / 290
	QA	2.95 / **321**	2.84 / 260	2.75 / 291	**3.06** / 308	2.51 / 283
	Summarization	**3.20 / 334**	2.93 / 260	2.89 / 296	3.13 / 306	2.74 / 298
	Roleplay	**2.50 / 270**	2.34 / 212	2.39 / 249	**2.50** / 250	2.17 / 242

Table 9: As Table 8, Qwen3-4B.

	Category	LiLiCorr	DFlash2	Domino	DSpark	DFlash
Greedy, $T=0$	Math	**7.41 / 1216**	7.22 / 1197	7.25 / 1137	7.14 / 1087	6.46 / 1111
	Coding	**6.38 / 1029**	6.25 / 1019	6.15 / 949	6.26 / 940	5.46 / 926
	Multilingual	**5.83 / 948**	5.43 / 891	5.17 / 801	5.27 / 794	4.75 / 810
	STEM	**5.75 / 950**	5.51 / 918	5.51 / 868	5.57 / 852	5.00 / 863
	Humanities	**5.33 / 878**	5.18 / 860	5.32 / 833	5.18 / 790	4.72 / 813
	Reasoning	**5.37 / 868**	5.25 / 855	5.28 / 813	5.29 / 793	4.71 / 798
	RAG	**4.47 / 702**	4.41 / 697	4.23 / 634	4.41 / 645	3.91 / 645
	Writing	**3.69 / 609**	3.52 / 585	3.65 / 572	3.65 / 556	3.27 / 565
	QA	3.55 / **584**	3.44 / 572	**3.59** / 561	3.55 / 541	3.16 / 546
	Summarization	**3.48 / 549**	3.25 / 519	3.21 / 483	3.27 / 480	3.00 / 497
	Roleplay	**3.44 / 557**	3.33 / 542	3.42 / 525	3.39 / 507	3.14 / 531
Sampling, $T=1$	Math	4.73 / **772**	4.59 / 579	4.19 / 651	**4.97** / 723	3.87 / 660
	Coding	5.20 / **826**	5.06 / 626	4.83 / 734	**5.25** / 747	4.28 / 715
	Multilingual	**4.17 / 671**	3.67 / 457	3.52 / 510	3.79 / 544	3.41 / 521
	STEM	4.17 / **684**	4.07 / 516	3.69 / 577	**4.26** / 624	3.42 / 587
	Humanities	3.85 / **627**	3.64 / 459	3.45 / 536	**3.89** / 565	3.24 / 552
	Reasoning	4.48 / **719**	4.29 / 535	4.17 / 640	**4.50** / 645	3.71 / 626
	RAG	**4.42 / 684**	4.20 / 507	3.91 / 582	4.32 / 600	3.78 / 615
	Writing	**3.36 / 546**	3.15 / 396	3.14 / 487	3.33 / 484	2.83 / 481
	QA	3.23 / **523**	3.07 / 385	3.08 / 473	**3.27** / 473	2.77 / 471
	Summarization	**3.39 / 523**	3.09 / 372	3.13 / 434	3.21 / 446	2.92 / 382
	Roleplay	**2.84 / 452**	2.52 / 311	2.54 / 358	2.64 / 376	2.39 / 370

above all AIME 2025 and HumanEval, which at 30 and 164 prompts are the two smallest in the suite. On the two splits of more than a thousand prompts no standard deviation exceeds 0.10.

Table 10: Seed variability behind the $T = 1$ panel of Table 5. Mean over seeds 42, 43 and 44, sample standard deviation in parentheses.

	Benchmark	LiLiCorr	DFlash2	Domino	DSpark	DFlash
Acceptance τ	GSM8K	6.86 (0.04)	6.51 (0.03)	6.22 (0.01)	6.87 (0.03)	5.58 (0.02)
	MATH-500	6.83 (0.03)	6.84 (0.04)	6.18 (0.02)	7.43 (0.07)	5.67 (0.05)
	AIME 2025	5.38 (0.04)	5.45 (0.05)	4.76 (0.13)	6.06 (0.17)	4.54 (0.20)
	HumanEval	8.28 (0.21)	7.83 (0.19)	7.90 (0.05)	7.85 (0.10)	7.09 (0.09)
	MBPP	5.18 (0.03)	5.06 (0.01)	4.77 (0.03)	5.46 (0.04)	4.36 (0.05)
	LiveCodeBench	6.02 (0.07)	5.88 (0.04)	5.62 (0.00)	6.16 (0.09)	5.13 (0.05)
	AlpacaEval	3.19 (0.01)	3.07 (0.02)	2.90 (0.01)	3.27 (0.01)	2.68 (0.00)
	MT-Bench	3.63 (0.01)	3.44 (0.03)	3.35 (0.03)	3.69 (0.02)	3.00 (0.03)
	SPEED	3.75 (0.02)	3.62 (0.02)	3.37 (0.03)	3.85 (0.01)	3.09 (0.01)
Throughput	GSM8K	719 (4.4)	574 (1.8)	636 (1.2)	669 (2.6)	609 (2.4)
	MATH-500	739 (2.2)	621 (3.6)	649 (3.1)	743 (6.1)	635 (4.3)
	AIME 2025	588 (4.7)	498 (3.7)	505 (13.8)	613 (16.6)	512 (22.3)
	HumanEval	646 (9.4)	540 (11.0)	606 (3.5)	586 (4.5)	591 (2.7)
	MBPP	552 (2.6)	452 (1.9)	494 (3.4)	538 (4.1)	482 (5.7)
	LiveCodeBench	622 (6.0)	479 (57.0)	565 (0.3)	590 (7.5)	551 (4.9)
	AlpacaEval	350 (1.3)	282 (1.2)	309 (0.9)	332 (1.0)	304 (0.5)
	MT-Bench	395 (1.9)	315 (3.5)	354 (4.7)	372 (3.1)	338 (4.9)
	SPEED	407 (1.8)	329 (1.5)	355 (2.9)	387 (1.4)	347 (1.8)

Table 11: As Table 10, Qwen3-4B.

	Benchmark	LiLiCorr	DFlash2	Domino	DSpark	DFlash
Acceptance τ	GSM8K	6.88 (0.01)	6.57 (0.01)	6.35 (0.01)	6.79 (0.04)	5.62 (0.02)
	MATH-500	6.89 (0.06)	6.92 (0.05)	6.26 (0.06)	7.32 (0.04)	5.68 (0.02)
	AIME 2025	5.52 (0.15)	5.45 (0.18)	4.84 (0.08)	5.95 (0.10)	4.41 (0.11)
	HumanEval	8.03 (0.14)	8.04 (0.24)	7.66 (0.07)	7.99 (0.18)	7.15 (0.20)
	MBPP	5.40 (0.05)	5.17 (0.05)	5.05 (0.02)	5.37 (0.03)	4.51 (0.06)
	LiveCodeBench	5.88 (0.01)	5.61 (0.01)	5.49 (0.05)	5.81 (0.07)	4.96 (0.06)
	AlpacaEval	3.32 (0.00)	3.11 (0.01)	3.13 (0.00)	3.32 (0.00)	2.83 (0.00)
	MT-Bench	3.67 (0.03)	3.41 (0.03)	3.48 (0.06)	3.67 (0.04)	3.15 (0.04)
	SPEED	3.98 (0.06)	3.77 (0.02)	3.62 (0.02)	3.97 (0.03)	3.33 (0.04)
Throughput	GSM8K	1063 (3.9)	793 (1.3)	943 (4.6)	939 (6.0)	921 (2.9)
	MATH-500	1024 (133.9)	850 (22.0)	951 (10.5)	1032 (31.2)	928 (48.0)
	AIME 2025	897 (24.6)	683 (23.3)	750 (11.7)	860 (13.8)	748 (18.2)
	HumanEval	908 (5.5)	732 (15.8)	857 (4.4)	837 (9.8)	877 (14.8)
	MBPP	849 (12.2)	632 (8.1)	760 (6.1)	755 (8.3)	748 (12.0)
	LiveCodeBench	885 (19.1)	652 (21.3)	805 (5.3)	792 (3.0)	793 (16.9)
	AlpacaEval	526 (27.2)	374 (29.4)	487 (0.3)	470 (14.7)	484 (0.1)
	MT-Bench	596 (8.8)	427 (4.2)	539 (11.3)	531 (7.3)	537 (9.4)
	SPEED	642 (11.4)	471 (1.7)	549 (7.2)	572 (4.4)	548 (27.2)

C The Convolution Variant

DFlash2 contributes a grouped causal convolution inside the draft layers alongside its candidate selector. The two are independent: the convolution improves the drafter, and is indifferent to how the candidates it proposes are afterwards correlated. We therefore place it under LiLiCorr as well and report the resulting arm, LiLiCorr + conv, here rather than in the main tables, since the question it answers is not how LiLiCorr compares with the baselines but how much a better drafter adds on top of it.

Table 12 gives the nine benchmarks at both target sizes and both decoding regimes. Its baseline columns are the same measurement as in Table 1, so the two tables may be read against one another. Table 13 isolates what the convolution itself buys. Under greedy decoding it raises acceptance length over LiLiCorr by 2.3% on Qwen3-8B and 2.5% on Qwen3-4B on the nine-benchmark average, and on every one of the eighteen benchmarks individually. It converts that into throughput at the larger

Table 12: LiLiCorr + conv against the same baselines, reported as in Table 1. The baseline columns are the same measurement, so the two tables read against one another.

		AR (TPS)		LiLiCorr + conv		DFlash2		Domino		DSpark		DFlash	
	Benchmark	8B	4B	8B	4B	8B	4B	8B	4B	8B	4B	8B	4B
Greedy, $T=0$	GSM8K	149	234	**5.35**×	**4.99**×	5.15×	4.90×	4.94×	4.56×	4.95×	4.50×	4.66×	4.43×
	MATH-500	150	235	**6.59**×	6.12×	6.54×	**6.14**×	6.29×	5.78×	6.20×	5.55×	5.93×	5.60×
	AIME 2025	149	233	**6.02**×	5.89×	5.90×	**5.90**×	5.75×	5.53×	5.64×	5.30×	5.42×	5.37×
	HumanEval	146	226	**4.13**×	**4.08**×	4.11×	3.99×	3.89×	3.73×	3.99×	3.67×	3.79×	3.64×
	MBPP	150	235	**4.21**×	3.80×	4.08×	**3.83**×	3.93×	3.63×	3.99×	3.58×	3.72×	3.50×
	LiveCodeBench	148	218	**5.48**×	5.57×	5.35×	**5.58**×	5.12×	5.18×	5.18×	5.16×	4.94×	5.03×
	AlpacaEval	150	236	**2.69**×	**2.53**×	2.58×	2.48×	2.59×	2.44×	2.52×	2.35×	2.46×	2.39×
	MT-Bench	150	234	**2.91**×	**2.78**×	2.77×	2.69×	2.81×	2.62×	2.76×	2.55×	2.64×	2.58×
	SPEED	149	232	**3.61**×	**3.52**×	3.44×	3.40×	3.38×	3.25×	3.37×	3.16×	3.24×	3.22×
	Average	149	232	**4.56**×	**4.36**×	4.44×	4.32×	4.30×	4.08×	4.29×	3.98×	4.09×	3.97×
Sampling, $T=1$	GSM8K	148	230	**4.82**×	**4.61**×	3.88×	3.45×	4.31×	4.10×	4.53×	4.09×	4.13×	4.01×
	MATH-500	148	228	4.95×	**4.86**×	4.20×	3.72×	4.40×	4.17×	**5.03**×	4.52×	4.30×	4.07×
	AIME 2025	147	228	3.99×	**3.97**×	3.38×	2.99×	3.43×	3.29×	**4.16**×	3.77×	3.47×	3.28×
	HumanEval	143	221	**4.56**×	**4.32**×	3.77×	3.32×	4.24×	3.89×	4.09×	3.80×	4.13×	3.97×
	MBPP	148	231	**3.73**×	**3.63**×	3.06×	2.74×	3.34×	3.29×	3.64×	3.27×	3.26×	3.24×
	LiveCodeBench	146	226	**4.33**×	**3.76**×	3.29×	2.88×	3.88×	3.56×	4.05×	3.50×	3.78×	3.51×
	AlpacaEval	148	229	**2.36**×	**2.35**×	1.90×	1.64×	2.08×	2.13×	2.24×	2.06×	2.05×	2.11×
	MT-Bench	148	230	**2.66**×	**2.58**×	2.13×	1.85×	2.39×	2.34×	2.51×	2.31×	2.29×	2.33×
	SPEED	147	228	**2.77**×	**2.78**×	2.23×	2.06×	2.41×	2.41×	2.63×	2.50×	2.36×	2.40×
	Average	147	228	**3.80**×	**3.65**×	3.10×	2.74×	3.39×	3.24×	3.65×	3.31×	3.31×	3.21×

Table 13: What the convolution buys, under greedy decoding. The two models differ only in the grouped causal convolution inside the draft layers, written + conv here. Parentheses give the percentage change from LiLiCorr, and the better of each pair is in bold.

	Qwen3-8B				Qwen3-4B			
	τ		TPS		τ		TPS	
	LiLiCorr	+ conv	LiLiCorr	+ conv	LiLiCorr	+ conv	LiLiCorr	+ conv
GSM8K	7.56	**7.72 (+2.1)**	795	**799 (+0.5)**	7.50	**7.63 (+1.8)**	**1171**	1170 (-0.1)
MATH	9.06	**9.24 (+2.0)**	987	**991 (+0.4)**	8.88	**9.04 (+1.9)**	**1438**	1437 (-0.0)
AIME	8.16	**8.28 (+1.6)**	899	**900 (+0.1)**	8.35	**8.52 (+2.0)**	1375	**1375 (+0.0)**
HEval	7.08	**7.39 (+4.5)**	588	**603 (+2.7)**	7.22	**7.84 (+8.7)**	882	**922 (+4.5)**
MBPP	5.85	**6.00 (+2.6)**	626	**633 (+1.0)**	5.78	**5.87 (+1.5)**	**920**	894 (-2.8)
LCB	7.75	**7.97 (+2.8)**	804	**812 (+1.1)**	7.86	**7.95 (+1.1)**	1202	**1211 (+0.8)**
Alpaca	3.66	**3.70 (+1.1)**	**405**	403 (-0.4)	3.63	**3.68 (+1.4)**	**609**	597 (-1.9)
MTB	3.94	**4.01 (+1.9)**	435	**436 (+0.2)**	3.99	**4.03 (+1.1)**	**655**	651 (-0.7)
SPEED	4.92	**4.99 (+1.4)**	538	**538 (-0.1)**	4.99	**5.11 (+2.4)**	812	**816 (+0.5)**
Average	6.44	**6.59 (+2.3)**	675	**679 (+0.6)**	6.47	**6.63 (+2.5)**	1007	**1008 (+0.1)**

target size, by 0.6% on the average, but not at the smaller, where its fixed per-block cost is a larger share of a cheaper block. Over the seventy-two settings of Section 4.2 the arm holds the highest throughput at 64, ties at 6 and trails at 2, against 63, 6 and 3 for LiLiCorr, and it removes both HumanEval deficits: at Qwen3-4B it accepts 5.4% more than DFlash2 where LiLiCorr accepts 3.0% less. On the concurrency sweep of Section 4.3 it leads all thirty-six operating points, as LiLiCorr does, with an acceptance margin over the strongest baseline in each cell of 2.2 to 10.4% against LiLiCorr's 1.1 to 5.7%. Table 14 places the two side by side there: the convolution accepts more at every one of the thirty-six points, by 0.3 to 5.2%, and serves faster at twenty-six of them. The convolution is not free: it adds 0.136 ms per block, raising the cost of the LiLiCorr head by about half (Table 3).

Table 14: The convolution on the concurrency sweep, Qwen3-8B, greedy. Acceptance length and throughput at the operating points of Table 2, for LiLiCorr and LiLiCorr + conv. Higher of each pair in bold.

		Low		Mixed		High	
	c	LiLiCorr	+ conv	LiLiCorr	+ conv	LiLiCorr	+ conv
1K	1	4.89 / 515	**5.10 / 528**	4.91 / 526	**5.00 / 527**	3.11 / **343**	**3.15** / 342
	2	4.85 / 955	**5.05 / 976**	4.93 / 993	**5.04 / 1000**	3.11 / **664**	**3.12** / 653
	4	4.87 / 1718	**5.11 / 1766**	4.91 / 1796	**5.00 / 1806**	3.09 / 1236	**3.16 / 1244**
	8	4.91 / 2744	**5.13 / 2822**	4.95 / 2913	**5.05 / 2928**	3.12 / **2109**	**3.15** / 2099
	16	4.88 / 3961	**5.07 / 4021**	4.88 / 4175	**4.94 / 4180**	3.11 / **3312**	**3.12** / 3281
	32	4.86 / 4528	**5.08 / 4649**	4.85 / 4976	**4.93 / 5008**	3.09 / 4043	**3.14 / 4102**
2K	1	5.45 / 560	**5.67 / 573**	5.16 / **541**	**5.25** / 541	3.12 / 339	**3.18 / 339**
	2	5.47 / 1025	**5.71 / 1046**	5.19 / **1001**	**5.25** / 997	3.15 / **649**	**3.19** / 648
	4	5.42 / 1751	**5.68 / 1800**	5.14 / **1724**	**5.22** / 1720	3.12 / 1180	**3.17 / 1183**
	8	5.48 / 2666	**5.74 / 2713**	5.20 / **2628**	**5.29** / 2623	3.16 / **1908**	**3.18** / 1907
	16	5.40 / 3550	**5.68 / 3649**	5.20 / 3594	**5.32 / 3615**	3.14 / 2870	**3.18 / 2877**
	32	5.41 / 4038	**5.64 / 4091**	5.18 / 4134	**5.26 / 4179**	3.14 / 3420	**3.19 / 3441**

Table 15: Loss-term ablation, Qwen3-8B target, greedy decoding, concurrency one, acceptance length τ. The two rows share the architecture, all other settings, and the total reranker weight, so the second reallocates that budget rather than enlarging it. It is the model of Section 4. Best per column in bold.

Objective	GSM8K	MATH	AIME	HEval	MBPP	LCB	Alpaca	MTB	SPEED
$\mathcal{L}_{\text{CE}} + \mathcal{L}_{\text{DFlash}}$	7.407	9.031	8.019	6.819	5.740	7.579	3.650	3.923	4.866
$+ \mathcal{L}_{\text{dist}}$ (full)	**7.557**	**9.064**	**8.156**	**7.077**	**5.849**	**7.754**	**3.656**	**3.939**	**4.925**

D Loss-Term Ablation

We train LiLiCorr twice, with and without the distractor penalty of Eq. (4), holding everything else fixed. The two arms share the architecture, every other setting, and the total weight of the reranker objective, so removing the penalty hands its budget to the hard cross-entropy rather than shrinking the objective. Table 15 gives the per-benchmark acceptance lengths at concurrency one. The penalty raises acceptance length on all nine benchmarks, by 1.5% on the average, most on HumanEval at 3.8% and least on AlpacaEval at 0.2%. It is what trains the reranker against the candidates that actually cost acceptances rather than against the ground-truth label alone, and Appendix E derives it as an expected target cost.

E Derivations for the Reranker Objective

The distractor penalty is an expected target cost. Write $P_\theta(k)$ for $P_\theta(c_s = k \mid c_{s-1} = g_{s-1})$, let $c_k = -\log q_k$ be the cost the target assigns to candidate k, low where the target is confident, and let $w_k = \mathrm{ReLU}(c_k - c_{g_s})$ be that cost relative to the ground truth after rectification. The penalty of Eq. (4) is then $\Omega_s = \sum_k P_\theta(k)\, w_k$, the expected rectified cost of the candidate LiLiCorr commits to. When g_s is the target's preferred candidate in the pool no cost is negative, the rectifier is inactive, and since $\sum_k P_\theta(k) = 1$,

$$\Omega_s \;=\; \sum_k P_\theta(k)\;(c_k - c_{g_s}) \;=\; H(P_\theta, q) - c_{g_s}, \qquad H(P_\theta, q) = \sum_k P_\theta(k)\, c_k. \tag{8}$$

$H(P_\theta, q)$ is the expected target cost under P_θ, and the offset c_{g_s} does not depend on the LiLiCorr parameters, so $\nabla_\theta \Omega_s = \nabla_\theta H(P_\theta, q)$: the penalty and a cross-entropy against the target have the same gradient. The penalty involves the target only through cost differences, so it is unchanged if q is renormalized over the pool.

Why this is not distribution matching. The target is frozen, so the costs c_k are constants and Ω_s is linear in P_θ. A linear function on the probability simplex attains its minimum at a vertex,

that is, at a distribution placing all of its mass on one candidate, and the minimizing vertex is the candidate of least cost. Under the assumption above that candidate is g_s, so minimizing the penalty drives LiLiCorr to commit to the target's preferred candidate rather than to reproduce the target's distribution over the pool. Committing is what the verifier rewards, since a block is accepted on the tokens the target would itself emit and mass placed elsewhere is not recovered.

The rectifier. At the occasional slot where g_s is not the target's preferred candidate, the rectifier clips the negative costs that would otherwise reward a candidate the target ranks above the supervised token. Such slots are rare, and arise from numerical differences between the offline generation of the supervision and the teacher-forced target pass used during training, rather than from a genuine target preference.

Complementary gradients. The hard cross-entropy and the distractor penalty act on the ground-truth logit z_g through complementary gradients. For cross-entropy,

$$\frac{\partial \mathcal{L}_{\mathrm{CE}}}{\partial z_g} = P_\theta(g) - 1, \tag{9}$$

which remains strong when $P_\theta(g)$ is small, exactly the hardest mistakes. For the penalty,

$$\frac{\partial \Omega_s}{\partial z_g} = -\,P_\theta(g)\,\mathbb{E}_{P_\theta}[w], \tag{10}$$

which vanishes as $P_\theta(g) \to 0$, so the penalty alone is weak precisely when the ground truth is badly misranked. It is also indifferent among candidates of zero cost, including target ties and candidates the target ranks above g, since the rectifier clips their costs to zero. Cross-entropy therefore robustly pulls the exact ground-truth token upward, while Ω_s preferentially pushes target-rejected confusers downward, so the penalty is best viewed as cost-aware shaping of cross-entropy rather than a replacement for it. Both terms are evaluated only on the oracle prefix and averaged over the same slots, so slots after the first coverage miss contribute to neither.

F Greedy Selection Versus the Highest-Scoring Path

Sections 3.3 and 3.5 argue that prefix acceptance rewards a greedy left-to-right decode over a search for the best-scoring block. We test the decode rule in isolation, holding the trained model fixed and varying only the rule that selects a path through the candidate lattice.

Path selection. Let $c = (c_1, \ldots, c_{B-1})$ denote a path, where c_s is the candidate committed at slot s and c_0 is the anchor. Since LiLiCorr couples only adjacent slots, a path scores as the sum of its matching terms,

$$g(c) \;=\; \sum_{s=1}^{B-1} \phi^{\mathrm{p}}(s{-}1,\, c_{s-1},\, c_s)\,. \tag{11}$$

The greedy rule of Section 3.5 commits each slot to the best continuation of what precedes it and never revisits an earlier choice. The alternative is to return the path that maximizes g. Enumeration is infeasible, as the lattice holds K^{B-1} paths, but the pairwise structure of Eq. (11) makes the exact maximizer reachable in $O(B\,K^2)$ time by dynamic programming. Sweeping left to right, one records for each candidate the score of the best path ending there, then traces the stored predecessors back from the best final candidate. This is the Viterbi algorithm (Viterbi, 1967), a strictly stronger optimizer of g than the greedy walk.

Why the stronger optimizer is not the better one here. Under prefix acceptance every slot contributes only while all the slots before it have matched, so the early ones carry most of the expected acceptance length, and a block that loses its first slot gains nothing from the rest. Maximizing g targets the block as a whole, which is essentially the probability that the entire block survives verification. Giving up an early slot to raise the total is rational for that objective and self-defeating here, because the improved tail is never reached. The greedy walk instead maximizes the leading decision exactly, and each later one conditioned on what was committed.

Table 16: Decode-rule ablation, Qwen3-8B target, greedy decoding, concurrency one, acceptance length τ. Both arms serve the same checkpoint and differ only in the rule that selects a path through the lattice, each as a single fused kernel in the same captured graph, and the two run as consecutive cells of one allocation. Δ is the change from greedy to exact, so negative favors greedy. Best per row in bold.

Benchmark	Greedy	Exact	Δ
SPEED	**4.925**	4.756	-3.43%
GSM8K	**7.557**	7.408	-1.97%
MATH-500	**9.064**	8.846	-2.40%
AIME 2025	**8.156**	7.792	-4.46%
HumanEval	**7.077**	7.018	-0.84%
MBPP	**5.849**	5.714	-2.30%
LiveCodeBench	**7.754**	7.633	-1.56%
Alpaca	**3.656**	3.531	-3.42%
MT-Bench	**3.939**	3.808	-3.32%
Mean Δ			$\mathbf{-2.64\%}$

Results. The comparison is between decode rules for one trained model, not between formulations: LiLiCorr is trained under the local objective of Eq. (2), so its path score is calibrated for the walk it was trained to take. What Table 16 shows is that this calibration is worth more than the stronger optimizer, at equal serving cost. The exact optimizer decodes worse. It lowers acceptance length on all nine benchmarks, by 0.84 to 4.46% and by 2.64% on average. Throughput follows, since the two rules are equally cheap to serve: each is a single fused kernel folded into the same captured draft graph, and the two rules run as consecutive cells of one allocation, where the block rate they sustain differs by at most 0.44% on any benchmark and by 0.06% on average. The stronger optimizer is therefore had for free and still accepts less. Its worst case is AIME 2025, though at 30 problems that split carries the direction rather than the magnitude; every other benchmark lies between -0.84% and -3.43%. The smallest effect is on HumanEval, which is also the smallest split and carries the largest block-rate excursion of the nine.

G Input-Length Sweep on the SPEED Throughput Split

Table 17 gives the acceptance lengths and absolute throughputs behind Table 2. Each block holds 512 requests at a fixed input length with $1{,}024$ generated tokens, in each of the three output-entropy tiers, and is swept from $c=1$ to $c=32$.

On the 1K and 2K blocks LiLiCorr leads on both axes at every one of the thirty-six operating points. Acceptance is ahead of the strongest system in each cell by 1.1 to 5.7%, throughput by 1.9 to 8.2%, and no cell falls inside the tie band.

Table 17: Acceptance length and throughput behind Table 2. Each cell holds 512 requests with 1,024 generated tokens. Reported as in Table 5.

	c	AR	LiLiCorr	DFlash2	Domino	DSpark	DFlash
1K, low	1	148	**4.89 / 515**	4.53 / 482	4.51 / 463	4.69 / 472	4.21 / 460
	2	283	**4.85 / 955**	4.50 / 898	4.50 / 863	4.69 / 879	4.20 / 863
	4	541	**4.87 / 1718**	4.55 / 1637	4.50 / 1559	4.69 / 1588	4.19 / 1558
	8	983	**4.91 / 2744**	4.54 / 2608	4.49 / 2479	4.72 / 2542	4.20 / 2493
	16	1705	**4.88 / 3961**	4.48 / 3730	4.50 / 3686	4.70 / 3725	4.19 / 3658
	32	2694	**4.86 / 4528**	4.53 / 4338	4.50 / 4319	4.69 / 4355	4.19 / 4212
1K, mixed	1	148	**4.91 / 526**	4.66 / 502	4.74 / 491	4.81 / 490	4.25 / 471
	2	283	**4.93 / 993**	4.67 / 949	4.74 / 928	4.81 / 918	4.27 / 894
	4	543	**4.91 / 1796**	4.71 / 1742	4.79 / 1705	4.82 / 1688	4.26 / 1631
	8	986	**4.95 / 2913**	4.66 / 2772	4.79 / 2764	4.84 / 2716	4.27 / 2651
	16	1709	**4.88 / 4175**	4.67 / 4091	4.73 / 4038	4.83 / 3974	4.29 / 3900
	32	2745	**4.85 / 4976**	4.61 / 4844	4.69 / 4870	4.76 / 4834	4.21 / 4634
1K, high	1	148	**3.11 / 343**	2.90 / 321	3.03 / 323	3.02 / 316	2.78 / 316
	2	285	**3.11 / 664**	2.89 / 617	3.05 / 628	3.02 / 606	2.77 / 606
	4	551	**3.09 / 1236**	2.92 / 1174	3.01 / 1161	3.00 / 1135	2.77 / 1148
	8	1016	**3.12 / 2109**	2.89 / 1965	3.03 / 1964	3.04 / 1943	2.80 / 1958
	16	1828	**3.11 / 3312**	2.89 / 3093	3.03 / 3172	3.01 / 3057	2.76 / 3061
	32	3041	**3.09 / 4043**	2.89 / 3877	3.01 / 3956	2.99 / 3846	2.77 / 3788
2K, low	1	146	**5.45 / 560**	4.98 / 518	4.94 / 495	5.17 / 509	4.61 / 493
	2	276	**5.47 / 1025**	5.00 / 952	4.98 / 915	5.17 / 928	4.62 / 907
	4	514	**5.42 / 1751**	4.95 / 1644	4.95 / 1587	5.17 / 1621	4.58 / 1573
	8	898	**5.48 / 2666**	4.98 / 2487	4.93 / 2402	5.20 / 2463	4.59 / 2393
	16	1486	**5.40 / 3550**	5.02 / 3449	4.90 / 3321	5.14 / 3371	4.60 / 3304
	32	2206	**5.41 / 4038**	4.96 / 3862	4.92 / 3826	5.14 / 3864	4.58 / 3749
2K, mixed	1	147	**5.16 / 541**	4.90 / 517	4.93 / 502	5.00 / 501	4.46 / 485
	2	278	**5.19 / 1001**	4.89 / 951	4.98 / 932	5.00 / 922	4.48 / 899
	4	520	**5.14 / 1724**	4.89 / 1665	4.92 / 1611	5.02 / 1610	4.49 / 1579
	8	907	**5.20 / 2628**	4.89 / 2520	4.97 / 2494	5.00 / 2439	4.48 / 2404
	16	1503	**5.20 / 3594**	4.94 / 3488	4.96 / 3466	5.00 / 3388	4.49 / 3322
	32	2267	**5.18 / 4134**	4.92 / 4026	4.97 / 4056	5.01 / 4017	4.49 / 3878
2K, high	1	147	**3.12 / 339**	2.92 / 318	3.04 / 319	3.03 / 313	2.80 / 313
	2	279	**3.15 / 649**	2.95 / 612	3.05 / 611	3.05 / 594	2.82 / 604
	4	527	**3.12 / 1180**	2.93 / 1116	3.04 / 1113	3.03 / 1090	2.83 / 1108
	8	936	**3.16 / 1908**	2.92 / 1804	3.05 / 1810	3.05 / 1759	2.83 / 1786
	16	1591	**3.14 / 2870**	2.93 / 2727	3.04 / 2753	3.06 / 2694	2.76 / 2663
	32	2461	**3.14 / 3420**	2.94 / 3264	3.05 / 3338	3.04 / 3290	2.81 / 3205