CHARD: EFFICIENT BLOCK-DIFFUSION SPECULATIVE DECODING VIA CHUNKED VERIFICATION AND IMMEDIATE REDRAFTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Block-parallel speculative decoding generates wide proposals efficiently, but requests in a serving batch can sustain markedly different acceptance lengths. When their proposals are verified in one target call, a low-acceptance request obtains little progress yet must await the shared completion boundary before redrafting. Prefix pruning reduces work within this interval, while request recovery remains tied to the selected batch's completion. We present CHARD (Chunk-Aware Redrafting and Decoding), which retains wide proposals while resolving requests at finer, target-confirmed boundaries. After each chunk, a continuing request keeps its valid suffix, whereas a terminated request becomes eligible to redraft and join a subsequent verification opportunity. This lets requests start new proposals while others continue consuming existing ones, and avoids evaluating invalid suffixes. We establish conditions under which incremental consumption preserves the target decoding distribution. Implemented in SGLang, CHARD is evaluated on Qwen3-8B and Qwen3.5-27B. Relative to same-width one-shot DFlash B16, it reduces the six-workload average of request-level p99 TPOT by up to 27.4% and improves average throughput by up to 30.4%. A controlled ablation shows that request-local recovery improves performance beyond reducing verification positions alone.

1 INTRODUCTION

Speculative decoding accelerates autoregressive generation by verifying multiple draft tokens in one target-model call (Leviathan et al., 2023; Chen et al., 2023). Multi-head, feature-based, and tree-structured drafters improve candidate quality (Cai et al., 2024; Li et al., 2024; 2026; Miao et al., 2024). Block-parallel drafters instead expose a wide proposal through a largely parallel generation path (Chen et al., 2026; Arriola et al., 2025; Nie et al., 2026). Their serving performance depends on both accepted tokens per proposal and the timing of feedback to each request.

Requests can sustain different acceptance lengths: some repeatedly consume long prefixes, whereas others terminate after a few tokens. In one-shot execution, both await the same target-call completion boundary before their states are resolved. A low-acceptance request therefore obtains less progress per shared interval and needs more such intervals per output token. Figure 1 illustrates this mismatch and the resulting difference in cumulative progress. We study how earlier request-local recovery changes this progress allocation and the resulting throughput and TPOT.

Dynamic verification prunes low-value suffixes before a target call (Liu et al., 2026; Cheng et al., 2026), reducing verification work and potentially shortening the call. When selected prefixes share one target call, their results become available together. A request with a short accepted prefix receives its correction at the same boundary as a request with a long prefix. Verification work and recovery timing therefore offer two distinct controls over serving performance.

CHARD (Chunk-Aware Redrafting and Decoding) makes proposal consumption incremental. It retains a B-wide proposal as request-local state and resolves the request after each C-wide target-confirmed chunk. A continuing request keeps its valid suffix; a terminated request becomes eligible to redraft and rejoin a subsequent verification opportunity. Requests can consequently consume different proposals within the same sequence of target calls (Figure 2).

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.

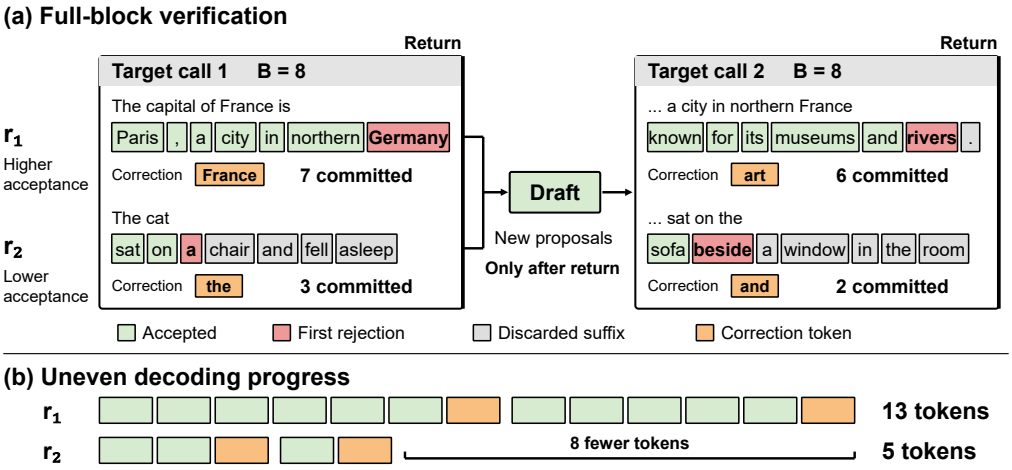


Figure 1: One-shot verification with unequal request progress. (a) Both requests await the same $B = 8$ target-call completion before redrafting. (b) The requests commit 13 and 5 tokens over two calls, including corrections. This illustrative execution uses words as token placeholders and omits committed input anchors.

This design relaxes *proposal-round atomicity*, the convention that a proposal's consumption and recovery form one indivisible round. Chunked early termination also avoids evaluating invalid suffixes. The schedule trades shorter verification calls against more frequent target and draft invocations; a controlled chunk-only ablation isolates recovery's benefit.

We implement CHARD in SGLang and evaluate Qwen3-8B and Qwen3.5-27B with DFlash drafters. Compared with same-width one-shot DFlash B16, CHARD reduces the six-workload average of request-level p99 TPOT by up to 27.4% and improves average throughput by up to 30.4%. Our contributions are:

- We identify shared verification completion as a mechanism that can amplify persistent acceptance differences into request-level TPOT disparity, beyond the cost of unusable suffixes.
- We introduce an incremental proposal lifecycle combining target-confirmed checkpoints, valid-suffix retention, and request-local recovery, and establish sufficient conditions for lossless execution.
- We measure end-to-end gains and isolate recovery at nearly unchanged target-position and call counts at BS32. Further experiments examine transfer to another draft head and the boundary against short blocks.

2 Background and Related Work

2.1 Speculative Decoding and Serving

Speculative decoding verifies multiple draft tokens with one target-model call and commits the longest accepted prefix, amortizing target execution while preserving the target distribution (Leviathan et al., 2023; Chen et al., 2023). Draft mechanisms include blockwise and multi-head prediction (Stern et al., 2018; Cai et al., 2024), feature-space and tree-structured drafting (Li et al., 2026; Miao et al., 2024), parallel Jacobi decoding (Fu et al., 2024; Santilli et al., 2023), self-speculation (Zhang et al., 2024), multi-token prediction (Gloeckle et al., 2024), and speculative early exiting (Xu et al., 2025). CHARD complements serving systems for continuous batching, KV-cache management, chunked prefill, and phase disaggregation (Yu et al., 2022; Kwon et al., 2023; Zheng et al., 2024; Agrawal et al., 2024; Zhong et al., 2024; Patel et al., 2024; Qin et al., 2026), as well as optimized attention kernels (Dao et al., 2022).

2.2 Block-Parallel Speculative Decoding

Discrete and masked diffusion language models enable parallel token generation (Lou et al., 2023; Sahoo et al., 2024; Nie et al., 2026), and block diffusion combines within-block parallelism with autoregressive dependence across blocks (Arriola et al., 2025). DFlash drafts a contiguous block in one diffusion pass (Chen et al., 2026); Domino adds lightweight causal refinement (Huang et al., 2026); and DSpark combines parallel generation with a lightweight sequential module for intra-block dependencies (Cheng et al., 2026).

2.3 Dynamic Verification

DISCO, SpecDec++, and SmartSpec adapt autoregressive speculation length (Mamou et al., 2024; Huang et al., 2024; Liu et al., 2024); AdaFlash predicts diffusion-draft length with a learned head (Qian et al., 2026). D-Cut allocates a shared verification budget (Liu et al., 2026), while DSpark uses prefix-survival estimates and engine profiles (Cheng et al., 2026). Such selection can improve both throughput and latency. When selected prefixes are verified in one call, however, request feedback still arrives at the batch completion boundary. CHARD retains proposal suffixes across target-confirmed checkpoints and interleaves their consumption with request-local recovery. It controls the feedback interval through the width of each target call.

3 Motivation

On Qwen3-8B with a DSpark head at BS32, one-shot B8 yields median TPOTs of 8.82 ms for low-acceptance requests and 5.84 ms for high-acceptance requests in a mixed workload (Section 7). We analyze this disparity through request-specific progress and shared verification completion.

Unequal progress over shared intervals. Let $\boldsymbol{\ell}_r$ contain the requests' verification widths in round r, and let $T_{\mathrm{V}}(\boldsymbol{\ell}_r)$ denote the batch latency. If request i participates in N_i rounds and commits $O_i = \sum_{r=1}^{N_i} G_{i,r}$ tokens, its verification contribution to TPOT is

$$\mathrm{TPOT}_i^{\mathrm{V}} \approx \frac{\sum_{r=1}^{N_i} T_{\mathrm{V}}(\boldsymbol{\ell}_r)}{\sum_{r=1}^{N_i} G_{i,r}} = \frac{\overline{T}_{\mathrm{V},i}}{\overline{G}_i}. \tag{1}$$

Total TPOT also includes drafting and scheduling. When requests encounter comparable round costs, persistently smaller $\overline{G}_i$ requires more shared intervals per output token, translating acceptance differences into request-level TPOT disparity.

Shared feedback after prefix selection. For an illustrative batch with selected widths $(1, 7)$, both requests await $T_{\mathrm{V}}(1, 7)$ before using their verification results, even if every retained position is useful. The target processes these prefixes in parallel and returns both results at the end of the call. Pruning changes the cost of this shared interval. Splitting the retained work across calls also changes the points at which a terminated request can obtain feedback and redraft.

Unusable work is an additional cost. With one-shot width B, let $G \in \{1, \ldots, B\}$ include accepted draft tokens and the correction or final bonus. The unusable-position fraction is

$$W_{\mathrm{block}} = 1 - \frac{\mathbb{E}[G]}{B}. \tag{2}$$

The B16 trace in Figure 3 has $\mathbb{E}[G] = 6.41$, implying 59.9% unusable positions. Pruning these suffixes reduces the position count. The surviving prefixes continue to share the return point of each target call, linking request recovery to the chosen verification granularity.

Design implication. CHARD separates proposal width from feedback granularity. It retains valid suffixes across shorter target calls and resolves requests after each chunk. A terminated request can then redraft while other requests continue consuming their current proposals. Recovery begins from the confirmed state returned by the completed chunk.

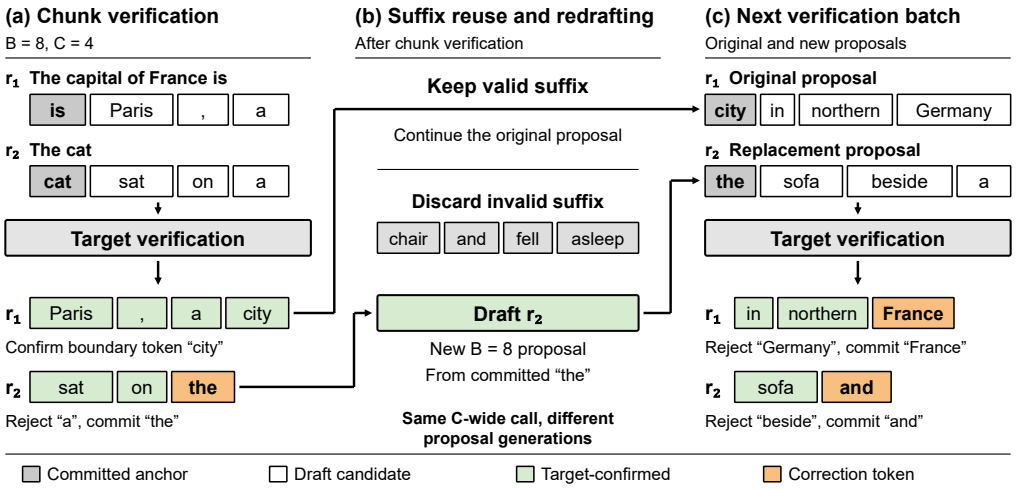


Figure 2: CHARD with $B = 8$ and $C = 4$. (a) Each chunk includes a committed input anchor; its final distribution verifies the next boundary token. (b) r_1 retains a valid suffix, while r_2 redrafts from its correction. (c) The next batch mixes retained and replacement proposals. Words illustrate token-level transitions across successive calls.

4 CHARD Overview

CHARD follows one principle: *generate wide, consume incrementally*. A B-wide proposal persists as a request-local queue with a consumption cursor. Each target call verifies at most C consecutive positions per request. At its completion, a request either advances its cursor and retains the valid suffix, or terminates the proposal and becomes eligible to redraft. A subsequent target batch can mix retained suffixes and replacement proposals.

Sections 5 and 6 define the verification semantics and the recovery policy.

Runtime realization. Each request records its queue, cursor, and proposal-generation version. Verify-ready and draft-ready sets track its next action; target batches group requests by logical chunk width. Committed target KV is retained, and proposal-local state is reused only when its confirmed prefix matches the request. The evaluated path uses fixed-width chunks and homogeneous target batches.

5 Target-Confirmed Incremental Verification

Target-confirmed checkpoints make recovery possible without discarding every unfinished proposal. The target advances a persistent B-wide queue by at most C positions per call. Unlike generating a fresh short block at every call, CHARD retains the unconsumed suffix whenever its continuation remains valid.

Verification semantics. Let

$$Q = (x_0, x_1, \ldots, x_{B-1}) \tag{3}$$

denote a DFlash verification block, where x_0 is a target-produced token that has already been committed and $x_1, \ldots, x_{B-1}$ are produced by the parallel drafter. For a chunk beginning at position s, a C-wide target call performs

$$[x_s, \ldots, x_{s+C-1}] \xrightarrow{\mathcal{M}_t} [p_{s+1}, \ldots, p_{s+C}], \tag{4}$$

where

$$p_i(\cdot) = P_t(\cdot \mid h, x_0, \ldots, x_{i-1}) \tag{5}$$

is the target distribution for x_i. The first $C - 1$ distributions in Equation 4 verify proposal tokens within the current input segment. Because next-token prediction is shifted by one position, the final distribution p_{s+C} verifies the first proposal token after the segment, x_{s+C}, whenever $s + C < B$.

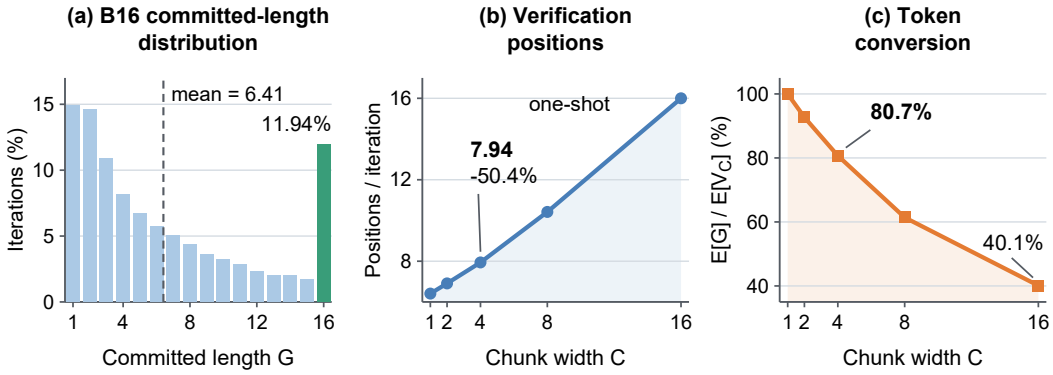


Figure 3: Chunked-verification opportunity at $B = 16$. (a) Measured committed length G, including a correction or bonus token. Replaying this distribution gives (b) expected evaluated positions and (c) conversion $\mathbb{E}[G]/\mathbb{E}[V_C]$. At $C = 4$, expected positions fall from 16 to 7.94, and conversion rises from 40.1% to 80.7%.

If this boundary token is accepted, it becomes the committed anchor of the next target call and verification proceeds to the following chunk. At the first rejection, CHARD commits the accepted prefix and the correction token, then terminates the current proposal; no later chunk is evaluated. If the entire proposal is accepted, p_B produces the final bonus token.

Reduction in verification work. Let $G \in \{1, \ldots, B\}$ be the number of tokens committed by one verification iteration, including either the correction token at the first rejection or the bonus token after full acceptance. Let $M = \lceil B/C \rceil$ and $w_k = \min(C, B - kC)$ be the logical width of chunk k. The number of evaluated target positions is

$$V_C(G) = \sum_{k=0}^{M-1} w_k \, \mathbf{1}\{G > kC\}. \tag{6}$$

Consequently,

$$\mathbb{E}[V_C] = \sum_{k=0}^{M-1} w_k \Pr(G > kC), \tag{7}$$

which reduces to $C \sum_{k=0}^{B/C-1} \Pr(G > kC)$ when C divides B. The final logical chunk is truncated when necessary, and requests are grouped by their next logical width before target execution. Equation 7 makes the early-termination benefit explicit: chunk $k+1$ is evaluated only if progress exceeds the first kC positions. Hence $G \le V_C(G) \le \min(B, G+C-1)$, whereas one-shot execution always evaluates B positions.

The measured B16 trace in Figure 3 has $\mathbb{E}[G] = 6.41$; 48.60% of iterations terminate within four positions. Replaying this distribution at $C = 4$ gives

$$\mathbb{E}[V_4] = 4[1 + \Pr(G > 4) + \Pr(G > 8) + \Pr(G > 12)] = 7.94. \tag{8}$$

Thus, evaluated positions decrease by 50.4%, while the conversion ratio $\rho_C = \mathbb{E}[G]/\mathbb{E}[V_C]$ rises from 40.1% to 80.7%. Realizing these position savings in wall-clock time depends on the added call overhead and short-sequence kernel efficiency. Section 7 measures their combined effect on throughput and TPOT.

Lossless proposal-lifecycle schedules. For request i, let H_i be its committed prefix and let $S_i = (Q_i, \kappa_i, \nu_i)$ record a token queue, cursor, and proposal-generation version. A schedule chooses the request, checkpoint width, and recovery time. It is *admissible* if: (i) it uses revealed acceptance outcomes and token-independent runtime metadata, and eventually serves every active request; (ii) it commits tokens in order by exact target correction with proposal probabilities conditioned on the revealed history and selection; and (iii) retained suffixes and caches are consistent with H_i, with every crossed boundary target-confirmed.

Theorem 1 (Losslessness of admissible schedules)**.** *Within the proposal-lifecycle model above, every admissible schedule—with arbitrary per-request checkpoint widths, recovery delays, and request interleavings—induces the target-only output distribution for every request.*

Proof. Condition on the revealed history before the next candidate and correction. Let $p(\cdot \mid H)$ be the target law at the current prefix H, and q the conditional proposal law for this step. The exact speculative kernel accepts candidate y with probability $\min(1, p(y)/q(y))$ and otherwise samples from the normalized residual $[p-q]_+$. Its output mass at y is

$$\min\{p(y), q(y)\} + \left(1 - \sum_v \min\{p(v), q(v)\}\right) \frac{[p(y) - q(y)]_+}{\sum_v [p(v) - q(v)]_+} = p(y). \tag{9}$$

When the residual mass is zero, rejection has zero probability and the second term is defined as zero. The deterministic DFlash case takes q to be a point mass.

Scheduling uses revealed information; draft, delay, continuation, and recovery change only uncommitted state. Ordered correction commits the next token from $p(\cdot \mid H_i)$ by Equation 9, and state consistency preserves this prefix condition at each subsequent step. With independent per-request sampling randomness, induction over commits yields the target-only law regardless of request interleaving. Eventual service extends the result to each request's output sequence. Greedy decoding gives token-wise identical output.

□

Corollary 1 (Semantic decoupling)**.** *Proposal horizon B, checkpoint width, request interleaving, and recovery delay may be selected independently without changing decoding semantics, provided the resulting schedule remains admissible. Fixed-C CHARD is one such schedule. Stochastic drafters require the exact proposal probabilities used by the correction kernel. Our evaluated implementation uses greedy decoding.*

6 Immediate Redrafting

Chunking supplies earlier target-confirmed feedback. A chunk-only execution policy leaves a terminated request inactive while the others finish their retained proposals. CHARD uses each chunk boundary to replenish terminated requests with new proposals.

After a chunk, a continuing request retains its valid suffix. A rejection commits the accepted prefix and correction, discards the suffix, and makes the request eligible to redraft. Full acceptance commits the final bonus and also triggers redrafting if generation continues. Replacement proposals join later calls alongside retained suffixes. Here, *immediate* means eligibility when the terminating chunk returns; the scheduler launches ready work according to device capacity.

6.1 How Recovery Changes Request Progress

We analyze proposal opportunities with a two-request execution model. Draft and target calls execute serially, with zero queueing delay and enough capacity for each scheduled batch. We assume $C \mid B$, constant draft-call cost T_D, and the same acceptance length for successive replacement proposals.

A high-acceptance request r_h commits B tokens per proposal, while r_l commits $g < B$. Let $T_V(w)$ be the target-batch time with width w per request. One-shot execution has common horizon

$$H_{\text{os}} = T_D + T_V(B), \tag{10}$$

giving

$$\frac{\text{TPOT}_l^{\text{os}}}{\text{TPOT}_h^{\text{os}}} = \frac{B}{g}. \tag{11}$$

Dividing the proposal into $m = B/C$ chunks without replenishing r_l changes the horizon to $T_D + mT_V(C)$ while retaining the progress pair (B, g) and the ratio B/g.

With recovery, suppose r_l commits $g \leq C$ tokens on each attempt and has a replacement ready for the next target opportunity. During the m opportunities needed by r_h, it completes m attempts and commits mg. Including replacement drafting, the common compute time is

$$H_{\text{rec}} = T_D + mT_V(C) + (m-1)T_D = m\big(T_D + T_V(C)\big), \tag{12}$$

so

$$\frac{\text{TPOT}_l^{\text{rec}}}{\text{TPOT}_h^{\text{rec}}} = \frac{B}{mg} = \frac{C}{g}. \tag{13}$$

For $B = 16$, $C = 4$, and $g = 2$, r_l commits eight tokens during four opportunities instead of two during one indivisible proposal round; the within-schedule ratio falls from eight to two.

Latency and recovery cost. For the low-acceptance request, this schedule gives $\text{TPOT}_l^{\text{rec}} = (T_D + T_V(C))/g$, compared with $(T_D + T_V(B))/g$ under one-shot execution. Its latency decreases when $T_V(C) < T_V(B)$: each attempt pays for a narrower target call. For the high-acceptance request, $\text{TPOT}_h^{\text{rec}} = m(T_D + T_V(C))/B$. Its latency decreases when $m(T_D + T_V(C)) < T_D + T_V(B)$, so the saved target time must cover the additional drafting and target calls. These expressions describe how recovery frequency redistributes progress under a fixed acceptance pattern. Section 7 measures throughput and the absolute TPOT of both acceptance groups under the implemented scheduler.

7 Evaluation

We first measure end-to-end throughput and request-level TPOT, then isolate request-local recovery from verification-position savings. A mixed-workload experiment measures the TPOT of each acceptance group, followed by comparisons with shorter proposals and fixed-budget dynamic verification.

7.1 Experimental Setup

Models and system. We implement CHARD in SGLang (Zheng et al., 2024) and evaluate it on NVIDIA H20 GPUs using Qwen3-8B (Yang et al., 2025) at TP1 and Qwen3.5-27B at TP4, each with its corresponding DFlash drafter (Chen et al., 2026). We test 16 and 32 concurrent requests (BS16/BS32). The default proposal width is $B = 16$; we also evaluate B8. A model-level sweep selects $C = 4$ for both targets, which we then fix across workloads and serving loads. D-Cut and its native short-block controls run in vLLM.

Workloads and protocol. The primary experiments use SpecBench, CodeAlpaca, GSM8K, HumanEval, MBPP, and MATH500. Compared methods receive identical prompts and request order, use greedy decoding, ignore early EOS, and generate 1,024 tokens per request. We warm up each configuration and run methods at the same deployment point serially. We use a decode-only, fixed-concurrency protocol.

Baselines and metrics. We compare with target-only decoding, EAGLE-3 (Li et al., 2026) for Qwen3-8B, MTP 3/1/4 (Gloeckle et al., 2024) for Qwen3.5-27B, and one-shot DFlash B8/B16. Same-width DFlash is the controlled baseline; we additionally evaluate native B4 and fixed-budget D-Cut below. We report throughput, request-level p99 TPOT, committed progress per proposal (τ), evaluated request positions (V), and target calls per proposal. For output-token completion times $t_{r,1}, \ldots, t_{r,N_r}$, request r has $\text{TPOT}_r = (t_{r,N_r} - t_{r,1})/(N_r - 1)$; p99 is taken across these request averages. This decode interval includes drafting, verification, and scheduler waiting, starting after prefill, TTFT, and pre-decode queueing. Tokens committed together share a timestamp. V counts packed request positions, excluding CUDA-graph bucket padding, whose cost is included in wall-clock metrics. Together with τ and calls per proposal, it separates target-work savings from scheduling effects. Table 1 reports six-workload averages. Within each width, CHARD and one-shot DFlash share the target and drafter, differing in verification and redraft eligibility.

Table 1: End-to-end performance averaged over six workloads. Throughput speedup and p99 change are computed relative to target-only decoding at the same deployment point. Throughput is measured in tokens/s (↑), p99 TPOT in ms (↓), and τ denotes mean committed progress per proposal.

Deployment	Method	Throughput	Speedup	p99 TPOT	Δ p99	τ
Qwen3-8B TP1, BS16	Target only	2132	1.00×	7.57	–	1.00
	EAGLE-3	1985	0.93×	8.31	+9.8%	3.10
	DFlash B8	3383	1.59×	4.95	−34.6%	5.49
	CHARD B8	**3439**	**1.61×**	**4.68**	**−38.2%**	**5.48**
	DFlash B16	2728	1.28×	6.32	−16.5%	7.75
	CHARD B16	**3307**	**1.55×**	**4.93**	**−34.9%**	**7.72**
Qwen3-8B TP1, BS32	Target only	3465	1.00×	9.27	–	1.00
	EAGLE-3	2314	0.67×	13.89	+49.8%	3.10
	DFlash B8	3778	1.09×	8.49	−8.4%	5.48
	CHARD B8	**3971**	**1.15×**	**7.96**	**−14.1%**	**5.47**
	DFlash B16	2922	0.84×	11.33	+22.2%	7.71
	CHARD B16	**3755**	**1.08×**	**8.46**	**−8.7%**	**7.70**
Qwen3.5-27B TP4, BS16	Target only	1833	1.00×	8.55	–	1.00
	MTP 3/1/4	3117	1.70×	5.08	−40.6%	3.61
	DFlash B8	2884	1.57×	5.70	−33.3%	5.62
	CHARD B8	**2931**	**1.60×**	**5.60**	**−34.5%**	**5.61**
	DFlash B16	2263	1.23×	7.69	−10.1%	7.59
	CHARD B16	**2715**	**1.48×**	**5.92**	**−30.8%**	**7.61**
Qwen3.5-27B TP4, BS32	Target only	2894	1.00×	10.30	–	1.00
	MTP 3/1/4	3795	1.31×	8.02	−22.1%	3.61
	DFlash B8	3376	1.17×	9.28	−9.9%	5.61
	CHARD B8	**3489**	**1.21×**	**8.87**	**−13.9%**	**5.57**
	DFlash B16	2520	0.87×	13.13	+27.5%	7.57
	CHARD B16	**3286**	**1.14×**	**9.53**	**−7.5%**	**7.51**

Table 2: Ablation at B8/C4. T_1 adds chunked verification; T_2 adds immediate redrafting. Work counters (V/prop., calls/prop.) are for BS32; paired cells report BS16 / BS32. Entries are six-workload means; p99/p50 is averaged per workload.

Deployment	Method	V/prop.	Calls/prop.	Thr.	p50	p99	p99/p50
Qwen3-8B TP1	One-shot B8	8.00	1.00	3383 / 3778	3.91 / 7.22	4.95 / 8.49	1.26 / 1.17
	$+T_1$: chunk-only	6.36	1.59	2953 / 3666	4.24 / 7.07	5.64 / 8.72	1.32 / 1.23
	$+T_1{+}T_2$: CHARD	6.36	1.59	3439 / 3971	4.01 / 7.14	4.68 / 7.96	1.16 / 1.11
Qwen3.5-27B TP4	One-shot B8	8.00	1.00	2884 / 3376	4.61 / 8.00	5.70 / 9.28	1.23 / 1.16
	$+T_1$: chunk-only	6.44	1.61	2467 / 3102	5.21 / 8.49	6.65 / 10.14	1.27 / 1.19
	$+T_1{+}T_2$: CHARD	6.45	1.61	2931 / 3489	4.94 / 8.08	5.60 / 8.87	1.13 / 1.10

End-to-end performance. At B16, CHARD improves both workload-averaged metrics at every reported deployment point. Throughput improves by 21.2%/28.5% on Qwen3-8B and 20.0%/30.4% on Qwen3.5-27B at BS16/BS32; p99 falls by 22.0%/25.3% and 23.0%/27.4%, respectively. Mean committed progress per proposal changes by at most 0.9%; the gains occur at nearly unchanged proposal acceptance yield. They also persist under TP4 despite tensor-parallel communication.

At B8, throughput/p99 gains over same-width one-shot verification are 1.7%/5.4% and 5.1%/6.2% on Qwen3-8B at BS16/BS32, and 1.6%/1.7% and 3.3%/4.5% on Qwen3.5-27B. CHARD improves both B8 and B16 configurations, with larger relative gains at B16.

Recovery beyond position savings. Table 2 compares one-shot B8, chunk-only verification (T_1), and CHARD with request-local recovery ($T_1 + T_2$), using $C = 4$. Chunk-only leaves terminated requests inactive until continuing requests finish their proposals. At BS32, the two chunked policies have nearly equal positions and calls per proposal, isolating replenishment at comparable target work.

Table 3: Coding-workload throughput against native block shrinking on Qwen3-8B TP1. Low-load (BS1–8) and higher-load (BS16/32) cases use separate coding workloads. Each row compares identical requests across methods. CHARD uses B8/C4.

Workload	Load	DFlash B4	CHARD B8/C4	Δ throughput
Coding	BS1	397	**419**	**+5.5%**
	BS2	746	**775**	**+3.8%**
	BS4	1364	**1400**	**+2.6%**
	BS8	2366	**2390**	**+1.0%**
	BS16	3682	**3711**	**+0.8%**
	BS32	**4184**	4144	−1.0%

At BS32, chunk-only reduces V from 8.00 to 6.36/6.44 but lowers throughput and raises p99/p50. Recovery improves throughput by 8.3–18.8% over chunk-only across all four deployment points. On Qwen3-8B BS32, throughput rises from 3,666 to 3,971 tokens/s and p99 falls from 8.72 to 7.96 ms with unchanged $V = 6.36$ and calls/proposal $= 1.59$. Across the four deployment points, CHARD lowers p99 versus one-shot B8. The p50 decreases on Qwen3-8B BS32 and increases at the other three points. Recovery therefore improves tail TPOT and throughput while changing the distribution of request progress.

Transfer to a DSpark draft head. We compare one-shot B8 and CHARD B8/C4 using the same DSpark block-7 head in SGLang, with Qwen3-8B TP1 on one H20. The mixture alternates high-acceptance math/coding prompts and low-acceptance chat prompts, grouped by prompt identity. Each of five repetitions uses matched prompt-ID sets with 76/60 requests at BS16/BS32 and fixed acceptance groups. Group medians pool request observations; throughput is reported as mean $\pm$ standard deviation across repetitions. CHARD B8/C4 improves throughput from $2{,}447 \pm 24$ to $2{,}553 \pm 19$ tokens/s at BS16 ($+4.3\%$) and from $2{,}856 \pm 18$ to $3{,}106 \pm 16$ at BS32 ($+8.7\%$). Low-acceptance median TPOT falls from 5.25 to 5.09 ms and from 8.82 to 8.52 ms, respectively; high-acceptance medians rise from 3.34 to 3.96 ms and from 5.84 to 6.67 ms. Consequently, low/high median ratios narrow from 1.57/1.51 to 1.28/1.28.

Separating proposal horizon from verification quantum. Native block shrinking changes both quantities together. Table 3 compares it with CHARD proposals that retain twice the horizon but use the same target-verification quantum on Qwen3-8B TP1.

In the low-load matched cases, B8/C4 exceeds native B4 by 1.0–5.5%; the BS16/32 comparisons show $+0.8\%/-1.0\%$. Retaining a valid second chunk saves a new draft invocation for requests that continue beyond the first chunk. The net gain depends on those saved invocations, kernel efficiency, and scheduling cost.

Fixed-budget dynamic verification. On Qwen3-8B TP1, D-Cut B8 (Liu et al., 2026) at a 50% verification budget has normalized p99 TPOT and inverse throughput of 1.304 and 1.098 relative to DFlash B4 in the average across all evaluated concurrency settings. The summary averages normalized p99 and throughput ratios; inverse throughput is the reciprocal of the latter. Thus, p99 rises by 30.4% and time per output token by 9.8%. In the BS4 comparison, CHARD B8/C4 reaches 0.966 and 0.974, reducing these metrics by 3.4% and 2.6%, respectively.

8 CONCLUSION

CHARD consumes persistent proposals in target-confirmed chunks and recovers terminated requests while others retain valid suffixes. Across Qwen3-8B and Qwen3.5-27B, it reduces average p99 TPOT by up to 27.4% and improves average throughput by up to 30.4% over same-width DFlash B16. Matched-runtime ablations isolate the benefit of request-local recovery. Future work will evaluate arrival-driven workloads and scheduling policies with explicit SLO objectives.

AI USE STATEMENT

In this work, we used generative AI tools to assist with language editing, including grammar correction, sentence refinement, and improvements to clarity and organization. The tools were not used to generate or alter experimental results. We reviewed and revised all AI-assisted text and independently verified the technical content, citations, and claims. We take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We document CHARD's algorithm, implementation, and evaluation protocol to support reproducibility. Sections 5 and 6 specify the verification semantics, suffix-retention conditions, and request-local recovery rules. Theorem 1 provides the assumptions and proof of losslessness, while Section 6.1 states the assumptions of the two-request analysis. Appendix C describes our SGLang implementation, including persistent proposal queues, request-state transitions, boundary confirmation, KV-cache management, immediate redrafting, host–device overlap, and CUDA Graph replay. Section 7.1 reports the models, hardware, tensor-parallel configurations, workloads, decoding settings, and metric definitions. Section 7 details the ablation configurations and the repetition and aggregation protocol for the DSpark-head experiment. Appendices A and B provide per-workload B16 results and the complete committed-length histogram with the equations used to reproduce the verification-position replay.

REFERENCES

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 117–134, 2024.

Marianne Arriola, Aaron Gokaslan, Justin Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, volume 2025, pp. 50726–50753, 2025.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

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. *arXiv preprint arXiv:2602.06036*, 2026.

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.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 2024.

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

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.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. Specdec++: Boosting speculative decoding via adaptive candidate lengths. *arXiv preprint arXiv:2405.19715*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph 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.

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. *arXiv preprint arXiv:2401.15077*, 2024.

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.

Tianyu Liu, Yuhao Shen, Rui Cen, Junhan Shi, Jiebin Zhang, Guangshuo Qin, Hong Liu, Song Liu, Guanghua Yu, and Jianchen Zhu. D-cut: Adaptive verification depth pruning for batched speculative decoding. *arXiv preprint arXiv:2607.14647*, 2026.

Xiaoxuan Liu, Cade Daniel, Langxiang Hu, Woosuk Kwon, Zhuohan Li, Xiangxi Mo, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Optimizing speculative decoding for serving large language models using goodput. *arXiv e-prints*, pp. arXiv–2406, 2024.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

Jonathan Mamou, Oren Pereg, Daniel Korat, Moshe Berchansky, Nadav Timor, Moshe Wasserblat, and Roy Schwartz. Dynamic speculation lookahead accelerates speculative decoding of large language models. *arXiv preprint arXiv:2405.04304*, 2024.

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

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.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132. IEEE, 2024.

Yu-Yang Qian, Hao-Cong Wu, Chen Chen, Jiacheng Sun, Zhenhua Dong, Peng Zhao, and Zhi-Hua Zhou. Adaflash: Adaptive speculative decoding via on-policy distilled diffusion drafters. *arXiv preprint arXiv:2607.19223*, 2026.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Heyi Tang, Feng Ren, Teng Ma, Shangming Cai, Yineng Zhang, Mingxing Zhang, et al. Mooncake: A kvcache-centric disaggregated architecture for llm serving. *ACM Transactions on Storage*, 22(4):1–38, 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.

Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodolà. Accelerating transformer inference for translation via parallel decoding. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12336–12355, 2023.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018.

Jiaming Xu, Jiayi Pan, Yongkang Zhou, Siming Chen, Jinhao Li, Yaoxiu Lian, Junyi Wu, and Guohao Dai. Specee: Accelerating large language model inference with speculative early exiting. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pp. 467–481, 2025.

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.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX symposium on operating systems design and implementation (OSDI 22)*, pp. 521–538, 2022.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282, 2024.

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

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 193–210, 2024.

A PER-WORKLOAD RESULTS AT B16

Tables A1 and A2 expand the B16 comparison in Table 1 into the six individual workloads. They include both evaluated targets and both concurrency levels, with CHARD fixed at $C = 4$. Each pair uses the same target, DFlash drafter, proposal width, prompts, and output length under the protocol in Section 7.1. Throughput is measured in tokens/s, and p99 TPOT is the 99th percentile of request-average TPOT in milliseconds. The Avg. rows reproduce the six-workload summaries in the main text.

Across all 24 paired B16 cases, CHARD increases throughput and reduces p99 TPOT relative to one-shot verification. The per-workload results show that the aggregate B16 gains occur across the evaluated workload types and both tensor-parallel deployments.

Table A1: B16 per-workload results on Qwen3-8B at TP1. CHARD uses C4; bold entries indicate the better value in each pair.

		Throughput (tokens/s) ↑		p99 TPOT (ms) ↓	
Load	Workload	DFlash	CHARD	DFlash	CHARD
BS16	SpecBench	1978	**2668**	8.84	**6.09**
	CodeAlpaca	2633	**3291**	6.55	**4.98**
	GSM8K	3020	**3479**	5.53	**4.61**
	HumanEval	3101	**3492**	4.99	**4.50**
	MBPP	2640	**3423**	6.38	**4.73**
	MATH500	2996	**3490**	5.64	**4.65**
	Avg.	2728	**3307**	6.32	**4.93**
BS32	SpecBench	2117	**2979**	15.85	**10.37**
	CodeAlpaca	2848	**3835**	11.39	**8.37**
	GSM8K	3232	**3966**	9.84	**7.89**
	HumanEval	3255	**3954**	9.41	**7.82**
	MBPP	2860	**3848**	11.34	**8.34**
	MATH500	3221	**3949**	10.13	**7.97**
	Avg.	2922	**3755**	11.33	**8.46**

Table A2: B16 per-workload results on Qwen3.5-27B at TP4. CHARD uses C4; bold entries indicate the better value in each pair.

		Throughput (tokens/s) ↑		p99 TPOT (ms) ↓	
Load	Workload	DFlash	CHARD	DFlash	CHARD
BS16	SpecBench	1637	**2313**	10.80	**6.94**
	CodeAlpaca	2039	**2658**	8.64	**6.18**
	GSM8K	2523	**2845**	6.49	**5.56**
	HumanEval	2582	**2802**	6.46	**5.70**
	MBPP	2195	**2771**	7.30	**5.67**
	MATH500	2602	**2901**	6.45	**5.47**
	Avg.	2263	**2715**	7.69	**5.92**
BS32	SpecBench	1848	**2738**	17.33	**11.03**
	CodeAlpaca	2299	**3270**	14.62	**9.83**
	GSM8K	2794	**3508**	11.37	**8.85**
	HumanEval	2825	**3259**	11.09	**9.45**
	MBPP	2435	**3376**	13.09	**9.24**
	MATH500	2916	**3565**	11.26	**8.76**
	Avg.	2520	**3286**	13.13	**9.53**

B COMMITTED-LENGTH TRACE AND VERIFICATION REPLAY

Table A3 provides the complete B16 committed-length histogram used in Figure 3. It contains 60,812 verification iterations. As in Section 5, G counts accepted draft tokens together with the correction token or the final bonus. The frequencies give $\mathbb{E}[G] = 6.4098$, with 48.60% of iterations committing at most four tokens and 11.94% committing all 16 tokens.

Table A3: Complete committed-length counts for the B16 trace in Figure 3. Percentages are computed over all 60,812 iterations.

G	Count	Share (%)	G	Count	Share (%)
1	9,080	14.93	9	2,202	3.62
2	8,903	14.64	10	1,942	3.19
3	6,617	10.88	11	1,721	2.83
4	4,954	8.15	12	1,416	2.33
5	4,080	6.71	13	1,224	2.01
6	3,496	5.75	14	1,202	1.98
7	3,045	5.01	15	1,019	1.68
8	2,652	4.36	16	7,259	11.94

Replay procedure. Let n_g be the number of iterations with committed length g and let $N = \sum_{g=1}^{16} n_g$. For each fixed chunk width, we apply Equation 6 to every histogram entry and compute

$$\widehat{\mathbb{E}[V_C]} = \frac{1}{N}\sum_{g=1}^{16} n_g V_C(g), \qquad \widehat{\mathbb{E}[G]} = \frac{1}{N}\sum_{g=1}^{16} g\, n_g.$$

Conversion is $\widehat{\mathbb{E}[G]}/\widehat{\mathbb{E}[V_C]}$, and position savings are $1 - \widehat{\mathbb{E}[V_C]}/16$. These calculations hold the measured committed-length distribution fixed and describe the logical verification work associated with each chunk width. End-to-end throughput and TPOT are reported separately in Section 7.

Table A4: Histogram replay at B16. Counts and conversion are computed from Table A3; the final row is full-width one-shot verification.

C	Expected positions	Position savings (%)	Conversion (%)
1	6.4098	59.9	100.0
2	6.8864	57.0	93.1
4	**7.9431**	**50.4**	**80.7**
8	10.3660	35.2	61.8
16 (one-shot)	16.0000	0.0	40.1

At $C = 4$, the expected position count is 7.9431 per proposal, compared with 16 for one-shot verification. This corresponds to 50.4% fewer evaluated positions and 80.7% conversion, matching the values highlighted in Figure 3. A surviving proposal keeps its unconsumed suffix; a terminated proposal avoids the positions in its later chunks. These are the position savings made available to the recovery schedule.

C SYSTEM IMPLEMENTATION

We implement CHARD in SGLang by separating proposal generation from target verification. The implementation follows the fixed-width execution described in Sections 5 and 6: a request maintains a persistent B-wide proposal, and each target call consumes at most C consecutive positions. Requests in a target batch share the same logical verification width, while their proposal cursors and recovery states may differ.

C.1 Request State and Scheduler Integration

A request-local `DraftQueue` stores proposal tokens, positions, cache-slot indices, and a consumption cursor. Its storage remains live across target calls, allowing an accepted chunk to advance the cursor without regenerating the remaining proposal. A `ChunkVerifyManager` maintains draft-ready, verify-ready, and in-flight request sets. A completed draft moves its request to the verify-ready set. After verification, a continuing request returns to that set with an advanced cursor; a terminated proposal moves its request to the draft-ready set if generation has not finished.

Two forward modes separate these operations: `CHUNK_VERIFY_DRAFT` generates and installs a proposal, whereas `CHUNK_VERIFY` executes one verification chunk. The scheduler consults the manager when its regular scheduling path does not produce a batch. Within the manager, ready verification work is prioritized; drafting can replenish the batch when the number of verify-ready requests is below the configured capacity. The integration is conditional, leaving the original scheduling path unchanged when CHARD is disabled.

C.2 Batch Assembly and Boundary Resolution

Proposal queues share contiguous GPU storage indexed by request-pool slot. Each request's queue is a view into this storage. To construct a verification batch, the manager gathers tokens, positions, and cache-slot indices from the selected rows and their current cursors. This avoids concatenating separate request tensors at every chunk. A request's position within its proposal does not determine its position in the batch: retained suffixes and newly generated proposals can therefore share the next target call. A shorter final logical chunk is grouped with requests of the same width.

Continuation requires target confirmation at the chunk boundary. Let s be the cursor, w the current logical width, g the number of newly committed tokens returned by the call, and y its last committed token. For greedy decoding, a proposal with a remaining suffix can continue only when

$$g = w, \qquad s + w < B, \qquad Q[s + w] = y.$$

The last check is essential: the final target distribution predicts the token immediately after the current input segment. Committing w tokens alone does not distinguish acceptance of that boundary token from a correction at the boundary. If the check succeeds, the cursor advances by w and the confirmed boundary token anchors the next call. Otherwise, the accepted prefix and correction are committed and the unused suffix is invalidated. Exhausting the proposal commits the final bonus token and also makes the request eligible for a replacement proposal.

C.3 KV-Cache Lifetime

CHARD tracks committed-prefix progress separately from proposal consumption. Target KV entries computed for accepted input positions are retained, and commitment advances the corresponding metadata rather than copying or recomputing the entire accepted prefix. A newly emitted correction or bonus token becomes the next input anchor; emitting that token does not imply that its target KV has already been materialized.

On rejection, cache entries beyond the valid prefix are invalidated or released, together with the unused proposal state. Continuing requests retain their valid caches and proposal suffixes. When a proposal is exhausted, its temporary reservations can be reclaimed without rolling back the committed prefix. Draft-side allocations are managed separately from target KV and remain associated with the proposal that owns them. Before launching a replacement draft, the runtime prepares its cache reservations, updated sequence position, and committed anchor. These updates allow one request to replace its proposal without resetting the state of other requests in the batch.

C.4 Immediate Redrafting and Host–Device Overlap

After a verification call, a GPU epilogue evaluates the continuation conditions and identifies rows requiring replacement proposals. It gathers their anchors, sequence lengths, and request indices into persistent staging buffers. This is a selection of requests for redrafting; it does not prune positions inside the verification chunk. The number of selected rows is transferred through pinned memory and

consumed at the existing verification-result synchronization point, avoiding a separate host round trip for each request.

The chain-draft path uses these staged inputs to launch replacement drafting within the current scheduler step when a compatible execution path is available. Otherwise, the requests remain draft-ready and use the regular draft path. Requests retaining valid suffixes need not finish their proposals before terminated requests become eligible for recovery. Once installed, replacement proposals join subsequent verification batches alongside those retained suffixes.

To overlap host bookkeeping with device execution, the worker schedules the verification-result transfer and records its completion event before enqueueing replacement drafting. The manager waits for that event, rather than for all subsequent work on the stream, and processes the completed verification results while the GPU executes drafting. Deferred bookkeeping is completed before its state is reused. This overlap removes host work from the launch path without requiring concurrent draft-model and target-model execution.

C.5 Persistent Buffers and CUDA Graph Replay

Repeated chunk execution uses persistent input, output, staging, and attention-metadata buffers. CUDA Graph replay updates these buffers in place, preserving the addresses captured by the graph. Draft and target execution retain their respective logical widths, B and C; changing request membership or proposal cursors changes buffer contents rather than the verification rule. Replacement drafting reuses compatible captured draft executions and falls back to the regular path when a suitable entry is unavailable.

Pinned host buffers are separated by purpose, including result transfer, batch assembly, and proposal installation. Their reuse is ordered with the associated transfers so that one operation cannot overwrite data still being read by another. Queue storage is shared between the manager and worker, allowing the GPU epilogue and host result processing to apply the same boundary checks without rebuilding per-request token arrays.

C.6 Draft-Head Interface

The draft worker exposes proposal tokens and their associated state to the same queue interface. The DSpark-head integration retains the head's sequential Markov correction when producing proposal tokens; its head-specific sampling is not replaced by DFlash's independent token selection. Once the proposal is installed, queue consumption, boundary confirmation, and request-local recovery use the same CHARD execution path. This separates the draft-head computation from the scheduling and verification mechanisms evaluated in the paper.

D Execution Examples and Measurement Conventions

This appendix illustrates the target-confirmed transitions used by CHARD and clarifies the counting and measurement conventions in the evaluation.

D.1 Worked Examples of Chunk Resolution

Consider greedy decoding with proposal width $B = 8$ and chunk width $C = 4$. Let

$$Q = (x_0, x_1, \ldots, x_7),$$

where x_0 is an already committed input anchor and $x_1, \ldots, x_7$ are draft candidates. The first target call takes (x_0, x_1, x_2, x_3) as input and produces next-token predictions (y_1, y_2, y_3, y_4). Each prediction y_j is conditioned on the preceding input prefix. Predictions after the first rejected candidate are not used to extend the committed sequence.

Rejection inside a chunk. Suppose $y_1 = x_1$ but $y_2 \neq x_2$. CHARD commits (x_1, y_2), where y_2 is the correction token, and terminates the proposal. Candidates $(x_2, \ldots, x_7)$ are discarded. The request becomes eligible to redraft from its updated committed prefix, using y_2 as the new anchor. Although this call evaluates four target input positions, it commits only two new tokens.

Rejection at the chunk boundary. Suppose $y_j = x_j$ for $j = 1, 2, 3$, but $y_4 \neq x_4$. CHARD commits (x_1, x_2, x_3, y_4) and discards the remaining proposal. This call commits four tokens, but continuation is invalid because its last token is a correction rather than the queued boundary token. Thus, committing C tokens is not by itself sufficient to retain a proposal suffix.

Continuation with a confirmed boundary. Suppose $y_j = x_j$ for all $j = 1, \ldots, 4$. CHARD commits (x_1, x_2, x_3, x_4) and advances the queue cursor by four positions. The next target call takes (x_4, x_5, x_6, x_7) as input. Here, x_4 is the confirmed input anchor: it is not counted again as a newly generated token. Candidates (x_5, x_6, x_7) are reused from the existing proposal without another draft invocation.

Proposal exhaustion. Following the continuation above, suppose the second target call accepts x_5, x_6, x_7 and produces a final bonus token y_8. It commits (x_5, x_6, x_7, y_8), completing the proposal with eight new output tokens across two target calls. The request then becomes eligible for a replacement proposal anchored at y_8, unless generation has finished.

These cases distinguish proposal continuation from proposal replacement. A retained suffix is consumed only after its boundary has been target-confirmed. A replacement proposal starts from the corrected or fully extended committed prefix. Different requests can take different transitions after the same target call.

D.2 Token and Verification-Work Accounting

Accepted candidates and committed progress. Accepted draft candidates are distinct from committed output tokens. For a proposal resolved by rejection or full acceptance, committed progress G includes the accepted candidates and one correction or final bonus token. The initial input anchor is excluded because it was already committed before verification began.

An intermediate chunk that continues an existing proposal need not produce a correction or bonus token. Its final committed token can instead be the accepted boundary candidate. Consequently, the rule "accepted candidates plus one" applies to a resolved proposal, not independently to every continuing chunk.

Logical positions and target calls. A target call of logical width w evaluates w input positions, including its input anchor. All w positions count toward verification work even if rejection occurs early within the chunk. CHARD avoids executing subsequent chunks of a terminated proposal; it does not retroactively remove computation from the call that detected rejection.

For a proposal consumed through M target calls with logical widths $w_1, \ldots, w_M$, its evaluated position count is

$$V = \sum_{j=1}^{M} w_j.$$

Calls per proposal count that request's participation in target calls, rather than the number of global batched launches divided by the number of requests. In the examples above, an immediate rejection has $V = 4$ and one call, whereas full acceptance has $V = 8$ and two calls.

Logical position counts exclude padding introduced for CUDA Graph execution. The elapsed time of that padding remains included in the end-to-end measurements. The runtime therefore records logical work and wall-clock performance separately.

D.3 Request-Level TPOT and Aggregation

Let request i produce $N_i > 1$ output tokens with completion timestamps $t_{i,1}, \ldots, t_{i,N_i}$. Its request-average time per output token is

$$\mathrm{TPOT}_i = \frac{t_{i,N_i} - t_{i,1}}{N_i - 1}.$$

This interval includes drafting, target verification, and scheduler waiting between the first and last output tokens. Prefill and time to first token are excluded. Tokens committed together share a completion timestamp.

Request-level p50 and p99 are computed over the resulting collection of request-average TPOT values. They are therefore distinct from percentiles of individual inter-token delivery gaps. For the six-workload summaries, metrics are first computed separately for each workload and then averaged; the reported average p99 is not a percentile of pooled requests from all workloads.

For the DSpark-head experiment, the repetition and aggregation protocol is specified in Section 7: throughput is summarized across repeated runs, while acceptance-group medians pool request observations within each fixed group. These summaries retain their respective units of observation rather than treating requests and repeated runs as interchangeable samples.