CANDIKV: PRESERVING CANDIDATE KV STATE ACROSS SELECTION AND CONTINUATION IN LLM WORKFLOWS

Anonymous authors
Paper under double-blind review

ABSTRACT

Branching LLM workflows score several candidate histories before continuing from a selected subset. Scoring has already materialized the KV state of every candidate, yet request-scoped serving separates that state from the continuation that may consume it. When selection resolves, the chosen histories return as new requests, and the engine must recover work it has just performed before their first continuation tokens. We introduce CandiKV, which treats selection as a handoff of candidate state and keeps the candidate group live until the choice is known. While selection is pending, incoming scores guide computation and KV residency toward likely continuations. After the choice, selected histories enter generation with their state and progress intact. On six reasoning and retrieval workloads, CandiKV reduces continuation TTFT by up to 90.7%. It also cuts the GPU KV byte-seconds wasted on unselected candidates by 23.7–30.5% compared with keeping every candidate resident, while regular-request TTFT changes by at most 0.33%.

1 INTRODUCTION

Many LLM workflows choose their next context from histories the model has already processed. Retrieval systems score question–passage histories before answering from one passage, and verifier-guided reasoning ranks partial solutions before extending a few paths (Lewis et al., 2020; Yao et al., 2023a). In each case, selection turns a processed candidate into the model state of the next generation stage. That state remains useful after scoring ends.

However, serving engines end each candidate's lifetime with the scoring request and treat its continuation as a new arrival. By the time selection reveals the winner, the branch-specific tail of the chosen history must be located, restored, or rebuilt before generation, even though scoring has just processed it. Recovery competes with unrelated requests, and its latency grows with load. Existing systems shorten terminal scoring (Du et al., 2025), recover matching prefixes (Kwon et al., 2023; Juravsky et al., 2024; Pan et al., 2025), or keep KV blocks outside GPU memory (Liu et al., 2025; Gao et al., 2025; Qin et al., 2026).

Across these approaches, a candidate is produced under one request lifetime and consumed under another (Figure 1). When scoring ends, the application knows that a few candidates will continue, but not which ones. Selection reveals the consumer only after the request that produced its state has ended. Scores nevertheless begin to distinguish likely continuations before selection completes. This early signal gives the engine time to prepare for the eventual winner. Making this preparation useful after selection presents two challenges.

The first challenge is deciding which candidates deserve memory and compute while several remain viable. Retaining every tail consumes KV capacity as the number of candidates grows, while discarding them all sends the winner back through prefill (Kwon et al., 2023; Pan et al., 2025). Computing ahead for every candidate also takes service from regular requests. Scores reorder likely consumers, while tails differ in footprint and saved work. A long tail may be costly to rebuild yet unlikely to survive, whereas a leading branch may be cheap to reconstruct. The useful set changes throughout selection, and neither score nor tail length alone identifies it.

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.

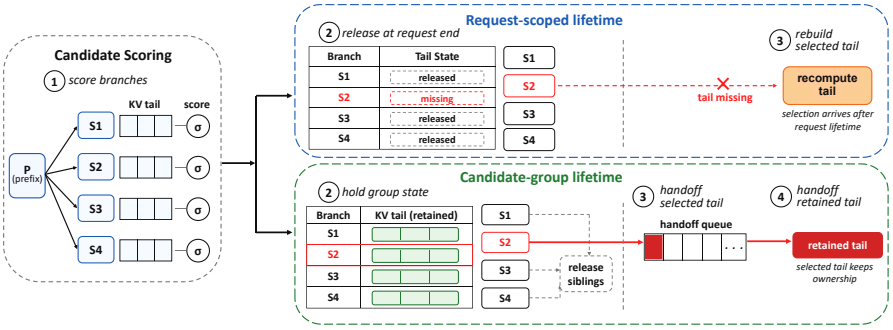


Figure 1: Selection creates a consumer for one of several materialized histories. Request-scoped serving closes the scoring requests before that consumer is known, so the selected tail must be recovered or rebuilt. CandiKV keeps the candidate group live, preserves ownership through selection, and transfers the selected record directly to generation.

The second challenge is carrying the prepared state and work into the continuation. The retained tail and any computation performed before selection must remain attached to the history that the continuation extends. The continuation must start where that history ends and resume from the progress already made. If admitted as a new request, even a ready branch can wait behind unrelated work under conventional request scheduling (Agrawal et al., 2024; Luo et al., 2025; Li et al., 2026). The handoff must transfer state, progress, and admission priority together; otherwise, recovery and queueing again delay the first token of the continuation.

In this paper, we present CandiKV, a selection-aware serving system that holds each candidate group under one lifetime until the choice resolves. For the first challenge, incoming scores guide state placement and model service before selection. For the second, once the winner is known, its branch record carries KV state and execution progress into generation while siblings are released.

This paper makes the following contributions.

- We identify selection as the boundary between materialized branch state and its eventual consumer, where request-scoped serving repeats work that scoring has just done.
- We introduce candidate-group lifetimes and branch records that preserve identity, KV state, and execution progress through selection.
- We develop selection-aware execution, retention, and handoff policies that direct model service and KV capacity toward the branches most likely to continue.
- We implement CandiKV on vLLM. On six reasoning and retrieval workloads, it lowers continuation TTFT by 26.2–90.7% relative to the strongest baseline.

2 Background and Motivation

2.1 Selection-conditioned generation

Selection-conditioned generation appears in both retrieval and reasoning, despite their different application semantics. In passage reranking, each candidate combines a shared question with a different passage. The model processes every question–passage history to score it, but answer generation extends only the selected one (Nogueira et al., 2020; Izacard & Grave, 2021). A verifier-guided reasoner follows the same pattern with partial solutions in place of passages (Cobbe et al., 2021; Yao et al., 2023a). Although the candidates differ, the next stage in both cases consumes the whole chosen history, not only the prefix shared by the group.

We call the interval between producing these candidate histories and resolving the selected set the selection window. Some selectors emit all scores at once, while verifiers can reveal the order incrementally. The application already knows that only a few histories will continue, but not which materialized branches will cross the boundary. The candidate states already exist; selection only decides which ones will serve the next stage.

2.2 Request-scoped KV reuse

This uncertainty becomes a serving problem because each scoring request ends inside the selection window. Scoring has already prefilled the candidate's entire history (Du et al., 2025). Prefix caching can recover context shared by the group, but each branch also has a distinct tail that its continuation needs (Kwon et al., 2023; Juravsky et al., 2024; Pan et al., 2025). Once the scoring request ends, the continuation must either rediscover that tail in the cache or prefill the tokens again before producing its first response token.

Figure 1 captures the mismatch between workflow continuity and request-scoped execution. The scoring request and its continuation name the same history under different request identities. An exact-match cache may retain the tail, and a host tier may keep it available for longer (Liu et al., 2025; Gao et al., 2025; Qin et al., 2026), but both react after the request boundary. Token matching reveals that a history has appeared before; it does not reveal that one member of this candidate group is about to consume its state. Shared-prefix reuse removes work common to the branches; selection provides the missing information by naming the branch-specific tail that will become useful next.

2.3 The value of candidate state

Once candidate identity remains visible, selection supplies a signal absent from reuse policies based on token matches, request pauses, or predicted reuse (Kwon et al., 2023; Yu et al., 2025; Pan et al., 2025). Before the choice, each branch is an option whose value depends on both its chance of continuing and the work its state would save. After the choice, the winner becomes the working context of the next stage and its siblings have completed their role. Incoming scores expose this shift early enough for the engine to respond. Score alone misses the amount of reusable work, while tail length alone misses the likelihood of future use.

Scores can guide computation as well as retention. The engine can advance branches still in contention and reserve KV capacity for histories that are both likely to continue and expensive to rebuild. Both decisions pay off only when the chosen history leaves the group with its state and progress intact while work on its siblings stops. CandiKV captures this interval as a candidate-group lifetime.

3 Related Work

3.1 Branching workflows

Retrieval-augmented generation constructs passage-conditioned histories and chooses among them before answer generation (Lewis et al., 2020; Karpukhin et al., 2020; Izacard et al., 2023; Shi et al., 2024; Asai et al., 2024; Izacard et al., 2022; Khattab & Zaharia, 2020; Santhanam et al., 2022; Formal et al., 2021; Nogueira et al., 2020; Izacard & Grave, 2021; Thakur et al., 2021). Reasoning methods create a similar branch structure through self-consistency or tree search, while ReAct and Toolformer interleave model decisions with external actions (Wang et al., 2023; Yao et al., 2023a;b; Schick et al., 2023). SGLang, Autellix, and HeraSys expose dependencies within and across these applications for program execution and scheduling (Zheng et al., 2024; Luo et al., 2025; Li et al., 2026). These systems organize the workflow graph. CandiKV operates at the transition where selection turns one materialized history into the input of the next generation stage.

3.2 Inference phases

Specialized engines optimize one-token requests whose useful work ends with the score (Du et al., 2025), while pause-aware systems preserve request state across tool calls (Abhyankar et al., 2024). Other serving systems coordinate prefill and decode within or across engines, multiplex models, and optimize execution or memory placement (Yu et al., 2022; Agrawal et al., 2024; Zhong et al., 2024; Patel et al., 2024; Li et al., 2023; Fu et al., 2024; Aminabadi et al., 2022; Sheng et al., 2023; Prabhu et al., 2025; Ye et al., 2025; NVIDIA, 2024; Crankshaw et al., 2017; Gujarati et al., 2020; Romero et al., 2021; Shen et al., 2019). Pause-aware execution preserves a known request across a gap, and phase schedulers act once the next consumer is fixed. During selection, several materialized histories remain possible. CandiKV serves this earlier interval, before the workflow knows which history will enter decode.

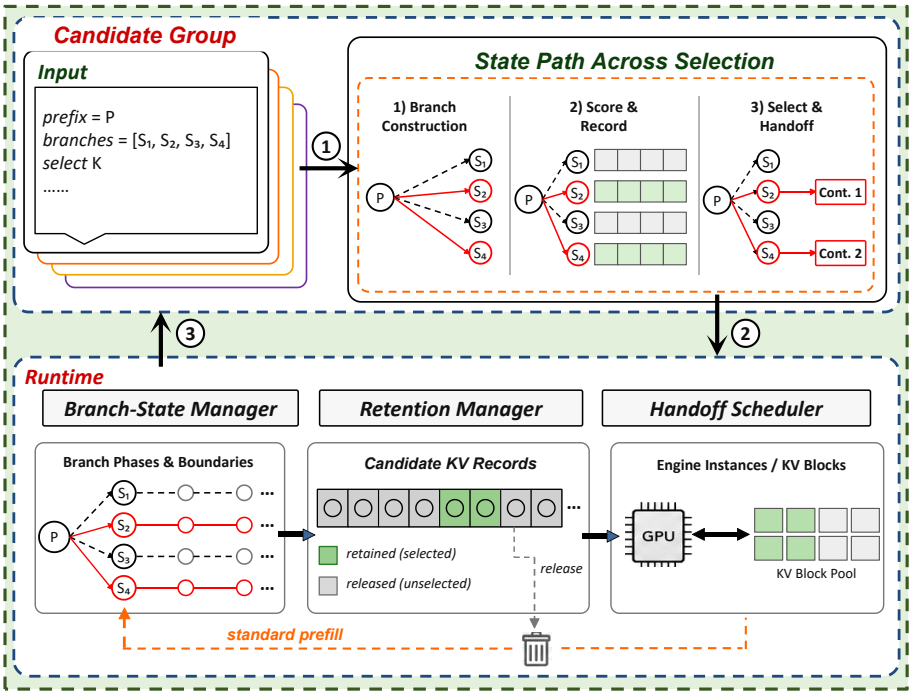


Figure 2: CandiKV keeps candidate histories under group ownership until the choice is resolved. Branch records carry their boundary and progress, allowing selected records to move directly to generation while sibling state is released.

3.3 KV REUSE AND PLACEMENT

PagedAttention and automatic prefix caching reuse exact token histories, while Hydragen batches work over shared prefixes (Kwon et al., 2023; Juravsky et al., 2024). CacheGen, LMCache, and Mooncake move or compress state across tiers, and CacheBlend fuses cached knowledge for RAG (Liu et al., 2024a; 2025; Qin et al., 2026; Yao et al., 2025). Pensieve and HCache restore state across pauses; prefix-cache admission, compression, and eviction methods reduce resident KV footprints (Yu et al., 2025; Gao et al., 2025; Pan et al., 2025; Liu et al., 2024b; Hooper et al., 2024; Li et al., 2024; Cai et al., 2025; Tang et al., 2024). These methods value reuse through token matches, predicted reuse, request pauses, or memory pressure. Selection provides an earlier signal by revealing which of several live histories is becoming the next request. CandiKV uses that signal to guide state placement before the continuation appears.

4 CANDIKV DESIGN

CandiKV groups the histories governed by one selection decision and keeps the group live until the choice resolves (Figure 2). Each branch record carries one history's state and progress until selection transfers the chosen records to generation and releases the rest.

4.1 PRESERVING BRANCH IDENTITY

The branch-state manager represents a candidate group as $G = (P, \{S_i\}_{i=1}^{W}, K, d, \mathcal{B})$, where P is the shared prefix, S_i a candidate tail, K the number selected, d the selection delay, and $\mathcal{B}$ the continuation budget. Candidate i follows history $P \parallel S_i$, and its continuation extends the same materialized history without reconstructing it.

The shared prefix is referenced once by the group, while each branch record names its candidate tail and the continuation boundary at its end. The record also carries placement, execution progress, and phase. The boundary remains fixed as work advances; a separate cursor records progress made

during the selection window. Selection can therefore transfer the record without reconstructing its history or losing completed work. The group remains live until selected records enter continuation and sibling records are released.

4.2 Advancing candidates before selection

The selection window gives the engine time to advance work that would otherwise follow the choice. CandiKV admits a bounded amount of residual prefill and first-token work at background priority before the selected set is known. Unscored branches remain eligible; each score update narrows the working set and stops further service to candidates that can no longer be selected. Work already completed remains attached to the corresponding branch record.

For the score-based top-K selector used by the runtime, let O_t contain the branches scored by time t; for $i \in O_t$, its observed rank within the set is

$$r_t(i) = 1 + \left|\{j \in O_t : (s_j, j) \succ (s_i, i)\}\right|, \tag{1}$$

where $\succ$ orders candidates by selector score and breaks ties by identity. Because O_t only grows, a branch with $r_t(i) > K$ cannot re-enter the final top-K. An external verifier exposes the same order through its branch scores. At the next scheduler event, CandiKV removes a branch that has left contention, so subsequent service follows histories that can still become continuations.

Pre-selection execution follows the standard first-token path through residual prefill and, within the branch budget, the first decode step. Output is exposed only after a branch is selected. A branch that has reached decoding continues from that point; otherwise it resumes from the latest state stored in its branch record.

4.3 Retaining useful candidate state

Not every unresolved tail deserves the same residency. The retention manager ranks candidate state by the continuation work it is expected to remove, so KV capacity follows likely future use rather than score or tail length alone. For a retained placement $a \in \{\mathrm{GPU}, \mathrm{CPU}\}$, it uses the utility

$$U_i(a, t) = p_i(t)\, \Delta_i(a) - C_i(a, t), \tag{2}$$

where $p_i(t)$ estimates the probability that branch i is selected, $\Delta_i(a)$ is the prefill work avoided at placement a, and $C_i(a, t)$ is the cost of footprint, movement, memory pressure, and queueing. The ranking changes with observed rank, score margin, tail coverage, and available capacity.

The product $p_i(t)\Delta_i(a)$ measures the expected work removed from the selected first-token path. A likely branch may have little prefill left, while a long tail may be unlikely to continue; the product distinguishes these cases. The cost term discounts placements whose footprint or movement would displace more useful state. As scores update $p_i(t)$ and execution reduces the work remaining in $\Delta_i(a)$, the ordering follows the work still at risk rather than assigning permanent priority to a branch.

A GPU placement removes both replay and transfer from the selected path. A host placement preserves the same tail under GPU pressure and pays restoration only if that branch is selected. As scores and memory pressure change, state may move between tiers or leave the group; branch identity and the continuation boundary remain unchanged.

4.4 Continuing the selected branch

Selection resolves the candidate-group lifetime. Pending sibling work is canceled and completed sibling state is reclaimed. A selected branch already in decoding continues in place; otherwise generation begins at its recorded boundary, either from resident state or by restoring that boundary from the host tier.

Selection therefore changes the role of a branch without changing the history it represents. The same record moves from pre-selection preparation to foreground generation, so work completed before the decision remains ahead of the unfinished continuation.

For top-K selection, this transfer applies independently to every chosen history. Selection supplies the branch identities directly, so each continuation inherits the state and progress recorded for that branch. The sibling records are released together.

The three decisions meet on the selected branch's first-token path. Let $R_i(a_i)$ be its residual prefill under placement a_i, $L_i(a_i)$ any restoration time, D_i the first decode step, $X_i(d)$ the first-token work completed during a selection window of length d, and Q_i post-selection queueing. Measured from selection, branch i has continuation TTFT

$$T_i^{\text{sel}} = Q_i + L_i(a_i) + [R_i(a_i) + D_i - X_i(d)]_+. \tag{3}$$

All work terms are expressed in time, and $[x]_+ = \max\{x, 0\}$. Retention determines R_i and L_i, pre-selection execution supplies X_i, and handoff determines Q_i. Keeping these terms on the same branch record preserves work completed before selection on the chosen path. For a released branch, R_i is full prefill and $L_i = X_i(d) = 0$, recovering the ordinary continuation path.

The handoff scheduler admits selected branches through a bounded foreground lane. Short bursts keep ready state from waiting behind unrelated queue work; between bursts, regular-request admission resumes. A selected branch reaches continuation with both its KV state and the progress accumulated during selection.

5 Evaluation

5.1 Experimental setup

Testbed. We implement CandiKV on vLLM and run the evaluation on an RTX 4090D. All methods use the same tokenizer, attention backend, memory budget, and workload trace.

Models. Math workloads use Qwen2.5-Math-1.5B-Instruct and Qwen2.5-Math-7B-Instruct; the other workloads use Qwen2.5-1.5B-Instruct and Qwen2.5-7B-Instruct (Qwen et al., 2025). Math branches are ranked by Skywork-o1-Open-PRM-Qwen-2.5-1.5B (Skywork Team, 2024), and general-reasoning branches by a Qwen2.5-1.5B-Instruct judge.

Workloads. The workloads cover mathematical reasoning (GSM8K and MATH-500), general reasoning (ARC-Challenge and CommonsenseQA), and retrieval (NQ-Open and MS MARCO) (Cobbe et al., 2021; Hendrycks et al., 2021; Clark et al., 2018; Talmor et al., 2019; Kwiatkowski et al., 2019; Bajaj et al., 2018). Retrieval branches use a one-token relevance score. The reasoning workloads continue two ranked branches, whereas retrieval continues the highest-scoring passage. Table 1 lists candidate width, selected count (W/K), and token caps for the branch tail and continuation across the six evaluated workloads.

Baselines. PrefillOnly (Du et al., 2025) executes one-token scoring as a terminal request and prefills the selected history for continuation. **vLLM** (vLLM Project, 2026) uses automatic prefix caching; **LMCache** (Liu et al., 2025) adds a matched host-memory tier. Component studies remove one part of CandiKV at a time. Together, these baselines recover the selected history through recomputation, GPU prefix lookup, or host-tier restoration, whereas CandiKV carries it across selection.

Metrics. We use B_q for a GPU KV budget equal to $q\%$ of the full pool. The main comparison runs at $0.9\lambda_{\text{ref}}$, where λ_{ref} is the sustainable arrival rate of the corresponding vLLM path, with the full pool B_{100}. TTFT is measured from selection completion to the first token of the chosen continuation, isolating the latency introduced after the choice from scoring work already completed. We report its 95th percentile unless stated otherwise. We also report group job completion time (JCT), post-selection prefill, and wasted GPU KV byte-seconds, which integrate the GPU KV residency of unselected branches over time.

5.2 End-to-end performance

Table 1 shows that CandiKV lowers TTFT for every model–workload pair, by 26.2–90.7% relative to vLLM and by up to 99.7% relative to PrefillOnly. The reduction over vLLM widens from 26.2–55.7% on 1.5B to 43.9–90.7% on 7B. vLLM reuses the shared prefix but replays the branch-specific tail, while LMCache restores that tail before request admission. Both costs grow with model size; CandiKV instead transfers the selected branch record onto the continuation path. The same ordering holds for top-2 reasoning and top-1 retrieval, despite their different tail and continuation lengths.

Figure 3 shows the same separation as load increases. Verifier scores redirect pre-selection service toward viable reasoning branches, while the longer tails in retrieval increase the value of retained

Table 1: Continuation TTFT on six workloads. Each entry reports the 1.5B/7B result in milliseconds; lower is better.

Dataset	W/K	Token caps	TTFT (ms), 1.5B / 7B			
			PrefillOnly	vLLM	LMCache	CandiKV
GSM8K	8/2	128/512	4.69k/41.7k	112/391	7.53k/8.61k	**49.6/159**
MATH-500	8/2	128/512	7.26k/34.2k	63.9/219	10.5k/3.35k	**33.2/107**
ARC-Challenge	8/2	96/256	2.98k/36.3k	77.1/1.21k	11.3k/13.7k	**45.2/113**
CommonsenseQA	8/2	96/256	413/25.5k	60.6/365	3.13k/4.75k	**37.2/160**
NQ-Open	8/1	512/128	242/1.07k	161/674	10.4k/1.49k	**103/303**
MS MARCO	8/1	512/128	78.9/297	57.2/162	1.07k/347	**42.2/90.9**

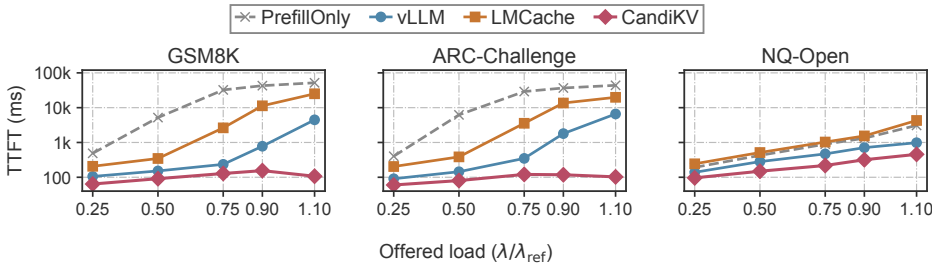


Figure 3: Continuation TTFT for the 7B models as offered load increases. The request-scoped paths recover or rebuild the selected tail and queue the continuation, whereas CandiKV carries the selected record across the same load range. The y-axis is logarithmic.

state. Avoiding replay also removes queue service and KV demand, so the request-scoped curves rise more sharply. Across the sweep, CandiKV lowers TTFT by 15.4–99.0%.

5.3 Candidate structure and selection window

Figure 4 shows that CandiKV's TTFT changes far less than the baselines as candidate width and branch-specific state grow. Wider groups add losing siblings but no new continuation. vLLM accumulates replay and queueing, while LMCache restores the winner before re-entering the queue. Rank updates instead focus CandiKV on histories still in contention. As tails lengthen, the selected history carries forward the state and progress that the baselines must recover after selection.

Width and tail length determine how much state is at stake, while selection delay determines how long the server can act on it. A short window mainly benefits from retention; a longer one lets scores narrow the active set and advance viable branches toward their first token. Across these sweeps, CandiKV lowers TTFT by 34.0–99.9%.

5.4 Mechanism attribution

Table 2 removes each mechanism in turn. The "w/o retention and handoff" row uses request-scoped prefix caching with first-come, first-served (FCFS) admission.

Every removal raises TTFT by at least 2.04×. Removing handoff has the largest effect on GSM8K because prepared work must enter generation promptly. Retention matters more for the longer passage state in NQ-Open. Without retention, handoff receives a branch that must rebuild prefill; without handoff, prepared work waits under FCFS. Full CandiKV keeps retained state and prepared work on the same branch, eliminates post-selection prefill, and cuts TTFT by 41.8–91.0% relative to request-scoped FCFS.

Pre-selection execution removes 8.4k tokens on GSM8K and 4.2k on NQ-Open from the post-selection path. The branch-state manager ablation roughly doubles TTFT because score updates, progress, and handoff no longer share one candidate record.

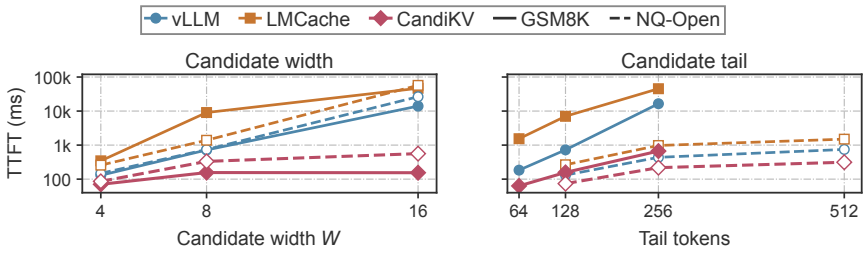


Figure 4: Continuation TTFT as candidate width and tail length vary on GSM8K (solid) and NQ-Open (dashed). CandiKV carries the selected state across both sweeps as request-scoped replay grows with either dimension.

Table 2: Mechanism ablation along the selected first-token path. TTFT is normalized to full CandiKV; post-selection prefill is in thousands of tokens.

Variant	Normalized TTFT		Post-selection prefill	
	GSM8K	NQ-Open	GSM8K	NQ-Open
w/o retention and handoff	2.61	2.11	8.4	4.2
w/o retention	3.27	3.63	109	92
w/o handoff	4.54	3.22	8.4	4.2
w/o pre-selection execution	3.43	3.23	8.4	4.2
w/o branch-state manager	2.07	2.04	8.4	4.2
CandiKV	**1.00**	**1.00**	**0**	**0**

5.5 State cost under a KV budget

Before selection, the retention policy must balance the value of a possible winner against state held for siblings. We compare Retain-all and Expire-all with policies that rank tails by score, length, or CandiKV's joint utility.

Figure 5 compares the two sides of the retention decision. Retain-all keeps latency low by holding every tail, whereas Expire-all minimizes residency but gives up reuse. Score-only identifies a likely consumer but not the work saved by its state; length-only values that work without knowing who will use it. CandiKV combines the two signals, staying near Retain-all latency while reducing wasted GPU KV byte-seconds by 23.7–30.5% relative to Retain-all at B_{100} and B_{75}. Score updates move capacity away from branches that leave contention, so CandiKV does not hold every sibling.

The relative order of score-only and length-only reverses between GSM8K and NQ-Open, reflecting their different tail and score distributions. Neither signal therefore gives a stable allocation across workloads. By ranking expected saved work, CandiKV retains a large tail only when its selection probability justifies the capacity it occupies.

5.6 Mixed traffic

Candidate groups share the engine with regular requests. We mix 75% ShareGPT-style traffic (Zheng et al., 2023) with 12.5% ARC and 12.5% NQ candidate work at two offered loads.

Figure 6 shows that CandiKV reduces continuation TTFT by 28.1–40.8% over vLLM and 47.3–61.8% over PrefillOnly, while regular-request TTFT changes by at most 0.33% and candidate-group JCT by at most 0.95%. PrefillOnly rebuilds the chosen history after selection, and LMCache restores its state before request admission. CandiKV carries the prepared branch into generation, so the gap grows when recovery competes with a busier queue. Score updates focus background work on viable branches, while bounded handoff bursts return the scheduler to regular requests. The shorter continuation path offsets this service, keeping group JCT close to the vLLM path at both loads.

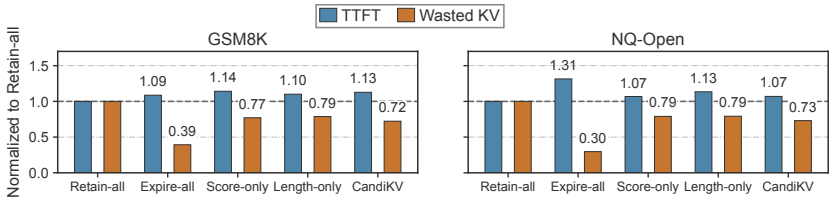


Figure 5: Retention policies across GPU KV budgets. CandiKV stays near Retain-all TTFT while reducing wasted GPU KV byte-seconds; both measures are normalized and lower is better.

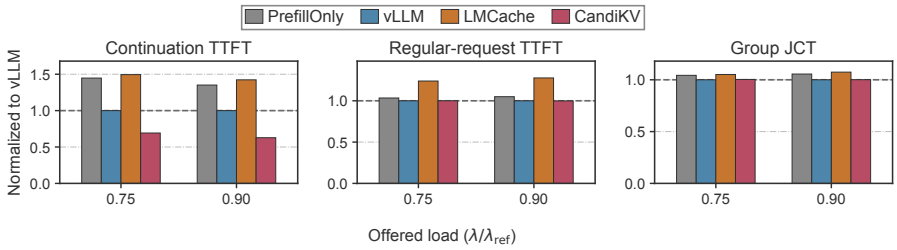


Figure 6: Mixed traffic at two offered loads. CandiKV lowers continuation TTFT while regular-request TTFT and group JCT remain close to vLLM; values are normalized at each load.

6 Discussion

Across the sweeps, the value of selection awareness follows the work at risk and the time available to preserve it. Longer tails increase the saved work, wider groups raise the cost of retaining every sibling, longer windows let scores focus service, and higher load amplifies the cost of restarting a winner. Token matching establishes that state can be reused; selection identifies its next consumer and when it will arrive.

The ablations show that state and progress are not interchangeable. Retention avoids reconstruction, pre-selection execution reduces unfinished work, and handoff prevents prepared branches from re-entering the ordinary queue. Their gains compound because all three decisions act on the same branch record. As scores reorder the candidate set and execution advances, the most valuable state changes with the work still remaining. Coordinating memory and service around that moving target converts work done before selection into lower continuation latency after it.

This distinction separates selection-aware serving from ordinary cache reuse. A cache recognizes a history after a request refers to it, whereas a selector reveals which member of a live candidate set is about to continue. That signal connects placement, pre-selection execution, and admission around the same consumer. Keeping state and progress with that consumer lowers TTFT and supports top-K choices. In a repeated workflow, each chosen history can become the starting point of the next candidate group without breaking continuity between stages.

7 Conclusion

Branching LLM workflows often continue from histories the model has already processed, yet request-scoped serving breaks their execution continuity at selection. CandiKV keeps candidate histories live, directs state and computation toward likely continuations, and transfers selected records into generation. Across six workloads, it reduces continuation TTFT by up to 90.7% and cuts wasted GPU KV byte-seconds by 23.7–30.5%, while leaving regular-request TTFT nearly unchanged. The load and candidate-structure sweeps show the same pattern as the selection window grows or the candidate state becomes larger. Keeping these records across selection lets the continuation reuse the state and progress that scoring has already produced.

AI USE STATEMENT

In this work, we used generative AI tools to assist with code generation and review, proof development, literature search, and suggestions for improving the clarity of the manuscript. The authors reviewed and verified all AI-assisted material and take full responsibility for the final paper, code, and reported results.

ETHICS STATEMENT

This work studies LLM serving infrastructure using publicly available models and benchmarks. It does not involve human participants or private user data.

REPRODUCIBILITY STATEMENT

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

REFERENCES

Reyna Abhyankar, Zijian He, Vikranth Srivatsa, Hao Zhang, and Yiying Zhang. InferCept: Efficient intercept support for augmented large language model inference. In *Proceedings of the 41st International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2024.

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.

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*, 2022.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations*, 2024.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. MS MARCO: A human generated MAchine reading COmprehension dataset. *arXiv preprint arXiv:1611.09268*, 2018.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2025.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. Clipper: A Low-Latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, 2017.

Kuntai Du, Bowen Wang, Chen Zhang, Yiming Cheng, Qing Lan, Hejian Sang, Yihua Cheng, Jiayi Yao, Xiaoxuan Liu, Yifan Qiao, Ion Stoica, and Junchen Jiang. PrefillOnly: An inference engine for prefill-only workloads in large language model applications. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, SOSP '25. Association for Computing Machinery, 2025.

Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. SPLADE: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21. Association for Computing Machinery, 2021.

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.

Shiwei Gao, Youmin Chen, and Jiwu Shu. Fast state restoration in LLM serving with HCache. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25. Association for Computing Machinery, 2025.

Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. Serving DNNs like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 2020.

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*, 2021.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2024.

Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Association for Computational Linguistics, 2021.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*, 2022.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 2023.

Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y. Fu, Christopher Ré, and Azalia Mirhoseini. Hydragen: High-Throughput LLM inference with shared prefixes. *arXiv preprint arXiv:2402.05099*, 2024.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2020.

Omar Khattab and Matei Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20. Association for Computing Machinery, 2020.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 2019.

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.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020.

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.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2024.

Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, 2023.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. CacheGen: KV cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*. Association for Computing Machinery, 2024a.

Yuhan Liu, Yihua Cheng, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Rui Zhang, Kuntai Du, and Junchen Jiang. LMCache: An efficient KV cache layer for enterprise-scale LLM inference. *arXiv preprint arXiv:2510.09665*, 2025.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2024b.

Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E. Gonzalez, and Ion Stoica. Autellix: An efficient serving engine for llm agents as general programs. *arXiv preprint arXiv:2502.13965*, 2025.

Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. Document ranking with a pre-trained sequence-to-sequence model. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, 2020.

NVIDIA. TensorRT-LLM. https://developer.nvidia.com/tensorrt-llm, 2024.

Rui Pan, Zhuang Wang, Zhen Jia, Can Karakus, Luca Zancato, Tri Dao, Yida Wang, and Ravi Netravali. Marconi: Prefix caching for the era of hybrid llms. In *Proceedings of Machine Learning and Systems*, 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 *ISCA*, 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.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Heyi Tang, Feng Ren, Teng Ma, Shangming Cai, Yineng Zhang, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: A kvcache-centric disaggregated architecture for llm serving. *ACM Trans. Storage*, 2026.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. INFaaS: Automated model-less inference serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 2021.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. ColBERTv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2022.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023.

Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. Nexus: A GPU cluster engine for accelerating DNN-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19. Association for Computing Machinery, 2019.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, 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.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. REPLUG: Retrieval-augmented black-box language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Association for Computational Linguistics, 2024.

Skywork Team. Skywork-o1-Open-PRM-Qwen-2.5-1.5B. https://huggingface.co/Skywork/Skywork-o1-Open-PRM-Qwen-2.5-1.5B, 2024.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of NAACL-HLT*, 2019.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2024.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

vLLM Project. Automatic Prefix Caching. https://docs.vllm.ai/en/stable/features/automatic_prefix_caching/, 2026.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25. Association for Computing Machinery, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023b.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In *Proceedings of Machine Learning and Systems*. MLSys, 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.

Lingfan Yu, Jinkun Lin, and Jinyang Li. Stateful large language model serving with Pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25. Association for Computing Machinery, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, 2023.

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.

A CANDIDATE HISTORIES ACROSS SELECTION

Request-level execution treats state retention and work performed before selection as separate decisions. Candidate-group serving joins them on the same first-token path while preserving branch identity through the choice.

A.1 CANDIDATE-GROUP LIFETIME

For a candidate group g, let P_g be its shared prefix, $\{S_{g,i}\}_{i=1}^{W_g}$ its candidate tails, I_g^* its selected set, t_s^g its selection time, and t_f^g its completion time. Before t_s^g, each branch may remain on the GPU, move to the host tier, or be released; write this placement as $a_{g,i}(t) \in \{\mathrm{GPU}, \mathrm{CPU}, \mathrm{drop}\}$ and let $b_{g,i}(t)$ be its state footprint.

Let $\mathcal{G}_t$ be the candidate groups active at time t. State placement across these groups must satisfy the aggregate GPU and host budgets

$$\sum_{g\in\mathcal{G}_t}\sum_{i} b_{g,i}(t)\,\mathbf{1}[a_{g,i}(t)=\mathrm{GPU}] \le B_{\mathrm{GPU}}, \qquad \sum_{g\in\mathcal{G}_t}\sum_{i} b_{g,i}(t)\,\mathbf{1}[a_{g,i}(t)=\mathrm{CPU}] \le B_{\mathrm{CPU}}. \tag{4}$$

Before selection, both placement and progress decisions are made without knowing I_g^*. After selection, state belonging to I_g^* joins the continuation path and the remaining state can be reclaimed. Capacity is therefore shared among active groups instead of being assigned to each scoring request throughout the selection window.

Omitting the group index, each history $P \,\|\, S_i$ carries an immutable application continuation boundary, together with its current placement and progress beyond that boundary. Once emitted, scores remain fixed and ties are broken by branch identity. Retention and pre-selection execution use the score order and resource state available at each scheduling event.

For a branch i, let ϕ_i denote its phase. Its record follows the transition

$$\phi_i:\ \text{scoring} \longrightarrow \text{pending selection} \longrightarrow \begin{cases}\text{continuation}, & i \in I_g^*,\\ \text{released}, & i \notin I_g^*.\end{cases} \tag{5}$$

When scoring completes, the request's block references move to the candidate-group record. Placement may change, but the record continues to name the same history and boundary. Selection transfers records in I_g^* to continuation and releases their siblings. A branch ruled out by the score order reaches the released state before t_s^g.

A.2 SELECTION ORDER AND STATE CONTINUITY

For a score-based top-K selector, let O_t contain the candidates scored by time t, and let $\succ$ be the strict order induced by score and identity tie-breaking. Equation 1 gives the observed rank $r_t(i)$ for $i \in O_t$. The same ordering applies when an external verifier supplies the branch scores.

Observed rank can only stay fixed or increase as O_t grows. Once $r_t(i) > K$, at least K candidates permanently precede branch i, so it cannot enter the final $\mathrm{TopK}_K\{(s_i, i)\}_{i=1}^{W}$. Conversely, a branch in the final selected set never meets this removal condition.

This order induces a nested set of candidates that may still continue. Let $\mathcal{V}$ be all branches in the group and define

$$\mathcal{E}_t = \big(\mathcal{V} \setminus O_t\big) \cup \mathrm{TopK}_K\{(s_i, i) : i \in O_t\}. \tag{6}$$

Unscored branches remain eligible because their position is not yet known; among scored branches, only the current top-K remain. When a new score arrives, that branch was already present in $\mathcal{V} \setminus O_t$. It either remains in $\mathcal{E}_t$ and displaces a lower-ranked branch, or leaves the set itself. Hence $\mathcal{E}_{t'} \subseteq \mathcal{E}_t$ for $t' > t$. Pre-selection execution can shrink its working set without later reopening a history that was removed by rank.

There is a short interval between observing a rank change and stopping work at the next scheduler event. If $\mathcal{R}_j$ is the set removed by update j, ϵ_j its cancellation delay, and ν_i the service rate of

branch i, the work issued after a branch becomes ineligible is bounded by

$$X_{\text{spill}} \le \sum_j \sum_{i \in \mathcal{R}_j} \nu_i \epsilon_j. \tag{7}$$

Batching removals at the next scheduling event shares this delay across the branches eliminated by the same score update.

Let $H_\theta(x_{1:\ell})$ denote the layerwise KV state produced by model θ for history $x_{1:\ell}$ through offset ℓ. For a stored branch record b and its continuation c, let ℓ, π, and σ denote the offset, positional policy, and score-token convention. Direct attachment uses the following match:

$$\text{match}(b, c) = 1 \quad \iff \quad (\ell_c, \pi_c, \sigma_c) = (\ell_b, \pi_b, \sigma_b) \ \wedge \ x^c_{1:\ell_b} = x^b_{1:\ell_b}. \tag{8}$$

Under this condition, $H_\theta(x^b_{1:\ell_b}) = H_\theta(x^c_{1:\ell_c})$: the retained KV is exactly the state produced by prefilling c. Including the score-token convention in the boundary preserves the attention inputs and logits of subsequent tokens.

A state handle carries the group and candidate identifiers, boundary hash, and continuation offset. The hash covers both the shared prefix and candidate tail, preserving lineage and the continuation position during lookup. Matching handles attach directly; other histories return through the standard prefill path used by ordinary requests.

A.3 Retention under shared capacity

Equation 2 combines selection probability, saved prefill work, and placement cost. Its value changes with the score order, tail coverage, queue occupancy, and available KV capacity.

CandiKV derives p_i from observed rank and score margin, normalized so the probability mass reflects K selections. The saving term combines uncovered tail tokens with an online estimate of their prefill time. State bytes, reload time, memory pressure, and queue slack enter the placement cost.

For a placement a with $\Delta_i(a) > 0$, branch i has positive retention utility exactly when

$$p_i(t) > \frac{C_i(a, t)}{\Delta_i(a)}. \tag{9}$$

A better observed rank raises the estimated selection probability, while greater tail coverage increases the work saved by a hit. Either can move a branch above the threshold, which explains why score-only and length-only rankings retain different tails. The choice between GPU and host placement follows from

$$U_i(\text{GPU}, t) - U_i(\text{CPU}, t) = p_i(t)\big[\Delta_i(\text{GPU}) - \Delta_i(\text{CPU})\big] - \big[C_i(\text{GPU}, t) - C_i(\text{CPU}, t)\big]. \tag{10}$$

GPU residence is preferred when its additional saved work exceeds the cost of holding the blocks there; otherwise the host tier preserves the same branch boundary at lower GPU occupancy.

For a fixed snapshot of unresolved branches, suppress the group index, let b_i be the footprint of branch i, and let $z_{i,a} \in [0, 1]$ denote the fraction assigned to placement $a \in \{\text{GPU}, \text{CPU}, \text{drop}\}$. With GPU and host capacities B_{GPU} and B_{CPU}, the fractional relaxation is

$$\begin{aligned} \max_z \quad & \sum_i \sum_{a \in \{\text{GPU},\text{CPU}\}} z_{i,a} U_i(a, t) \\ \text{s.t.} \quad & \sum_i b_i z_{i,\text{GPU}} \le B_{\text{GPU}}, \qquad \sum_i b_i z_{i,\text{CPU}} \le B_{\text{CPU}}, \\ & z_{i,\text{GPU}} + z_{i,\text{CPU}} + z_{i,\text{drop}} = 1, \qquad z_{i,a} \ge 0. \end{aligned} \tag{11}$$

For a fixed tier a, positive-utility states are ordered by $U_i(a, t)/b_i$. Moving an amount δ of capacity from i to j changes the objective by $\delta(U_j(a, t)/b_j - U_i(a, t)/b_i)$; the relaxation therefore fills capacity in decreasing utility density, with at most one partial item. The runtime applies this order at block granularity and releases records with non-positive utility in both tiers.

Given the GPU-resident fraction $z_{i,\mathrm{GPU}}$ and selection probability $p_i = \Pr(i \in I^*)$, the useful and unselected snapshot footprints take the form

$$W^{\mathrm{snap}}_{\mathrm{use}} = \sum_i p_i z_{i,\mathrm{GPU}} b_i, \qquad W^{\mathrm{snap}}_{\mathrm{waste}} = \sum_i (1-p_i) z_{i,\mathrm{GPU}} b_i. \tag{12}$$

Once I^* is known, setting $z_{i,\mathrm{GPU}} = 0$ for $i \notin I^*$ removes the unselected footprint while preserving the selected records.

The snapshot view extends over the full selection window. Let t_0 be group admission and $q_i(t) = \mathbf{1}[a_i(t) = \mathrm{GPU}]$. GPU residence before selection decomposes according to the eventual consumer:

$$W^{\mathrm{pre}}_{\mathrm{selected}} = \sum_{i \in I^*} \int_{t_0}^{t_s} b_i(t) q_i(t)\, \mathrm{d}t, \tag{13}$$

$$W^{\mathrm{pre}}_{\mathrm{sibling}} = \sum_{i \notin I^*} \int_{t_0}^{t_s} b_i(t) q_i(t)\, \mathrm{d}t. \tag{14}$$

The first term preserves state that enters continuation. The second is the price of making the decision before the winner is known. Given the scores observed at time t, sibling residency accumulates at the expected rate

$$\mathbb{E}\left[\frac{\mathrm{d}W^{\mathrm{pre}}_{\mathrm{sibling}}}{\mathrm{d}t}\right] = \sum_i \big(1 - p_i(t)\big) b_i(t) q_i(t). \tag{15}$$

As the score order sharpens, lowering $q_i(t)$ for unlikely branches reduces this term while retaining histories whose state is likely to cross selection.

For an unselected branch resident at selection, let $t_i \geq t_s$ be its reclamation time and t_f the group completion time. Its post-selection GPU KV byte-seconds are

$$W_{i,\mathrm{post}} = \int_{t_s}^{\min\{t_i, t_f\}} b_i(t)\, \mathrm{d}t. \tag{16}$$

Immediate sibling reclamation sets this term to zero.

A.4 Work during the selection window

Retention decides where a history resides; pre-selection execution decides how far that history advances. For a branch at placement a_i, write $H_i = R_i(a_i) + D_i$ for the remaining work through its first continuation token. Let $x_i \in [0, H_i]$ be the work completed before selection, and let $C(d)$ be the service available to a candidate group during a window of length d. At a fixed score snapshot, the expected useful progress follows

$$\begin{aligned} \max_{x} \quad & \sum_i p_i(t) x_i \\ \text{s.t.} \quad & \sum_i x_i \leq C(d), \qquad 0 \leq x_i \leq H_i. \end{aligned} \tag{17}$$

The objective is linear until a branch reaches its first token. Execution therefore follows the highest selection probabilities among eligible branches, completing at most H_i work for each before moving to the next. When a score update makes $r_t(i) > K$, that branch leaves the eligible set and subsequent capacity shifts to histories that can still continue. Retention and pre-selection execution use the same score signal for different resources: retention prices saved work against state bytes, while execution uses model time to advance toward the first token.

Applying these allocations across scheduler events yields $X_i(d)$, the work branch i completes over the full selection window.

After the selected set I^* is known, completed work separates into

$$X_{\mathrm{use}}(d) = \sum_{i \in I^*} X_i(d), \qquad X_{\mathrm{discard}}(d) = \sum_{i \notin I^*} X_i(d). \tag{18}$$

The residual selected prefill is

$$R_{\text{post}}(d) = \sum_{i \in I^*} [R_i(a_i) - X_i(d)]_+. \tag{19}$$

Removing a losing branch before another δ of service subtracts that work from X_{discard}. Assigning the service to a selected branch reduces $R_{\text{post}}(d)$ until residual prefill completes, then advances its first decode step.

A.5 The selected first-token path

Equation 3 decomposes selected-branch TTFT into queueing, restoration, and remaining first-token work. Let F_i be the branch-specific prefill time after a shared-prefix hit. Placement a_i leaves residual prefill $R_i(a_i) \in [0, F_i]$ and incurs restoration time $L_i(a_i)$. The work $X_i(d)$ completed within the branch budget is capped at $R_i(a_i) + D_i$.

Placement determines $L_i(a_i)$, pre-selection execution determines $X_i(d)$, and handoff determines Q_i. Section 5.4 separates these terms experimentally.

For comparison, a fresh continuation has

$$T_i^0 = Q_i^0 + F_i + D_i, \tag{20}$$

where Q_i^0 is its queueing delay without a selected-state handoff. When $0 \le X_i(d) \le R_i(a_i) + D_i$, subtracting Equation 3 from Equation 20 gives

$$T_i^0 - T_i^{\text{sel}} = (Q_i^0 - Q_i) + [F_i - R_i(a_i)] + X_i(d) - L_i(a_i). \tag{21}$$

Equation 21 shows where the gain comes from. Retained coverage contributes $F_i - R_i(a_i)$, useful pre-selection progress contributes $X_i(d)$, and direct handoff reduces queueing; restoration is paid once when the state resides off GPU.

For fixed placement and queueing, useful progress translates one-for-one into lower TTFT until the first-token path is complete. For $0 \le X_i'(d) \le X_i(d) \le R_i(a_i) + D_i$,

$$T_i^{\text{sel}}(X_i') - T_i^{\text{sel}}(X_i) = X_i(d) - X_i'(d). \tag{22}$$

The TTFT benefit saturates once $X_i(d) = R_i(a_i) + D_i$.

For fixed queueing, progress, and decoding terms, placement contributes

$$T_i^{\text{sel}}(a_1) - T_i^{\text{sel}}(a_2) = L_i(a_1) - L_i(a_2) + [R_i(a_1) + D_i - X_i(d)]_+ - [R_i(a_2) + D_i - X_i(d)]_+. \tag{23}$$

For equal retained coverage, GPU and host state differ by restoration time. The fresh-prefill case follows directly from Equation 3.

A.6 Coupling state and progress

Retention and pre-selection execution consume different resources, but they act on the same first-token path. A placement determines how much of the branch remains to execute and whether restoration is required; pre-selection execution uses model time on that residual path. Their common value can be written as the first-token work removed relative to fresh continuation. For placement a and pre-selection progress $x \in [0, R_i(a) + D_i]$, define

$$G_i(a, x) = F_i + D_i - L_i(a) - [R_i(a) + D_i - x]_+ = F_i - R_i(a) - L_i(a) + x. \tag{24}$$

The first two terms value retained coverage after restoration; the final term values progress made while selection is still open.

At one score snapshot, let $z_{i,a} \in \{0,1\}$ assign branch i to GPU or host state, and let $x_{i,a}$ be its pre-selection progress under that placement. Expressed in units of time, the joint objective is

$$\begin{aligned}
\max_{z,x} \quad & \sum_i \sum_{a\in\{\mathrm{GPU},\mathrm{CPU}\}} z_{i,a}\big[p_i(t)\big(F_i - R_i(a) - L_i(a)\big) - C_i(a,t)\big] + \sum_i \sum_a p_i(t) x_{i,a} \\
\text{s.t.} \quad & \sum_i b_i z_{i,\mathrm{GPU}} \le B_{\mathrm{GPU}}, \qquad \sum_i b_i z_{i,\mathrm{CPU}} \le B_{\mathrm{CPU}}, \\
& \sum_a z_{i,a} \le 1, \qquad 0 \le x_{i,a} \le z_{i,a}\big(R_i(a) + D_i\big), \\
& \sum_i \sum_a x_{i,a} \le C(d).
\end{aligned} \tag{25}$$

Branches without retained state are not advanced before selection; if selected, they enter through standard prefill. The formulation connects the runtime's placement and service decisions. For fixed placements, each unit of service has marginal value $p_i(t)$ until the branch reaches its first token, recovering Equation 17. For fixed service, moving a branch between tiers compares retained coverage and restoration against the change in placement cost, recovering Equations 9 and 10. Both decisions therefore follow the same score order, although retention spends memory and execution spends model time.

A symmetric candidate group makes the role of branch structure explicit. Suppose a group has $W > K$ equally likely candidates, selects K, and retains m of them on the GPU. If every tail has footprint b and avoids prefill time F, the expected selected prefill saved is

$$\mathbb{E}[S_{\mathrm{saved}}] = m\frac{K}{W}F, \qquad \mathbb{E}[B_{\mathrm{sibling}}] = m\left(1 - \frac{K}{W}\right)b. \tag{26}$$

The saved-prefill return per unselected resident byte is consequently

$$\frac{\mathbb{E}[S_{\mathrm{saved}}]}{\mathbb{E}[B_{\mathrm{sibling}}]} = \frac{KF}{(W-K)b}. \tag{27}$$

Longer tails raise the value of preserving a winner, while greater fanout makes indiscriminate retention less attractive when K stays fixed. Once scores break the symmetry, branch i contributes saved work $p_i F_i$ and expected unselected footprint $(1-p_i)b_i$. Score-only ranking sees p_i but not F_i; length-only ranking sees F_i but not p_i. Their product is the quantity that connects the retention decision to the workflow.

Selection delay determines how much model time is available before the choice. The budget $C(d)$ grows with the window, so useful progress increases until retained branches reach their first token. Beyond that point, Equation 24 has no further first-token value. Rank updates make this service more selective over time because the eligible sets in Equation 6 are nested. Candidate width, tail length, and selection delay therefore affect different parts of the same first-token path.

B Serving the Candidate Group

The candidate-group abstraction becomes a scheduler record that persists across score and selection events. While selection is open, score updates change which branches remain worth serving and where their state should reside. Selection then transfers the chosen records to continuation and releases the rest. The scheduler acts on this record directly instead of reconstructing the relationship through a later cache lookup.

B.1 State Records and Event Order

Each branch record carries the group and candidate identifiers, boundary hash, application continuation offset, placement, phase, accumulated progress, and KV-block handle. Let η_i be the boundary hash and ℓ_i the application continuation offset. Tier movement and pre-selection execution leave this pair unchanged throughout the candidate-group lifetime:

$$\big(\eta_i(t), \ell_i(t)\big) = \big(\eta_i(t_0), \ell_i(t_0)\big) \tag{28}$$

Algorithm 1 Updating branch eligibility and service while selection is open. Score events shrink the active set before the next service allocation.

Require: Group G, observed score events, GPU/host capacities, service budget
1: Apply new scores and form $\mathcal{E}_t$ with Equation 6.
2: Stop branches that left $\mathcal{E}_t$ and release their candidate state.
3: Recompute $p_i(t)$ and $U_i(a, t)$ for the remaining records.
4: Update GPU, host, and released placements under Equation 4.
5: Let c be the model service available before the next event.
6: **while** $c > 0$ and an eligible retained branch has unfinished first-token work **do**
7: Choose the branch with highest current selection probability.
8: Advance it by the smaller of c and its remaining first-token work.
9: Decrease c and record the accumulated progress.
10: **end while**

Here ℓ_i marks the end of the scored history, not the scheduler's current execution cursor. Placement and accumulated progress may change, but the record continues to name the same history and origin boundary. GPU-resident records draw from the block pool; the host tier holds a demoted record at that boundary together with its progress. Release returns its storage to the corresponding allocator.

Three events govern the record. A score event updates rank and selection probability. A scheduling event changes placement or advances the branch along its first-token path. A selection event closes the group: records chosen by the application change owner from the candidate group to their continuations, while the others are released. Once this event is applied, later score arrivals do not reopen the group. At any time, a state record belongs to a scoring request, an unresolved group, or a selected continuation.

B.2 Serving while selection is open

At each scheduling event, CandiKV first applies new scores and shrinks the eligible set. It then revisits its state placement and spends the available background service on retained branches. Algorithm 1 summarizes this loop.

Unscored branches remain eligible until their score is known. Once a scored branch falls outside the eligible set, the nesting property of Equation 6 makes its removal final. Placement may still change among the remaining branches as memory pressure, score margins, and retained coverage change. The work budget is bounded at every event; CandiKV stops advancing a branch when it reaches its first token, even if the selection window remains open.

B.3 Resolving selection

Selection changes ownership in one step. Algorithm 2 stops pre-selection work, reclaims siblings, and attaches each compatible selected record to its continuation. A resident record can continue in place; a host record keeps the same boundary while its blocks return to the GPU.

When $K > 1$, group resolution promotes K independent branch records. Each record keeps its own continuation boundary, placement, and pre-selection progress, and Equation 8 applies separately to each continuation. The remaining $W - K$ records are reclaimed when the selected set is known; the group closes after all selected records enter continuation.

Compatibility is determined by the selected branch and boundary. A matching record transfers its handle to the continuation; other histories enter through the standard prefill path. Moving the handle preserves the state equality in Equation 8 without copying group state into a new request record.

B.4 Handoff on the first-token path

Selected continuations and regular requests use separate admission lanes. Once the selection event supplies an immediate consumer, CandiKV admits a short selected burst and then revisits the regular-request lane. Ready state moves quickly onto the first-token path, while the next regular-request visit remains bounded by the work in that burst.

Algorithm 2 Resolving selection and handing the selected records to generation. Unselected siblings release their state before the handoff lane runs.

Require: Unresolved candidate group G, selected set I^*
1: Close the selection window and stop pending pre-selection execution.
2: Release every record $i \notin I^*$ and reclaim its state.
3: **for** each $i \in I^*$ **do**
4: Check the history and boundary using Equation 8.
5: Transfer ownership of the matching record to continuation i.
6: Resume from resident state or restore the same host-tier boundary.
7: Enqueue the continuation with its accumulated pre-selection progress.
8: **end for**
9: Admit a bounded selected burst, then return to the regular-request lane.
10: Close G after all records in I^* have entered continuation.

Let $\mathcal{H}_t$ be the selected records admitted in one burst and let s_i be the service quantum granted before the scheduler returns to regular requests. The work placed ahead of the next regular-request visit is

$$\Delta Q_{\mathrm{reg}}(t) \le \sum_{i \in \mathcal{H}_t} s_i. \tag{29}$$

Bounding both the burst size and its per-branch quantum bounds this delay at each visit. For a selected branch, the same admission removes part of Q_i in Equation 3; for a regular request, the return to its lane prevents selected groups from occupying every subsequent admission decision. The mixed-traffic result in Figure 6 reflects this division: continuation TTFT falls while regular-request TTFT and group completion remain near the request-scoped path.

Handoff completes the preparation made by retention and pre-selection execution. Retention determines whether state is resident or requires restoration, pre-selection execution determines how much first-token work remains, and the selected lane determines when that work runs. The state handle carries these decisions to one continuation, so their gains add along the path in Equation 21.

B.5 Scheduling across groups

The capacity constraints in Equation 4 span all live candidate groups. Each unresolved group exposes its eligible branch records, their placement utilities, and a bounded amount of first-token work. The memory manager orders these records in the common GPU and host pools; it does not reserve a private cache for each group.

Model service is shared in the same way. For the active groups $\mathcal{G}_t$, let $c_g(t)$ be the portion of group g's continuation budget $\mathcal{B}_g$ available during one scheduling interval, and let C_t be the interval's total background capacity. With remaining first-token work $H_{g,i}$, the cross-group allocation is

$$\begin{aligned} \max_{x} \quad & \sum_{g \in \mathcal{G}_t} \sum_{i \in \mathcal{E}_{g,t}} p_{g,i}(t) x_{g,i} \\ \text{s.t.} \quad & \sum_{g} \sum_{i} x_{g,i} \le C_t, \qquad \sum_{i} x_{g,i} \le c_g(t), \\ & 0 \le x_{g,i} \le H_{g,i}. \end{aligned} \tag{30}$$

The group budget prevents a wide fanout from consuming the entire interval, while selection probability determines which histories within that group advance first. With fixed group budgets, the linear objective again serves eligible branches in decreasing $p_{g,i}(t)$ until a branch or group reaches its cap. Score events can remove branches between intervals, and newly admitted groups join the next allocation without changing work already completed.

For a group that selects several branches, the selection probabilities sum to its expected selected count rather than one. Equation 30 therefore values all likely winners, while the per-group cap keeps their total service comparable with other active groups. After selection, those winners leave the background allocation and enter the bounded handoff lane; siblings release both their state and any remaining group budget.

Let $N_t = \sum_g |\mathcal{E}_{g,t}|$ be the number of eligible branch records. Ordering their placement utilities and service priorities costs $O(N_t \log N_t)$ when rebuilt globally. A score event only changes one group:

for width W, updating its ranks and utility order costs $O(W \log W)$, followed by an $O(W)$ scan for placement and service. Group resolution is linear, touching K selected records and $W - K$ siblings. Metadata storage is $O(N_t)$.

Before selection, candidate work follows the same prefill and decode path as a continuation. The scheduler decides which eligible branch to advance and how much capacity it receives; score events revise that order as the selected set takes shape.

B.6 Repeated selection stages

Reasoning and retrieval workflows may repeat generation, scoring, and selection several times. The candidate-group lifetime composes across these stages. Let stage r begin with prefix P_r, choose branch i_r, and emit continuation tokens Y_r before the next set of candidates is formed. The next stage starts from

$$P_{r+1} = P_r \parallel S_{r,i_r} \parallel Y_r. \tag{31}$$

The record transferred at stage r therefore becomes the parent state of the next group. Its selected continuation first consumes the retained branch state, and later candidates fork from the resulting history. When a stage selects several branches, each transferred record can start its own descendant group in the same way.

The history check remains local to each selection boundary. Equation 8 attaches the selected record at stage r; the tokens produced after that point extend the history and define the boundary used at stage $r + 1$. Sibling records from stage r are no longer part of either path and release their state. Thus, each new group inherits a concrete token history rather than a collection of KV fragments from earlier candidates.

The first-token effect also composes along a selected path. For a path $\mathcal{P} = (i_1, \ldots, i_R)$ through R sequential selection stages, applying Equation 21 at each stage gives

$$\begin{aligned}\Delta T_{\mathcal{P}} = \sum_{r=1}^{R} \Big[& Q^0_{r,i_r} - Q_{r,i_r} + F_{r,i_r} - R_{r,i_r}(a_{r,i_r}) \\ & + X_{r,i_r}(d_r) - L_{r,i_r}(a_{r,i_r}) \Big].\end{aligned} \tag{32}$$

Each stage can have a different fanout, tail length, and selection window. The runtime makes its retention and service decisions with the scores available at that stage, while the ownership transfer preserves the selected lineage between stages.