

Correctness Forensics for Batch Speculative Decoding: Diagnosing the Ragged Tensor Problem

Ranran Haoran Zhang¹, Soumik Dey², Ashirbad Mishra²,
Hansi Wu², Binbin Li², Rui Zhang¹

¹The Pennsylvania State University, University Park, PA, USA

²eBay Inc., San Jose, CA, USA

{hzz5361, rmz5227}@psu.edu

Code: https://github.com/eBay/spec_dec

Abstract

Inference optimizations are routinely evaluated by throughput alone, without verifying output correctness. We conduct a forensic analysis of batch speculative decoding and find that several widely-used implementations silently produce corrupted outputs (repetitive tokens, <unk> symbols) while reporting competitive speed; failures invisible to metrics like ROUGE. We trace the root cause to the ragged tensor problem: variable token acceptance desynchronizes position IDs, attention masks, and KV-cache across a batch. We formalize the synchronization invariants (rectangular alignment and position/KV-cache correspondence) that valid contiguous batching must preserve and show that maintaining them incurs superlinear alignment overhead. EQSPEC enforces the invariants without custom kernels; EXSPEC schedules same-length sequences to bypass realignment. On SpecBench across three model families, EXSPEC reaches 3× throughput at batch size 8. Our methods remove catastrophic synchronization failures but achieve 90.8–97.3% exact match against batch-size-1 non-speculative decoding. The residual mismatches are consistent with batch-dependent floating-point effects.

1 Introduction

LLM inference optimizations are routinely evaluated by throughput alone, with output correctness assumed rather than verified. We argue that *output equivalence*, the requirement that an optimized system reproduce the same distribution as standard decoding, should be a first-class evaluation criterion. A system that produces garbage quickly is not an improvement.

This paper conducts a forensic analysis of batch speculative decoding using an observe→diagnose→formalize→validate audit protocol. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) has a small draft

model propose tokens that a target model verifies in parallel; its defining requirement is output equivalence (Leviathan et al., 2023; Xia et al., 2024). Extending speculation to batches raises the ragged tensor problem: sequences accept different numbers of draft tokens, desynchronizing position IDs, attention masks, and KV-cache on the standard contiguous layout.

Figure 1 shows that several widely-used implementations, BSP (Su et al., 2023) and DSD (Yan et al., 2025), report competitive throughput while producing repetitive tokens or <unk> symbols under greedy decoding. These are fundamental correctness violations that metrics such as ROUGE (Qian et al., 2024) can mask, raising a **reproducibility concern**: published speedups may not be comparable unless output equivalence is verified. We use exact match and partial match (fraction of tokens before first divergence) to detect corruption and localize its cause: immediate divergence indicates position-ID errors; gradual decay reveals KV-cache drift.

We trace these failures to the ragged tensor problem (Qian et al., 2024). For contiguous layouts, we formalize rectangular alignment (I1) and position/KV-cache correspondence (I2) as synchronization conditions for algorithmic output equivalence (Section 3.1). EQSPEC enforces them without custom kernels. The invariants require realigning the batch after every verification round; this overhead grows superlinearly, reaching 40% at batch size 8 (Section 3.2), a cost intrinsic to contiguous layouts that explains why prior methods either break equivalence or fail to scale. EXSPEC groups same-length sequences to bypass realignment, reaching 3× throughput at batch size 8 (Section 4.3).

We instantiate this audit for batch speculative decoding: task-specific diagnostics, synchronization invariants tied to the ragged-tensor problem, and empirical validation. The invariants concern prop-

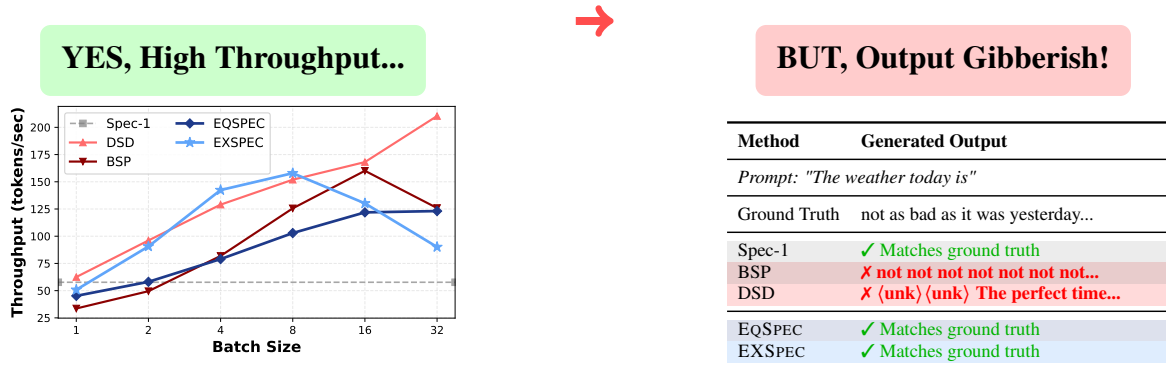


Figure 1: Batch speculative decoding on Vicuna-7B/68M: Existing methods achieve high throughput but **violate the fundamental requirement of output equivalence** by producing corrupted outputs. EQSPEC and EXSPEC avoid this catastrophic corruption while retaining competitive performance.

erties of tensor-batched inference (position tracking, mask alignment, cache synchronization) that arise whenever sequences grow at different rates; we validate them only for batch speculative decoding. Our contributions:

- **A forensic audit** revealing that several batch speculative decoding implementations produce corrupted outputs invisible to standard metrics, with diagnostics that distinguish failure signatures and isolate root causes (Appendix A).
- **A formal correctness specification:** two synchronization invariants for output equivalence, enforced by EQSPEC without custom kernels. Across three model families, our methods achieve 90.8–97.3% exact match against batch-size-1 non-speculative decoding.
- **A cost analysis and scheduling-based mitigation:** realignment overhead grows super-linearly under contiguous layouts; EXSPEC groups same-length sequences to bypass it, reaching 3× throughput at batch size 8.

2 Failure Modes and the Audit Protocol

Inference optimizations that modify batch-level tensor state (position IDs, attention masks, KV-cache) are prone to silent correctness failures. We organize our analysis as a four-stage audit: *observe* divergence from a reference oracle under deterministic decoding; *diagnose* via exact and partial match, separating immediate position-ID failures from gradual KV-cache drift; *formalize* the invariants each failure violates (Section 3.1); *validate* that enforcing them restores equivalence and quantify the cost (Section 4).

We apply this to batch speculative decoding. When sequences accept different numbers of draft

tokens, the resulting tensors become irregular: the ragged-tensor problem (Figure 2). Approaches divide by *KV-cache layout* (Table 1). **Contiguous** layouts (Family A) maintain a rectangular [batch, seq] tensor and must restore that shape after every verification round. **Variable-length** layouts (Family B) pack sequences into a flat buffer indexed by cumulative lengths, dissolving the alignment requirement by construction; this is the route taken by production systems (vLLM v1, SGLang).

Family A: contiguous layout. Four strategies attempt to keep the rectangular tensor, each producing a distinct failure signature. *Masking* reassigns position IDs for rejected tokens but leaves the KV-cache desynchronized; BSP (Su et al., 2023) and EAGLE’s experimental batch code¹ (Li et al., 2026) take this route and corrupt output (Figure 1). Our partial-match diagnostic reveals BSP’s signature: initially correct output before KV-cache drift triggers repetitive loops (gradual decay, not immediate failure). Making masking correct requires custom kernels (Qian et al., 2024). *Roll-back* truncates all sequences to the batch-minimum accepted length (Wolf et al., 2020; kamilakesbi, 2024): correct but discards verified tokens, with waste growing until throughput collapses. *Batch expansion* (vLLM v0) duplicates each sequence into K draft prefixes: correct but $K \times$ memory, deprecated due to OOM failures. *Dynamic padding* realigns the batch via unpad–append–repad each round; DSD (Yan et al., 2025) attempts it but desynchronizes position IDs/KV-cache and samples the bonus token from the draft model. Our diagnostic shows near-zero partial match, indicating immediate divergence. Only dynamic padding is correct, token-preserving, portable, and memory-bounded,

¹<https://github.com/SafeAILab/EAGLE/issues/250>

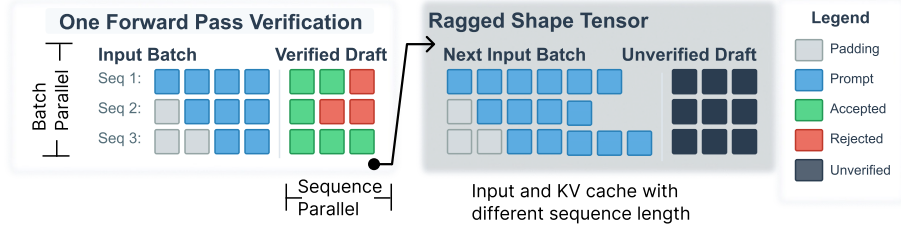


Figure 2: The ragged tensor problem in batch speculative decoding. Differing numbers of accepted draft tokens across sequences in the same batch lead to ragged-shaped input IDs tensors and KV-cache that disrupt subsequent batch operations.

Family	Strategy	Correct (output-equiv.)	Keeps all tokens	Portable (std. kernels)	Bounded mem. (no $K \times$)
A: contiguous	Masking	✗	✓	✗	✓
	Rollback	✓	✗	✓	✓
	Batch expansion	✓	✓	✓	✗
	Dynamic padding (ours)	✓	✓	✓	✓
B: variable-length	Flat / paged packing	✓	✓	✗	✓

Table 1: Design space for the ragged-tensor problem, organized by KV-cache memory layout.

provided synchronization is done correctly, which none of the implementations we tested achieves (Appendix A).

Family B: variable-length layout. Continuous-batching systems (vLLM v1, SGLang) store the KV-cache in a flat buffer indexed by cumulative lengths, so ragged acceptance never requires repadding and I1 does not arise. Position consistency (I2) is still required: rejected tokens must be rolled back in both position IDs and the KV-cache. This layout depends on paged-attention kernels available only on CUDA/Triton; on other accelerators (Ascend, Intel Gaudi, Apple Metal) and for offline batch processing, the contiguous family remains the only option. Family B is thus the production default where its kernels exist; the contiguous family, our focus, covers the remaining hardware and workloads (Appendix F).

2.1 Output Equivalence as a Diagnostic Criterion

At the algorithmic level, *output equivalence* requires greedy decoding to produce the same token sequence as standard autoregressive generation; for temperature > 0 , it requires preserving the target distribution. Batched floating-point execution can alter argmax decisions, so empirical exact match diagnoses corruption but does not guarantee bit-exact identity. We use greedy decoding because it exposes synchronization errors more clearly than stochastic sampling. BSP and DSD fail this diagnostic regardless of throughput (Figure 1).

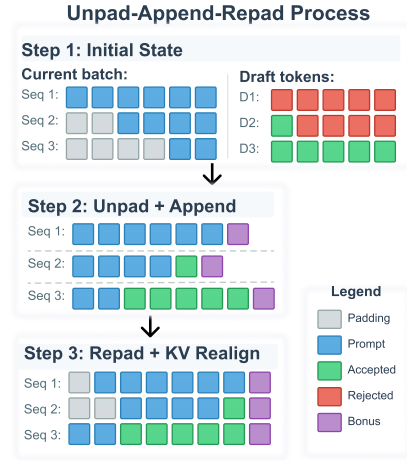


Figure 3: EQSPEC synchronizes via unpad-append-repad.

3 Correctness Specification and Cost

Having classified failure modes (Stage 1–2), we now formalize the invariants correctness requires and quantify their cost (Stage 3–4). EQSPEC specifies and enforces the minimal synchronization invariants (Section 3.1); EXSPEC reduces their overhead by grouping same-length sequences for which realignment vanishes (Section 3.2).

3.1 EQSPEC: Formalizing the Correctness Specification (Stage 3)

The diagnostic stage identified *what* fails; we now formalize *why*—the invariants whose violation produces each observed failure signature.

Rectangular Alignment. Let B denote the batch size, and let sequence i have padding length p_i , content length c_i , and a_i newly accepted tokens (including the bonus token) after verification round t . A valid implementation must maintain:

$$\forall i \in [1, B], \quad p_i^{(t)} + c_i^{(t)} = L^{(t)} \quad (\text{I1})$$

where $L^{(t)}$ is the padded sequence length at round t .

Position and KV-Cache Correspondence. Position IDs must be derived consistently from the attention mask:

$$\text{pos}_i[j] = \max \left(0, \sum_{k=0}^j \mathbf{1}[\text{tok}_i[k] \neq \text{pad}] - 1 \right) \quad (\text{I2})$$

which guarantees contiguous position IDs starting from zero at the first content token, as required by RoPE (Su et al., 2024) and other position-aware attention mechanisms. At each content position, the KV-cache entry must also be the target-computed entry for the token at that logical position. Continuous-batching systems (vLLM, SGLang) sidestep I1 through variable-length packing (Family B, Section 2), but I2 remains a prerequisite for correctness regardless of batching strategy.

After batch verification, sequence i accepts a_i tokens, breaking rectangular alignment since a_i varies across sequences. To restore I1, the new padding must satisfy:

$$\forall i \in [1, B], \quad p_i^{(t+1)} + c_i^{(t)} + a_i = L^{(t+1)} \quad (1)$$

where $L^{(t+1)} = \max_j (c_j^{(t)} + a_j)$. This yields the per-sequence padding offset:

$$\delta_i = p_i^{(t+1)} - p_i^{(t)} = (L^{(t+1)} - L^{(t)}) - a_i \quad (2)$$

We prove that this update rule preserves rectangular alignment (I1) and the cache-correspondence clause of I2 across all verification rounds (Theorem B.1): because realignment relocates cached entries rather than recomputing them, it introduces no numerical drift. The position-ID clause of I2 is enforced by recomputing position IDs after each repadding operation. Together with target-model bonus sampling, maintaining I1 and I2 is sufficient for algorithmic output equivalence (Corollary B.2).

Algorithm 1 summarizes EQSPEC’s three-phase loop.

Algorithm 1 EQSPEC: Minimalism Batch Spec

Require: Draft model $\mathcal{M}_d$, Target model $\mathcal{M}_t$, Prompts $\mathcal{P}$, Max tokens T , Draft length K
Ensure: Generated sequences $\mathcal{S}$
1: $\mathcal{S}, L \leftarrow \text{Tokenize}(\mathcal{P}) \triangleright$ Left-pad batch to width L
2: $KVCache \leftarrow \emptyset$
3: **while** until max new tokens **do**
4: **Phase 1: Draft Generation**
5: $D \leftarrow \mathcal{M}_d.\text{Generate}(\mathcal{S}, K)$
6: **Phase 2: Batch Verification**
7: $\triangleright$ See Alg. 3 in Appendix for BatchVerify
8: $A, B, KVCache \leftarrow \text{BatchVerify}(\mathcal{M}_t, \mathcal{S}, D, KVCache)$ $\leftarrow$
9: $\triangleright$ See Figure 3 for illustration on index offset.
10: **Phase 3: Realign by Offset δ**
11: **for** each sequence i in batch **do**
12: $\mathcal{S}[i] \leftarrow \mathcal{S}[i] \oplus A[i] \oplus B[i] \triangleright$ Append accepted + bonus; $a_i = |A[i]| + 1$
13: **end for**
14: $L' \leftarrow \max_j (c_j + a_j) \triangleright$ New batch width $L^{(t+1)}$
15: $\delta_i \leftarrow (L' - L) - a_i \quad \forall i \triangleright$ Per-sequence offset, Eq. 2
16: $\mathcal{S}, KVCache, L \leftarrow \text{Realign}(\mathcal{S}, KVCache, \delta) \triangleright$ Shift padding & KV by δ_i ; reset pos-IDs (I2)
17: **end while**
18: **return** $\mathcal{S}$

Unpad-Append-Repad. Figure 3 illustrates the realignment phase. The offset δ_i realigns all batch state: token sequences receive fresh left padding, attention masks are recomputed, position IDs are recalculated via I2, and KV-cache entries are shifted by δ_i positions. The bonus token, sampled from the target distribution after the forward pass completes (Algorithm 3, Appendix D), lacks KV-cache entries and must be included in the next forward pass.

3.2 EXSPEC: Quantifying and Reducing the Cost of Correctness (Stage 3–4)

Alignment Overhead. Beyond the drafting and verification costs (c_{draft} , c_{verify}) of single-sequence speculation (Leviathan et al., 2023), batched speculation adds an alignment overhead $c_{\text{overhead}}(B) = c_{\text{pad}}(B) + c_{\text{kv}}(B)$, absent from single-sequence analysis, where $c_{\text{pad}}(B)$ covers unpad–append–repad and $c_{\text{kv}}(B)$ KV-cache realignment. The rest of this section quantifies it.

Theorem 3.1 (Alignment Overhead under Contiguous Layout). *Restoring rectangular alignment (Invariant I1) each round pads every sequence up to the batch’s longest acceptance $M_B = \max_i a_i$, so the per-round realignment work is the total gap $\sum_{i=1}^B (M_B - a_i)$. With acceptances $a_i \in [0, K]$ drawn i.i.d. with mean μ , where K is the per-round draft budget, this work has expectation $B(E[M_B] - \mu)$. Since $E[M_B]$ rises with B , it grows strictly faster than linearly over the batch sizes of interest, reaching the linear rate*

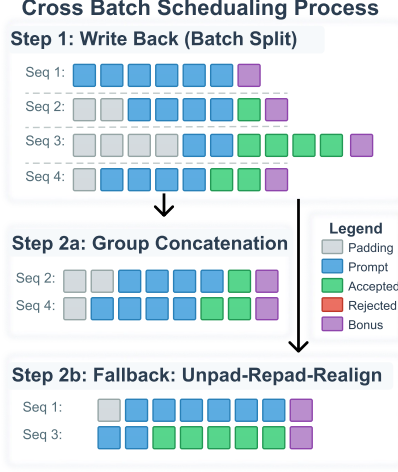


Figure 4: EXSPEC pools ragged sequences by length, avoiding realignment; only unmatched sequences need syncing, turning fixed overhead into optional cost.

$(K - \mu)B$ only as $E[M_B] \rightarrow K$. The alignment cost $c_{\text{coverhead}}(B)$ scales with this work.

This cost is a property of the contiguous layout that variable-length layouts (Family B) avoid; within it, the KV-cache dominates, since its rank-4 tensors (batch $\times$ heads $\times$ sequence $\times$ dimension) are reallocated on every padding change while c_{draft} and c_{verify} stay stable under GPU parallelism. The alignment overhead also grows with variance in acceptances across sequences; Section 4.3 confirms this empirically. This explains the pattern the audit’s diagnostic stage surfaced: prior methods either skip synchronization (breaking equivalence) or pay the full superlinear cost (failing to scale)—no middle ground under contiguous layouts.

However, the speedup analysis reveals a key insight: when all sequences accept the same number of tokens, $\delta_i = 0$ and realignment overhead vanishes (Corollary B.4). Rather than accelerating these operations, EXSPEC sidesteps them through scheduling.

Cross-Batch Scheduling. EXSPEC differs from EQSPEC in how it manages sequence lifecycles. Rather than maintaining fixed batches that require realignment after every verification, we introduce a *SequencePool* that holds sequences individually in their ragged states. This enables three optimizations: (i) sequences that complete (reach EOS) are immediately removed, avoiding wasted computation; (ii) *lazy realignment* defers synchronization until strictly necessary; and (iii) *dynamic batch formation* over a sliding window of $W > B$ sequences greatly increases the chance of finding

same-length groups.

Figure 4 illustrates the flow. After verification, ragged sequences return directly to the pool without realignment. The scheduler scans the active window and attempts to form batches of identical length. Same-length sequences concatenate directly (no padding adjustments, no position-ID recomputation, no KV-cache realignment), bypassing $c_{\text{coverhead}}(B)$ entirely. Only when same-length grouping fails do we fall back to unpad–append–repad. Combined with prompt-length sorting, grouping rates approach unity for similar workloads, turning constant realignment into the rare case. The complete procedure is detailed in Algorithm 2 (Appendix C).

4 Experimental Validation

We validate the diagnostics and the correctness specification along two dimensions conflated in prior work: **output equivalence** and **throughput**.

4.1 Experimental Setup

Models. To test robustness across architectures, we evaluate three target–draft pairs: Vicuna-7B/68M (Zheng et al., 2023a), Qwen3-8B/0.6B (Yang et al., 2025), and GLM-4-9B/0.6B (GLM et al., 2024). Unless otherwise noted, experiments use NVIDIA A100 80GB GPUs, PyTorch 2.7, HuggingFace Transformers 4.51.3, five draft tokens per speculation round, and greedy decoding for determinism.

Evaluation and Datasets. We use SpecBench (Xia et al., 2024), a widely adopted benchmark for speculative decoding research. We also use Multi30k (Elliott et al., 2016) for a controlled EXSPEC study that contrasts random sampling with an identical-length subset, isolating sequence-length diversity as the driver of grouping rate. For the main evaluation, we measure: (1) *Throughput*: tokens/s across batch sizes; and (2) *Output Equivalence*: exact-match rate (full-sequence equivalence with non-speculative decoding) and partial-match rate (fraction of tokens matching until the first divergence). The partial-match metric helps localize failure modes. Early divergence typically indicates position-ID or KV-cache misalignment.

Batch Speculative Decoding Compared. Following our failure-mode taxonomy (Section 2), we evaluate: (1) *Masking approaches*: **BSP** (Su

et al., 2023) attempts masking with adaptive speculation but suffers position-ID inconsistencies (BASS (Qian et al., 2024) also follows this approach but requires custom CUDA kernels, limiting generality); (2) *Dynamic-padding approaches*: **DSD** (Yan et al., 2025) explores padding but mishandles the KV-cache, while **EQSPEC** implements correct synchronization and **EXSPEC** adds cross-batch scheduling; (3) *Reference baselines*: **Spec-1** (batch-size-1 speculation from Hugging Face Transformers), which does not support batch speculative decoding². Production systems (vLLM, SGLang; see also Sections 2 and 3.1) employ continuous batching with paged attention, which sidesteps the ragged tensor problem through different architectural choices. SGLang supports only EAGLE-family drafters; vLLM v1 serves as a numerical control in Section 4.2, while broader vLLM and SGLang results remain in Appendix F. No existing implementation uses rollback due to its inherent wastefulness.

4.2 Validating the Diagnostics (Stage 1–2)

The first two audit stages ask: do our diagnostics detect failures standard metrics miss, and do the signatures localize root causes? ROUGE (Qian et al., 2024) is unsuitable as a correctness oracle: corrupted repetitive text can still score reasonably.

Table 2 validates the diagnostics on two fronts. First, the metrics *detect*: DSD and BSP show catastrophic degradation invisible to ROUGE. Second, they *distinguish* root causes. DSD’s near-zero scores on both metrics indicate immediate position-ID misalignment (failure from the first token). BSP shows higher partial match (up to 39.7%) but low exact match: gradual decay as KV-cache drift misdirects attention, triggering repetition. These complementary signatures (immediate vs. gradual) map to the bug taxonomy (Table 3): DSD suffers from batch-independent errors compounded by synchronization failures; BSP accumulates KV-cache drift from position-ID desynchronization.

Enforcing the invariants removes the catastrophic failure signatures, but our methods achieve 90.8–97.3% exact match against the batch-size-1 non-speculative oracle rather than bit-exact identity (Table 2). The remaining divergence is consistent with batch-dependent floating-point effects and argmax tie-breaking (He and Lab, 2025; Gond et al., 2026; Liu et al., 2026); Table 4 shows that

even non-speculative batched decoding can diverge from a batch-size-1 anchor.

vLLM v1 numerical control. Table 4 shows that non-speculative batched decoding also diverges from the batch-size-1 anchor when batch-invariant kernels are unavailable, separating numerical divergence from speculative-decoding synchronization. Because this vLLM configuration differs from our main setup, we treat it as a control rather than a matched performance comparison.

From a reproducibility perspective, these results underscore that published throughput numbers for batch speculative decoding should not be taken at face value without accompanying correctness verification. BSP and DSD reported competitive throughput in their respective publications, yet our audit reveals near-zero output equivalence: the measured tokens/s, while real, correspond to corrupted outputs rather than valid generation. We provide additional analysis of production systems (vLLM, SGLang) in Appendix F.

4.3 Validating the Cost Analysis (Stage 4)

Five studies validate the cost predictions from the formalization stage: batch scaling efficiency, alignment overhead growth, sequence diversity as bottleneck, mechanistic profiling, and latency–throughput tradeoff.

Batch scaling efficiency. Figure 5(a) normalizes each method’s throughput to its own BS=1 baseline, isolating batch-level scaling from absolute performance. The *No-Ragged-Scaling* reference line is non-speculative decoding normalized the same way. EXSPEC initially exceeds this reference because speculation converts memory-bound generation into compute-bound verification that benefits more from GPU parallelism; at larger batch sizes, alignment overhead dominates and the advantage inverts. The key finding: valid batch speculative decoding need not sacrifice scaling efficiency up to BS=8, whereas prior methods either produce corrupted outputs or fail to scale.

Alignment overhead growth. Figure 5(b) quantifies realignment costs via two metrics: percentage of total time spent on alignment (OH%) and per-round alignment time (OH/|Spec|). Overhead rises from $\sim 13\%$ at BS=1 to nearly 40% at BS=32, with per-round costs increasing even more. This matches our prediction that $c_{\text{pad}}(B) + c_{\text{kv}}(B)$ grows superlinearly with B . This is a structural cost of the

²<https://github.com/huggingface/transformers/issues/32165>

Method	Vicuna				Qwen3				GLM4			
	Batch 1		Batch 4		Batch 1		Batch 4		Batch 1		Batch 4	
	E	P	E	P	E	P	E	P	E	P	E	P
Non-Spec-Batch	—	—	53.8	98.2	—	—	92.9	96.5	—	—	93.3	97.2
Spec-1	97.1	98.4	—	—	94.6	97.2	—	—	96.0	98.0	—	—
EQSPEC	97.3	98.6	92.1	98.6	94.6	96.9	92.3	95.7	96.7	98.1	96.5	98.3
EXSPEC	97.3	98.6	90.8	97.6	94.6	96.9	95.0	97.1	96.7	98.1	95.2	97.7
DSD	0.0	8.1	0.0	2.2	0.2	2.2	0.0	0.6	0.0	1.0	0.0	0.8
BSP	1.9	39.7	0.2	31.3	3.5	19.9	2.1	12.6	1.0	15.3	0.6	8.1

Table 2: Output equivalence of speculative-decoding implementations, measured by exact (E) and partial (P) match against batch-size-1 non-speculative decoding. EQSPEC and EXSPEC do not achieve bit-exact identity; DSD and BSP exhibit near-zero exact match.

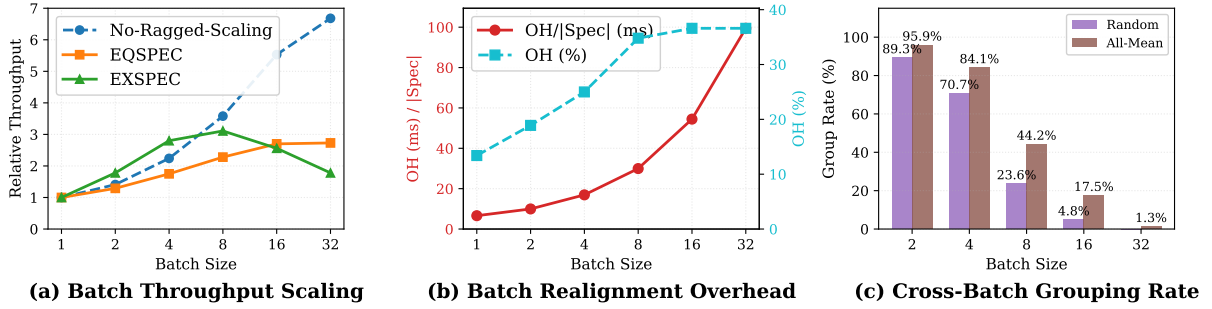


Figure 5: Decomposing batch speculative decoding performance. (a) Batch scaling efficiency: each method’s throughput normalized to its own BS=1 baseline, isolating scaling behavior from absolute performance. The No-Ragged-Scaling line shows measured autoregressive decoding throughput (not a theoretical bound), serving as a reference for how standard batching scales without ragged tensor overhead. (b) Alignment overhead grows superlinearly with batch size, consuming up to 38% of inference time, validating that $c_{\text{overhead}}(B)$ dominates at scale. (c) Cross-batch grouping rates on Multi30k for random vs. uniform-length sequences, showing that length homogeneity transforms grouping effectiveness.

contiguous layout: the very operations required to maintain the invariants (unpad–append–repad and KV-cache realignment) become increasingly expensive as sequence lengths diverge. The superlinear growth is not an implementation inefficiency but a property of maintaining synchronization under contiguous layouts (Theorem 3.1), which helps explain why prior methods either break equivalence (skipping the expensive synchronization) or fail to scale (paying the full cost).

Grouping rate $\times$ sequence-length distribution.

Figure 5(c) probes whether cross-batch scheduling limits are algorithmic or circumstantial via Multi30k. Comparing random sampling to an All-Mean subset with identical lengths isolates the bottleneck: sequence diversity, not the method. Random sampling shows grouping rates collapsing with batch size, whereas the All-Mean configuration maintains high grouping effectiveness even at moderate scales, substantially reducing $c_{\text{overhead}}(B)$. This contrast suggests that length-based bucketing or dynamic sorting can push real workloads toward the ideal. Production traces are

length-heterogeneous (Bai et al., 2026), and grouping similar-length requests is an established and effective serving technique (Zheng et al., 2023b; Bai et al., 2026); EXSPEC applies this principle under the synchronization invariants.

Overhead Anatomy. Table 5 shows EXSPEC achieves 64% higher throughput by cutting verification calls by one-third and halving alignment overhead through same-length grouping. The tradeoff is memory locality: dynamic batching scatters KV-cache entries, making each verification slower, but fewer total operations wins.

Latency–throughput tradeoff. Simulating online serving with dynamic arrivals from shuffled multi-turn SpecBench conversations, Table 6 reveals two operating points. EXSPEC’s grouping raises throughput but delays early requests until matching partners arrive, inflating tail latency, worst at small batch sizes where throughput is low; EQSPEC realigns every round for stable, predictable tail latency. The tradeoff is deployment-driven: EXSPEC for offline throughput, EQSPEC

Bug Category	Specific Issue	System	BS=1	BS>1
Batch-independent implementation errors	Bonus token sampled from draft model instead of target model	DSD	✓	✓
	KV-cache contains rejected tokens; position IDs auto-calculated incorrectly	BSP	✓	✓
Ragged tensor synchronization (batch-specific)	Unaligned KV-cache across sequences with different accepted lengths	DSD, BSP	✗	✓
	Position ID desynchronization across ragged sequences	BSP	✗	✓

Table 3: Taxonomy of implementation errors in batch speculative decoding. Batch-independent errors corrupt outputs at any batch size and could be fixed with careful code review. Ragged tensor synchronization errors are batch-specific and represent the core challenge this paper addresses.

GPU	Batch-inv.	Non-spec BS>1	Spec BS=1	Spec BS>1
H100	on	80/80	80/80	80/80
	off	66/80	80/80	62/80
A100	on	62/80	58/80	56/80
	off	58/80	61/80	63/80

Table 4: vLLM v1 numerical control (Qwen3-4B/0.6B, $K=3$, 80 MT-Bench prompts, greedy): exact matches against the non-speculative BS=1 output.

Method	TPS	Spec	ms/V	V%	D%	OH%
EQSPEC	95.6	1469	24.9	44.9	27.4	27.7
EXSPEC	156.4	952	29.1	55.9	29.5	14.6

Table 5: Overhead anatomy of batch speculation methods. V%=verification, D%=draft, OH%=overhead. Despite slower per-verification time, EXSPEC achieves higher throughput by reducing verification calls and alignment overhead.

for latency-sensitive serving.

5 Related Work

Speculative Decoding and Batch Extension. Speculative decoding verifies draft tokens in parallel, either sequentially (Xia et al., 2023; Santilli et al., 2023; Yang et al., 2023; Hooper et al., 2023; Zhang et al., 2024; Fu et al., 2024) or via tree structures (Miao et al., 2024; Spector and Re, 2023; Sun et al., 2023; He et al., 2024; Cai et al., 2024; Li et al., 2024b,a, 2026; Tang et al., 2025). Exact variants preserve the target distribution; approximate schemes (Kim et al., 2023; Zhong et al., 2025) trade exactness for speed. We assume lossless verification so that any deviation signals a bug. Recent work parallelizes multiple drafts per request (Stewart et al., 2024; Lee et al., 2024) but still verifies independently. Extending speculation to a batch of requests introduces the ragged tensor problem: when sequences accept different numbers of draft

BS	Method	Tok/s	P90	P99
1	EQSPEC	15.20	12.96	16.53
	EXSPEC	15.78	8.32	9.42
2	EQSPEC	26.70	14.78	19.06
	EXSPEC	30.54	114.89	134.03
4	EQSPEC	46.11	16.23	19.13
	EXSPEC	52.44	53.21	70.35
8	EQSPEC	76.03	19.23	22.50
	EXSPEC	77.54	19.13	33.93

Table 6: Latency-throughput tradeoff under simulated online serving with heterogeneous request lengths. Throughput in tokens/s; P90/P99 are the 90th/99th-percentile request-completion latency (s). EXSPEC maximizes throughput while EQSPEC keeps tail latency stable; setup and full breakdown in Appendix E.

tokens, the resulting variable-length tensors break GPU-friendly rectangular operations (Qian et al., 2024). Existing approaches face fundamental limitations detailed in Section 2. Position ID/masking approaches like BSP (Su et al., 2023) suffer from position ID inconsistencies across iterations. Dynamic padding approaches reveal critical implementation errors: DSD (Yan et al., 2025) samples bonus tokens from the draft model’s distribution rather than the target model’s, violating output equivalence. This error compounds with improper KV-cache handling, producing corrupted outputs: BSP generates repetitive tokens, DSD produces <unk> symbols. BASS (Qian et al., 2024) sidesteps these issues through custom CUDA kernels but sacrifices portability. Production systems (vLLM, SGLang) avoid the problem through a different memory layout, continuous batching with variable-length and paged attention, rather than the synchronization we study (Appendix F). To our knowledge, no prior work formalizes the synchronization invariants that correctness requires; failures have been

observed but not systematically analyzed.

Correctness Verification and Reproducibility in LLM Inference. Verifying the correctness of inference optimizations remains underexplored. Liu et al. (2026) report large gaps between theoretical and measured speculative-decoding speedup on vLLM and note output-length variation across batch sizes, attributing it to numerical non-determinism without further diagnosis; He and Lab (2025) characterize floating-point non-determinism without addressing algorithmic correctness. Our diagnostic metrics separate this numerical noise from the synchronization failures we identify, drawing on fault-localization ideas from software engineering (spectrum-based localization (Jones and Harrold, 2005; Abreu et al., 2007) and delta debugging (Zeller and Hildebrandt, 2002)) that localize root causes from observable failure patterns.

6 Conclusion

Throughput without correctness verification is unreliable. Our forensic audit of batch speculative decoding followed four stages: *observe* corrupted outputs in published implementations invisible to ROUGE; *diagnose* via exact/partial match, distinguishing immediate position-ID failures from gradual KV-cache drift; *formalize* the synchronization invariants whose violation produces each signature, proving superlinear realignment cost under contiguous layouts; *validate* that enforcing the invariants removes catastrophic synchronization failures without bit-exact identity (90.8–97.3% exact match), while EXSPEC cuts the cost via same-length scheduling (3× throughput at batch size 8). For practitioners: verify I1/I2 under greedy decoding; use exact match to detect corruption and partial match to localize it (high partial / low exact $\Rightarrow$ cache drift; both low $\Rightarrow$ position-ID errors). We hope the audit approach transfers to other tensor-batched settings, though validated here only for batch speculative decoding.

Limitations

Our case study is limited to the contiguous KV-cache setting; we do not integrate EQSPEC or EXSPEC into continuous-batching systems such as vLLM or SGLang, whose variable-length and paged-attention layouts avoid the ragged-tensor problem by construction (Section 2). Within our case study, the advantage is confined to moderate batch sizes, since realignment overhead

grows superlinearly and EXSPEC’s gains depend on sequence-length homogeneity that heterogeneous workloads erode, while absolute speedup is bounded by draft-model quality. EXSPEC further trades tail latency for throughput: delaying requests to form same-length groups suits offline, throughput-oriented batch processing, whereas EQSPEC realigns each round and offers stable per-request latency for latency-sensitive online serving. We do not achieve bit-exact identity with batch-size-1 non-speculative decoding (90.8–97.3% exact match); residual mismatches are consistent with batch-dependent floating-point effects.

Acknowledgments

This work is supported by an eBay’s Research and University Partnership for Technology (eRUPT) grant.

References

- Rui Abreu, Peter Zoetewij, and Arjan J.C. Van Gemund. 2007. On the accuracy of spectrum-based fault localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*, pages 89–98. IEEE.
- Fengyao Bai, Hongbin Zhang, Zhitao Chen, Jiangsu Du, Zhiguang Chen, and Yutong Lu. 2026. Aligned-serve: Orchestrating prefix-aware batching to build a high-throughput and computing-efficient llm serving system. *Proceedings of the ACM on Management of Data*, 4(3 (SIGMOD)):1–25.
- Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. *Medusa: Simple LLM inference acceleration framework with multiple decoding heads*. *CoRR*, abs/2401.10774.
- Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.
- Desmond Elliott, Stella Frank, Khalil Sima’an, and Lucia Specia. 2016. *Multi30k: Multilingual english-german image descriptions*. In *Proceedings of the 5th Workshop on Vision and Language*, pages 70–74. Association for Computational Linguistics.
- Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. *Break the sequential dependency of llm inference using lookahead decoding*. *Preprint*, arXiv:2402.02057.
- Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiada

- Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, and 37 others. 2024. [Chatglm: A family of large language models from glm-130b to glm-4 all tools](#). *Preprint*, arXiv:2406.12793.
- Raja Gond, Aditya K Kamath, Arkaprava Basu, Ramachandran Ramjee, and Ashish Panwar. 2026. [Llm-42: Enabling determinism in llm inference with verified speculation](#). *Preprint*, arXiv:2601.17768.
- Horace He and Thinking Machines Lab. 2025. [Defeating nondeterminism in llm inference](#). *Thinking Machines Lab: Connectionism*. <https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/>.
- Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D. Lee, and Di He. 2024. REST: retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1582–1595.
- Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Hasan Genc, Kurt Keutzer, Amir Gholami, and Yakun Sophia Shao. 2023. [SPEED: speculative pipelined execution for efficient decoding](#). *CoRR*, abs/2310.12072.
- James A. Jones and Mary Jean Harrold. 2005. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, pages 273–282. ACM.
- kamilakesbi. 2024. Enable speculative decoding with batch size > 1 (github issue #32165). <https://github.com/huggingface/transformers/issues/32165>. Hugging Face *transformers*. Opened 2024-07-23. Accessed 2025-09-24.
- Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W. Mahoney, Amir Gholami, and Kurt Keutzer. 2023. [Speculative decoding with big little decoder](#). In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.
- Minjae Lee, Wonjun Kang, Minghao Yan, Christian Classen, Hyung Il Koo, and Kangwook Lee. 2024. [In-batch ensemble drafting: Toward fast and robust speculative decoding for multimodal language models](#).
- Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.
- Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024a. Eagle-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7421–7432.
- Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. Eagle: Speculative sampling requires rethinking feature uncertainty. In *International Conference on Machine Learning*, pages 28935–28948. PMLR.
- Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2026. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. *Advances in Neural Information Processing Systems*, 38:136737–136756.
- Xiaoxuan Liu, Jongseok Park, Langxiang Hu, Woosuk Kwon, Zhuohan Li, Chen Zhang, Kuntai Du, Xiangxi Mo, Kaichao You, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. 2025. [Turbospec: Closed-loop speculation control system for optimizing llm serving goodput](#). *Preprint*, arXiv:2406.14066.
- Xiaoxuan Liu, Jiaxiang Yu, Jongseok Park, Ion Stoica, and Alvin Cheung. 2026. Speculative decoding: Performance or illusion? *arXiv preprint arXiv:2601.11580*.
- Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunshen Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2024. [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, ASPLOS '24*, page 932–949, New York, NY, USA. Association for Computing Machinery.
- Haifeng Qian, Sujun Kumar Gonugondla, Sungsoo Ha, Mingyue Shang, Sanjay Krishna Gouda, Ramesh Nallapati, Sudipta Sengupta, Xiaofei Ma, and Anoop Deoras. 2024. Bass: Batched attention-optimized speculative sampling. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 8214–8224.
- Andrea Santilli, Silvio Severino, Emiliano Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodolà. 2023. [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)*, ACL 2023, Toronto, Canada, July 9-14, 2023, pages 12336–12355. Association for Computational Linguistics.

- SGLang Team. 2025. Enabling deterministic inference for sglang. <https://lmsys.org/blog/2025-09-22-sglang-deterministic/>. LMSYS Org Blog, published Sep 22, 2025.
- Benjamin Spector and Chris Re. 2023. [Accelerating LLM inference with staged speculative decoding](#). *CoRR*, abs/2308.04623.
- Lawrence Stewart, Matthew Trager, Sujun Kumar Gonugondla, and Stefano Soatto. 2024. The n-grammys: Accelerating autoregressive inference with learning-free batched speculation. *arXiv preprint arXiv:2411.03786*.
- Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.
- Qidong Su, Christina Giannoula, and Gennady Pekhimenko. 2023. The synergy of speculative decoding and batching in serving large language models. *arXiv preprint arXiv:2310.18813*.
- Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix X. Yu. 2023. [Spectr: Fast speculative decoding via optimal transport](#). In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.
- Bangsheng Tang, Carl Chengyan Fu, Fei Kou, Grigory Sizov, Haoci Zhang, Jason Park, Jiawen Liu, Jie You, Qirui Yang, Sachin Mehta, and 1 others. 2025. Efficient speculative decoding for llama at scale: Challenges and solutions. *arXiv preprint arXiv:2508.08192*.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. [Transformers: State-of-the-art natural language processing](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.
- Zhaoxuan Wu, Zijian Zhou, Arun Verma, Alok Prakash, Daniela Rus, and Bryan Kian Hsiang Low. 2025. Tetris: Optimal draft token selection for batch speculative decoding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*.
- Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. 2023. [Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation](#). In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 3909–3925. Association for Computational Linguistics.
- Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. [Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding](#). In *Findings of the Association for Computational Linguistics ACL 2024*, pages 7655–7671, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.
- Minghao Yan, Saurabh Agarwal, and Shivaram Venkataraman. 2025. Decoding speculative decoding. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6460–6473.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- Seongjun Yang, Gibbeum Lee, Jaewoong Cho, Dimitris S. Papailiopoulos, and Kangwook Lee. 2023. [Predictive pipelined decoding: A compute-latency trade-off for exact LLM decoding](#). *CoRR*, abs/2307.05908.
- Andreas Zeller and Ralf Hildebrandt. 2002. Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering*, 28(2):183–200.
- Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2024. 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)*, pages 11263–11282.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023a. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.
- Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, and 1 others. 2024. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583.
- Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. 2023b. Response length perception and sequence scheduling: An llm-empowered llm inference pipeline. *Advances in Neural Information Processing Systems*, 36:65517–65530.

Meiyu Zhong, Noel Teku, and Ravi Tandon. 2025. Speeding up speculative decoding via sequential approximate verification. *arXiv preprint arXiv:2502.04557*.

Appendix Contents

- A Bug Taxonomy in Prior Implementations
- B Proof of Synchronization Invariants
- C EXSPEC Cross-Batch Scheduling
- D BatchVerify Algorithm Details
- E Latency-Throughput Tradeoff in Online Serving
- F Continuous Batching Systems
- G Discussion
- H Reproducibility Statement
- I Disclose on LLM Usage

A Bug Taxonomy in Prior Implementations

We categorize the implementation errors found in prior batch speculative decoding systems by whether they manifest at batch size 1 (Table 3).

Batch-independent errors. The first category consists of relatively straightforward implementation mistakes: sampling from the wrong distribution or failing to exclude rejected tokens. While these corrupt outputs at any batch size, they could be fixed with careful code review. DSD incorrectly samples the bonus token from the draft model’s distribution rather than the target model’s, violating the fundamental output equivalence guarantee of speculative decoding. BSP’s KV-cache retains rejected tokens and relies on framework auto-calculation for position IDs, which produces incorrect values when padding patterns change.

Batch-specific synchronization errors. The second category represents the core contribution of this paper: errors that *only* manifest when batch size exceeds one. These arise from the ragged tensor problem: when sequences accept different numbers of draft tokens, maintaining synchronized position IDs, attention masks, and KV-cache state across the batch becomes non-trivial. Our synchronization invariants (I1, I2) in Section 3 formalize exactly what must be preserved, and our EQSPEC algorithm provides the first correct implementation.

This taxonomy clarifies that while prior implementations suffer from multiple bug types, our contribution specifically addresses the batch-specific synchronization challenge that has no analog in single-sequence speculative decoding.

B Proof of Synchronization Invariants

Theorem B.1 (EQSPEC: Preservation of Alignment and Cache Correspondence). *Given a batch of size B , let $p_i^{(t)}$ and $c_i^{(t)}$ denote the padding and content lengths of sequence i at verification step t , and let realignment apply the offset δ_i of Eq. (2) as a relocation: a value-preserving move of cached entries, with rejected entries dropped. If at step t (i) the alignment invariant $p_i^{(t)} + c_i^{(t)} = L^{(t)}$ holds for all $i \in [1, B]$, and (ii) the cache clause of I2 holds, so every content position stores the key/value the target model computed for its token, then the EQSPEC update rule guarantees both properties at step $t + 1$. The position-ID clause of I2 is restored directly by recomputing position IDs from the attention mask after repadding.*

Proof. We prove this by induction on the verification step t .

Base Case ($t = 0$): At initialization, all sequences are left-padded to the maximum prompt length $L^{(0)}$. Thus, $\forall i, p_i^{(0)} + c_i^{(0)} = L^{(0)}$. The invariant holds. At $t = 0$ the cache holds only the prompt entries from the initial target forward pass, which are correct by construction, so (ii) holds as well.

Inductive Step: Assume the invariant holds at step t . During verification, sequence i accepts a_i new tokens. The new content length becomes:

$$c_i^{(t+1)} = c_i^{(t)} + a_i \quad (3)$$

The new target batch length is determined by the longest updated sequence:

$$L^{(t+1)} = \max_{j \in [1, B]} (c_j^{(t)} + a_j) \quad (4)$$

The EQSPEC algorithm calculates the padding offset δ_i as defined in Eq. (2):

$$\delta_i = (L^{(t+1)} - L^{(t)}) - a_i \quad (5)$$

The new padding length is updated as $p_i^{(t+1)} = p_i^{(t)} + \delta_i$. We verify the invariant at $t + 1$ by substituting these terms:

$$\begin{aligned} p_i^{(t+1)} + c_i^{(t+1)} &= (p_i^{(t)} + \delta_i) + (c_i^{(t)} + a_i) \\ &= p_i^{(t)} + (L^{(t+1)} - L^{(t)} - a_i) \\ &\quad + c_i^{(t)} + a_i \\ &= (p_i^{(t)} + c_i^{(t)}) - L^{(t)} + L^{(t+1)} \end{aligned}$$

By the inductive hypothesis, $p_i^{(t)} + c_i^{(t)} = L^{(t)}$. Substituting this yields:

$$\begin{aligned} &= L^{(t)} - L^{(t)} + L^{(t+1)} \\ &= L^{(t+1)} \end{aligned}$$

Thus, $\forall i, p_i^{(t+1)} + c_i^{(t+1)} = L^{(t+1)}$. The rectangular alignment is preserved.

Cache clause of I2. EQSPEC realignment is a relocation: it moves each retained entry to the offset induced by δ_i without recomputation, drops the entries of rejected draft tokens, and keeps the entries the target produced for accepted tokens during verification. Relocation leaves stored values unchanged, so each retained entry still equals the target-computed value from the round in which its token was processed; by the inductive hypothesis this is the value required at $t + 1$. This round’s accepted tokens take their entries directly from the verification pass, and rejected tokens contribute none. Hence every content position holds the correct key/value at $t + 1$. (The bonus token carries no entry until the next forward pass; see Appendix D.) $\square$

Corollary B.2 (Output Equivalence under Exact Arithmetic). *If a batched run maintains I1 and I2 at every round and samples each bonus token from the target model, then under exact arithmetic and a deterministic policy each sequence produces the same tokens as standard per-sequence decoding.*

Proof. At each round the target reads, for every sequence, the reference positions and keys/values (I2) within a rectangular batch (I1). Under identical arithmetic, it therefore computes the same logits as standard decoding. A deterministic policy then selects the same accepted tokens and the same bonus token. Induction over rounds yields identical token sequences. $\square$

Finite-precision batched kernels may alter argmax decisions; Section 4.2 and Table 4 evaluate this numerical divergence separately.

Corollary B.3 (Zero Overhead for Single Sequence). *For batch size $B = 1$, no padding adjustment is required.*

Proof. When $B = 1$, the batch consists of a single sequence with content length $c_1^{(t)}$. We can set padding $p_1^{(t)} = 0$ without loss of generality, satisfying the alignment invariant naturally as

$L^{(t)} = c_1^{(t)}$. Given acceptance a_1 , the new length is $L^{(t+1)} = c_1^{(t)} + a_1$. Substituting into Eq. (2):

$$\begin{aligned} \delta_1 &= (L^{(t+1)} - L^{(t)}) - a_1 \\ &= (c_1^{(t)} + a_1 - c_1^{(t)}) - a_1 = 0 \end{aligned} \quad (6)$$

Since $\delta_1 = 0$, no unpad-append-repad or KV-cache shifting is required. $\square$

Corollary B.4 (EXSPEC: Zero Overhead for Uniform Acceptance). *If all sequences in a batch accept the same number of tokens $k \geq 1$, no padding realignment is required ($\delta_i = 0$ for all i), regardless of their initial padding or content lengths.*

Proof. Let any sequence i have arbitrary padding $p_i^{(t)}$ and content $c_i^{(t)}$. The batch length at step t is defined as $L^{(t)} = \max_j (p_j^{(t)} + c_j^{(t)})$. Recall that the invariant $p_j^{(t)} + c_j^{(t)} = L^{(t)}$ holds for all sequences.

If every sequence accepts an identical number of tokens $a_i = k$, then the new content length is $c_i^{(t+1)} = c_i^{(t)} + k$. The new batch length becomes:

$$\begin{aligned} L^{(t+1)} &= \max_j (p_j^{(t)} + c_j^{(t+1)}) \\ &= \max_j (p_j^{(t)} + c_j^{(t)} + k) \\ &= (\max_j (p_j^{(t)} + c_j^{(t)})) + k \\ &= L^{(t)} + k \end{aligned}$$

The realignment offset δ_i is calculated as:

$$\begin{aligned} \delta_i &= (L^{(t+1)} - L^{(t)}) - a_i \\ &= ((L^{(t)} + k) - L^{(t)}) - k \\ &= k - k \\ &= 0 \end{aligned}$$

Since $\delta_i = 0$, the existing padding structure is preserved exactly. No memory reallocation or shifting is required, even if the sequences started with different amounts of padding. $\square$

Proof of Theorem 3.1. Write $C(B) = \sum_{i=1}^B (M_B - a_i)$ for the per-round realignment work. By linearity of expectation, $E[C(B)] = B(E[M_B] - \mu)$. Adding a sequence can only raise the running maximum, so $E[M_{B+1}] \geq E[M_B]$, with strict inequality whenever acceptances are non-degenerate; the per-sequence cost $E[M_B] - \mu$ is therefore increasing in B , and the total $B(E[M_B] - \mu)$ is superadditive (it grows strictly faster than linearly).

Algorithm 2 EXSPEC: Cross-Batch Scheduling

Require: Draft and Target model $\mathcal{M}_d, \mathcal{M}_t$, Prompts $\mathcal{P}$, Window size W , Batch size B
Ensure: Generated sequences $\mathcal{S}$

```
1:  $Pool \leftarrow \text{InitSequencePool}(\mathcal{P})$ 
2:  $\triangleright$  Tokenize and optionally sort by length
3:  $Window \leftarrow \text{RefillWindow}(Pool, W)$ 
4: while  $Pool.hasActive()$  do
5:   Phase 1: Lazy Realignment
6:    $\mathcal{B}, mask, KV \leftarrow \text{GetBatch}(Window, B)$ 
7:    $\triangleright$  Try same-length concatenation
8:    $\triangleright$  Fallback to Unpad-Repad Realignment
9:   Phase 2: Draft Generation
10:   $D \leftarrow \mathcal{M}_d.Generate(\mathcal{B}, mask, K)$ 
11:  Phase 3: Batch Verification
12:   $\triangleright$  See Alg. 3 in Appendix for BatchVerify
13:   $A, B, KV \leftarrow \text{BatchVerify}(\mathcal{M}_t, \mathcal{B}, D, KV)$ 
14:  Phase 4: Write-Back and Window Refill
15:  for  $i \in \mathcal{B}$  do
16:     $Pool[i] \leftarrow Pool[i] \oplus A[i] \oplus B[i]$ 
17:     $Pool.KV[i] \leftarrow KV[i]$ 
18:    if  $isComplete(Pool[i])$  then  $Pool.deactivate(i)$ 
19:  end for
20:   $Window \leftarrow \text{RefillWindow}(Pool, W)$ 
21: end while
22: return  $Pool.sequences$ 
```

Because acceptances are capped at K and some sequence attains K with positive probability, $P(M_B < K) = [1 - P(a_i = K)]^B \rightarrow 0$, so $E[M_B] \rightarrow K$ and the per-sequence cost saturates at $K - \mu$. $\square$

C EXSPEC: Cross-Batch Scheduling Algorithm

Algorithm 2 gives the complete EXSPEC procedure summarized in Section 3.2. It maintains a *SequencePool* of active sequences and, over a sliding window of width $W > B$, forms batches of identical length that concatenate without realignment, falling back to the unpad-append-repad path of EQSPEC only when same-length grouping fails.

D BatchVerify Algorithm Details

Algorithm 3 implements batch verification through a single forward pass over all draft tokens. On the first iteration, the input concatenates the full sequences with their draft tokens ($\mathcal{X} \leftarrow S \oplus D$) to build the initial KV-cache; subsequent iterations process only the draft tokens since prior context is cached. The target model produces logits for all positions, from which we extract predicted tokens via argmax. Mismatch detection is vectorized: we compare predicted tokens against draft tokens and identify the first disagreement position j per sequence. The accepted tokens $A[i]$ consist

of all draft tokens up to (but not including) the first mismatch, while the bonus token $B[i]$ is sampled from the target model’s distribution at the mismatch position. Critically, this must come from the target model, not the draft model, to preserve output equivalence. Note that because bonus token sampling occurs after the forward pass completes, the bonus token lacks KV-cache entries and must be included in the next forward pass.

An alternative approach, previously employed by vLLM’s v0 engine, sidesteps variable acceptance lengths through batch expansion: rather than handling ragged outputs, each sequence is duplicated K times with progressively longer draft prefixes (e.g., $[t_1]$, $[t_1, t_2]$, $[t_1, t_2, t_3]$), all verified in one pass, keeping only the longest correct prefix per sequence. While this maintains rectangular tensors, it incurs $K \times$ memory and compute overhead, resources wasted when early tokens are rejected. This approach was deprecated in vLLM v1 due to GPU memory overflow at scale and incompatibility with CUDA graphs. Our BatchVerify instead accepts the ragged output directly and delegates synchronization to the subsequent unpad-append-repad phase (Section 3.1), avoiding the multiplicative overhead of batch expansion.

E Latency-Throughput Tradeoff in Online Serving

To evaluate performance under realistic online serving conditions, we conducted experiments simulating dynamic request arrivals using full multi-turn conversations from SpecBench. We shuffled conversation turns to maximize length diversity and inserted new turns into the next available batch, measuring both throughput (tokens/s) and request completion latency (s) at P50/P90/P99 percentiles. Table 6 compares EQSPEC against EXSPEC across batch sizes 1, 2, 4, and 8.

Both methods achieve positive speedups under heterogeneous multi-turn workloads, with EXSPEC obtaining 2–14% higher throughput than EQSPEC through cross-batch grouping of same-length sequences. However, EXSPEC suffers 1.5–7.7 $\times$ worse P90/P99 latencies because delaying early requests until same-length partners arrive causes head-of-line blocking. The penalty peaks at batch size 2 (P99 134s) and recedes as batch size grows (34s at batch size 8), since larger batches sustain the throughput (30.5 to 77.5 tokens/s) needed to drain the request queue. In contrast, EQSPEC

maintains predictable tail latency with P99 under 23 seconds across all batch sizes, making it suitable for latency-sensitive online serving where service-level objectives must be met. These results demonstrate that the choice between EQSPEC and EXSPEC depends on deployment requirements: EQSPEC provides stable, predictable latency for interactive applications, while EXSPEC maximizes throughput for offline batch processing where individual request latency is less critical. By offering both algorithms with explicit algorithmic correctness guarantees, our work enables practitioners to match their batch speculation strategy to their specific operational constraints.

F Continuous Batching Systems

Scope

Continuous-batching systems (vLLM, SGLang) address raggedness through memory layout rather than synchronization: variable-length packing with paged attention stores sequences in a flat buffer, so Invariant I1 never arises (Family B, Section 2). We report two regimes of vLLM: the v0 engine used in our main comparison (Table 7), and vLLM v1, which added external-draft speculative decoding in January 2026 (Table 4). vLLM v1 is output-equivalent on H100, confirming that the varlen/paged layout is correct where its kernels are available; the divergences reported below reflect numerical precision and feature scope.

Architectural differences. Static batching processes a fixed batch of B sequences together, requiring rectangular tensor alignment (Invariant I1) and facing the ragged tensor synchronization challenge when sequences accept different numbers of draft tokens. Continuous batching systems (e.g., vLLM, SGLang) process requests using variable-length packing, sidestepping I1, but still require scatter-gather across requests and rollback both position-ID and KV-cache for rejected tokens, a one-dimensional analog of alignment. Critically, I2 remains a prerequisite for algorithmic correctness regardless of batching strategy. When we ran our main comparison, neither engine exposed external-draft speculative decoding through continuous batching; vLLM v1 has since added it (Ta-

Algorithm 3 BatchVerify: Single Forward Pass

Require: Target model $\mathcal{M}_t$, Sequences $\mathcal{S}$, Draft tokens D , KV cache
Ensure: Accepted tokens A , Bonus tokens B

```

1: if first iteration then
2:    $\mathcal{X} \leftarrow \mathcal{S} \oplus D$ 
3: else
4:    $\mathcal{X} \leftarrow D$ 
5: end if
6:  $\text{logits}, \text{KVCache} \leftarrow \mathcal{M}_t(\mathcal{X}, \text{KVCache})$ 
7:  $\text{pred\_tokens} \leftarrow \arg \max(\text{logits}, \text{dim}=\text{vocab})$ 
8:  $\triangleright$  Vectorized first mismatch detection
9:  $\text{matches} \leftarrow (\text{pred\_tokens} = D)$ 
10:  $J \leftarrow \arg \max(\neg \text{matches}, \text{dim}=\text{seq})$ 
11:  $\triangleright$  Ragged shape acceptance, no vectorization
12: for each sequence  $i$  in batch do
13:    $A[i] \leftarrow D[i]: j$ 
14:    $\triangleright$  Get bonus token from first mismatch
15:    $\text{bonus\_logit} \leftarrow \text{logits}[i, |\mathcal{S}[i]| + j]$ 
16:    $B[i] \leftarrow \arg \max(\text{bonus\_logit})$ 
17: end for
18: return  $A, B, \text{KVCache}$ 

```

ble 4).

Systems evaluated. For informational comparison, we evaluated: **vLLM** (Kwon et al., 2023) (which subsumes TETRIS (Wu et al., 2025) and TurboSpec (Liu et al., 2025) as vLLM forks) and **SGLang-EAGLE** (Zheng et al., 2024; Li et al., 2024a, 2026). We used the vLLM v0 engine, the only version supporting external-draft speculative decoding when we ran these experiments; v1 added a varlen replacement in January 2026 (Table 4). SGLang is compatible only with the EAGLE family; we compare Vicuna-7B/EAGLE2 and Qwen3-8B/EAGLE3 (no available weights for GLM-4). We further tested **SGLang-EAGLE-Deterministic** (SGLang Team, 2025), which enables deterministic execution to reduce numerical variance.

Throughput observations. Figure 6 shows throughput characteristics of production frameworks. These systems use continuous batching with dynamically varying effective batch sizes, so the x-axis is a “maximum batch size” rather than the fixed batch size of our experiments. The observed patterns (vLLM’s speculative decoding underperforming its baseline at high concurrency, SGLang+EAGLE being slower than non-speculative generation) reflect the challenges of integrating speculation with continuous batching. Independent evaluation by Liu et al. (2026) confirms these challenges: they report EAGLE speedup degrading from $1.73\times$ to $1.21\times$ as batch size scales from 1 to 128, with verification consum-

ing 42–95% of execution time. vLLM’s v0 engine used batch expansion, deprecated in v1 (vllm-project/vllm#17984) due to $K \times$ memory overhead.

Accuracy observations. Table 7 shows accuracy metrics for vLLM and SGLang. Both systems are accurate on Vicuna but degrade on Qwen3, suggesting model-specific sensitivities to position encoding or numerical precision. SGLang-EAGLE-Deterministic (SGLang Team, 2025) helps disambiguate the cause: the improvement from 69.8% to 85.0% on Vicuna confirms that most divergences stem from floating-point non-determinism (He and Lab, 2025) rather than algorithmic bugs. Concurrent work by Liu et al. (2026) reaches similar conclusions using output length correlation, though our exact and partial match metrics provide finer-grained verification by distinguishing complete sequence match from partial divergence. Note that SGLang only supports EAGLE-family drafters, not external draft models.

The vLLM v1 batch-invariant-kernel results are reported in Section 4.2.

Architectural implications. The contrast between batch expansion and our approach illustrates two strategies for handling ragged tensors: resource duplication versus explicit synchronization. Batch expansion maintains rectangular tensors by duplicating work, trading $K \times$ memory for implementation simplicity. Our unpad-append-repad approach instead accepts raggedness and pays synchronization costs only when sequences diverge. Integrating this synchronization-based approach with continuous batching and paged attention remains an open problem for production deployment.

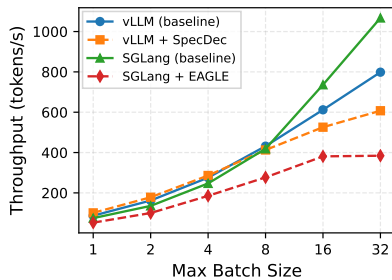


Figure 6: Speculative decoding lags non-speculative baselines, with larger batches further degrading throughput.

G Discussion

Batch parallelism and per-sequence speculative decoding represent two orthogonal acceleration strategies that should, in principle, multiply together: batching exploits GPU parallelism across sequences while speculation reduces sequential token dependencies within each sequence. However, when combining these techniques in practice, the ragged tensor problem emerges as a fundamental obstacle that breaks this multiplicative relationship. Different sequences accept different numbers of draft tokens, desynchronizing position IDs, attention masks, and KV-cache state across the batch. A correctly implemented batch speculative decoding, as we provide through EQSPEC and EXSPEC, preserves the multiplicative gains but incurs a realignment cost that consumes up to 40% of computation. Our work establishes the synchronization invariants required for algorithmic correctness, quantifies their cost under contiguous layouts, and demonstrates through cross-batch scheduling that intelligent system design can mitigate, though not eliminate, the overhead of maintaining output equivalence at scale.

H Reproducibility Statement

We provide our complete implementation in the Supplementary Material, including all hyperparameters, experimental configurations, and model specifications detailed in Section 4. Our output equivalence verification framework using exact and partial match metrics enables deterministic validation of both our results and future implementations. All experiments use publicly available models and datasets for reproducibility.

Model Configurations. We evaluate three target-draft model pairs: Vicuna-7B (lmsys/vicuna-7b-v1.3) with a 68M draft model (double7/vicuna-68m), Qwen3-8B (Qwen/Qwen3-8B) with Qwen3-0.6B (Qwen/Qwen3-0.6B), and GLM-4-9B (zai-org/GLM-4-9B-0414) with a 0.6B draft model (jukofyork/GLM-4.5-DRAFT-0.6B-v3.0). All models were loaded in FP16 precision with greedy decoding (temperature=0, top_p=1.0) to ensure deterministic outputs. We use five draft tokens per speculation round across all experiments unless otherwise specified.

Production Systems Configuration. We evaluated two production inference systems for comparison: vLLM version 0.9.1 and SGLang com-

Method	Vicuna		Qwen3		GLM4	
	E	P	E	P	E	P
vLLM + Spec	96.9	98.0	65.6	78.5	72.7	84.5
SGLang + EAGLE	69.8	79.5	47.7	65.4	–	–
SGLang + EAGLE + Det	85.0	90.0	50.6	69.5	–	–

Table 7: Correctness of continuous batching systems. Exact match (E) and partial match (P) scores compared against each system’s own non-speculative mode.

mit c4e314f (the deterministic decoding had not merged to the stable version when we ran these experiments, so we compiled it from source). For vLLM, we used the V0 engine with speculative decoding enabled, as the V1 engine did not yet support draft-model speculative decoding³ (added in January 2026; see Table 4). For SGLang, we used EAGLE-based speculation with model-specific draft models: yuhuli/EAGLE-Vicuna-7B-v1.3 for Vicuna-7B and Tengyunw/qwen3_8b_eagle3 for Qwen3-8B. GLM-4 was not evaluated with SGLang due to the unavailability of compatible EAGLE draft models. We tested both standard SGLang inference and SGLang with deterministic mode enabled (SGLang Team, 2025) to isolate floating-point non-determinism from algorithmic correctness issues.

I Disclose on LLM Usage

We used Large Language Models as assistive tools in preparing this manuscript: GPT-5 and Claude Opus 4.1 for polishing writing (grammar correction and sentence restructuring), generating conceptual diagram illustrations, and writing unit tests; NVIDIA/Parakeet-TDT-0.6B-v3 for voice input transcription. LLMs were not used for research ideation, experimental design, data analysis, or scientific conclusions. All core algorithmic implementations and scientific contributions are solely from the authors. We take full responsibility for all content in this paper.

³<https://github.com/vllm-project/vllm/issues/21797>