STRATASPEC: SELF-SPECULATIVE DECODING WITH STRUCTURED ROUTES FOR EFFICIENT LLM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Efficient LLM serving under limited compute depends on batching, which amortizes each model pass across several requests. Self-speculative decoding (SSD) reduces proposal cost further without a separate draft model by drafting through a route, a subset of the target model's attention and MLP sub-layers, before full-model verification. Existing methods adapt this route to each request from verifier feedback, but assess each choice in isolation. A shared proposal pass executes the union of those routes. Because independently chosen routes rarely skip the same sub-layers, their union can reactivate almost the full model and give different row sets to its sub-layers. Much of the per-request saving then disappears at the batch level. We introduce StrataSpec, which retains request-specific route choice within one compact, regular proposal pass. Its routes form a nested family in which every richer route contains the preceding one. The union of any mixture is exactly its richest assigned route, and sorting requests by route gives each sub-layer a contiguous suffix of rows. A batch planner chooses routes and draft length from the incremental work they add to the active batch. StrataSpec improves throughput by up to 4.40× over the strongest of three recent adaptive SSD baselines and sustains 98.7–99.3% SLO attainment under continuous arrivals, while baseline queues grow by orders of magnitude.

1 INTRODUCTION

LLM services increasingly run near users and their data, where limited compute and communication latency shape how requests are served (Chen et al., 2026a; Park et al., 2025; Xu et al., 2025; Choi et al., 2026; Song et al., 2024). Continuous batching groups ready requests into shared decoding iterations (Yu et al., 2022; Kwon et al., 2023), while self-speculative decoding (SSD) reuses the target model to generate proposals without a separate draft model (Elhoushi et al., 2024; Zhang et al., 2024; Kim et al., 2025). Adaptive SSD uses verifier feedback to vary the amount of target-model computation assigned to each request (Xia et al., 2025; Chen et al., 2025; Cha et al., 2026). Batching supplies reuse while routing adapts computation to request difficulty under tight resource budgets.

However, the two mechanisms interact once a shared proposal pass combines routes chosen for different requests. The cost of one route then depends on the other routes in the batch. A common route is regular but treats requests alike; independent routes preserve choice but fill one another's skips (Figure 1). At batch size sixteen, individual routes skip 22–27% of the optional sub-layers, yet fewer than 4% remain skippable across the batch (Figure 2). Their differing row sets also require repeated calls or regrouping (Selvam et al., 2024; Liu et al., 2025). Request-specific routing can retain its advantage only if its routes share a compact, regular batch structure. Nesting provides that structure, although adapting the resulting frontier to a changing batch presents two challenges.

The first challenge is how to evaluate a route in the context of the active batch. Activating a sub-layer loads its weights once, while each request that uses it adds row and KV-cache work. A route can therefore be inexpensive when its weights are already active and costly when it expands the union. Existing selectors use standalone latency and acceptance (Xia et al., 2025; Chen et al., 2025; Cha et al., 2026), so they cannot price the work a route adds to the current batch. A batch planner must therefore compare committed progress against the shared and request-dependent work of the complete batch assignment executed in that round.

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.

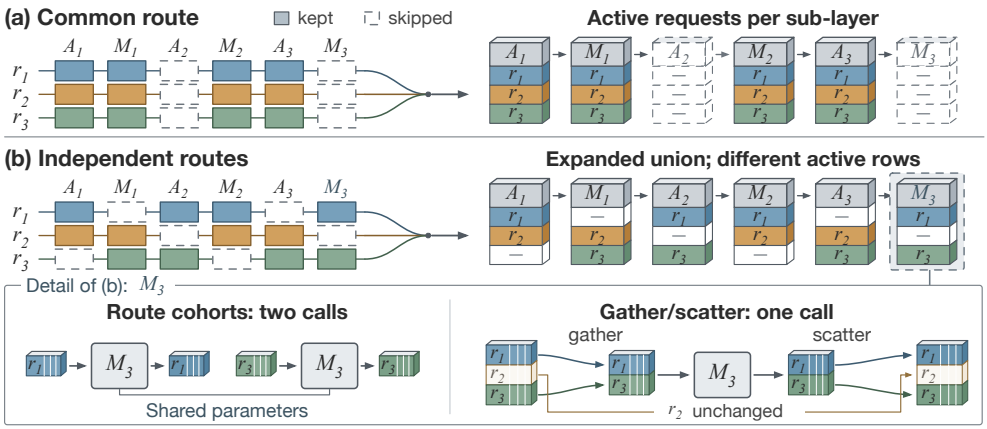

Figure 1: Batch execution under a common route and independent routes. The common route is regular but fixed across requests. Independent routes preserve per-request choice, yet expand the union and change active rows. A and M denote attention and MLP sub-layers.

The second challenge is how to build a nested family that still fits changing requests. Routes derived independently from verifier states need not be ordered by containment, and imposing that order afterward can remove the sub-layers that sustain acceptance. The useful routes also change as requests arrive, finish, and move through generation. Regrouping after selection cannot shrink an expanded union (Selvam et al., 2024; Liu et al., 2025), while repeating route search every round would consume the savings it seeks. Construction must enforce containment from the outset and refresh the family on a slower timescale than request assignment.

We propose StrataSpec, a self-speculative serving system that organizes request routes as a nested frontier built from committed verifier states. Its batch planner prices each tier by the work it adds to the current batch. The frontier refreshes across proposal rounds, while assignments update at verification boundaries. Requests can occupy different tiers while the batch retains a shared suffix order. Across HumanEval, GovReport, and AIME, StrataSpec improves throughput by 1.73–4.40× over the strongest of three adaptive SSD baselines. Under continuous ShareGPT arrivals, StrataSpec maintains above 98% SLO attainment, while KnapSpec falls to 71.1% at the highest offered load.

The main contributions are as follows:

- We identify how per-request routes lose sparsity and regularity when batched, and formulate their marginal cost from shared weight activation, request rows, and KV-cache pages.
- We introduce a nested route frontier that preserves per-request choice while bounding the batch union and exposing a shared suffix order.
- We design a batch planner that assigns tiers and draft length from their marginal work in the active batch, and a verifier-driven procedure that builds and refreshes the nested frontier.
- We develop an end-to-end implementation of StrataSpec that reaches up to 4.40× higher throughput and maintains 98.7–99.3% SLO attainment under continuous arrivals.

2 Background and Motivation

2.1 From request-local routing to batch composition

An SSD round drafts multiple tokens along a route, then verifies them with the full model. The accepted prefix and a correction or bonus token are committed, and the resulting states guide the next routing decision (Stern et al., 2018; Leviathan et al., 2023; Chen et al., 2023).

Adaptive routing balances proposal cost against the prefix expected to survive verification. A lighter route drafts cheaply but may shorten that prefix; a richer route spends more work to improve acceptance. Existing methods revisit this balance as verifier feedback changes (Xia et al., 2025; Chen

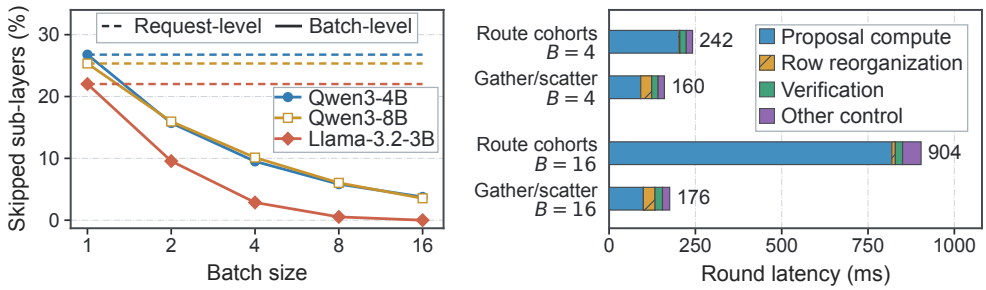

Figure 2: Request-level sparsity does not translate directly into batch-level savings. Independent routes fill one another's skips. Route cohorts repeat shared calls, while gather/scatter reorganizes the active rows of each sub-layer.

et al., 2025; Cha et al., 2026). In this request-local view, the measured route latency is the proposal cost paid by the GPU.

Continuous batching leaves these route choices intact but changes their cost. Recent batch-oriented systems coordinate draft-token selection and speculative phases across requests (Wu et al., 2025; Chen et al., 2026b). At the route level, the batch pays once to activate a sub-layer, then adds the rows and KV-cache pages of every request that uses it. A route's cost is therefore the work it adds to the routes already present, rather than the latency measured for that route alone.

2.2 Batch-level sparsity and execution

Once request-local routes enter a shared proposal pass, their costs begin to interact. A sub-layer disappears from the batch only when every request omits it, so complementary choices progressively fill one another's skips. Figure 2 compares the resulting batch-level sparsity with individual routes and exposes the cost of the two natural execution layouts.

Across these batch sizes, request-level sparsity stays fairly stable, but the batch-level union rapidly approaches the full set of optional sub-layers. Complementary skip masks leave almost every sub-layer active, so routes that are individually sparse can demand nearly the full weight union when executed together in one proposal pass.

The execution measurements reveal a second cost. Route cohorts keep rows contiguous but repeat shared sub-layer calls; gather/scatter uses one call but reorganizes rows as its active set changes (Selvam et al., 2024; Liu et al., 2025). At batch sixteen, cohorts issue 12.1× as many calls, while reorganization consumes 13.5% of a gather/scatter round. Neither layout shrinks the route union.

2.3 Structured routes for batched drafting

The bottleneck therefore precedes kernel execution. Once independently chosen routes from adaptive SSD systems (Xia et al., 2025; Chen et al., 2025; Cha et al., 2026) expand the union, a different row layout cannot recover the missing sparsity. Route composition must instead be determined as part of selection. A nested family provides this structure. Requests can still occupy different tiers, but their routes now differ by ordered increments that the batch can share. The richest assigned tier bounds the union, lower tiers add rows and KV-cache pages to active weights, and one tier order gives every sub-layer a contiguous active suffix. Marginal cost and row layout then follow from the same frontier. Within that frontier, crossing the richest assigned tier introduces new weights, while other moves add work to weights already in memory.

3 StrataSpec

StrataSpec follows the cycle in Figure 3. Verification commits output and updates the states and acceptance history used to maintain the frontier. Before the next proposal, the planner revises request tiers and draft length; sorting those tiers produces the suffix views consumed by the kernels.

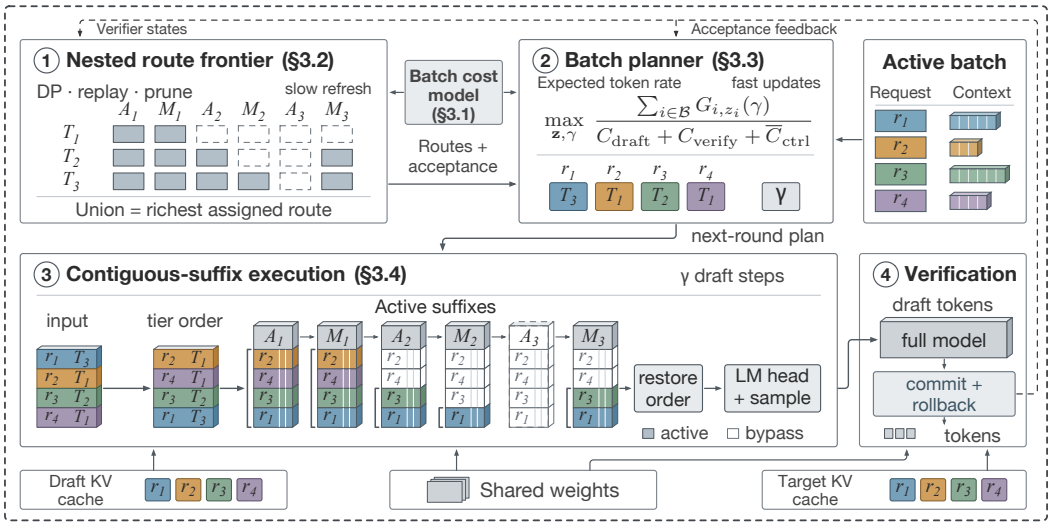

Figure 3: Overview of StrataSpec. Verification provides committed states and acceptance feedback for a nested route frontier. The batch planner selects tiers and draft length from measured batch cost, and tier order exposes contiguous active suffixes during proposal.

3.1 Batch cost model

For a decoder with M attention/MLP sub-layers, let $R_i \subseteq \{1, \dots, M\}$ denote the route assigned to request i in the active batch $\mathcal{B}$, and let γ be the draft length. At draft position s, let $Q_{i,s}$ be the number of KV-cache pages visible to request i, and define $\mathcal{A}_{m,s} = \{i \in \mathcal{B} : m \in R_i\}$, $n_{m,s} = |\mathcal{A}_{m,s}|$, and $Q_{m,s} = \sum_{i \in \mathcal{A}_{m,s}} Q_{i,s}$. The draft cost is

$$C_{\text{draft}}(\mathbf{R}, \gamma) = C_{\text{base}}(B, \gamma) + \sum_{s=1}^{\gamma} \sum_{m=1}^{M} \tau_m(n_{m,s}, Q_{m,s}) \,, \tag{1}$$

where $B = |\mathcal{B}|$, $\tau_m(0, \cdot) = 0$, and C_{base} covers full-batch operations outside the sub-layers. The measured function τ_m gives sub-layer latency for the active rows and their aggregate KV-cache pages; its nonzero entries include the cost of activating the weights.

3.2 Nested route frontier

To construct the frontier, StrataSpec replays candidate routes from committed verifier states, balancing probes across requests. An attention/MLP-separated dynamic program builds a chain in which each richer tier contains its predecessor by construction:

$$R^{(1)} \subseteq R^{(2)} \subseteq \cdots \subseteq R^{(K)}. \tag{2}$$

The union of any assignment is its richest assigned tier. Attention and MLP sub-layers may enter at different tiers, so tier increments need not be uniform layer cuts. Replay prunes dominated choices while preserving the nested chain order.

Replay also estimates the utility of each tier. For greedy decoding, $\hat{a}_{i,k}$ is top-1 agreement with full verification; for sampling it is the accepted probability mass

$$\hat{a}_{i,k} = \frac{1}{|P_i|} \sum_{p \in P_i} \sum_{v} \min(p_{i,p}(v), q_{i,k,p}(v)) \,, \tag{3}$$

where P_i is the probe set and $p_{i,p}$ and $q_{i,k,p}$ are the target and tier distributions. This mass is the acceptance probability under speculative rejection sampling.

3.3 BATCH PLANNER

The batch planner uses recent verification feedback to estimate the progress request i obtains at tier k. For a draft of length γ,

$$G_{i,k}(\gamma) = \sum_{j=0}^{\gamma} \hat{a}_{i,k}^{j}. \tag{4}$$

The $j = 0$ term is the correction or bonus token from verification. Given the frontier, StrataSpec jointly selects request tiers $\mathbf{z}$ and draft length γ to maximize the expected committed-token rate:

$$\max_{\mathbf{z},\gamma} \frac{\sum_{i\in\mathcal{B}} G_{i,z_i}(\gamma)}{C_{\text{draft}}(\{R^{(z_i)}\},\gamma) + C_{\text{verify}}(\mathcal{B},\gamma) + \overline{C}_{\text{ctrl}}}. \tag{5}$$

The denominator includes assignment, row ordering, and amortized frontier construction along with the shared sub-layer work. Moving within the active frontier changes its row and KV-cache page counts; crossing its richest tier also activates new weights. For each γ, coordinate ascent updates only the thresholds crossed by a tier move and retains the highest-rate plan across candidate draft lengths.

3.4 CONTIGUOUS-SUFFIX EXECUTION

Each sub-layer first appears at one tier and remains active in every richer one, so a single tier order serves the draft without per-sub-layer regrouping.

Let π stably sort requests by nondecreasing tier index, and define the zero-based suffix offset $o_k = |\{i : z_i < k\}|$. Define the activation threshold and active set of sub-layer m by

$$\kappa_m = \min\{k : m \in R^{(k)}\}, \qquad \mathcal{A}_m = \{i : z_i \geq \kappa_m\},$$

using $\kappa_m = K + 1$ for sub-layers absent from the frontier. Nesting makes $\mathcal{A}_m$ a suffix that begins at the corresponding offset o_{κ_m}.

At each draft position, equal-threshold segments reuse one hidden-state view and its attention metadata. A single inverse permutation restores request positions before the LM head, while physical KV blocks remain in place.

Frontier construction and batch planning operate at different timescales. The cost of constructing the frontier is amortized over committed progress, while assignments react at verification boundaries to arrivals, completions, and acceptance feedback.

4 EVALUATION

4.1 EXPERIMENTAL SETUP

System and models. We implement StrataSpec in vLLM 0.9.2 (Kwon et al., 2023) using PyTorch 2.7, CUDA 12.4, FlashAttention-2 (Dao, 2024), and BF16 weights. Experiments use Qwen3-4B, Qwen3-8B, and Llama-3.2-3B-Instruct (Llama-3.2-3B in figures and tables) (Yang et al., 2025; Grattafiori et al., 2024) on an RTX 4090 with 24 GB of memory.

Baselines. We compare with three adaptive SSD methods. **KnapSpec** (Cha et al., 2026) selects attention and MLP sub-layers using measured hardware cost; **CLaSp** (Chen et al., 2025) derives routes from recent verifier states; and **SWIFT** (Xia et al., 2025) updates its skip pattern during generation. All methods use the same target verifier, kernels, KV-cache allocator, and scheduler. For each independent-route baseline, we report the faster of route-cohort and gather/scatter execution.

Workloads and metrics. Fixed-batch evaluation uses HumanEval (Chen et al., 2021) for code, GovReport (Huang et al., 2021) for long-context summarization, and AIME for long-form mathematical reasoning. Online evaluation replays ShareGPT conversations with Poisson arrivals and a burst trace. Fixed-batch throughput counts committed output tokens per second; online results use 500 ms time to first token (TTFT) and 200 ms time per output token (TPOT) SLOs and report attainment, goodput, P95 queueing delay, and P95 TPOT.

Table 1: End-to-end throughput (tok/s) across workloads and models. B denotes batch size, and speedup is measured over KnapSpec.

Workload	Model	Input / B	KnapSpec	CLaSp	SWIFT	StrataSpec	Speedup
HumanEval	Qwen3-4B	– / 2	49.1	42.0	41.1	142.8	**2.91**×
		– / 4	98.0	86.2	54.2	258.6	**2.64**×
		– / 8	174.8	157.0	121.4	466.7	**2.67**×
		– / 16	282.2	261.7	182.3	780.7	**2.77**×
	Qwen3-8B	– / 2	44.5	38.9	32.2	98.0	**2.20**×
		– / 4	88.4	74.3	51.5	189.9	**2.15**×
		– / 8	149.8	134.1	108.6	335.6	**2.24**×
	Llama-3.2-3B	– / 2	84.5	71.4	45.8	184.2	**2.18**×
		– / 4	152.9	127.4	86.3	348.8	**2.28**×
		– / 8	266.8	222.6	142.9	629.3	**2.36**×
		– / 16	431.0	372.1	222.6	1036.9	**2.41**×
GovReport	Qwen3-4B	16K / 2	31.9	28.9	25.4	58.5	**1.83**×
		8K / 4	63.6	57.6	31.2	124.3	**1.95**×
		4K / 8	123.5	115.1	64.6	250.1	**2.02**×
	Qwen3-8B	8K / 2	32.0	29.0	20.5	60.8	**1.90**×
		4K / 4	58.2	55.4	30.6	121.5	**2.09**×
		2K / 8	109.5	99.5	56.6	234.5	**2.14**×
	Llama-3.2-3B	8K / 4	95.6	83.6	63.7	165.6	**1.73**×
		4K / 8	184.5	158.0	111.3	328.3	**1.78**×
		2K / 8	224.5	184.1	130.8	448.5	**2.00**×
AIME	Qwen3-4B	– / 8	87.8	48.5	39.4	385.9	**4.40**×
	Qwen3-8B	– / 4	45.0	23.9	17.9	165.8	**3.68**×
	Llama-3.2-3B	– / 8	227.2	202.3	111.2	467.1	**2.06**×

4.2 End-to-end throughput

Across the multi-request settings in Table 1, StrataSpec outperforms all three baselines, improving over KnapSpec by 1.73–4.40× and by as much as 9.79× over SWIFT. The concurrency curves in Figure 4 show how this advantage develops on HumanEval.

As the batch grows, independent routes fill one another's skips and approach a dense proposal pass. In StrataSpec, a new request adds rows along the active frontier; the weight footprint grows only when an assignment crosses the richest assigned tier. Most arrivals therefore reuse active weights, so their marginal cost remains small as concurrency rises. Independent routes instead keep enlarging the union even when each request skips many sub-layers, and regrouping after selection cannot recover sparsity once that union is fixed. The consistent separation across all three backbones identifies batch composition as the source of the larger-batch gain.

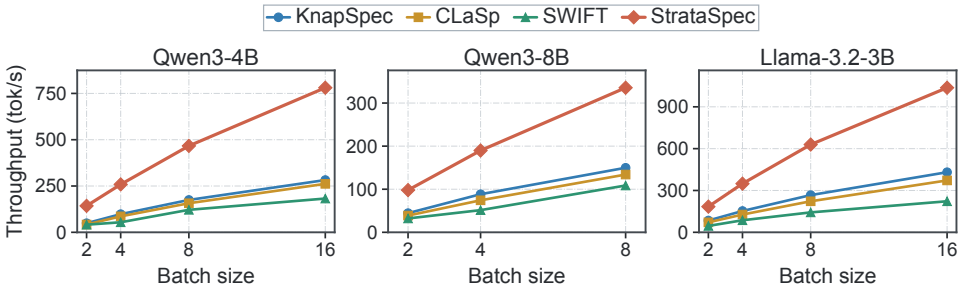

Figure 4: End-to-end throughput on HumanEval. StrataSpec continues to scale with batch size because new requests reuse the route frontier already active in the proposal pass.

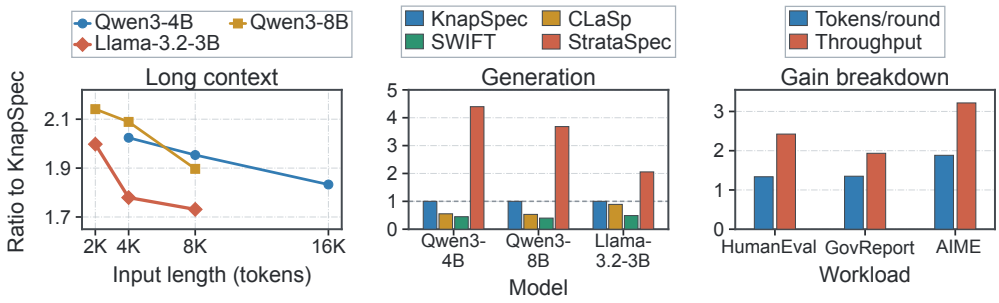

Figure 5: Performance across context and generation regimes, normalized to KnapSpec (1.0). The right panel separates the gain in committed tokens per round from the larger throughput gain enabled by a smaller route union and contiguous-suffix execution.

4.3 Workload and Online Performance

Workload shape changes the balance between shared weights and row-dependent attention (Figure 5). On GovReport, longer inputs shift more proposal time into attention, narrowing but not erasing the gain. Routes with similar skip counts can still carry different amounts of KV-cache work, which the batch cost model captures but standalone route latency does not.

Long generations expose a different source of reuse. On AIME, richer tiers go to requests that convert additional proposal work into accepted tokens, while easier spans remain on the shared lower tier. Because nesting preserves their common lower tiers, this split can persist across rounds without expanding a new independent-route union at every step.

The right panel separates proposal quality from execution efficiency. Relative to KnapSpec, committed tokens per round improve by 1.34–1.88×, whereas throughput improves by 1.93–3.22×. Better tier choices increase accepted progress, and the larger throughput gain comes from executing those choices through a smaller active union and a regular row layout.

Under continuous arrivals, proposal-round length also controls when the next request can enter service. A long round delays both the requests being decoded and those waiting in the queue. Figure 6 follows this interaction as the offered load increases.

At low load, service is arrival-limited and all methods complete most requests within the SLO. Once the GPU is saturated, complementary baseline routes lengthen the proposal pass and postpone later admissions. The delay then compounds over successive decode rounds. At the highest Poisson load, StrataSpec retains 98.7% SLO attainment while reducing P95 queueing delay by more than two orders of magnitude relative to KnapSpec at the same load.

As the queues grow, baseline throughput no longer translates into timely completions. StrataSpec instead keeps P95 TPOT nearly flat because mixed requests reuse the active frontier and avoid forming a different irregular route composition each round. Table 2 shows the same pattern under a burst trace.

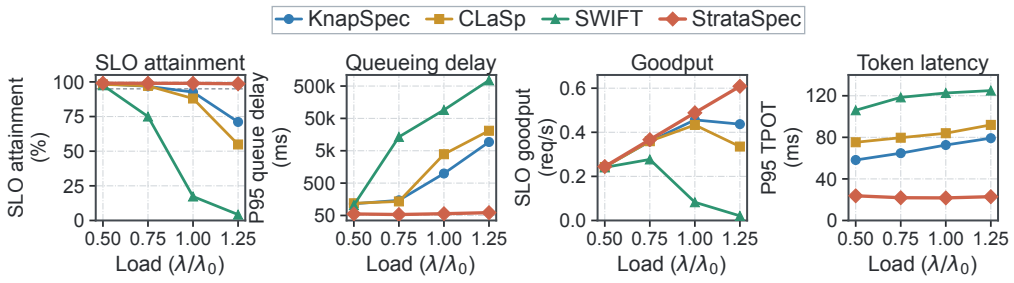

Figure 6: Online performance under increasing offered load. Predictable proposal work prevents queue buildup, preserving goodput and SLO attainment as adaptive baselines saturate.

Table 2: Continuous-arrival performance. Each entry pairs SLO attainment (%) with P95 queueing delay (ms) at one load.

Load (λ/λ_0)	KnapSpec	CLaSp	SWIFT	StrataSpec
0.50	98.6 / 115	98.2 / 119	97.6 / 104	**99.1 / 56**
0.75	96.8 / 147	97.2 / 136	75.0 / 13,377	**99.0 / 53**
1.00	92.7 / 990	88.0 / 3,907	17.3 / 91,198	**99.0 / 57**
1.25	71.1 / 9,319	54.9 / 20,792	4.3 / 749,584	**98.7 / 61**
Burst	98.5 / 129	98.6 / 104	77.6 / 12,193	**99.3 / 50**

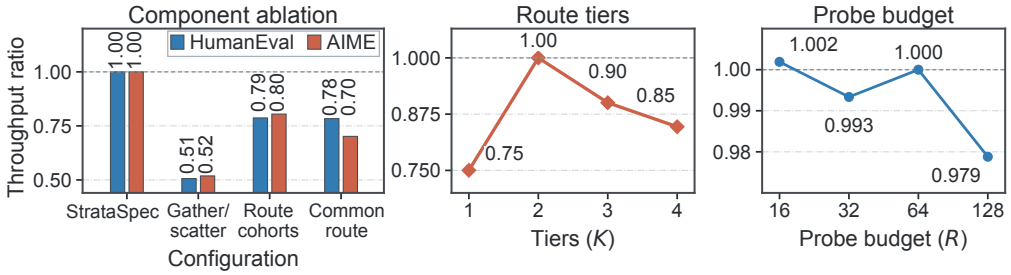

Figure 7: Throughput falls without suffix execution or request-adaptive tiers. Two tiers perform best, while larger probe budgets change little.

StrataSpec absorbs the arrival spike without leaving a persistent queue and maintains 99.3% SLO attainment throughout the burst trace.

4.4 Component Analysis

End-to-end gains depend on both route composition and row layout. Table 3 separates these effects, while Figure 7 shows how the routing and execution choices interact end to end. Fixed-route results expose the MLP work created by route composition; controlled decoding holds tier choices fixed and changes only how active rows reach the kernels.

The fixed-route microbenchmark isolates the work determined by route composition. Nesting reduces MLP latency by 1.91–2.01×, while the two KnapSpec layouts have nearly identical MLP cost because both inherit the same expanded union. Route cohorts repeat weight reads, and gather/scatter changes how rows reach the kernels, but neither changes which sub-layers are active. The nested frontier removes this extra work before layout becomes a concern.

The controlled-decoding columns isolate the remaining layout cost. For fixed tier assignments, contiguous suffixes improve throughput by 44–52% over gather/scatter. A tier boundary selects the active rows while hidden states, page tables, and attention metadata stay aligned, preserving the savings from the smaller union.

Table 3: Nested routes reduce fixed-route MLP latency, while contiguous-suffix execution improves throughput and proposal latency in controlled decoding.

Configuration	Fixed routes		Controlled decoding			
	MLP latency (ms)		Throughput (tok/s)		Proposal latency (ms/tok)	
	$B = 4$	$B = 16$	$B = 4$	$B = 16$	$B = 4$	$B = 16$
KnapSpec (route cohorts)	6.698	7.031	62.5	57.7	14.24	16.52
KnapSpec (gather/scatter)	6.655	7.068	87.9	200.3	9.55	4.20
KnapSpec	–	–	82.1	196.5	8.09	3.21
StrataSpec (gather/scatter)	–	–	66.0	175.3	12.31	4.35
StrataSpec	**3.480**	**3.504**	**94.8**	**266.8**	**6.85**	**2.38**

Gather/scatter retains the tier choices but loses their shared row order, leaving about half of StrataSpec's throughput. Route cohorts preserve contiguous rows, yet give up roughly one fifth by repeating shared work. A common route avoids these layout penalties but loses 22–30% because one route for all requests cannot follow request difficulty.

Two tiers capture most of this variation: easy spans use a lighter route, while harder spans receive a richer proposal without imposing that work on every request. Nesting confines the richer tier's incremental sub-layers to the rows that need them. Additional tiers mostly divide requests with similar acceptance, adding active rows for little committed progress; the probe sweep likewise changes little once replay identifies this coarse separation.

5 Related Work

5.1 Speculative decoding

Speculative decoding amortizes verification across proposals. Candidate generation uses prediction heads, parallel streams, target features, or trees (Cai et al., 2024; Bhendawade et al., 2025; Li et al., 2024a; 2025; 2024b; Miao et al., 2024; Chen et al., 2024), while TETRIS schedules verification (Wu et al., 2025). SSD routes range from fixed paths (Elhoushi et al., 2024; Zhang et al., 2024) to adaptive or model-specific designs (Xia et al., 2025; Chen et al., 2025; Cha et al., 2026; Kim et al., 2025; Song et al., 2026). These methods choose routes per request. StrataSpec also chooses routes so that requests can share that computation in one proposal pass.

5.2 Adaptive execution

Input-dependent depth and width reduce per-input work, but divergent paths complicate batching (Graves, 2016; Teerapittayanon et al., 2016; Wang et al., 2018; Xin et al., 2020; Hou et al., 2020). BatchCond and DREX regroup paths after selection (Selvam et al., 2024; Liu et al., 2025), restoring contiguous rows but not shrinking the union. Regrouping repairs row contiguity only after the active union has been formed, leaving route composition outside the execution objective. After regrouping, selection and execution remain separate. FASER coordinates speculative phases (Chen et al., 2026b), HeraSys schedules reusable workflow nodes (Li et al., 2026), and online SSD adapts the draft model (Liu et al., 2024). StrataSpec structures routes before assignment, so containment controls the union and the row layout within one proposal pass.

5.3 Resource-efficient LLM serving

Autoregressive generation (Brown et al., 2020; Kaplan et al., 2020; Hoffmann et al., 2022; Touvron et al., 2023; Vaswani et al., 2017) has prompted attention and KV-cache optimization (Dao et al., 2022; Dao, 2024; Kwon et al., 2023; Prabhu et al., 2025; Aminabadi et al., 2022), request scheduling (Yu et al., 2022; Zheng et al., 2024; Agrawal et al., 2024), phase disaggregation (Zhong et al., 2024; Patel et al., 2024), execution and placement (Zhu et al., 2025; Fu et al., 2024; Borzunov et al., 2023), and memory-efficient execution (Sheng et al., 2023; Alizadeh et al., 2024; Song et al., 2024; Xue et al., 2024; Sadhukhan et al., 2025; Tiwari et al., 2025). Edge systems combine resource-aware placement and batching (Xu et al., 2025; Choi et al., 2026; Park et al., 2025; Chen et al., 2026a); StrataSpec composes request-dependent draft work inside one proposal pass, where its nested frontier fixes both active weights and row order.

6 Conclusion

StrataSpec makes request-adaptive SSD compatible with regular batch execution by organizing routes as a nested family. The richest assigned tier bounds the active union, while a shared suffix order keeps each sub-layer's rows contiguous. The batch planner prices assignments by marginal work and refreshes the frontier more slowly than it updates tiers, turning route savings into higher throughput across fixed batches. Under continuous arrivals, shorter, more regular proposal rounds prevent queueing delay from compounding and keep SLO attainment near 99%. By making route composition part of the serving policy, StrataSpec preserves request-level adaptation together with the reuse needed for resource-constrained serving.

AI USE STATEMENT

In this work, generative AI tools assisted with code generation and review, mathematical derivations and proof development, literature search, and suggestions for improving the clarity of the manuscript. The authors reviewed and verified all AI-assisted material and remain responsible for the final manuscript, source code, and reported results.

ETHICS STATEMENT

This work evaluates LLM serving infrastructure using publicly available models, benchmarks, and conversation traces. It involves neither human participants nor private user data.

REPRODUCIBILITY STATEMENT

Code for reproducing the reported results is available in the anonymous repository.

REFERENCES

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, 2024.

Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, Karen Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. LLM in a flash: Efficient large language model inference with limited memory. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2024.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and Yuxiong He. DeepSpeed-Inference: Enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022.

Nikhil Bhendawade, Irina Belousova, Qichen Fu, Henry Mason, Antonie Lin, Mohammad Rastegari, and Mahyar Najibi. Speculative Streaming: Efficient and scalable speculative decoding with multi-stream attention. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2025.

Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Maksim Riabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and Colin Raffel. Petals: Collaborative inference and fine-tuning of large models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Association for Computational Linguistics, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2024.

Seongjin Cha, Gyuwan Kim, Dongsu Han, Tao Yang, and Insu Han. KnapSpec: Self-speculative decoding via adaptive layer selection as a knapsack problem. In *Proceedings of the 43rd International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2026.

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.

Hao Mark Chen, Zhiwen Mo, Guanxi Lu, Shuang Liang, Lingxiao Ma, Wayne Luk, and Hongxiang Fan. FastTTS: Accelerating test-time scaling for edge LLM reasoning. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. Association for Computing Machinery, 2026a.

Longze Chen, Renke Shan, Huiming Wang, Lu Wang, Ziqiang Liu, Run Luo, Jiawei Wang, Hamid Alinejad-Rokny, and Min Yang. CLaSp: In-context layer skip for self-speculative decoding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, 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.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2024.

Soongyu Choi, Yuntae Kim, Muyoung Son, and Joo-Young Kim. Cassandra: Enabling reasoning LLMs at edge via self-speculative decoding. In *2026 ACM/IEEE 53rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2026.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, 2024.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2022.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2024.

Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. ServerlessLLM: Low-Latency serverless inference for large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karén Simonyan, Erich Elsen, Oriol Vinyals, Jack Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2022.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. DynaBERT: Dynamic BERT with adaptive width and depth. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2021.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Juhyeong Kim, Sangyeon Yu, Gyunyeop Kim, and Sangwoo Kang. FractalLLM: Lossless self-speculative decoding with layer embedded self-compression. In *Findings of the Association for Computational Linguistics: EMNLP 2025*. Association for Computational Linguistics, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23. Association for Computing Machinery, 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2023.

Size Li, Zhiqing Tang, Hongrui Liang, Jianxiong Guo, Jiong Lou, Tian Wang, and Weijia Jia. HeraSys: Collaborative serving of multiple LLM workflows via fine-grained end-to-end optimization. In *Proceedings of the 43rd International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2026.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2024b.

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*. Curran Associates, Inc., 2025.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In *Proceedings of the 41st International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2024.

Xuting Liu, Daniel Alexander, Siva Kesava Reddy Kakarla, Behnaz Arzani, and Vincent Liu. Dynamic rebatching for efficient early-exit inference with DREX. *arXiv preprint arXiv:2512.15705*, 2025.

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, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24. Association for Computing Machinery, 2024.

Jinwoo Park, Seunggeun Cho, and Dongsu Han. SpecEdge: Scalable edge-assisted serving framework for interactive LLMs. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2025.

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

Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. vAttention: Dynamic memory management for serving LLMs without PagedAttention. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '25. Association for Computing Machinery, 2025.

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian En-Hsu Yen, Avner May, Tianqi Chen, and Beidi Chen. MagicDec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In *International Conference on Learning Representations*, 2025.

Surya Selvam, Amrit Nagarajan, and Anand Raghunathan. Efficient batched inference in conditional neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Re, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2023.

Mingbo Song, Heming Xia, Jun Zhang, Chak Tou Leong, Qiancheng Xu, Wenjie Li, and Sujian Li. KNN-SSD: Enabling dynamic self-speculative decoding via nearest neighbor layer set optimization. In *Findings of the Association for Computational Linguistics: EACL 2026*. Association for Computational Linguistics, 2026.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. PowerInfer: Fast large language model serving with a consumer-grade GPU. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, SOSP '24. Association for Computing Machinery, 2024.

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

Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016.

Rishabh Tiwari, Haocheng Xi, Aditya Tomar, Coleman Richard Charles Hooper, Sehoon Kim, Maxwell Horton, Mahyar Najibi, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. QuantSpec: Self-speculative decoding with hierarchical quantized KV cache. In *Proceedings of the 42nd International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2025.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. SkipNet: Learning dynamic routing in convolutional networks. In *European Conference on Computer Vision*. Springer International Publishing, 2018.

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 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2025.

Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. SWIFT: On-the-fly self-speculative decoding for LLM inference acceleration. In *International Conference on Learning Representations*, 2025.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020.

Daliang Xu, Wangsong Yin, Hao Zhang, Xin Jin, Ying Zhang, Shiyun Wei, Mengwei Xu, and Xuanzhe Liu. EdgeLLM: Fast on-device LLM inference with speculative decoding. *IEEE Transactions on Mobile Computing*, 2025.

Zhenliang Xue, Yixin Song, Zeyu Mi, Xinrui Zheng, Yubin Xia, and Haibo Chen. PowerInfer-2: Fast large language model inference on a smartphone. *arXiv preprint arXiv:2406.06282*, 2024.

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

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, 2022.

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

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2024.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, 2024.

Kan Zhu, Yufei Gao, Yilong Zhao, Liangyu Zhao, Gefei Zuo, Yile Gu, Dedong Xie, Zihao Ye, Keisuke Kamahori, Chien-Yu Lin, Ziren Wang, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. NanoFlow: Towards optimal large language model serving throughput. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*. USENIX Association, 2025.

A NESTED-ROUTE COMPOSITION

A nested route family bounds the active route union and induces one shared suffix order. These properties connect the route construction to the planner's progress objective, row layout, and control overhead.

A.1 MARGINAL COST IN A SHARED BATCH

For a route union U, write its unique weight footprint as

$$W(U) = \sum_{m \in U} w_m,$$

where w_m is the weight size of sub-layer m. Adding a request with route R to an existing batch activates the weights outside its current union:

$$W(U \cup R) - W(U) = \sum_{m \in R \setminus U} w_m. \tag{6}$$

If $U \subseteq V$, then

$$W(U \cup R) - W(U) \;\geq\; W(V \cup R) - W(V),$$

because $R \setminus V \subseteq R \setminus U$. The first request activating a sub-layer introduces its weights; subsequent requests reuse that activation. The same request route has different marginal costs in different batches.

Timing also depends on the rows that reuse those weights. If request i contributes Q_i KV-cache pages and activates a previously unused sub-layer, the planner evaluates $\tau_m(1, Q_i)$. If the sub-layer is already active, it evaluates

$$\tau_m(n+1, Q+Q_i) - \tau_m(n, Q).$$

The latency function $\tau_m(n, Q)$ in Equation 1 captures both cases, including shared activation, active rows, and KV-cache pages when the planner compares request tiers.

A.2 UNION AND SUFFIX STRUCTURE

Active route union. Nesting gives exact control of the activated weight union. For a nonempty batch assigned to routes $R^{(1)} \subseteq \cdots \subseteq R^{(K)}$, let $z_{\max} = \max_{i \in \mathcal{B}} z_i$. Then

$$\bigcup_{i \in \mathcal{B}} R^{(z_i)} = R^{(z_{\max})}. \tag{7}$$

All assigned routes lie in $R^{(z_{\max})}$, which is itself assigned to at least one request, so equality follows.

For any route collection, the union footprint is at least that of its largest member:

$$W\left(\bigcup_i R_i\right) \geq \max_i W(R_i).$$

A nested family attains equality. Adding requests at existing tiers changes only row-dependent work; activating a richer tier introduces exactly its difference from the previous richest route.

The nested chain also exposes the weight footprint shared across tiers. Let $R^{(0)} = \varnothing$, define the incremental tier footprint $d_k = W(R^{(k)}) - W(R^{(k-1)})$, and let $N_k = |\{i : z_i \geq k\}|$. For $z_{\max} = \max_i z_i$, the isolated execution footprint and the nested-batch footprint are

$$\sum_i W(R^{(z_i)}) = \sum_{k=1}^{z_{\max}} N_k d_k, \qquad W\left(\bigcup_i R^{(z_i)}\right) = \sum_{k=1}^{z_{\max}} d_k. \tag{8}$$

Their difference,

$$\sum_{k=1}^{z_{\max}} (N_k - 1) d_k,$$

is the weight footprint repeated by isolated execution. Sharing grows with the number of requests reaching a tier and the size of its increment.

Shared suffix order. Nesting also gives a request order shared by every sub-layer. Let $\pi(j)$ map sorted position $j \in \{0, \dots, B-1\}$ to a request, with $z_{\pi(j)}$ nondecreasing. Define

$$\kappa_m = \min\{k : m \in R^{(k)}\}, \qquad o_k = |\{i : z_i < k\}|,$$

using $\kappa_m = K + 1$ for a sub-layer absent from the frontier. The active requests for sub-layer m are exactly

$$\mathcal{A}_m = \{\pi(j) : o_{\kappa_m} \le j < B\}. \tag{9}$$

Nesting gives $m \in R^{(z_i)}$ exactly when $z_i \ge \kappa_m$. Sorted order places the o_{κ_m} lower-tier requests first, leaving the stated suffix. For $\kappa_m = K + 1$, $o_{\kappa_m} = B$ and the suffix is empty.

The same permutation works for every sub-layer. Assigned tiers $(3, 1, 2, 1)$, for example, become $(1, 1, 2, 3)$, and thresholds one, two, and three start at offsets zero, two, and three. Thresholds may vary with model depth; adjacent sub-layers sharing a threshold reuse one view.

This ordering property also characterizes a nested family. If one order makes every active set a suffix, then for positions $u < v$, every sub-layer in $R_{\pi(u)}$ is active for $\pi(v)$. Hence

$$R_{\pi(0)} \subseteq R_{\pi(1)} \subseteq \cdots \subseteq R_{\pi(B-1)}. \tag{10}$$

Equation 9 and this reverse implication establish the equivalence between nesting and a common suffix order. The route family that bounds the union also supplies its row layout.

Layout cost. The shared order also minimizes the number of active-row fragments. For a permutation π, let $r_m(\pi)$ be the number of contiguous active-row runs for sub-layer m. If $\mathcal{A}_m$ is nonempty, every permutation has $r_m(\pi) \ge 1$; if it is empty, $r_m(\pi) = 0$. The shared-suffix result makes every active set a suffix under π_{tier}, giving

$$\sum_{m=1}^{M} r_m(\pi_{\text{tier}}) = |\{m : \mathcal{A}_m \neq \varnothing\}| = \min_{\pi} \sum_{m=1}^{M} r_m(\pi). \tag{11}$$

Every nonempty $\mathcal{A}_m$ requires at least one run, and tier sorting makes it a single block. The global order attains all per-sub-layer lower bounds. Each active sub-layer receives one invocation per draft position with cost $\tau_m(n, Q)$.

Let d be hidden width and J the number of maximal equal-threshold segments. Sorting tiers and finding segments costs $O(B \log B + M)$. Across γ positions, row permutations move $O(\gamma B d)$ elements and update $O(\gamma J)$ segment boundaries; these measurements form the layout term.

A.3 Committed progress from acceptance

Route structure determines batch cost, while acceptance determines how much useful progress that cost buys. Let A_j be the event that the first j draft tokens are accepted, with A_0 certain. In a complete speculative round, the accepted prefix contributes one token for each surviving position, and the target contributes a correction or bonus token. Hence

$$G = 1 + \sum_{j=1}^{\gamma} \mathbf{1}\{A_j\}, \qquad \mathbb{E}[G] = \sum_{j=0}^{\gamma} \Pr(A_j). \tag{12}$$

Define $a_j = \Pr(A_j \mid A_{j-1})$. The chain rule gives

$$\mathbb{E}[G] = 1 + \sum_{j=1}^{\gamma} \prod_{\ell=1}^{j} a_\ell. \tag{13}$$

For planning, we replace the position-specific a_j values with the route's current acceptance estimate a:

$$G(a, \gamma) = \sum_{j=0}^{\gamma} a^j = \begin{cases} (1 - a^{\gamma+1})/(1 - a), & a < 1, \\ \gamma + 1, & a = 1. \end{cases} \tag{14}$$

Increasing draft length from γ to $\gamma + 1$ adds expected progress $a^{\gamma+1}$. The planner compares this decreasing marginal gain with the additional computation. Probe top-1 agreement supplies a for greedy decoding; accepted probability mass supplies it for sampling.

Choosing draft length. For the current assignment $\mathbf{z}$, let G_γ and C_γ denote the numerator and denominator of Equation 5, and write $F_\gamma = G_\gamma / C_\gamma$. Extending the draft by one position adds

$$\Delta G_\gamma = \sum_{i \in \mathcal{B}} \hat{a}_{i,z_i}^{\gamma+1}, \qquad \Delta C_\gamma = C_{\gamma+1} - C_\gamma. \tag{15}$$

Comparing the two token rates gives

$$\frac{G_\gamma + \Delta G_\gamma}{C_\gamma + \Delta C_\gamma} > \frac{G_\gamma}{C_\gamma} \quad \Longleftrightarrow \quad \Delta G_\gamma > F_\gamma \Delta C_\gamma. \tag{16}$$

The extra position is worthwhile when its committed progress exceeds the work it adds at the current batch rate. This work includes the extra draft step and the change in verification cost. As acceptance declines, the marginal progress decays more quickly and favors shorter drafts. Sharing more of the added computation lowers its cost and makes a longer draft more attractive. The planner evaluates tier assignments for each candidate length so that these decisions use the same batch objective.

Acceptance from probe replay. Condition on a request prefix, and let p and q be the target and draft distributions after applying the generation policy. A proposal $x \sim q$ is accepted with probability $\min(1, p(x)/q(x))$. Summing over proposals,

$$A(p, q) = \sum_{x : q(x) > 0} q(x) \min(1, p(x)/q(x)) = \sum_x \min(p(x), q(x)). \tag{17}$$

This is the quantity averaged over probe positions in Equation 3. It also satisfies $A(p, q) = 1 - \frac{1}{2} \sum_x |p(x) - q(x)|$, connecting acceptance to distribution overlap under speculative rejection sampling (Leviathan et al., 2023; Chen et al., 2023).

A.4 Batch planning

Every sub-layer belongs to one frontier threshold, so a tier move can be priced by updating only the thresholds it crosses. The planner can therefore compare the resulting cost change with the progress estimate above without recomputing the full batch.

Suffix cost statistics. Sub-layers with the same activation threshold process the same requests. Let $Q_{i,s}$ be the KV-cache page count of request i at draft position s. For each threshold k, define

$$N_k = \sum_i \mathbf{1}\{z_i \geq k\}, \qquad V_{k,s} = \sum_{i : z_i \geq k} Q_{i,s}, \qquad D_k(n, Q) = \sum_{m : \kappa_m = k} \tau_m(n, Q). \tag{18}$$

Reindexing the sub-layer sum in Equation 1 yields

$$C_{\text{draft}}(\mathbf{z}, \gamma) = C_{\text{base}}(B, \gamma) + \sum_{s=1}^{\gamma} \sum_{k=1}^{K} D_k(N_k, V_{k,s}). \tag{19}$$

The planner prices all sub-layers from K suffix counts and page totals. Requests at a tier share these statistics even when the sub-layers they activate are separated in model depth. Cached values of D_k are reused as the search visits assignments with the same active-row and KV-cache page totals.

Marginal tier cost. For fixed routes and draft length, write the planner objective as $F(\mathbf{z}) = G(\mathbf{z})/C(\mathbf{z})$, where $G(\mathbf{z}) = \sum_i G_{i,z_i}(\gamma)$ and C is the denominator of Equation 5. A candidate move changes request i from tier k to k'. Define

$$\delta_{i,m} = \mathbf{1}\{k' \geq \kappa_m\} - \mathbf{1}\{k \geq \kappa_m\}.$$

The updated row and KV-cache page counts at position s are

$$n'_{m,s} = n_{m,s} + \delta_{i,m}, \qquad Q'_{m,s} = Q_{m,s} + \delta_{i,m} Q_{i,s},$$

where $Q_{i,s}$ is the page count contributed by request i. Only thresholds between the two tiers change. The draft-time increment is

$$\Delta C_{\text{draft}} = \sum_{s=1}^{\gamma} \sum_{m : \delta_{i,m} \neq 0} \left[\tau_m(n'_{m,s}, Q'_{m,s}) - \tau_m(n_{m,s}, Q_{m,s}) \right]. \tag{20}$$

Algorithm 1 Joint tier and draft-length selection.

Require: Acceptance scores $\hat{a}$, round-cost function C, lengths Γ, starting assignments $\{\mathcal{S}_\gamma\}$, scan budget J, rate fraction β
Ensure: Request tiers $\mathbf{z}^*$ and draft length γ^*

```
for each γ ∈ Γ do
    Compute G_{i,k}(γ); initialize f_γ ← −∞.
    for each starting assignment z^(0) ∈ S_γ do
        Set z ← z^(0) and initialize suffix counts and page totals.
        for t = 1, ..., J do
            Set z_old ← z.
            for each request i ∈ B do
                Evaluate alternative tiers using Equation 20.
                Commit the highest-rate move satisfying Equation 21, if any.
                Update the affected suffix counts, page totals, and batch progress.
            end for
            if z = z_old then
                break
            end if
        end for
        if F_γ(z) > f_γ then
            Store z_γ ← z and f_γ ← F_γ(z).
        end if
    end for
end for
Select γ* and z* by Equation 22.
return (z*, γ*)
```

Batch size and draft length stay fixed during the move, so C_{base} and verification cost are unchanged. Assignment-dependent control cost is included in the total increment ΔC.

Cross-multiplying the positive round costs in $(G + \Delta G)/(C + \Delta C) > G/C$ gives the improving-move condition

$$\Delta G > F(\mathbf{z})\Delta C, \qquad \Delta G = G_{i,k'}(\gamma) - G_{i,k}(\gamma). \tag{21}$$

The condition prices extra progress at the current batch rate. A richer tier is useful when its extra committed progress exceeds the time it adds at that rate; a cheaper tier is useful when its time saving compensates for the progress it removes. The same calculation applies when comparing neighboring draft lengths, with draft and verification costs both updated.

Coordinate ascent. For each draft length, Algorithm 1 evaluates a set of initial assignments $\mathcal{S}_\gamma$, writing the rate as $F_\gamma(\mathbf{z})$. Full updates start from both the request-local best tiers and the best common tier; routine updates start from the current assignment. Each accepted move updates the suffix statistics between its old and new tiers, so subsequent moves see the work already active in the batch.

Coordinate ascent returns the best assignment $\mathbf{z}_\gamma$ and rate f_γ for each draft length. The planner then selects the shortest draft whose rate reaches a fraction β of the best candidate rate:

$$\gamma^* = \min\{\gamma \in \Gamma : f_\gamma \geq \beta \max_{\gamma' \in \Gamma} f_{\gamma'}\}, \qquad \mathbf{z}^* = \mathbf{z}_{\gamma^*}. \tag{22}$$

When candidate rates are close, the controller favors a shorter round and an earlier opportunity to update the batch.

Every accepted move strictly increases $F(\mathbf{z})$, so the search cannot revisit an assignment. A scan with no accepted move evaluates all single-request alternatives at the same state and identifies a coordinate-wise optimum. Each scan examines at most BK moves, each touching at most $M\gamma$ latency entries, for a cost of $O(BKM\gamma)$.

A.5 Amortizing frontier refresh

Frontier construction changes the route family and runs less often than tier assignment. Let T_{refresh} be the measured cost of candidate synthesis and replay, and let H be the mean committed tokens

Algorithm 2 Online tier selection and suffix execution.

Require: Active requests $\mathcal{B}$, nested frontier $\{R^{(k)}\}_{k=1}^{K}$, latency profiles τ, draft lengths Γ
Ensure: Committed output tokens and updated request states
while the serving batch is nonempty **do**
  Update batch membership, KV-cache page counts, and acceptance statistics.
  if frontier refresh is due and verification probes are available **then**
    Balance probes; refresh the frontier and its tier utilities.
  end if
  if initial round or tier reassignment is due **then**
    Initialize from the current plan; add request-local and common-tier starts for a full update.
    Select request tiers and draft length with Algorithm 1.
    Compute the stable tier order π and suffix offsets.
  end if
  for $s = 1, \ldots, \gamma$ **do**
    Permute hidden states and request metadata by π.
    Execute equal-threshold segments on their active suffixes; bypass inactive rows.
    Apply π^{-1}, then the LM head, and select the next draft token.
  end for
  Verify the draft, commit accepted output, and record acceptance feedback.
end while

per request between refreshes and $\overline{G} = B^{-1} \sum_i G_{i,z_i}(\gamma)$ the predicted mean progress per round. A refresh is amortized over approximately $N_H = H/\overline{G}$ rounds. With per-round assignment cost T_{assign} and layout cost $T_{\text{layout}}(\gamma)$,

$$\overline{C}_{\text{ctrl}} = T_{\text{assign}} + T_{\text{layout}}(\gamma) + \frac{T_{\text{refresh}}\overline{G}}{H}. \tag{23}$$

The layout term includes every draft-position permutation and inverse permutation. If their measured combined cost per position is T_{perm}, then $T_{\text{layout}}(\gamma) = \gamma T_{\text{perm}}$, with plan construction charged when the assignment changes.

For planning fraction ρ and measured round time T_r, the refresh budget gives

$$H_{\text{plan}} = \left\lceil \frac{T_{\text{refresh}}\overline{G}}{\rho T_r} \right\rceil. \tag{24}$$

The controller updates this interval from observed refresh time and committed progress. More expensive refreshes are amortized over longer intervals, while tier assignment responds to the current batch between refreshes.

A.6 Online control

StrataSpec updates its plan after verification commits the request state. Algorithm 2 combines frontier refresh, tier assignment, and suffix execution around that boundary.

B Frontier construction and adaptation

Full verification produces both the states used to build the frontier and the feedback used to select among its tiers. Frontier construction revises the route family; between those revisions, verification feedback updates the utilities of its existing tiers.

B.1 Probe-balanced route construction

Probe scoring. Each request contributes a recent contiguous span of committed positions. The probe budget is divided across requests before candidate routes are scored, so the frontier reflects the batch rather than the longest context. For probe set P_i from request i, the candidate score is

$$S_{\text{batch}} = \frac{1}{|\mathcal{B}_P|} \sum_{i \in \mathcal{B}_P} \frac{1}{|P_i|} \sum_{p \in P_i} s_{i,p},$$

where $\mathcal{B}_P$ contains requests with available probes and $s_{i,p}$ is the candidate's state-similarity score at that position. Averaging within each request and then across requests gives every contributing request equal weight.

The reference states come from the verifier's attention and MLP residual outputs; its logits supply the target distribution for acceptance scoring.

Shared-prefix replay. Candidate routes share the committed prefix before each probe span. For attention replay, prefix entries are read from the existing paged KV cache, while each candidate forms its own KV entries for the short span. Let o_P, o_S be the attention outputs over the prefix and the causally visible span, and let ℓ_P, ℓ_S be their log-normalizers. Their combined output is

$$o = e^{\ell_P - \ell} o_P + e^{\ell_S - \ell} o_S, \qquad \ell = \log(e^{\ell_P} + e^{\ell_S}). \tag{25}$$

The two terms partition the keys of the same softmax, so the merged output equals attention over their concatenation. Candidate branches reuse the long prefix and allocate candidate-specific KV entries for the replay span.

Candidate search. Route construction follows the attention/MLP-separated dynamic program of KnapSpec (Cha et al., 2026). Let H^*_m be the verified hidden states after sub-layer m, and let $H_{m,b}$ be the retained candidate states after b skips among the first m sub-layers. Each state contains the batch's probe positions. With unit skip costs, a reachable budget b has two possible transitions:

$$H^{\text{exec}}_{m,b} = \mathcal{F}_m(H_{m-1,b}), \qquad H^{\text{skip}}_{m,b} = H_{m-1,b-1}, \tag{26}$$

where $\mathcal{F}_m$ includes the residual update. The execute transition retains the skip budget; the bypass transition adds one skip. Required sub-layers use the execute transition.

Write $S_m(H)$ for the request-balanced cosine similarity between candidate H and H^*_m. For similarity threshold θ, the DP retains from the available transitions $\mathcal{H}_{m,b}$

$$H_{m,b} = \underset{H \in \mathcal{H}_{m,b} : S_m(H) \geq \theta}{\arg\max} S_m(H). \tag{27}$$

Budget zero follows the recorded verifier states, $H_{m,0} = H^*_m$; required boundary sub-layers use the execute transition. Each retained state stores its parent decision, so tracing the final state recovers a route mask for every feasible skip budget. Attention and MLP transitions are scored separately, allowing a route to retain either part of a decoder block according to its contribution on the probes.

Execute and bypass transitions are evaluated together across the reachable skip budgets. With J_b budget states, R probe positions, and hidden width d, two rolling hidden-state buffers and the parent decisions require $O(J_b R d + M J_b)$ storage.

One dynamic-programming pass produces a candidate pool $\mathcal{C}$ across feasible skip budgets. The planner selects evenly spaced budget targets, starting with the route that skips the most sub-layers. Each subsequent tier is the candidate nearest its budget target that contains the preceding tier:

$$\mathcal{R}_k = \{R \in \mathcal{C} : R^{(k-1)} \subseteq R\}.$$

All tiers reuse the same candidate pool. The containment condition preserves nesting at every selection step.

After construction, the planner replays candidate routes on the probe spans to estimate per-request agreement or acceptance mass. It removes duplicate and dominated choices while preserving the chain order.

Latency model. Planning uses a latency profile for each sub-layer, indexed by active rows, aggregate KV-cache pages, draft length, and verification batch size. During planning, the planner interpolates between neighboring entries: attention and verification entries depend on active rows and aggregate context, while MLP entries depend primarily on active rows. Row-permutation cost is included as a separate layout term. Sub-layers with the same activation threshold reuse one set of row and page statistics, so a tier move updates only the affected suffixes.

B.2 FEEDBACK AND STATE TRANSITIONS

A frontier remains active across many speculative rounds. Its tier order and suffix offsets are reused as batch membership changes, while verification feedback updates the request-specific utilities used to select among its tiers.

Acceptance feedback. Replay provides an acceptance profile across tiers. The planner projects these scores onto a nondecreasing sequence and averages them across requests to form priors $\bar{a}_k$. Between refreshes, verification updates a request-specific difficulty offset d_i. With smoothing rate η, if request i accepts s of γ proposed tokens at tier k, its observed prefix acceptance is $\tilde{a}_i = s/(s + \mathbf{1}\{s < \gamma\})$. The update is

$$d_i \leftarrow (1-\eta)d_i + \eta(\tilde{a}_i - \bar{a}_k), \qquad \hat{a}_{i,k'} = \mathrm{clip}_{[0,1]}(\bar{a}_{k'} + d_i). \tag{28}$$

The update keeps the tier gaps calibrated while allowing a request-specific offset to follow the observed prefix acceptance. A verification outcome at one tier therefore informs the request's scores at the other tiers.

Tier reassignment. Tier assignment runs at a verification boundary when membership changes or acceptance feedback calls for replanning. New requests receive an initial tier, while continuing requests carry their previous tiers into the next coordinate ascent. Periodic full updates also consider a common-tier starting point before selecting the next assignment and draft length with Equation 22.

Refresh scheduling. A candidate frontier is evaluated over several speculative rounds using measured tokens per time and acceptance feedback. Refresh intervals grow with committed progress and context length; the next interval is the larger of the context-based interval and the measured planning interval in Equation 24. Longer contexts thus amortize replay, while tier assignment continues at the faster cadence.

KV cache across tier changes. Drafting and verification share parameters but maintain separate KV caches. Prefill populates the committed draft cache. Within a speculative round, fixed tier assignments determine which attention sub-layers append tentative entries. Full verification then commits the accepted prefix and target token and releases the rejected suffix; tier changes take effect only after this commit.

At that boundary, the target cache contains the full committed history. Promoting a request materializes missing blocks for newly active attention sub-layers, while demotion leaves existing blocks unused. Request permutation changes block-table metadata rather than physical KV blocks (Kwon et al., 2023), so unaffected cache state remains in place.