Where to Speculate: Cross-Request Survival-Value Allocation for Batched Speculative Decoding

Anonymous authors
Paper under double-blind review

Abstract

Many batched speculative decoders apply a common proposal width to requests whose candidates have sharply different survival probabilities. PACE (Prefix-Aware Compute Allocation) estimates prefix survival before proposal generation, allocates a shared candidate budget across requests, and materializes request-specific widths without batch-wide padding. The verifier remains unchanged. For equal-cost, prefix-consistent proposals, PACE optimizes the supplied survival objective under a fixed budget. The fused runtime is evaluated as an empirical realization of this contract.

On the controlled heterogeneous workload, PACE supports 24–28% more concurrent requests than the best-static capacity envelope at per-request rate floors of 40–50 tok/s. Across a seven-point sweep that varies within-batch survival separation, pre-generation estimates improve equal-budget progress by 12.7–13.1% over the offline per-budget best-fixed envelope. In the fused I-DLM-8B runtime, materializing request-specific widths within the same $N = 8$ container improves throughput by 27.3–33.3% at concurrency 80–128, showing the effect of omitting unselected candidate positions. Offline replay on external proposal stacks confirms that the allocation relationship transfers beyond the fused decoder; these replay results do not measure physical work removal. Serving gains emerge when the saved candidate computation is large enough to amortize ragged-execution overhead.

1 Introduction

Speculative decoding trades candidate computation for fewer sequential generation rounds. Candidates may come from a smaller draft model (Leviathan et al., 2023; Chen et al., 2023), a subset of the target (Zhang et al., 2024a; Kim et al., 2023), or a fused parallel decoder (Christopher et al., 2025; Arriola et al., 2025). Existing adaptation largely asks how far one request, or an entire batch, should speculate. Batched serving raises a different question: where should the next unit of speculative compute go?

An ordered proposal advances only through its longest accepted prefix. If L_i is request i's leading accepted length, then $s_i(d) = \Pr(L_i \geq d)$ is the marginal expected progress of candidate d under the PACE allocation contract (§ 2). Concurrent requests can value the same-cost slot very differently: in our fused stack, $\Pr(L \geq 7)$ is 0.915 for copyediting and 0.053 for chat. A uniform width can therefore spend scarce capacity on low-survival suffixes while withholding useful candidates elsewhere.

PACE (Prefix-Aware Compute Allocation) ranks pre-generation request–depth survival estimates and selects prefix-feasible widths under a shared candidate budget. The original verifier remains the authority over committed tokens.

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.

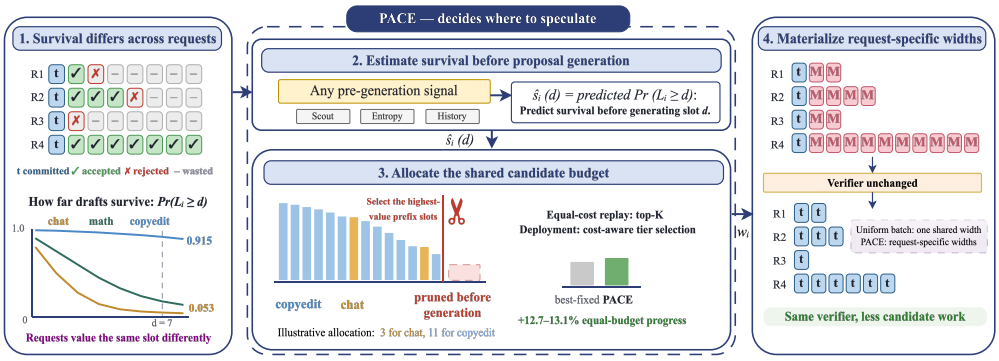

Figure 1: PACE estimates request–depth survival before proposal generation, reallocates candidate work across requests, and materializes the selected prefixes with an unchanged verifier.

Survival heterogeneity creates allocation value when estimates arrive before candidate generation. Request-specific execution converts that value into serving capacity when the saved computation amortizes ragged-execution overhead.

Our experiments test these links in order. A controlled seven-point sweep varies within-batch survival separation without exposing domain labels to PACE; the allocation gain tracks this separation. On I-DLM-8B, a fused Qwen3-8B-based checkpoint, pre-generation estimates improve equal-budget progress by 12.7–13.1%. Physical PACE raises matched-$N = 8$ throughput by 27.3–33.3% at concurrency 80–128 and supports 24–28% more concurrency than the best-static capacity envelope at 40–50 tok/s per-request rate floors.

PACE therefore combines comparable request–depth values, pre-generation decisions, and request-specific execution.

Our contributions are:

1. We formulate speculative width as a shared cross-request resource and identify prefix survival as the marginal value of each candidate position under the PACE allocation contract.
2. We introduce PACE, a prefix-feasible allocator that assigns a shared candidate budget to the highest-survival request–depth positions before generation.
3. We materialize the selected widths, separate allocation from candidate computation, and characterize how survival separation, candidate cost, and concurrency determine whether the allocation gain translates into serving capacity.

2 Batched Speculation as a Shared Resource

Speculative decoding commits only the longest accepted prefix of an ordered proposal: the first rejection ends a round's verified progress, and every deeper candidate is discarded. Many batch-uniform speculative runtimes assign all concurrent requests the same width. This creates waste within the shared candidate budget: slots that one request will almost surely reject could have advanced another.

Ordered proposals. For a batch of B requests, let L_i denote request i's leading acceptance length, $\mathcal{F}_t$ the information available when its width is chosen, and $\mathbf{w} = (w_1, \ldots, w_B)$. Cumulative survival and expected committed progress satisfy

$$s_i(d) := \Pr(L_i \geq d \mid \mathcal{F}_t), \qquad U_i(w_i) := \mathbb{E}[\min(L_i, w_i) \mid \mathcal{F}_t] = \sum_{d=1}^{w_i} s_i(d). \tag{1}$$

Thus $s_i(d)$ is the non-increasing marginal value of candidate position d. Writing $a = (i, d)$ as a slot, a prefix-feasible allocation has value $U(A) = \sum_{a \in A} s(a) = \sum_i U_i(w_i)$. Replacing a low-survival suffix with a higher-survival slot increases expected progress at fixed budget.

This additive form is the PACE allocation contract; its exactness for the deployed execution is assessed separately.

Allocation scope and timing. A request-local controller chooses each width independently, whereas a cross-request allocator couples widths through one hard budget so that a slot removed from one request becomes available to another. Timing is a separate axis. A pre-generation decision uses state available before candidate tokens are generated and can avoid computing them. A current-block decision uses fresher state from an existing block to select its carry and verification prefix, after those candidates have been generated. We evaluate both decision times; physical candidate savings use the pre-generation setting. Allocation chooses retained slots, while execution determines whether omitted slots save wall-clock time.

Proposal depth and physical width. At the allocation level, $w_i \in \{0, \ldots, N-1\}$ denotes proposal depth and N the container limit. Equal-budget replay reserves one depth-one proposal per active request and allocates the residual budget $K - B$; the physical-width mapping appears in § 3 and Appendix E.

3 PACE: Cross-Request Allocation by Prefix Survival

PACE allocates proposal work across requests while preserving the original verification procedure. Its design separates candidate value, budget exchange, and physical execution.

3.1 The PACE contract

Component	Required condition
Value	Prefix-consistent ordered proposals make $s_i(d)$ the marginal value of request–depth slot (i, d).
Budget	Equal-cost slots share one hard batch budget K, enabling cross-request exchange.
Execution	Scores precede the corresponding work, and selected prefixes are physically materialized without padding.

Value and Budget define the allocation problem; Execution determines whether the allocation gain appears in serving throughput. PACE changes proposal work while commitment remains with the original verifier (Appendix A.1). The fixed-budget theorem is exact for prefix-consistent proposal chains. Fused bidirectional execution is an empirical realization of this contract; we measure its consistency and evaluate allocation and serving behavior directly in Appendix A.2 and D.6.

3.2 Cross-request top-K allocation

For request i, the allocator receives a maximum depth D_i and a nonnegative, non-increasing value sequence $1 \geq \widehat{s}_i(1) \geq \cdots \geq \widehat{s}_i(D_i) \geq 0$ over slots $(i, 1), \ldots, (i, D_i)$. For the fixed-budget problem, the controller supplies a slot budget K; PACE returns prefix depth limits $\mathbf{w} = (w_1, \ldots, w_B)$. With estimated survival values, PACE solves

$$\max_{\mathbf{w}} \sum_{i=1}^{B} \sum_{d=1}^{w_i} \widehat{s}_i(d) \quad \text{s.t.} \quad \sum_i w_i \leq K, \quad 0 \leq w_i \leq D_i, \quad w_i \in \mathbb{Z}. \tag{2}$$

Replacing $\widehat{s}_i(d)$ with $s_i(d)$ gives the expected-progress objective in Equation (1). PACE selects the K largest estimated values with shallow-first tie-breaking. Monotonicity makes this selection prefix-feasible. Varying K traces how a given amount of candidate work is distributed across requests.

Next-block scores are available before proposal generation; fresher current-block scores are used to clip carry and verification. § 5.2 compares both.

Frontier implementation. The top-K set need not be formed by sorting all $\sum_i D_i$ slots. PACE keeps only the next eligible slot of each request in a max-priority queue:

Algorithm 1 PACE frontier allocation

Input: monotone values $\{\widehat{s}_i(d)\}_{d=1}^{D_i}$ available or revealed at the chosen decision point, maximum depths $\{D_i\}_{i=1}^{B}$, and slot budget K.
Output: prefix-feasible depth limits $\mathbf{w}$.

$\mathbf{w} \leftarrow \mathbf{0}$; initialize an empty max-heap H.
for $i = 1, \ldots, B$ with $D_i \geq 1$ do push$(H, (\widehat{s}_i(1), i, 1))$.
for $t = 1, \ldots, \min\{K, \sum_i D_i\}$ do
  $(v, i, d) \leftarrow$ popmax(H), with deterministic shallow-first tie-breaking.
  $w_i \leftarrow d$.
  if $d < D_i$ then read or compute $\widehat{s}_i(d{+}1)$ and push$(H, (\widehat{s}_i(d{+}1), i, d{+}1))$.
return $\mathbf{w}$.

For next-block allocation, $\mathbf{w}$ sets the proposal depths.
For current-block allocation, $\mathbf{w}$ sets the carry/verification prefixes.

The heap yields exact top-K allocation in $O(B + K \log B)$ time and $O(B)$ memory. Fused replay reserves mandatory depth-one proposals before applying the allocator to the residual budget; implementation details are given in Appendix A.2.

Budget realization. Replay treats K as an externally swept variable and compares every policy at the same budget. Deployment instead ranks feasible, nested tier increments by predicted progress per added slot. Let $\widehat{U}_0$ denote predicted committed progress with no optional fresh proposals, let $\Delta\widehat{U}_{(j)}$ denote the predicted progress increment of ordered tier j, and let $k_i(q)$ be the fresh-proposal count after selecting the first q tiers. It selects

$$q^* = \arg\max_q \frac{\widehat{U}_0 + \sum_{j \leq q} \Delta\widehat{U}_{(j)}}{a + c \sum_i (1 + m_i + \max\{k_i(q), 1\})}, \qquad K^*_{\text{opt}} = \sum_i k_i(q^*). \tag{3}$$

Here $a + cW$ is a frozen step-cost model for the pre-menu footprint $W = \sum_i (1 + m_i + \max\{k_i, 1\})$, where m_i is the carry depth. The runtime maps the resulting decisions to $p_i = Q_N(1 + m_i + \max\{k_i, 1\})$, where Q_N rounds up to a width supported by the container, and packs different p_i into one ragged forward. Menu rounding makes deployment a cost-aware implementation of the allocation rule rather than an exact fixed-K instance; controller details are in Appendix E.1.

3.3 Optimality and boundary-sensitive regret

Theorem 1 (Optimality under the PACE contract). For prefix-consistent ordered proposals, nonnegative non-increasing values, and equal slot costs, shallow-first top-K selection is prefix-feasible and optimal. With true survival values, it maximizes expected committed progress under budget K.

The exchange proof and tie treatment appear in Appendix A.2. This is a contract-level allocation result: it characterizes the ideal equal-cost frontier, while the fused runtime is evaluated as a cost-aware realization of that frontier. Deployment selects a rounded operating point using the profiled cost model.

Strict improvement over uniform allocation. Let A_0 be the strongest uniform allocation under budget K, padded with zero-value dummy slots if necessary, and let A^* be the true top-K allocation. With $G = A^* \setminus A_0$ and $O = A_0 \setminus A^*$,

$$\mathcal{D}(A_0) = U(A^*) - U(A_0) = \sum_{g \in G} s(g) - \sum_{o \in O} s(o) \geq 0. \tag{4}$$

The gain is positive exactly when a selected uniform slot can be exchanged for a higher-survival omitted slot; survival separation therefore predicts the gain measured in § 5.

Theorem 2 (Boundary-sensitive allocation regret). Assume nonnegative, non-increasing estimates $\widehat{s}_i(\cdot)$ and the same deterministic tie rule, so both selections are prefix-feasible. Let A^* and $\widehat{A}$ be the true- and estimated-value top-K allocations, let $R_K := U(A^*) - U(\widehat{A})$ be the true-value regret of acting on estimates, and define $e(a) = |\widehat{s}(a) - s(a)|$. Then

$$R_K \leq \sum_{a \in A^* \triangle \widehat{A}} e(a). \tag{5}$$

If $m = |A^* \setminus \widehat{A}|$ and $\|\widehat{s} - s\|_\infty \leq \varepsilon$, then $R_K \leq 2m\varepsilon \leq 2K\varepsilon$.

Only slots crossing the cutoff contribute to the first bound (Appendix A.4 and B.1). These guarantees concern a fixed equal-cost budget; physical throughput is governed by Equation (6).

3.4 Decision-time survival estimates

PACE accepts any comparable request–depth survival estimate. Fused deployment uses Scout, calibrated entropy, or request history; Scout adds about 0.4 ms per round. Signal construction and calibration appear in Appendix B and D.8.

4 Experimental Setup

Decoders and hardware. We study three fused checkpoints: I-DLM-8B, a fused Qwen3-8B-based checkpoint (Yu et al., 2026); Qwen3-1.7B with an R-ISD LoRA; and I-DLM-32B. Physical execution runs on I-DLM-8B. A shared Qwen3-8B target compares Qwen3-0.6B, dFlash-b16 (Chen et al., 2026a), EAGLE-3 (Li et al., 2025), and MTP-TTT3 (AngelSlim Team, 2026) proposal traces. Primary experiments use A100-SXM4-80GB GPUs.

Workloads. The primary controlled workload pairs deep-survival copyediting with shallow-survival chat. A frozen seven-point composition sweep covers both homogeneous endpoints and multiple degrees of within-batch heterogeneity; additional workloads appear in Appendix D and D.6.

Equal-budget replay. At $B = 32$, equal-budget experiments hold selected candidate count fixed and score $\sum_i \min(L_i, w_i)$; best-fixed reselects the strongest feasible uniform width at every budget. Adaptive-rule comparisons instead match a total analytic proposal-plus-verification budget, charging each rule for the work required by its decision time. Replay fixes resource use to isolate allocation quality; serving experiments retain executed-width differences so that physical work removal can be measured.

End-to-end protocol. The fused implementation uses saved request state to score next-block width increments. The frozen controller in Equation (3) selects an operating point from the feasible allocation choices, and the resulting request-specific widths are rounded up to the container's supported menu. Closed-loop experiments report aggregate throughput and mean active-request progress; open-loop experiments require TTFT ≤ 1 s and TPOT ≤ 25 ms for at least 90% of requests. Serving uses greedy decoding, the original verifier, and matched sessions, including scheduler scoring and allocation cost. Complete replay, serving, and reproducibility protocols appear in Appendix C, C.5, D and E.

5 Allocation Gains from Survival Heterogeneity

We test whether survival heterogeneity creates allocation value, how much remains available before generation, and whether cross-request coupling improves over request-local control. The composition sweep varies within-batch survival separation while keeping the proposal and serving stack fixed.

5.1 Gain tracks survival separation

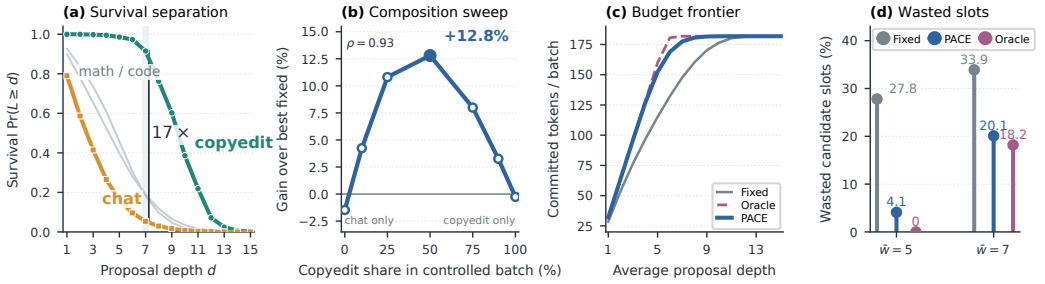

Figure 2: Survival heterogeneity drives allocation gain: (a) survival curves, separated 17× at depth seven; (b) pre-generation gain across seven composition points tracks exchange value ($\rho = 0.93$), peaks in the heterogeneous interior, and collapses at both homogeneous ends; (c) the equal-budget frontier; and (d) PACE moves budget from post-rejection positions toward the oracle level.

Define the per-request oracle exchange value at the sweep's mid-budget point as $D := \mathcal{D}(A_0)/B$, using the exchange gain in Equation (4). Pre-generation gain peaks in the heterogeneous interior, collapses at both homogeneous endpoints, and tracks D with Spearman correlation 0.93 across the seven composition points (Table 11). The sweep varies survival separation through composition, but PACE never observes domain labels. The predictive variable is therefore separation between within-batch survival curves rather than copyedit or chat identity.

5.2 Timing and exact budget coupling

At the controlled operating point, pre-generation entropy achieves a 13.1% gain over the offline per-budget best-fixed envelope, and deployed Scout reaches 12.7%. Fresher current-block information reaches 17.3%, showing that most of the allocation opportunity is available before candidate generation; the realized-acceptance oracle reaches 19.6%.

In equal-budget replay, request-local thresholds satisfy the candidate budget only in expectation: overflow requires clipping without prioritization, while underflow leaves capacity idle. PACE spends exactly K on the globally highest-valued slots, beating the SpecDec++ threshold port on all four stacks by 3.00–6.48%. Decision timing determines a different advantage: post-draft ranking sees fresher information but cannot recover proposal work already performed.

5.3 Controlled comparison with adaptive allocation rules

We compare controlled rule ports on fixed proposal traces with a shared Qwen3-8B target. Deep stacks use entropy through $d-1$ to score slot d; the shallow EAGLE-3 and MTP columns use same-depth entropy and therefore test allocation granularity rather than slot-level cost avoidance. Full accounting appears in Appendix D.8.

On the external 0.6B drafter, PACE gains 28.41% over the offline per-budget best-fixed envelope and has positive margins against all four controlled ports, including 2.47% against TETRIS with a 0.93 paired-batch win fraction. Across four stacks, 13 of 16 mean margins are positive with paired 95% intervals above zero; two margins against TETRIS have intervals below zero when drafting is cheap or the useful chain is shallow, and EAGLE-3–AngelSpec is statistically indistinguishable from zero. This ordering reflects the tradeoff between avoiding candidate computation early and using fresher post-draft information. Drafting-cost and request-disjoint sensitivity results appear in Appendix D.8.

Table 1: Controlled allocation-rule ports on a shared Qwen3-8B target. Entries are mean PACE-minus-comparator margins; parentheses show paired win fractions; † marks the only interval crossing zero. Full paired intervals appear in Table 17.

	PACE margin (paired win fraction)			
Comparator	External 0.6B	dFlash	EAGLE-3	MTP
Offline best-fixed envelope	+28.41% (1.00)	+24.24% (0.99)	+3.06% (0.78)	+22.10% (1.00)
TETRIS (post-draft top-K)	+2.47% (0.93)	−0.61% (0.18)	−1.16% (0.16)	+2.44% (0.83)
AngelSpec (D-cut)	+4.76% (0.99)	+1.01% (0.84)	−0.20%† (0.54)	+4.27% (0.97)
DSpark (binary admission)	+10.80% (0.99)	+12.99% (0.99)	+1.42% (0.62)	+4.20% (0.90)
SpecDec++ (local threshold)	+4.75% (0.94)	+4.35% (0.98)	+3.00% (0.76)	+6.48% (0.87)

§ 6 tests whether physical execution converts this allocation gain into a serving gain.

6 From Allocation Gain to Serving Gain

Serving gains require the runtime to omit unselected positions and save enough candidate computation to amortize ragged overhead. We test this in the fused 8B runtime under closed-loop and open-loop load.

Break-even execution condition. For B_s requests at p_L and B_d at $p_H > p_L$, let $B = B_s + B_d$, $\bar{p} = (B_s p_L + B_d p_H)/B$, $a + cBp$ be linear round cost, e_ℓ ragged overhead, and $\rho = Q_{\text{rag}}/Q_{\text{fix}}$. Under this model, request-specific execution is faster exactly when

$$e_\ell < \rho\, e^*(B, c), \qquad e^*(B, c) := \frac{a + cBp_H}{a + cB\bar{p}}. \tag{6}$$

For $a > 0$, larger B or c increases the ragged overhead that saved width-token work can absorb; see Appendix A.5.

6.1 Request-specific execution avoids candidate computation

Saved request state provides next-block width scores, and the frozen controller packs the selected prefixes into request-specific tensors (Equation (3) and Appendix E.1). Unselected positions are omitted from the target forward.

Using the same $N = 8$ container, PACE improves aggregate throughput by 27.3–33.3% at $C = 80$–128 and mean active-request progress by 33.7–36.2% over the same range (Table 2); each cell is a within-session three-seed measurement at a shared configuration. Logical allocation with full padding matches Fixed ($N = 8$) throughput, whereas physical widths improve throughput by 18.6% at $C = 32$ (Table 18).

Attributing assignment and execution gains. The padding control isolates omitted computation; a width-matched permutation preserves each round's width multiset and execution cost while destroying the request–width correspondence. The realized gain splits into

$$\text{pace} - \text{fixed} = \underbrace{(\text{pace} - \text{perm})}_{\mathcal{D}_{\text{alloc}}} + \underbrace{(\text{perm} - \text{fixed})}_{\mathcal{D}_{\text{decouple}}}. \tag{7}$$

At C=80, assignment explains 88.1% of the active-request gain; assignment and batch-max decoupling have consistent signs across seeds (Appendix E.9). The speedup therefore depends on both correct assignment and physical omission of unselected positions. Output-quality checks appear in Appendix E.10.

At fixed 40–50 tok/s active-request rate floors, PACE supports 24–28% more concurrency than the best-static capacity envelope (Table 3, Appendix E.3). This is the primary system comparison; matched-container throughput isolates the benefit of omitting unselected candidate positions.

Table 2: Matched-container fused execution on A100. PACE and the fixed baseline use the same container with generation block size $N = 8$, so the comparison isolates request-specific allocation and omission of unselected candidate positions within the same action space. Values are mean±sd over three within-session seeds; throughput is measured in tok/s.

C	Fixed ($N = 8$)	PACE	Aggregate gain	Active-request gain
80	1939.0±9.9	2468.4±23.0	+27.3%	+34.4%
96	1995.7±18.0	2586.4±30.6	+29.6%	+33.7%
128	2104.1±9.0	2805.4±9.4	+33.3%	+36.2%

Table 3: Maximum concurrency at each active-request rate floor. The best-static capacity envelope selects its strongest fixed container per floor.

Floor (tok/s)	PACE	Best static	Selected N	Gain
40	106.1	83	2 or 4 (tie)	+28%
45	94.0	75	4	+25%
50	82.0	66	4	+24%

6.2 Operating at the SLO capacity knee

Closed-loop request progress does not expose arrival queues or tail latency. We therefore replay paired Poisson traces on the frozen heterogeneous workload and measure P95 TTFT, P95 TPOT, joint-SLO attainment, and SLO-compliant goodput (Figure 3). The primary SLO, fixed before the sweep, requires TTFT ≤ 1 s and TPOT ≤ 25 ms for at least 90% of requests.

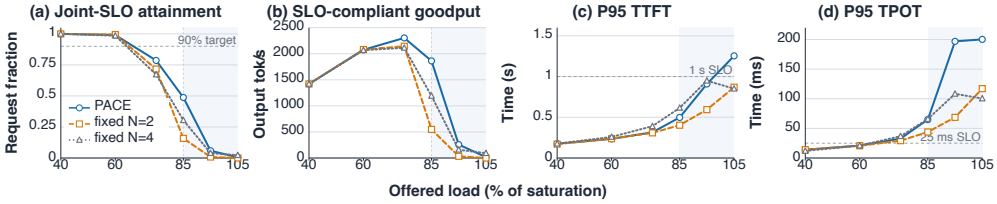

Figure 3: Open-loop serving under paired Poisson arrivals; policies share each trace. Shading marks loads beyond the 90% attainment knee, where no policy meets the target; (c, d) report raw tail percentiles in this overload regime.

PACE sustains 70.6% of saturation at the prespecified primary SLO in the open-loop $N = 4$ configuration, versus 69.9% for Fixed ($N = 2$) and 68.3% for Fixed ($N = 4$). The 95% CI on the PACE–Fixed ($N = 2$) capacity gap is $[+0.20, +1.35]$ saturation points, excluding zero.

Complete overload and tail-latency results appear in Appendix E.7.

Workload-dependent serving regime. On the low-separation math–code–chat workload, a load-specific fixed width can lead at some operating points, while PACE's rate-floor capacity is more stable across compositions (Appendix E.5). Serving gains arise when survival separation creates allocation value and saved candidate computation amortizes ragged execution. The outcome depends on three factors: survival separation determines the available allocation value; candidate cost and concurrency determine whether saved computation exceeds ragged-execution overhead; and a workload-specific fixed width can still lead in aggregate throughput when it is already well matched to the load.

7 Transfer Across Proposal Stacks

PACE requires an ordered proposal interface, not a particular decoder or estimator. Across fused checkpoints, equal-budget gains track request-level survival separation: the indepen-

dently trained 1.7B checkpoint retains a 5.5–6.9% pre-generation gain, while I-DLM-32B approaches zero on its homogeneous control (Appendix D.5 and Table 14).

External drafters and learned proposal heads show this relationship as well. Holding the 0.6B drafter and protocol fixed, scaling the target from 4B to 32B raises heterogeneous-workload gain from 47.8% to 53.9%, while homogeneous copyediting remains near zero (Table 15 and Appendix D.7). Cheap or shallow proposals are expected exceptions (Appendix D.8). Thus, the allocation rule transfers across stacks in offline replay; physical serving gains are evaluated on the fused 8B runtime.

8 Related Work

Proposal generation. Speculative decoding verifies multiple candidates together (Stern et al., 2018; Leviathan et al., 2023; Chen et al., 2023). Proposals can come from EAGLE/Medusa heads (Li et al., 2024; 2025; Cai et al., 2024), target self-speculation (Zhang et al., 2024a; Kim et al., 2023), trees (Miao et al., 2024), masked parallel decoders (Christopher et al., 2025), or diffusion-LM systems that reduce the cost or number of parallel positions (Nie et al., 2025; Wu et al., 2026; Israel et al., 2025). PACE complements these techniques by allocating candidate depth across concurrent requests.

Adaptation scope and timing. SpecDec++, DISCO, and Kangaroo adapt drafting within one request (Huang et al., 2025a; Mamou et al., 2024; Liu et al., 2024a). AdaEAGLE, PEARL, PACER, and FASER adapt proposal structure or serving control (Zhang et al., 2024b; Liu et al., 2025; Zhang et al., 2026; Chen et al., 2026b), while TurboSpec and BASS select batch-level operating points (Liu et al., 2024b; Qian et al., 2024). The distinction relevant here is a shared, cross-request candidate budget selected before generation.

TETRIS, AngelSpec, and DSpark study cross-request control after or around candidate formation (Wu et al., 2025; Liu et al., 2026; Cheng et al., 2026). Their control interfaces rank materialized candidates, prune to retention tiers, or admit requests through a learned confidence head. SpecDec++ acts before generation, but stops each request independently rather than reallocating a shared hard budget. PACE combines pre-generation decisions with cross-request budget allocation. We use dFlash (Chen et al., 2026a) and native MTP heads (AngelSlim Team, 2026) as additional proposal generators in § 5.3.

Other resource controls. AdaSpec combines SLO-aware draft control with confidence selection (Huang et al., 2025b), and concurrent LibraSpec selects diffusion speculation length from marginal gain (Lin et al., 2026). Such controllers can supply PACE's round budget, which PACE distributes across requests. Adaptive-depth models allocate layers (Elbayad et al., 2020; Schuster et al., 2022; Raposo et al., 2024); serving systems schedule iterations, memory, or prefill/decode work (Yu et al., 2022; Kwon et al., 2023; Agrawal et al., 2024; Zhong et al., 2024). PACE treats candidate depth as another cross-request resource.

9 Conclusion

PACE turns measurable survival separation into an allocation gain by assigning request-specific widths before generation. On the fused runtime, that gain becomes serving capacity when the saved computation amortizes ragged execution; external stacks provide replay evidence for transfer of the allocation rule.

Physical serving is evaluated on I-DLM-8B; the other proposal stacks provide offline replay evidence for allocation transfer. Tree proposals and unequal slot costs require a cost-aware extension of the allocator. PACE changes where candidate computation is spent while leaving the verifier unchanged.

AI use statement

Generative AI tools assisted with language editing, manuscript organization, and plotting-code development. The authors reviewed all assisted material, verified the code and technical

claims, and take responsibility for the final submission; no experimental measurements were generated by AI.

References

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve. In USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2024.

AngelSlim Team. Qwen3-8b-mtp-ttt3: Multi-token-prediction module for qwen3-8b. https://huggingface.co/AngelSlim/Qwen3-8B-MTP-TTT3, 2026.

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In International Conference on Learning Representations (ICLR), 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. In Proceedings of the International Conference on Machine Learning (ICML), 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. In Proceedings of the International Conference on Machine Learning (ICML), 2026a.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021.

Wenyan Chen, Chengzhi Lu, Yanying Lin, and Dmitrii Ustiugov. FASER: Fine-grained phase management for speculative decoding in dynamic LLM serving. arXiv preprint arXiv:2604.20503, 2026b.

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.

Jacob K Christopher, Brian R Bartoldson, Tal Ben-Nun, Michael Cardei, Bhavya Kailkhura, and Ferdinando Fioretto. Speculative diffusion decoding: Accelerating language generation through diffusion. In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), 2025.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In International Conference on Learning Representations (ICLR), 2020.

Bennett Fox. Discrete optimization via marginal analysis. Management Science, 13(3): 210–216, 1966.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track, 2021.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. Specdec++: Boosting speculative decoding via adaptive candidate lengths. In Conference on Language Modeling (COLM), 2025a.

Kaiyu Huang, Hao Wu, Zhubo Shi, Han Zou, Minchen Yu, and Qingjiang Shi. AdaSpec: Adaptive speculative decoding for fast, SLO-aware large language model serving. In ACM Symposium on Cloud Computing (SoCC), 2025b.

Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating diffusion llms via adaptive parallel decoding. In Advances in Neural Information Processing Systems (NeurIPS), 2025.

Hans Kellerer, Ulrich Pferschy, and David Pisinger. Knapsack Problems. Springer, 2004.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W. Mahoney, Amir Gholami, and Kurt Keutzer. Speculative decoding with big little decoder. In Advances in Neural Information Processing Systems (NeurIPS), 2023.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles (SOSP), 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Proceedings of the International Conference on Machine Learning (ICML), 2023.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. In Proceedings of the International Conference on Machine Learning (ICML), 2024.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. In Advances in Neural Information Processing Systems (NeurIPS), 2025.

Zexun Lin, Yuan Feng, Junlin Lv, Kevin S. Zhou, and Xike Xie. Libraspec: Dynamic diffusion-based speculative decoding via marginal-gain-driven optimization. arXiv preprint arXiv:2608.08721, 2026.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding via double early exiting. In Advances in Neural Information Processing Systems (NeurIPS), 2024a.

Hong Liu, Rui Cen, Junhan Shi, Guangshuo Qin, Jiebin Zhang, Tianyu Liu, Runzhi Fan, Guoliang Zhao, Ruobing Xie, Kai Zhang, Song Liu, Guanghua Yu, and Jianchen Zhu. AngelSpec: Towards real-world high performance inference with speculative decoding. arXiv preprint arXiv:2607.25852, 2026.

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. Pearl: Parallel speculative decoding with adaptive draft length. In International Conference on Learning Representations (ICLR), 2025.

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. TurboSpec: Closed-loop speculation control system for optimizing LLM serving goodput. arXiv preprint arXiv:2406.14066, 2024b.

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. In NeurIPS Workshop on Efficient Natural Language and Speech Processing (ENLSP), PMLR 262, 2024.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, et al. Specinfer: Accelerating generative large language model serving with tree-based speculative inference and verification. In Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 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. In Advances in Neural Information Processing Systems (NeurIPS), 2025.

Haifeng Qian, Sujan Kumar Gonugondla, Sungsoo Ha, Mingyue Shang, Sanjay Krishna Gouda, Ramesh Nallapati, Sudipta Sengupta, Xiaofei Ma, and Anoop Deoras. BASS: Batched attention-optimized speculative sampling. In Findings of the Association for Computational Linguistics: ACL, 2024.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. arXiv preprint arXiv:2404.02258, 2024.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In Advances in Neural Information Processing Systems (NeurIPS), 2022.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In Advances in Neural Information Processing Systems (NeurIPS), 2018.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. In International Conference on Learning Representations (ICLR), 2026.

Zhaoxuan Wu, Zijian Zhou, Arun Verma, Alok Prakash, Daniela Rus, and Bryan Kian Hsiang Low. TETRIS: Optimal draft token selection for batch speculative decoding. In Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), 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 USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp. 521–538, 2022.

Yifan Yu, Yuqing Jian, Junxiong Wang, Zhongzhu Zhou, Donglin Zhuang, Xinyu Fang, Sri Yanamandra, Xiaoxia Wu, Qingyang Wu, Shuaiwen Leon Song, Tri Dao, Ben Athiwaratkun, James Zou, Fan Lai, and Chenfeng Xu. Introspective diffusion language models. In Conference on Language Modeling (COLM), 2026.

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 Annual Meeting of the Association for Computational Linguistics (ACL), 2024a.

Situo Zhang, Hankun Wang, Da Ma, Zichen Zhu, Lu Chen, Kunyao Lan, and Kai Yu. Adaeagle: Optimizing speculative decoding via explicit modeling of adaptive draft structures. arXiv preprint arXiv:2412.18910, 2024b.

Situo Zhang, Yifan Zhang, Zichen Zhu, Hankun Wang, Da Ma, Danyang Zhang, Lu Chen, and Kai Yu. PACER: Blockwise pre-verification for speculative decoding with adaptive length. IEEE Transactions on Audio, Speech and Language Processing, 34:3583–3596, 2026.

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 USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2024.

A Proofs and Additional Theory

This appendix proves the results in § 3.3 and derives the ragged-execution break-even condition used in the system analysis.

Theorem 3 (Conditional greedy-output invariance). Fix a deterministic reference decoder G and assume proposal width and layout do not change its verifier logits at a committed prefix. A proposal is accepted only if it equals G's greedy next token under the current committed prefix. If emitted tokens are accepted proposals or verifier fallback tokens, positions after the first mismatch are discarded, and unfinished requests are eventually advanced, decoding terminates with the same output as G.

Lemma 1 (Tail-sum identity). Assume proposal-prefix consistency: for every d, the first d proposal positions and their acceptance outcomes are invariant to the selected maximum width $w_i \geq d$. Then $\mathbb{E}[\min(L_i, w_i)] = \sum_{d=1}^{w_i} s_i(d)$.

Proposition 1 (Exact exchange decomposition). Let A_0 be the highest-value uniform allocation satisfying $|A_0| \leq K$ and augment it with $K - |A_0|$ zero-value dummy units. Let A^* be the true top-K allocation in this augmented universe and define $G = A^* \setminus A_0$, $O = A_0 \setminus A^*$. Then $|G| = |O|$ and

$$\mathcal{D}(A_0) := U(A^*) - U(A_0) = \sum_{g \in G} s(g) - \sum_{o \in O} s(o) \geq 0.$$

There is a bijection $\phi : O \to G$ with $s(\phi(o)) \geq s(o)$; the inequality is strict exactly when one such paired inequality is strict. In particular, at uniform width $w \geq 1$, $s_i(w+1) > s_j(w)$ yields gain $s_i(w+1) - s_j(w)$.

A.1 Losslessness and liveness

Proof of Theorem 3. Let y^* be the greedy output of G. We induct on the length of the committed prefix, maintaining the invariant that the committed sequence of every request is always a prefix of y^*.

Base case. The empty sequence is trivially a prefix of y^*.

Inductive step. Suppose a round begins with committed prefix $y^*_{\leq t}$. Consider the candidate block committed in this round, which may be empty. We apply an inner induction over depths within the block. Assume that the first $d-1$ candidates have been accepted and equal $y^*_{t+1}, \ldots, y^*_{t+d-1}$; for $d = 1$, this premise is vacuous. The d-th candidate is evaluated against precisely the committed prefix $y^*_{\leq t+d-1}$ and is committed only if it equals the fixed verifier's greedy token,

$$G(x, y^*_{\leq t+d-1}) = y^*_{t+d}.$$

Thus, whenever committed, it must equal y^*_{t+d}, preserving the inner induction. No deeper candidate is committed after the first rejection. The candidates committed in this round therefore form a contiguous extension of y^* beginning at position $t+1$. After a rejection, the fallback token is the verifier's next greedy token. The invariant consequently holds at the end of the round.

The argument imposes no restriction on the source of a candidate. Candidates may come from any proposer, and the scheduler may choose arbitrary widths and verification depths based on arbitrary history; the fixed verifier remains the sole authority for commitment. This is precisely what permits policy freedom.

Termination. Every unfinished request is eventually advanced, so its prefix grows until EOS or the length limit is reached. The final sequence is thus y^*. □

Why liveness is needed. Prefix safety alone does not imply completion: a scheduler may emit no incorrect token simply by starving a request. Eventual rescheduling and fallback progress supply termination.

Scope. The theorem concerns discrete greedy-decoding semantics. Numerical effects, including bf16 ties, batch-dependent reduction order, and kernel differences, fall outside its scope and are handled empirically by the conformance protocol in Appendix C.

A.2 Tail-sum identity and top-K optimality

The theorem is exact for autoregressive draft chains. In a fused bidirectional block, changing the block width may perturb shallower states; our common-depth measurements find a 1–4% conditional-survival deviation (Appendix D.6). We treat this as a measured departure from the contract, not as a theorem-level regret value: fused allocation and serving behavior are evaluated empirically.

For the finite set $\mathcal{U} = \{(i, d) : 1 \le d \le D_i\}$ and budget $1 \le K \le \sum_i D_i$, the true single-round problem is

$$\max_{\mathbf{w}} \sum_i \sum_{d=1}^{w_i} s_i(d) \quad \text{s.t.} \quad \sum_i w_i \le K,\ 0 \le w_i \le D_i,\ w_i \in \mathbb{Z}. \tag{8}$$

Scores may be precomputed or revealed incrementally; both use the same frontier queue. Mandatory base proposals are reserved before allocating the residual budget, and shallow-first tie breaking makes the result deterministic.

Proof of Lemma 1. For any nonnegative integer-valued random variable L and integer $w \ge 0$, the pointwise identity

$$\min(L, w) = \sum_{d=1}^{w} \mathbf{1}\{L \ge d\}$$

holds. Taking conditional expectations with respect to $\mathcal{F}_t$ turns each term on the right into $\Pr(L_i \ge d \mid \mathcal{F}_t) = s_i(d)$, yielding Eq. (1). □

Proof of Theorem 1. The argument uses only that the value table is non-increasing. When the table is the true survival table, the objective is expected committed progress; when it is an estimate table, the same argument gives optimality for the estimated objective.

Assign value $s_i(d)$ to each candidate unit (i, d), where $1 \le d \le d_{\max}$. By Lemma 1, the objective of any prefix-feasible width vector $\mathbf{w}$ is the sum of its selected unit values. Thus, Eq. (8) is equivalent to selecting at most K units of maximum total value, subject to the selected depths of each request forming a prefix.

Step 1: a prefix-feasible optimum exists. Suppose an optimal unit set S contains (i, d) but not $(i, d-1)$ for some $d \ge 2$. Because $\{L_i \ge d\} \subseteq \{L_i \ge d-1\}$, $s_i(d-1) \ge s_i(d)$; replacing (i, d) with $(i, d-1)$ therefore neither decreases the objective nor increases the number of units. Repeating this replacement for every violation terminates in finitely many steps because each replacement strictly decreases $\sum_{(i,d)\in S} d$. The resulting optimum is prefix-feasible.

Step 2: top-K exchange. Let T be the K highest-valued units, breaking ties in favor of shallower positions, and let τ be the K-th largest value. Monotonicity and this tie-breaking rule make T prefix-feasible. Let $S \ne T$ be any prefix-feasible solution; without loss of generality $|S| = K$, as zero-valued units may otherwise be added, so $|S \setminus T| = |T \setminus S| \ge 1$. Because T holds the K largest values, $s(g) \ge \tau \ge s(o)$ for every $g \in T \setminus S$ and $o \in S \setminus T$.

We exchange one specific pair. Pick any row that meets $S \setminus T$ and let $o = (i_o, d_o)$ be that row's deepest unit in S; this o lies in $S \setminus T$, because a deeper selected unit of the row cannot belong to T without forcing $o \in T$ by prefix-feasibility of T. Pick any $g = (i_g, d_g) \in T \setminus S$ and let $g' = (i_g, d')$ be the shallowest depth of row i_g missing from S. Then $d' \le d_g$, so $g' \in T$ because T is prefix-feasible, and $s(g') \ge s(g)$ by monotonicity within the row.

Both operations preserve prefix-feasibility: deleting o is safe because o is the deepest selected unit of its row, and inserting g' is safe because every shallower depth of row i_g already lies in S. The two rows are distinct, so the operations do not interact. Suppose instead $i_g = i_o$.

Since S is prefix-feasible with deepest selected depth d_o in that row and $g \notin S$, we would need $d_g > d_o$; but $o \notin T$ bounds the row's depth in T strictly below d_o, contradicting $g \in T$.

Hence $S' = (S \setminus \{o\}) \cup \{g'\}$ is prefix-feasible with $|S'| = K$ and $U(S') - U(S) = s(g') - s(o) \geq 0$. Moreover $g' \in T$ and $o \notin T$, so $S' \setminus T = (S \setminus T) \setminus \{o\}$ and $|S' \setminus T| = |S \setminus T| - 1$. The quantity $|S \setminus T|$ is a nonnegative integer bounded by K, so at most K exchanges drive it to zero; with $|S| = |T| = K$ this gives $S = T$. No prefix-feasible set therefore exceeds T in value, and T is globally optimal.

Threshold form. Let τ be the K-th largest unit value; if the budget is not exhausted, interpret τ as any positive lower bound in the $\tau = 0^+$ limit. The top-K set contains every unit with value strictly greater than τ, contains no unit with value strictly less than τ, and uses deterministic shallow-first tie-breaking to select enough units equal to τ to reach cardinality K. Values within each row are nonincreasing with depth, so the selected units still form prefixes. If no units tie at the budget boundary, selecting values at least τ gives exactly K units and $w_i^* = \max\{d : s_i(d) \geq \tau\}$; with boundary ties, that simple formula can exceed the budget and the tie-breaking clause is necessary. The same sorted value list exposes the top-K' prefix of every row for each $K' < K$, so one sort serves every budget and budget selection composes directly with the allocator. □

Fixed width as a restricted class. Every uniform vector $w\mathbf{1}$ satisfies the constraints of Eq. (8) as a feasible point under budget $K = Bw$. Fixed-width policies therefore form a subset of the feasible set, and their optimal value cannot exceed that of Eq. (8). If no budget-preserving exchange improves the objective, the uniform allocation is already optimal and the values coincide. Otherwise, one profitable exchange yields a strict separation. □

For a separable cost depending only on total selected slots, the cutoff has the usual shadow-price interpretation from separable concave resource allocation (Fox, 1966; Kellerer et al., 2004). Nonseparable batched costs require a different cost-aware optimization; wall-clock claims therefore rest on Equation (6) and Appendix A.5 rather than on an extension of top-K optimality.

A.3 Heterogeneity gain

Proof of Proposition 1. After adding dummy units, both allocations contain K units, hence $|G| = |O|$. Their common units cancel under the additive objective from Lemma 1, giving

$$U(A^*) - U(A_0) = \sum_{g \in G} s(g) - \sum_{o \in O} s(o).$$

Sort G and O in nonincreasing value and pair equal ranks. Because A^* contains the K largest values, each gained unit is no less valuable than its paired removed unit (up to the fixed boundary rule). This defines the claimed bijection ϕ, proves nonnegativity, and shows strictness exactly when one paired inequality is strict. Dummy units simply price unused baseline budget at zero. For the stated local exchange, $w \geq 1$ makes $w_j = w - 1 \geq 0$ feasible and preserves total budget. Only two marginals change:

$$\Delta U = \Big[\sum_{d \leq w+1} s_i(d) + \sum_{d \leq w-1} s_j(d)\Big] - \Big[\sum_{d \leq w} s_i(d) + \sum_{d \leq w} s_j(d)\Big] = s_i(w+1) - s_j(w),$$

which is positive under the proposition's condition. □

A.4 Boundary-error regret and margin guarantee

Corollary 1 (Exact recovery under a margin). Under the hypotheses of Theorem 2, and with $\|\widehat{s} - s\|_\infty \leq \varepsilon$: if the smallest true value inside A^* exceeds the largest value outside A^* by more than 2ε, then $\widehat{A} = A^*$.

Proof of Theorem 2. Let $G = A^* \setminus \widehat{A}$, $O = \widehat{A} \setminus A^*$, and $m = |G| = |O|$. Monotonicity makes both top-K sets prefix-feasible. Since $\widehat{A}$ is estimated top-K, every selected $o \in O$

has $\widehat{s}(o) \geq \widehat{s}(g)$ for every unselected $g \in G$ (with the deterministic boundary rule). For any bijection $\phi : G \to O$, writing $e(a) = |\widehat{s}(a) - s(a)|$ gives

$$s(g) - s(\phi(g)) \leq \widehat{s}(g) + e(g) - \widehat{s}(\phi(g)) + e(\phi(g)) \leq e(g) + e(\phi(g)).$$

Common units cancel in the regret. Summing the paired inequalities over G therefore yields

$$R_K \leq \sum_{a \in A^* \triangle \widehat{A}} e(a) \leq 2m\varepsilon \leq 2K\varepsilon.$$

□

Proof of Corollary 1. Suppose $\min_{a \in A^*} s(a) - \max_{b \notin A^*} s(b) > 2\varepsilon$. For every $a \in A^*$ and $b \notin A^*$,

$$\widehat{s}(a) \geq s(a) - \varepsilon > s(b) + \varepsilon \geq \widehat{s}(b).$$

Every unit in A^* therefore ranks strictly above every unit outside A^* under the estimated values. The top K units under $\widehat{s}$ are exactly A^*; prefix feasibility follows from the same monotonicity argument as above. Hence $\widehat{A} = A^*$. □

The error $e(a)$ concerns cumulative survival, motivating direct calibration of that quantity. AUC and ECE remain empirical diagnostics rather than substitutes for the theorem's per-slot error bound.

A.5 Ragged-execution break-even condition

For a policy π and a baseline b at operating point ℓ, let Q denote expected committed tokens per round, T the expected wall-clock round cost, and $\Theta(\ell) = Q/T$ the resulting throughput.

Break-even identity. $\Theta_\pi(\ell) > \Theta_b(\ell)$ is equivalent to

$$\frac{Q_\pi}{T_\pi} > \frac{Q_b}{T_b}.$$

All four quantities are positive. Cross-multiplication gives $Q_\pi T_b > Q_b T_\pi$. Dividing by $Q_b T_b > 0$ yields

$$\frac{Q_\pi}{Q_b} > \frac{T_\pi}{T_b}.$$

Subtracting 1 from both sides shows that π improves throughput exactly when its relative progress gain exceeds its relative time overhead. Each transformation is reversible. □

Proposition 2 (Same-container batch-max coupling, expanded form). Suppose B_s requests use physical width p_L and B_d use $p_H > p_L$, while the fixed container charges the entire batch for p_H. Let $\rho = Q_{\text{rag}}/Q_{\text{fix}} > 0$, and assume the linear costs below. Ragged execution improves throughput exactly when Equation (9) holds.

Derivation of Equation (6) and proof of Proposition 2. Let the two executions have expected committed tokens Q_{rag} and Q_{fix} per step. Then

$$\frac{\Theta_{\text{rag}}}{\Theta_{\text{fix},p_H}} = \frac{Q_{\text{rag}}/C_\ell^{\text{rag}}(p_L^{B_s}, p_H^{B_d})}{Q_{\text{fix}}/C_\ell^{\text{fix}}(p_H)} = \rho \frac{C_\ell^{\text{fix}}(p_H)}{C_\ell^{\text{rag}}(p_L^{B_s}, p_H^{B_d})}.$$

Under the linear cost model, the fixed container charges the full batch for physical width p_H,

$$C_\ell^{\text{fix}}(p_H) = a + c\,(B_s + B_d)\,p_H;$$

ragged execution charges linearly for the realized widths and incurs per-round overhead factor e_ℓ, which aggregates packing, indexing, scheduling, and execution-mode differences,

$$C_\ell^{\text{rag}}(p_L^{B_s}, p_H^{B_d}) = e_\ell\big[a + c\,(B_s p_L + B_d p_H)\big].$$

Ragged execution improves throughput if and only if $C_\ell^{\text{rag}} < \rho C_\ell^{\text{fix}}$:

$$e_\ell\big[a + c(B_s p_L + B_d p_H)\big] < \rho\big[a + c(B_s + B_d)p_H\big] \iff e_\ell < \rho \frac{a + c(B_s + B_d)p_H}{a + c(B_s p_L + B_d p_H)}, \quad (9)$$

which is Eq. (6) with $e^*(B, c)$ written out.

Let $B = B_s + B_d$ and $\bar{p} = (B_s p_L + B_d p_H)/B < p_H$. The admissible overhead is

$$e^*(B, c) = \frac{a + cBp_H}{a + cB\bar{p}}.$$

Holding the mixture fixed, direct cross-multiplication gives $e^*(B+1, c) > e^*(B, c)$ for integer $B \geq 1$. Under a continuous relaxation,

$$\frac{\partial e^*}{\partial B} = \frac{ca(p_H - \bar{p})}{(a + cB\bar{p})^2} \geq 0, \qquad \frac{\partial e^*}{\partial c} = \frac{Ba(p_H - \bar{p})}{(a + cB\bar{p})^2} \geq 0.$$

These derivatives are nonnegative, and are strictly positive when $a > 0$ (with $c > 0$ for the derivative in B). Moreover, $e^* \to 1$ as $B \to 0$ and $e^* \to p_H/\bar{p}$ as $B \to \infty$. If $a = 0$, then $e^* = p_H/\bar{p}$ and these load/cost comparative statics disappear: a load-dependent crossing must then come from variation in e_ℓ/ρ. □

The comparator is fixed-p_H in the same container, concurrency comparisons hold mixture composition fixed, and ρ captures any change in committed progress induced by width or state. The realized $e_\ell(B)/\rho(B)$ crossing $e^*(B, c)$ determines the operating point at which ragged execution becomes faster.

B Survival Estimation and Calibration

This appendix supplies the estimator implementation and the calibration and decision-timing diagnostics behind § 3.4.

B.1 Cutoff-local allocation diagnostics

We measure how estimator errors alter the selected set at the budget cutoff. For each replay batch, the ex-post reference A_K^* is the top-K set formed from realized prefix indicators, while $\widehat{A}_K$ is selected from the estimated survival values using the same deterministic shallow-first tie rule. This realized-oracle diagnostic is distinct from the conditional-survival oracle in Theorem 2, but exposes the same operational question: how many slots cross the cutoff and how much committed progress those exchanges lose. Mean calibration error alone does not answer either question.

The diagnostic uses complete N16 test groups with $B = 32$ and 200 request resamples. At the most selective common budget, current-block Scout has the lowest membership error and realized regret; pre-generation signals form a tighter cluster. Complete values through $K = 256$ will be included in the public release.

Table 4: Cutoff-local allocation quality at $K = 96$ (N16 test, $B = 32$, 200 request resamples). Membership error is one minus top-K overlap with the realized oracle; regret is oracle-minus-policy committed tokens. Intervals are 95% resampling intervals for membership error.

Signal	Membership error [95% interval]	Regret
Scout, current-block	0.197 [0.104, 0.333]	7.2
History EMA	0.333 [0.250, 0.438]	21.1
Population prior	0.343 [0.260, 0.438]	22.1
Entropy, pre-generation	0.348 [0.240, 0.459]	22.9
Scout, pre-generation	0.366 [0.260, 0.490]	24.6

Finally, a synthetic rank-preserving transformation changes mean absolute survival error by 0.089 but leaves the selected allocation and its regret exactly unchanged relative to the untransformed survival scores. Boundary jitter of comparable global magnitude changes both, demonstrating why global calibration error and allocation regret are not interchangeable.

B.2 Scout architecture and two-stage isotonic calibration

Labels and training data. Labels are by-products of existing verification forwards. The math–code–chat collection records cross-round tuples (h, accept, c^*) for $N \in \{4, 8, 16\}$, where h is the last-layer hidden state at a MASK position before its proposal is realized. Records failing the KV-accounting audit are removed before training; all reported results use the cleaned Scout checkpoint.

Architecture. Scout is a lightweight MLP ($4096 \to 256 \to 1$). It is trained jointly across all three widths, with depth d embedded in the input, and scores all MASK positions of a request in one batch. Measured decision overhead is 0.4 ms per round. This scoring is part of the serving scheduler path; the isolated measurement is reported separately and is not double-counted in replay accounting.

Two-stage isotonic calibration. Raw sigmoid scores become cross-request comparable probabilities through two isotonic stages: the first calibrates conditional acceptance by depth, and the second directly calibrates cumulative survival. On the cleaned N16 test slice, pooled AUC is 0.8506, peak position-wise ECE is 1.33%, and peak cumulative ECE is 0.80%.

B.3 Signal quality by width and domain

Table 5 exposes a width-dependent crossover hidden by pooled AUC: entropy is stronger at N4 and nearly tied at N8, while Scout leads at N16. Scout is therefore most useful for deeper proposals, while entropy remains competitive at shallow widths.

Table 5: In-domain discrimination and Scout position-wise ECE by container width. Bold marks the stronger of Scout and entropy AUC.

Width	Domain	Scout AUC	Entropy AUC	Margin AUC	Scout ppECE max/mean
N4	math	0.883	0.904	0.898	0.66/0.49%
N4	code	0.862	0.888	0.882	1.25/1.08%
N4	chat	0.840	0.862	0.834	0.46/0.43%
N8	math	0.886	0.889	0.857	1.32/1.02%
N8	code	0.877	0.884	0.845	2.44/1.70%
N8	chat	0.848	0.852	0.797	0.88/0.69%
N16	math	0.859	0.810	0.790	1.72/1.19%
N16	code	0.858	0.774	0.784	3.38/2.05%
N16	chat	0.841	0.717	0.718	1.43/0.86%

B.4 Sample efficiency of online recalibration

On new domains, ranking can transfer while calibration fails (Appendix B.5). We compare isotonic-only adaptation, which freezes Scout and refits both isotonic stages, with head fine-tuning plus isotonic refitting, allocating three quarters of samples to training and one quarter to calibration. Evaluation uses the copyedit test-224 slice; its exact zero-shot ECE differs slightly from the full slice in Table 7.

Table 6: Online recalibration on a frozen copyedit test slice using nested subsamples.

Arm	requests	AUC	max position ECE	max survival ECE
Zero-shot	0	0.9043	16.85%	11.98%
Iso-only	32	0.9087	3.02%	4.42%
Iso-only	128	0.9096	1.78%	1.82%
Iso-only	512	0.9099	1.62%	1.19%
Head-ft	32	0.9242	7.68%	5.57%
Head-ft	128	0.9454	2.32%	2.28%
Head-ft	512	0.9526	1.70%	1.42%

The shift primarily affects calibration rather than ranking. Refitting isotonic maps on 32 requests reduces ECE by 4–5×; 128 requests brings cumulative and position-wise ECE below 2%. Head fine-tuning further improves discrimination but requires more samples.

B.5 Calibration under distribution shift

Table 7: Calibration of Scout without domain-specific adaptation. Dashes denote unavailable cumulative-survival ECE.

Metric	cleaned in-domain test	copyedit zero-shot	chat think-off	SQL zero-shot	toolcall zero-shot
pooled AUC (threshold ≥ 0.8)	0.8506	0.9040	0.8322	0.8078	0.8197
max position ECE (threshold $\leq 1.5\%$)	1.33%	17.70%	2.12%	16.10%	9.94%
mean position ECE	—	—	—	6.58%	5.07%
max cumulative ECE (threshold $\leq 2\%$)	0.80%	12.44%	0.85%	—	—

Distribution shift affects probability calibration more than ranking. The early depths that determine most allocations transfer well, while error concentrates in rarely selected deep positions. Copyedit, SQL, and tool-use retain useful discrimination but require domain-specific recalibration for probability-valued survival estimates; Table 6 shows that a small verified sample is sufficient for this correction.

C Conformance and Measurement Protocols

This appendix distinguishes semantic correctness under an unchanged verifier from performance measurement across execution shapes.

C.1 Serial conformance

We compare every generated token at batch size one under greedy decoding. Two independent references validate the chain: a token-by-token Hugging Face forward pass with an explicit causal mask, and an in-stack $N = 1$ autoregressive configuration. Both match the fused verification path when run with the same execution shape.

Changing kernel geometry can alter reduction order near numerical ties. At any cross-shape divergence, we recompute the top logits from the shared prefix in fp32. Here Δ is the fp32 logit gap between the two competing top tokens. We classify a divergence as numerical only when $|\Delta| \leq 0.5$ and neither execution shape consistently favors one token under fp32 arbitration. This test distinguishes numerical tie breaking from a change in verifier semantics.

C.2 Adaptive-execution checks

With all adaptive mechanisms disabled, the implementation matches the baseline byte for byte on 20/20 prompts. With variable-width execution enabled, 17/20 prompts match exactly; the remaining three divergences are confined to the fp32-arbitrated tie band. A separate 2688-request copyedit audit verifies completion, structured-output constraints, truncation, and request-level width accounting. These checks support task-level conformance under the tested execution and tie-arbitration protocol; they do not establish bitwise identity for every variable-width execution. Concurrent experiments are used for latency and throughput evaluation.

C.3 Serving configuration

The server uses greedy decoding with temperature: 0, fast_verify: false, and verify_alpha: 1.0. We report per-request and aggregate throughput separately, set max_running_requests at least as high as the largest tested concurrency, and exclude cold-start cells. Thinking mode is disabled in both the template and server configuration.

C.4 Workload and concurrency controls

Records failing KV-accounting or completion audits are excluded before model fitting or evaluation. Every concurrent workload additionally checks request completion, accounting consistency, admission capacity, and thinking-mode leakage. All reported cells pass these checks.

C.5 Reproducibility and artifacts

Artifact contents. Upon acceptance, we will publicly release the PACE source code, configuration files, Scout checkpoint, request-split manifests, workload reconstruction and validation scripts, and result manifests used to generate the reported tables and figures. The release will record file checksums and the upstream I-DLM revision a23c1a1; the bundled SGLang tree is installed in editable mode from that revision.

Recorded software environment. The primary A100 measurements use a four-GPU A100-SXM4-80GB NVLink node with NVIDIA driver 570.133.20; nvidia-smi reports CUDA 13.0. The environment uses Python 3.12, PyTorch 2.9.1+cu129, FlashInfer 0.6.3, and sgl-kernel 0.3.21. Model execution uses bfloat16 and the FlashInfer attention backend. The H100 measurements are a secondary hardware check and are not used for the primary performance claims.

Splits and randomization. Scout is constructed from 2,664 requests using a request-level, domain-stratified 80/10/10 split: 2,131 training, 266 calibration, and 267 test requests. Workload construction uses shuffle seed 20260731, and the split uses seed 20260803. Scout training uses seed 42; its four epoch permutations use seeds $42+e$ for epoch index e. Offline allocation replay uses seed 0 unless a result manifest states otherwise. Splits are made by request ID, so proposal positions from one request cannot cross partitions.

Replay and physical serving. Replay operates on frozen proposal pools and uses 300 paired resamples. It measures allocation under either an equal selected-slot budget or a matched analytic proposal-plus-verification cost model; it does not constitute a wall-clock system measurement. External-drafter and external-head results are reported only in this replay setting. Physical throughput, latency, and capacity are measured on the fused I-DLM-8B implementation. These serving comparisons intentionally retain differences in executed width-token work, because removing that work is the system effect under study.

Table 8: Independent-run structure for the physical serving results. Arms in each row share prompts and are interleaved or rotated within matched sessions.

Experiment	Replication and ordering
Matched-container headline	Three within-session seeds
Fixed-family comparison at $C = 80$	Five matched-session runs
Rate-floor capacity sweep	Three runs per measured point
Padding and permutation controls	One interleaved session for each of three seeds
Open-loop SLO sweep	Shared arrival traces and prompt order with Latin-square policy/GPU rotation; three paired seeds at L40 and L105, five at all other loads

The open-loop knee adds five paired seeds at each of L63, L66, and L69. Exact command lines, replicate identifiers, prompt orders, and per-run random seeds are retained in the result manifests. Statistical resamples are never counted as independent serving runs.

D Additional Allocation and Transfer Results

This appendix expands the allocation results with mixture, timing, and estimator diagnostics for the fused 8B stack, and reports the complete 32B, cross-stack, and same-target evidence summarized in § 7.

D.1 Workload construction and sample counts

The math–code–chat workload uses three public sources in strict round-robin order. Math draws the 500 examples in the HuggingFaceH4/MATH-500 test split (Hendrycks et al., 2021) and appends an instruction to reason step by step and box the final answer. Code draws the 164 examples in the openai/openai_humaneval test split (Chen et al., 2021) and asks for only the completed function. Chat scans the allenai/WildChat-1M training split and retains the first 500 English first-turn prompts containing 10–2000 characters. Closed-loop runs cycle through these frozen pools, use greedy think-off decoding, set max_tokens=1536, and admit at least the largest tested concurrency. The open-loop variant uses the same pools and order, with per-domain limits of 384/384/320 output tokens for math/code/chat. For WildChat, workload construction reads only the first-turn prompt text; it does not use user metadata, conversation identifiers, or later turns. The paper does not reproduce individual prompts. Reconstruction uses source indices or hashes rather than redistributing the raw WildChat text, subject to the dataset's license and terms of use.

The controlled heterogeneous workload is a strictly stratified 50/50 mixture of chat from the same frozen WildChat pool and 96 deterministic copyediting prompts. The latter span three equally generated families: document-wide word substitution, verbatim extraction of named sections, and JSON key reordering. The generation templates are deterministic. Substitution prompts replace every occurrence of the with this; extraction prompts request Section 2 or 3 and require the selected section verbatim while omitting Sections 1 and 4; JSON prompts reorder keys as [service, action, target, id, note] without changing their values. The reconstruction templates and copyedit_quality_check.py reconstruct the expected answer for every prompt. Across the audited fixed-width conditions, all 2,688 outputs pass these task-specific checks. Closed-loop runs use max_tokens=1536. Open-loop generation limits are 768 tokens for copyediting and 320 for chat. After the audit in Appendix C.4, the open-loop source pools contain 96 copyediting and 480 chat prompts. All policies use identical prompt orders, generation limits, and completion filters.

Replay protocol. At B=32, equal-budget studies score $\sum_i \min(L_i, w_i)$ over frozen proposal pools and use 300 paired resamples. The fused N16 frontier spans D=15 and $K \in [32, 480]$; the external 0.6B and dFlash stacks use D=15, EAGLE-3 uses D=8, and MTP uses D=3. The draft–target and EAGLE-3 collections contain 42,000 and 115,000 proposal groups. External-stack entropy estimators reserve 25% of request IDs for calibration. The fused frontier resamples proposal groups, whereas timing comparisons resample request IDs to preserve within-request dependence. Appendix tables name the estimator directly: Scout, calibrated entropy, or the per-request exponential moving average of past acceptance depth. Each supplies cumulative-survival estimates to the same top-K allocator.

D.2 8B allocation frontiers

Table 9 gives the equal-budget frontier summarized by Figure 2. On the heterogeneous mixture, the share of candidate slots placed after a request's first rejection at matched budgets $\bar{w}$=5/7 is 27.8/33.9% for best-fixed, 4.1/20.1% for current-block PACE, and 0.0/18.2% for the realized-acceptance oracle; these are the values plotted in Figure 2(d).

Table 9: Equal-budget allocation with current-block signals. best-fixed is the offline envelope that reselects the strongest uniform width at each budget.

Slice	Scout vs. best-fixed	Oracle gap recovered
copyedit+chat 50/50	+16.8%	91.7%
math–code–chat	+3.8%	90.1%
math / code / chat	+5.0 / +4.3 / +2.7%	88.4 / 85.0 / 96.5%
xfer-chat	+4.4%	96.0%
copyedit (homogeneous)	+0.75%	—

D.3 Signal timing and allocation gain

Table 10 reports the comparison behind the timing result in the main text. Per-domain fixed receives true labels and solves the hindsight budget problem independently in each replay batch. Current-block Scout exceeds this labeled comparator on both workloads; pre-generation signals retain most of the controlled-workload gain but are weaker on math–code–chat. The realized-acceptance oracle reaches +19.6% over best-fixed on the controlled mixture and +4.4% on math–code–chat.

Table 10: Allocation by signal timing. Intervals compare PACE with hindsight per-domain fixed under paired request resampling.

Slice	Estimator	Timing	vs. best-fixed	vs. per-domain fixed [95% CI]
Math–code–chat	Scout	current-block	+3.9%	+3.0% [+0.4, +6.5]
Math–code–chat	Entropy	current-block	+3.4%	+2.5% [+0.1, +6.0]
Math–code–chat	Scout	pre-generation	−2.0%	−2.8% [−7.8, +2.1]
Math–code–chat	Entropy	pre-generation	+0.8%	−0.1% [−3.4, +3.5]
Controlled 50/50	Scout	current-block	+17.3%	+2.5% [+0.6, +4.4]
Controlled 50/50	Entropy	current-block	+17.0%	+2.3% [+0.2, +4.5]
Controlled 50/50	Scout	pre-generation	+12.7%	−1.4% [−4.0, +1.6]
Controlled 50/50	Entropy	pre-generation	+13.1%	−1.0% [−3.4, +1.7]

Table 11: Frozen copyedit–chat ratio sweep. D is the mid-budget oracle exchange value per request; gains are relative to the per-budget best-fixed envelope. Oracle recovery uses the current-block arm.

Copyedit share	D	Current-block gain	Pre-generation gain	Oracle gap recovered
0%	0.091	+3.8%	−1.5%	97.5%
10%	0.282	+10.3%	+4.2%	97.1%
25%	0.582	+16.9%	+10.8%	94.6%
50%	0.794	+17.4%	+12.8%	92.5%
75%	0.662	+10.3%	+8.0%	85.2%
90%	0.414	+4.6%	+3.3%	73.5%
100%	0.201	+0.7%	−0.3%	51.6%

The pre-generation sweep preserves the same response structure: gains exclude zero across the interior 25–90% band, approach zero at both homogeneous endpoints, and track exchange value D with Spearman correlation 0.93 (0.89 for entropy). At the 50/50 point, this sweep estimates Scout and entropy gains of 12.8% and 13.2%. The timing table uses request-level rather than composition-sweep resampling and gives the corresponding 12.7% and 13.1% headline estimates; the same protocol difference yields 17.4% here and 17.3% in the timing table for current-block Scout (Table 10).

D.4 Survival and signal diagnostics

Table 12: Survival and zero-shot signal discrimination on I-DLM-8B at N16. AUC is pooled position-wise acceptance discrimination; $P(L{\geq}7)$ and $\bar{L}$ are measured from proposal groups. Math and code are held-out in-domain slices; the other rows use Scout without head retraining.

Domain	Scout AUC	Entropy AUC	$P(L{\geq}7)$	$\bar{L}$	labeled requests
copyedit	0.904	0.913	0.915	8.94	741
math	0.859	0.810	0.182	4.23	240
code	0.858	0.774	0.190	3.97	240
chat	0.832	0.693	0.053	2.43	185
sql	0.808	0.748	0.152	4.31	240
toolcall	0.820	0.776	0.351	5.13	240

The table separates two roles: survival depth determines the available allocation opportunity, while signal ranking determines how much an estimator recovers. Entropy is stronger

on copyediting and at shallow N4/N8 operating points (Table 5); Scout leads at deeper widths and remains discriminative on shifted domains. Probability calibration transfers less reliably than ranking and is restored by lightweight recalibration (Appendix B).

D.5 Replication on Qwen3-1.7B with R-ISD

We repeat proposal collection, Scout training, calibration, and request-level replay on Qwen3-1.7B with a runtime R-ISD LoRA adapter. The checkpoint has mean additional acceptance length $L_d = 1.09$ and uses N4/$D = 3$ with 259 test requests. Current-block Scout improves over the per-budget best-fixed envelope by 22.4% and recovers 63.5% of the oracle gap. Pre-generation Scout retains 5.5%, and history reaches 6.9%. In this checkpoint, the allocation gain remains positive despite the short mean proposal depth, where request-level marginals remain separated.

Table 13: Equal-budget replication on the independently trained Qwen3-1.7B fused checkpoint (N4/$D = 3$; 259 test requests).

Signal	Timing	vs. best-fixed	Oracle gap recovered
Scout	current-block	+22.4%	63.5%
History EMA	pre-generation	+6.9%	—
Scout	pre-generation	+5.5%	—

D.6 Fused I-DLM-32B equal-budget allocation

Protocol. We collect N4 proposal groups from I-DLM-32B with TP=2 and thinking disabled over copyedit, math, code, and chat pools of 200–240 requests each. The frozen 8B replay protocol is adapted only by reducing maximum optional depth from 15 to 3. We use B=32, budgets $K \in \{32, 64, 96\}$, and 300 paired request-bootstrap replicates. The K=96 point allocates all three positions to every request and is therefore identical for all policies. Entropy is calibrated by depth using PAV on this checkpoint's disjoint training split. Scout is not evaluated because the 5120-dimensional head was not trained.

Table 14: Complete informative-budget results for fused I-DLM-32B. Brackets are paired 95% intervals.

Workload	Policy	K=32 vs. best-fixed	K=64 vs. best-fixed
Math–code–chat	per-domain fixed	+0.9%	+1.1%
	history EMA	+1.6% [−3.9,8.0]	+1.9% [−4.6,10.8]
	entropy, pre-generation	+0.4% [−11.6,15.1]	+1.9% [−7.6,11.1]
	Entropy, current-block	+13.3% [1.5,28.0]	+11.7% [3.8,21.7]
	oracle	+15.1% [3.2,33.3]	+22.2% [12.9,31.6]
Controlled 50/50	per-domain fixed	+12.6%	+12.4%
	history EMA	+6.5%	+10.5%
	entropy, pre-generation	+9.1%	+9.8%
	Entropy, current-block	+12.8%	+16.7%
	oracle	+13.1%	+24.4%
Homogeneous copyedit	Entropy, current-block	0.0%	+0.2%

At K=64, current-block entropy on math–code–chat also improves over hindsight per-domain fixed by 10.4% [2.0, 18.9]. Its between-domain component is only 0.8/1.1 points at $K = 32/64$, compared with 12.2/12.1 points on the controlled mixture. The result therefore captures both between-domain and within-domain request variation.

Measured survival. The N4 test-set survival probabilities $P(L \geq 1/2/3)$ are copyedit 1.000/0.981/0.834, math 0.932/0.736/0.456, code 0.902/0.676/0.420, and chat 0.765/0.426/0.186. Copyedit is nearly saturated at mean $L = 2.812$. An N8 diagnostic shows why the deeper container is unsuitable here: on math, survival falls from 0.92 at depth one to 0.21 at depth four and 0.01 at depth seven. At common depths 1–3, N4 and N8

Table 15: Allocation transfer on public external proposal traces (B=32, 300 replicates). Each gain is relative to that stack's offline per-budget best-fixed envelope. Signals are recorded after proposal generation and before verification; the table measures ranking value, not physically avoided same-block drafting cost. The two draft–target blocks share one protocol, drafter, and workload set, so the 4B and 32B rows are directly comparable to each other; EAGLE-3 differs in depth, survival separation, and draft vocabulary, so its magnitudes support a qualitative comparison only. The width-two EAGLE-3 entry is the controlled 50/50 mixture at $\bar{k}$=2, the same workload as its row.

Proposal stack	Target	Workload	PACE gain	Oracle	History
Draft–target	Qwen3-4B	controlled 50/50	+47.8%	96.8%	+44.5%
		math–code–chat	+30.5%	75.4%	+13.5%
		homogeneous copyedit	+2.8%	—	+1.1%
	Qwen3-32B	controlled 50/50	+53.9%	97.5%	+50.7%
		math–code–chat	+31.7%	81.3%	+15.2%
		homogeneous copyedit	+4.9%	—	+3.4%
EAGLE-3	Qwen3-4B	controlled 50/50	+4.1% (width-2: +23.2%)	74.4%	+0.3%
		math–code–chat	+1.1%	62.0%	−1.5%

conditional survival differs by only 1–4%, providing an additional check of depth-conditional stability across container widths.

Boundary diagnostics. At the reported cutoff, estimated allocation makes 8.8 membership errors among 32 slots while incurring only 0.5 committed-token regret. Many ranking errors therefore exchange nearly equal survival values and have little effect on committed progress.

D.7 Draft–target replication with a 32B target

We pair Qwen3-0.6B with Qwen3-32B and reuse the 4B-target protocol verbatim: greedy chains of D=15, B=32, 300 replay replicates, and the same four workload pools. The collection contains approximately 48,000 proposal groups. Deep survival at depth seven is copyedit 0.937, math 0.455, code 0.306, and chat 0.058, closely matching 0.962/0.484/0.337/0.081 with the 4B target.

Table 16: Complete equal-budget replay with the Qwen3-32B target. PACE uses calibrated draft-chain probability.

Workload	PACE gain	Oracle gap recovered	History gain
copyedit+chat 50/50	+53.9%	97.5%	+50.7%
math+code+chat	+31.7%	81.3%	+15.2%
math	+25.8%	65.6%	+1.9%
code	+28.4%	69.3%	+8.0%
chat	+9.2%	80.7%	−1.7%
copyedit	+4.9%	99.2%	+3.4%

Raw and calibrated chain probabilities differ by less than 0.1 percentage point. The proposal pool contains one greedy collection pass, with uncertainty estimated through paired replay resampling. Copyediting has mean $L = 14.1$ near the $D = 15$ cap, strengthening the contrast with shallow chat requests. The matched 4B- and 32B-target experiments use the same protocol, making their 47.8% and 53.9% heterogeneous-workload gains directly comparable.

D.8 Same-target comparison with adaptive rules

Protocol and decision-time signals. Table 1 holds the Qwen3-8B target, equal math–code–chat benchmark, decoding mode, request samples, and replay seed fixed while varying the proposal generator and allocation rule. External 0.6B and dFlash use D=15, EAGLE-3 uses D=8, and MTP uses D=3. We draw 300 paired batches of B=32 from held-out request IDs and aggregate each replicate over the middle half of that stack's budget frontier.

On the two D=15 stacks (external 0.6B and dFlash), slot $d \geq 2$ is scored from entropy observations through $d-1$; slot one is generated unconditionally and scored using the population prior. The resulting frontier can draft a position only after selecting it. On the shallow EAGLE-3 and MTP stacks, same-depth entropy becomes available in the forward that produces the slot. Those columns therefore isolate allocation granularity and exact budget coupling rather than slot-level timing; selected-slot cost is a common analytic budget convention there, not a measured generation saving. All estimators and calibrators are fit on a disjoint 25% calibration split, and cumulative survival is projected to be non-increasing.

Matched cost accounting. We compare at a total analytic envelope T in target-token equivalents. A pre-generation arm selecting K slots pays $T = K(1 + r_d)$: one unit of verification and r_d units of drafting per selected slot. PACE and SpecDec++ receive this accounting. The controlled DSpark port also uses the preceding-depth signal and pays only for admitted slots, isolating binary allocation granularity from decision timing. TETRIS and AngelSpec retain their fresher post-draft scores; because those scores require the proposal pool, they first pay BDr_d and allocate the remaining $T - BDr_d$ to verification. SpecDec++ chooses a request-local threshold whose expected spend is K; an over-budget realization is clipped in arrival order, and an under-budget realization leaves residual capacity idle. The analytic proxies scale proposer parameters by forward passes per exposed slot: $0.6/8 = 0.075$ for autoregressive drafting, $(1/8)/15 = 0.0083$ for the 15-slot dFlash block, and $1/36 = 0.0278$ for the shallow EAGLE-3 and MTP heads. They are architecture-ratio proxies rather than latency measurements.

Results and interpretation. On the external 0.6B drafter, PACE's margins over TETRIS, AngelSpec, DSpark, and SpecDec++ are +2.47%, +4.76%, +10.80%, and +4.75%. Their paired-batch win fractions are .93, .99, .99, and .94, and all four 95% intervals on the mean exclude zero; against TETRIS, the interval is $[+2.29, +2.64]$%. Thus the result holds against the strongest controlled port without using the entropy of the slot being selected.

Across stacks, paired 95% intervals on the mean are above zero in 13 of 16 comparisons, below zero for TETRIS on dFlash and EAGLE-3, and cross zero only for AngelSpec on EAGLE-3. Table 17 reports every interval. The comparison can be read through the following qualitative balance:

$$\begin{aligned} \text{PACE margin} \;\approx\; & \text{recoverable drafting cost} \\ & + \text{allocation-granularity advantage} \\ & - \text{post-draft information advantage.} \end{aligned} \tag{10}$$

TETRIS has the finest and freshest post-draft ordering, so it approaches or slightly exceeds PACE when drafting is nearly free (dFlash, −0.61%) or the useful proposal chain is shallow (EAGLE-3, −1.16%). AngelSpec pays the same full-pool cost but restricts retention to coarse D-cut tiers; PACE wins three stacks and is tied on EAGLE-3 (−0.20%, with the interval on the mean crossing zero). DSpark uses preceding-depth information and selected-slot cost in this comparison, leaving its binary width as the distinguishing constraint; PACE consequently has its largest margins against this port. SpecDec++ also avoids unselected drafting, but its request-local thresholds enforce the budget only in expectation; exact top-K coupling yields positive PACE margins on all four stacks. The ports isolate rule behavior under shared traces and accounting.

Table 17: Paired 95% intervals for the mean PACE-minus-comparator margins in Table 1. All entries are percentage points.

Comparator	External 0.6B	dFlash	EAGLE-3	MTP
TETRIS	$[+2.29, +2.64]$	$[-0.69, -0.52]$	$[-1.41, -0.91]$	$[+2.20, +2.67]$
AngelSpec	$[+4.54, +4.98]$	$[+0.89, +1.12]$	$[-0.46, +0.05]$	$[+4.03, +4.50]$
DSpark	$[+10.28, +11.33]$	$[+12.46, +13.52]$	$[+1.10, +1.74]$	$[+3.86, +4.54]$
SpecDec++	$[+4.21, +5.30]$	$[+3.74, +4.96]$	$[+2.61, +3.40]$	$[+5.64, +7.32]$

Drafting-cost sensitivity. We sweep each stack's drafting-cost proxy r_d from zero to twice its nominal value while holding signals, batches, and accounting fixed (Figure 4). Comparisons with DSpark and SpecDec++ are invariant because both sides pay only for selected slots. Against post-draft rules, the ordering follows the predicted tradeoff between early cost avoidance and fresher post-draft information: TETRIS is stronger when drafting is nearly free, while PACE gains as avoidable drafting cost grows.

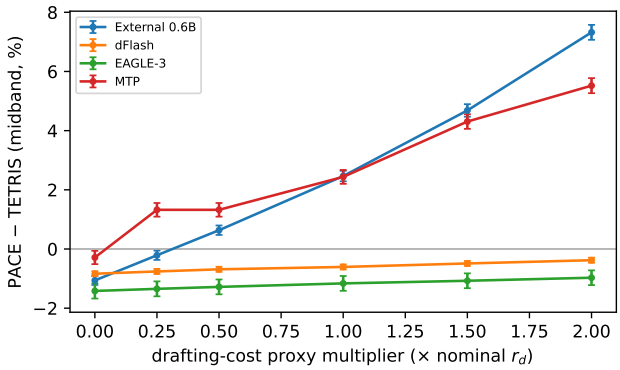

Figure 4: PACE-minus-TETRIS margin as the drafting-cost proxy is swept from 0 to 2× its nominal value per stack (300 paired batches; error bars are 95% intervals on the mean). Selected-slot comparators (DSpark, SpecDec++) are invariant to this sweep.

Request-disjoint sensitivity analysis. We split request IDs into disjoint fitting and evaluation sets, refit the estimator, isotonic maps, and population prior on the fitting set, and report results on the evaluation set. On the external 0.6B drafter, the margins over TETRIS, AngelSpec, DSpark, and SpecDec++ remain positive at +3.18%, +5.51%, +7.75%, and +3.91%, with win fractions of .997, .997, .993, and .967. Across all four stacks, the same 13 of 16 margins are positive; the exceptions remain concentrated on the cheap or shallow stacks. The qualitative ordering is unchanged under request-level separation between estimator fitting and evaluation.

E Additional End-to-End Results

This appendix reports the fused-system evidence behind § 6: closed-loop capacity, the open-loop SLO frontier, matched-container controls, and execution-cost attribution.

Primary runs use greedy decoding and the original verifier. Comparisons interleave arms within matched sessions, and each table reports its replicate count. Workload limits and conformance checks follow Appendix C and D.1.

In equal-budget replay, a container with generation block size N exposes at most $N-1$ proposal positions. Each active request receives a mandatory depth-one proposal, so a round reserves B slots before allocating the residual budget; reported K and w_i include that reserved position. In the fused deployment, m_i is the carried current-block depth and k_i the fresh-proposal count selected for the next block. The runtime sets $p_i = Q_N(1 + m_i + \max\{k_i, 1\})$, where Q_N rounds up to the container's supported odd-width menu. An N4 container therefore supports physical widths $p_i \in \{3, 5, 7\}$, while an N8 container supports widths through 15. Before each next block, the scheduler ranks feasible k_i increments across requests and chooses a prefix of this ordering by predicted progress per profiled step cost. The selected prefix determines an implicit optional count K^*_{opt}; the main serving configuration fixes the carry threshold at $\tau_s = 0.5$.

E.1 Deployment controller and cost profiling

The serving controller implements Equation (3) on the discrete fresh-proposal tiers $\{0, 1, 2, 4, 8\}$ for the N8 container. It orders feasible nested tier increments by predicted

marginal value, evaluates every prefix, and maximizes predicted committed progress divided by

$$\widehat{C}(q) = a + c\left[\sum_i (1 + m_i + \max\{k_i(q), 1\})\right]. \tag{11}$$

The reported configuration fixes $a = 14.0$ ms and $c = 0.072$ ms per pre-menu physical width-token. These constants were obtained before the headline serving runs from isolated fixed-width A100 decode profiles using math prompts, physical widths $\{3, 7, 15, 31\}$, and concurrency 32. The fit uses step latency only—no acceptance labels or controlled-workload throughput outcomes—and is not refit by test composition or test concurrency. Fixed baselines execute their stated containers and are measured directly, so this model selects only PACE's operating point and does not rescore baseline measurements. The controller uses the pre-menu footprint for selection; the rounded physical widths are charged during execution and in all reported serving measurements.

For the selected prefix, the runtime records the induced optional count $K^*_{\text{opt}} = \sum_i k_i$ and maps each request to $p_i = Q_N(1 + m_i + \max\{k_i, 1\})$ over the supported odd-width menu $\{3, 5, 7, 9, 11, 13, 15\}$. Menu rounding may add physical width-token slack and therefore need not preserve the exact pre-rounding top-K count; all serving measurements charge and execute the rounded p_i. The decision reads only the current Scout curves, carry depths, live batch size, and frozen cost coefficients. Acceptance outcomes of the newly allocated candidates are not available until after the decision.

E.2 Execution ablation: ranking versus kernel

The ablation of Table 18 separates the ranking from the kernel at C=32 on the frozen controlled workload (within-session three-seed cells, zero request errors and zero thinking leakage). Bare ($N = 8$) and logical allocation both execute the full window—logical allocation changes only the widths the scheduler books, not what the kernel runs—so the padding control earns nothing and the two arms sit at 68.2% wasted slots. Physically materializing request-specific widths removes that work instead of masking it, raising aggregate rate by 18.6% over Bare ($N = 8$) and cutting waste to 56.3%. This is direct evidence that the serving gain requires omitted candidate computation rather than favorable accounting.

Table 18: Execution ablation at C=32 on the controlled workload (aggregate tok/s and mean active-request tok/s, mean±sd, n=3). Waste is the share of materialized candidate slots lying beyond a request's first rejection.

Variant	Aggregate	Per-req.	Waste
Bare ($N = 8$)	1710 ± 17	85.8	68.2%
Logical alloc.	1692 ± 35	86.1	68.2%
Physical PACE	$\mathbf{2028 \pm 28}$	**115.4**	**56.3%**

E.3 Controlled-workload rate-constrained capacity

We interpolate measured per-request rate in $\log_2 C$ to obtain the largest closed-loop concurrency C^* sustaining a stated rate floor. The Fixed ($N = 2$) sweep uses $C \in \{80, 88, 96\}$ with $n = 3$; two $C = 80$ anchors agree with the primary session within 0.6%. The PACE sweep uses the same $N = 8$ ragged configuration at every point, with $n = 3$ at $C \in \{80, 84, 88, 92, 96, 100, 104, 112\}$. Its measured per-request rate declines from 51.4 to 38.1 tok/s, while aggregate throughput remains between 2632 and 2807 tok/s. All three replicates at $C = 104$ exceed 40 tok/s per request (mean 40.7 ± 0.3), giving the directly measured lower bound $C^* \geq 104$; interpolation between $C = 104$ and 112 gives $C^* = 106.1$.

Relative to the best-static capacity envelope at each rate floor, the interpolated PACE crossings support 28%, 25%, and 24% more concurrent requests. Rate-constrained goodput is 1.8% below that envelope at 40 tok/s and 2.7% and 1.6% above it at 45 and 50 tok/s. The main text reports the resulting 24–28% capacity range; the directly measured 40 tok/s lower

Table 19: Rate-floor capacity and rate-constrained goodput on A100. Each entry is interpolated C^*/G, where G is aggregate tok/s at the same crossing. PACE uses one ragged N8-container configuration throughout the sweep.

Floor (tok/s)	Fixed ($N = 2$)	Fixed ($N = 4$)	PACE ($N = 8$)
40	83/2800	83/2560	106/2750
45	72/2717	75/2708	94/2789
50	< 64/–	66/2630	82/2671

bound is 104 versus 83, and interpolation places the crossing at 106.1. These measurements characterize closed-loop active-request rate targets.

E.4 Math–code–chat closed-loop frontier

We next run the same fused system on the equal math–code–chat mixture with thinking disabled. The deployment-level action space is restricted to the N4 container, while PACE retains the frozen Scout signal and request-level allocation rule. All cells use three repetitions and have zero request errors and zero thinking leakage.

Table 20: Math–code–chat closed-loop sweep (aggregate tok/s / mean active-request tok/s, mean±sd, $n = 3$). Bold marks the largest point estimate in each metric at a given concurrency.

C	Policy	Aggregate tok/s	Active-request tok/s
32	PACE ($N = 4$)	2132.4 ± 19.2	96.8 ± 0.8
	Fixed ($N = 4$)	$\mathbf{2326.4 \pm 17.9}$	$\mathbf{98.9 \pm 0.7}$
	Fixed ($N = 2$)	2253.4 ± 64.2	88.0 ± 0.2
48	PACE ($N = 4$)	$\mathbf{2530.7 \pm 23.1}$	$\mathbf{80.5 \pm 0.6}$
	Fixed ($N = 4$)	2496.2 ± 7.4	73.0 ± 0.1
	Fixed ($N = 2$)	2468.1 ± 9.9	67.8 ± 0.4
64	PACE ($N = 4$)	2702.1 ± 29.9	$\mathbf{63.2 \pm 0.1}$
	Fixed ($N = 4$)	2642.2 ± 19.8	56.7 ± 0.1
	Fixed ($N = 2$)	$\mathbf{2892.1 \pm 12.5}$	58.8 ± 0.3
80	PACE ($N = 4$)	2783.0 ± 24.3	50.8 ± 0.1
	Fixed ($N = 4$)	2699.1 ± 16.3	45.6 ± 0.1
	Fixed ($N = 2$)	$\mathbf{3154.5 \pm 6.6}$	$\mathbf{52.5 \pm 0.1}$
	PACE ($N = 8$)	2430.5 ± 9.1	45.7 ± 0.1

At the evaluated $C \in \{48, 64, 80\}$ points, PACE ($N = 4$) improves both reported metrics over Fixed ($N = 4$). The aggregate margins are 1.4%, 2.3%, and 3.1%; the active-request margins are 10.3%, 11.5%, and 11.4%. Fixed ($N = 2$) enters the aggregate frontier at $C = 64$ and is strongest on both axes at $C = 80$. Thus the sweep identifies a load-dependent frontier rather than a global ranking.

E.5 Math–code–chat rate-floor capacity

We extend the math–code–chat sweep to $C = 112$ and interpolate mean active-request rate in $\log_2 C$, using the same procedure as Appendix E.3. The result separates interactivity-oriented targets from the looser saturated-throughput regime.

Table 21: Math–code–chat closed-loop capacity C^* at active-request rate floors. These floors are not latency SLOs.

Rate floor	PACE ($N = 4$)	Fixed ($N = 2$)	Fixed ($N = 4$)
40 tok/s	105.7	109.0	93.8
45 tok/s	93.5	87.4	81.2
50 tok/s	81.7	82.5	< 80

At 45 tok/s, PACE increases capacity by 7.0% over Fixed ($N = 2$) and 15.1% over Fixed ($N = 4$). At 50 tok/s it approximately ties Fixed ($N = 2$), while at the looser 40 tok/s

floor Fixed ($N = 2$) supports 3.1% more concurrency. The interactivity target therefore determines the preferred operating point, as predicted by the break-even analysis.

E.6 Math–code–chat container ablation

An N8 diagnostic showed 65–66% unused proposed positions and negligible realized value from depths 9–15. We therefore compare N8 with an N4 container while holding Scout, calibration, allocation, engine settings, and workload fixed, and use the same N4 configuration for subsequent sweeps.

At $C = 80$, the N4-container PACE arm reaches 2783.0 ± 24.3 aggregate tok/s and 50.80 ± 0.10 active-request tok/s. Holding the signal, workload, and engine fixed while exposing the N8 action space yields 2430.5 ± 9.1 and 45.67 ± 0.06. The N4 container therefore improves the two metrics by 14.5% and 11.2%. Its selected-width histogram is approximately balanced across the available nontrivial widths (17,356/18,677/17,375 selections at widths 3/5/7), showing that the signal distinguishes requests inside the useful range; the loss from N8 instead comes from provisioning depths 9–15 whose physical cost is rarely repaid on this workload.

E.7 Open-loop Poisson frontier

The open-loop experiment replays identical Poisson traces for every policy on the frozen 50/50 copyedit–chat workload. Offered loads L40–L105 are fractions of the common 7.86 requests/s saturation anchor. The primary joint SLO is TTFT ≤ 1 s and TPOT ≤ 25 ms, with capacity defined by 90% attainment. The anchor is frozen before the main sweep as the largest completed-request rate observed across Fixed ($N = 2/4/8$) closed-loop calibration runs at $C \in \{32, 64, 96, 128\}$. Each trace has a 60 s warm-up, a 360 s measurement window, and at most 300 s of drain; membership in the measurement window is determined by scheduled arrival time. Requests have a 300 s timeout, and timeouts or requests unfinished after drain count as SLO failures. All policies replay the same arrivals and prompt order at a given seed, with a Latin-square policy/GPU rotation. L40 and L105 use three paired seeds; all other loads use five. So that the 90% capacity crossing is measured rather than interpolated across the L60–L75 gap, the load grid includes dense knee points at L63, L66, and L69 (five paired seeds each). Queue and running-request counts are sampled every 5 s.

TTFT is scheduled-arrival-to-first-nonempty-token time and therefore includes client-visible queueing. For requests with at least two output tokens, TPOT is $(t_{\text{last}} - t_{\text{first}})/(n_{\text{out}} - 1)$, where n_{out} is the server-reported completion-token count rather than the number of streaming chunks. We report per-seed request-level P50/P95/P99 TTFT and TPOT, completed requests/s, aggregate output tok/s, joint attainment, and SLO-compliant goodput. The workload construction, generation limits, and observed length distributions are given in Appendix D.1.

Table 22: Open-loop P95 latency. Entries are P95 TTFT in seconds / P95 TPOT in milliseconds, averaged across paired seeds.

Policy	L40	L60	L63	L66	L69	L75	L85	L95	L105
PACE ($N = 4$)	0.175/14.0	0.236/20.9	0.240/21.5	0.258/23.8	0.275/26.2	0.314/32.7	0.498/64.9	0.909/196.9	1.254/200.2
Fixed ($N = 2$)	0.171/14.1	0.235/20.1	0.242/21.8	0.259/23.8	0.277/25.8	0.308/28.6	0.403/43.9	0.594/68.7	0.869/117.1
Fixed ($N = 4$)	0.176/12.0	0.257/20.8	0.261/21.6	0.296/25.5	0.324/28.8	0.386/36.1	0.615/65.8	0.948/108.3	0.851/100.4

Table 23: Open-loop joint-SLO attainment / SLO-compliant goodput in output tok/s. L85 lies beyond every policy's 90% attainment knee.

Policy	L40	L60	L63	L66	L69	L75	L85	L95	L105
PACE ($N = 4$)	1.00/1420	0.99/2073	0.99/2213	0.97/2246	0.94/2324	0.80/2299	0.49/1863	0.06/258	0.01/17
Fixed ($N = 2$)	1.00/1420	1.00/2078	1.00/2217	0.97/2240	0.93/2310	0.75/2198	0.16/549	0.01/33	0.00/0
Fixed ($N = 4$)	1.00/1420	0.99/2067	0.99/2204	0.95/2213	0.89/2250	0.69/2124	0.30/1191	0.04/159	0.02/92

With the dense knee points in the grid, the 90% crossing is bracketed by measured loads (attainment 0.94 at L69, 0.80 at L75 for PACE ($N = 4$)); interpolation between adjacent measured points gives primary capacities of 70.6%, 69.9%, and 68.3% of saturation for

PACE ($N = 4$), Fixed ($N = 2$), and Fixed ($N = 4$). A paired bootstrap over the shared seed axis (B=2000) places the PACE–Fixed ($N = 2$) capacity gap at [+0.20, +1.35] points of saturation (95% CI, excluding zero). The complete latency and goodput curves show the same capacity ordering near the primary crossing; higher-load cells characterize overload behavior rather than the headline operating point.

Table 24: Open-loop SLO-capacity sensitivity at a 1.0-s TTFT bound (% of saturation). Varying the TTFT bound from 0.5 to 2.0 s changes these crossings by at most 0.7 points.

TPOT	Attain.	PACE ($N = 4$)	Fixed ($N = 2$)	Fixed ($N = 4$)	PACE ($N = 8$)
20 ms	90%	61.6	60.7	61.4	50.9
20 ms	95%	54.5	55.1	55.8	45.4
25 ms	90%	66.6	65.1	64.1	61.2
25 ms	95%	63.0	62.4	61.7	55.2
33 ms	90%	76.4	77.1	72.6	63.8
33 ms	95%	72.9	75.9	66.2	61.7

The threshold grid confirms a banded frontier. PACE ($N = 4$) is strongest throughout the 25 ms TPOT band at 90% attainment; a looser 33 ms target favors Fixed ($N = 2$), and the strict 20 ms/95% condition favors a fixed width. The original N8 container reaches 61.2% primary capacity, reinforcing the importance of matching the execution container to the useful depth range.

E.8 Matched-container execution boundary

In a batch-uniform-width engine, the deepest request determines batch width. The matched-N8 control isolates the benefit of removing this max coupling before the fused forward.

Table 25: Aggregate-throughput change from removing matched-N8 max coupling. A100 entries are matched-session measurements; H100 provides an additional execution-boundary point.

C	A100	H100
16	-1.8% / $+2.8\%$ (two sessions)	—
64	—	-2.1%
80	$+27.3\%$ (2468.4 ± 23.0 vs 1939.0 ± 9.9)	—
96	$+29.6\%$ (2586.4 ± 30.6 vs 1995.7 ± 18.0)	—
128	$+33.3\%$ (2805.4 ± 9.4 vs 2104.1 ± 9.0)	—

On A100, candidate work grows enough with load that narrowing shallow requests offsets ragged packing overhead; mean per-request rate improves by 33.7–36.2% over $C = 80$–128. The fitted marginal width-token cost is $c \approx 0.072$ ms on A100 but only 0.012 ms on H100, where the same matched-container comparison falls 2.1% below break-even. This reversal is consistent with Equation (6): allocation value and physical execution cost are distinct quantities. The H100 point is a directional two-run measurement.

E.9 Attribution: allocation versus decoupling

The width-matched permutation control of § 6 recomputes the PACE widths for a round and permutes them across that round's requests, so the two arms differ only in which request receives which width. Equation (7) splits the matched-container gain accordingly. Arms run in one interleaved session per seed, three seeds, A100, frozen controlled 50/50 copyedit–chat workload, greedy, MEM_FRACTION = 0.75; all cells have zero request errors and zero thinking leakage.

The C=16 row explains the null aggregate cell of Table 25. At that load the two components have opposite signs and nearly cancel: ranking still earns +10.2%, while ragged packing is no longer amortized and costs −6.8%. This is the same crossing as the H100 reversal above, reached by lowering B rather than c — exactly the regime Equation (6) identifies, since $e^*(B, c) \to 1$ as cB falls. Two independent sessions place that cell at −1.8% and +2.8% (per-seed $[-2.4, +7.0]$), i.e. at zero within measurement scatter. Active-request progress is not ambiguous there: PACE remains +13.1% over Fixed ($N = 8$) in every seed, because $\mathcal{D}_{\text{alloc}}$ is large enough on that axis to absorb the execution penalty.

Table 26: Matched-N8 gain decomposed into survival-based assignment ($\mathcal{D}_{\text{alloc}}$) and removal of batch-max coupling ($\mathcal{D}_{\text{decouple}}$). Brackets give per-seed ranges.

		aggregate	active-request
C=80	$\mathcal{D}_{\text{alloc}}$	+11.8% [+9.3, +13.2]	+32.7% [+32.0, +33.2]
	$\mathcal{D}_{\text{decouple}}$	+15.8% [+14.2, +16.6]	+4.6% [+4.3, +4.9]
C=16	$\mathcal{D}_{\text{alloc}}$	+10.2% [+6.2, +14.2]	+34.5% [+34.3, +34.8]
	$\mathcal{D}_{\text{decouple}}$	−6.8% [−8.1, −5.9]	−15.9% [−16.2, −15.7]

The permutation preserves the width multiset within each round. Subsequent widths evolve with the preceding acceptance outcomes, so the reported attribution describes the realized trajectories rather than a frozen single-round permutation.

E.10 Output-quality check under concurrent serving

Both arms answer the same 200 GSM8K questions at C=16 on the same pod, engine tree, and day, paired per question: think-off at a 1536-token cap (primary) and think-on at 8192 (anchor). Accuracy is 92.0 vs. 91.5 (think-off) and 95.0 vs. 95.5 (think-on) for PACE vs. Fixed ($N = 4$); paired disagreements are 2/1 and 2/3 questions with sign-test p=1.0 in both regimes, and think-on truncations are symmetric (5/3). The comparison uses greedy verification, matching the conformance protocol.

Measurement scope. Avoided candidate computation is measured on the fused 8B runtime. External stacks evaluate allocation under a matched cost model; tree proposals and unequal slot costs require corresponding extensions of the allocator.