LATTICETTS: DEADLINE-AWARE FRONTIER COMPILATION FOR VERIFIER-GUIDED TEST-TIME SCALING

Anonymous authors
Paper under double-blind review

ABSTRACT

Verifier-guided test-time scaling (TTS) improves the reasoning of compact models by expanding and scoring several candidate paths. Under a time budget, a candidate helps only if it is generated and verified before the deadline. Existing serving systems make each stage faster but treat the requested frontier as a fixed amount of work. Our measurements show this cost is discontinuous, since one more child can push either stage across a batch boundary and raise its latency by up to 26% while leaving the other almost unchanged. We present LatticeTTS, a serving layer that keeps every selected parent and decides only how many children to run beneath each, without retraining or changing the ranking rule. A joint service lattice records the generator, verifier, and key–value (KV) cache state of each round, and an online planner moves along a nested sequence of child counts, admitting only children whose complete cycle fits the remaining deadline. Across three model pairs, three reasoning benchmarks, and three search policies, LatticeTTS achieves a 5.75× geometric-mean end-to-end speedup, up to 18.36× higher throughput under concurrent load, and up to 16.7 percentage points higher fixed-deadline accuracy under tight budgets.

1 INTRODUCTION

Reasoning-oriented language models gain multi-step capabilities from chain-of-thought training and reinforcement learning (Wei et al., 2022; Guo et al., 2025; Yang et al., 2024). Test-time scaling (TTS) extends these capabilities by exploring candidate solutions at inference (Wang et al., 2023; Brown et al., 2024; Snell et al., 2025; Yao et al., 2023; Guan et al., 2025). In verifier-guided TTS, a verifier scores partial solutions and directs later expansion toward promising paths (Cobbe et al., 2021; Uesato et al., 2022; Lightman et al., 2024; Setlur et al., 2025). This flexibility is especially useful in resource-constrained deployments, where compact models fit in memory and harder prompts can receive more local compute (Sheng et al., 2023; Song et al., 2024; Lin et al., 2024). Additional compute is useful only when generation and verification finish before the deadline and return verified evidence to the next search step.

However, current serving systems treat the frontier chosen by search as fixed work. Batching, KV management, speculative execution, and phase-aware scheduling reduce the cost of generation and verification (Kwon et al., 2023; Zheng et al., 2024; Miao et al., 2024; Zhong et al., 2024; Agrawal et al., 2024; Cai et al., 2024), while FastTTS specializes these optimizations for verifier-guided reasoning (Chen et al., 2026a). Once the frontier reaches the serving backend, its size no longer responds to the deadline. This couples parent selection, which determines the reasoning paths in play, with child count, which determines how much evidence each path gathers. A child that finishes after the deadline consumes time without entering the next selection step. Allowing the backend to vary that count while preserving the parent set could prevent such late work, but doing so raises three challenges.

The first challenge is how to model the cost of a logical frontier across both model stages. The generator and verifier see different batch shapes and sequence lengths (Yu et al., 2022; Aminabadi et al., 2022; Ye et al., 2025), and KV residency carries request history into the next round (Cai et al., 2025; Hooper et al., 2024). Consequently, one more child may leave one stage unchanged but push the other into a new batch, which a scalar width-to-latency curve can miss (Chen et al., 2026b;

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.

Bhendawade et al., 2025). A useful cost model should couple the two stage states and expose the boundaries that determine whether another complete cycle can finish.

The second challenge is how to reduce frontier work without changing the search decision. Branch pruning and adaptive proposal methods narrow the work that reaches execution (Miao et al., 2024; Zheng et al., 2024; Juravsky et al., 2024; Cha et al., 2026; Chen et al., 2025; Xia et al., 2025). However, in verifier-guided search, removing a selected parent changes the alternatives seen by the verifier (Yao et al., 2023; Cobbe et al., 2021; Lightman et al., 2024). Figure 1 illustrates the available freedom. The frontiers of four and five children retain the same parents, but the fifth child opens a new verifier batch and returns no score before the deadline. The executable frontier should remove only such late work while retaining every selected parent and every sample that a smaller child allocation of the same round already includes.

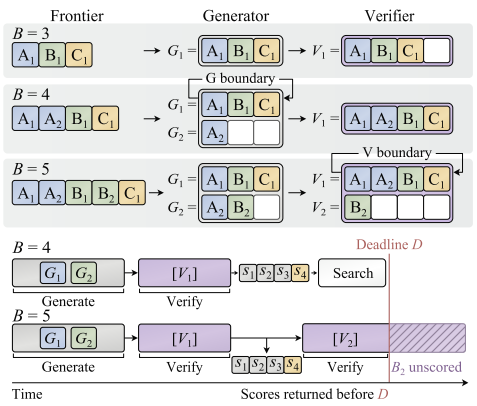

Figure 1: Deadline-aware frontier allocation under discrete service boundaries. A logical expansion becomes useful only when its generator–verifier cycle completes before the deadline.

The third challenge is how to revise an allocation whose true cost becomes visible only at runtime. Adaptive inference and stateful serving already react to budgets or runtime state (Snell et al., 2025; Gujarati et al., 2020; Fu et al., 2024; Gao et al., 2025; Yu et al., 2025), including systems for resource-limited reasoning (Choi et al., 2026; Park et al., 2025; Chen et al., 2026a). Yet step length, termination, verifier shape, and KV residency evolve during a request, so a static profile loses accuracy. The next allocation should update both stages from observed costs while reserving enough time for the admitted children to complete verification and inform the next search decision.

Therefore, we introduce LatticeTTS, a serving layer that adapts the executable frontier to the deadline without changing the paths selected by search. Its key idea is to keep the selected parents fixed and vary only how many children are executed beneath them. For each parent set, LatticeTTS forms a nested family of parent-preserving allocations and evaluates their complete generation–verification cost from the joint generator, verifier, and KV state. Costs observed from completed rounds refine the next choice as the request evolves. Using these estimates, LatticeTTS selects an allocation whose verified evidence can return within the remaining budget while preserving every selected path.

The main contributions of this paper are as follows:

- We identify the gap between a logical frontier and the verified work it can return on time, and capture its coupled generator, verifier, and KV cost with a joint service lattice.
- We develop a frontier compiler that preserves every selected parent while producing nested child allocations. Execution moves among them without changing the search decision.
- We design a deadline-aware planner that values complete generator–verifier cycles, corrects stage costs from completed rounds, and applies the same compilation rule across search policies and changing load.
- We implement LatticeTTS on vLLM and evaluate it over three model pairs, three benchmarks, three search policies, and concurrent load, where it delivers a 5.75× geometric-mean end-to-end speedup, up to 18.36× higher throughput, and up to 16.7 percentage points higher fixed-deadline accuracy.

2 Background and Motivation

2.1 Useful progress in verifier-guided search

A verifier-guided round expands retained parents, scores their children, and selects the next frontier (Cobbe et al., 2021; Uesato et al., 2022; Lightman et al., 2024; Setlur et al., 2025). Parent identity de-

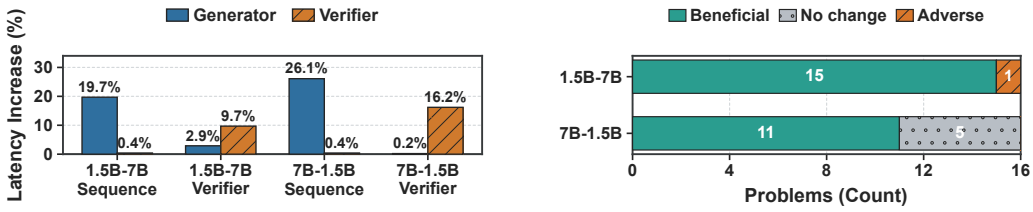

Figure 2: Effect of service-boundary crossings on stage latency (left) and boundary-aware allocation on problem-level outcomes (right).

termines which solutions stay in play; child multiplicity determines how much evidence is gathered beneath each one. Yet a single frontier width conflates these roles.

At execution time, only a scored child can influence selection or seed the next round, as Figure 1 illustrates. Candidate count measures admitted work, whereas completed verification measures returned evidence. Near a deadline, an unfinished verifier pass yields neither an answer nor a successor path, so equal search budgets can produce different progress.

2.2 Logical width is not service cost

Execution turns one frontier into two workloads: the generator groups paths by decoding state and length, the verifier scores their texts, and both inherit KV state from earlier rounds. Batching, attention, prefix reuse, and cache management reduce the cost of these workloads (Chen et al., 2026a; Kwon et al., 2023; Zheng et al., 2024; Ye et al., 2025; Tang et al., 2024; Cai et al., 2025; Zhang et al., 2023; Xiao et al., 2024). However, they do not align the two stages' boundaries. One child can open a batch in either stage while leaving the other unchanged. Consequently, logical width cannot reveal the next cost increase.

The left panel of Figure 2 isolates this mismatch at one frontier decision. Generated work and verifier text remain fixed while a one-child increase moves one stage across its service boundary.

The measurements expose this asymmetry. At sequence boundaries, generator latency rises by 19.7% and 26.1%, while verifier latency changes by 0.4%. At verifier boundaries, verifier latency instead rises by 9.7% and 16.2%. The same one-child increment can therefore enter a different service region at each stage. Visible boundaries distinguish useful added evidence from an expansion that merely opens another batch.

2.3 Execution freedom after parent selection

After parent selection, the serving layer can still decide how many children to execute beneath each parent. Removing one changes which solutions contribute children and which hypotheses the verifier compares (Yao et al., 2023; Cobbe et al., 2021; Uesato et al., 2022; Lightman et al., 2024). One child per parent preserves that decision; later slots follow the search policy's expansion shares. Nested allocations then let each richer frontier extend the preceding one without discarding samples (Miao et al., 2024; Zheng et al., 2024; Juravsky et al., 2024).

The deadline determines how far that nesting should be followed. Deadline-aware inference already varies computation with the available time (Snell et al., 2025; Crankshaw et al., 2017; Gujarati et al., 2020; Yu et al., 2025), yet verifier-guided search must reserve time for both stages. The right panel of Figure 2 compares decisions from the same state with and without the active boundary. Boundary information improves or preserves the outcome in 15 of 16 cases for one model pair and all 16 for the other. Each parent can still contribute a successor, while children unlikely to finish are withheld. Therefore, the useful frontier is the largest parent-preserving allocation whose evidence can return before the cutoff, not the widest frontier or a uniform reduction.

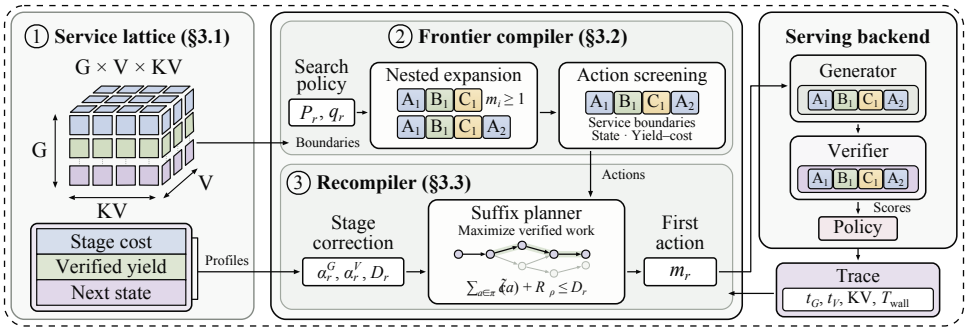

Figure 3: The overview of LatticeTTS.

3 METHOD

Verifier-guided search decides which paths remain; LatticeTTS decides how much work beneath them can finish in time. In Figure 3, the policy selects parents and consumes scores, while LatticeTTS forms and revises parent-preserving choices from execution state and returned traces. Appendix A gives the full derivations and compiler properties.

In round r, the policy returns $P_r = \{p_{r,1}, \ldots, p_{r,K_r}\}$, a set of K_r parents, with normalized expansion shares $\mathbf{q}_r$. An action assigns $m_{r,i} \geq 1$ children to parent $p_{r,i}$, so $B_r = \sum_i m_{r,i} \leq B_{\max}$ is its total width. For request deadline D, the remaining time is $D_r = \max\{0, D - T_r^{\text{elapsed}}\}$. Thus, the parent set stays fixed while the child counts vary.

3.1 JOINT SERVICE LATTICE

A frontier can move into a new generator regime without moving into a new verifier regime. For an executed trace τ_r, LatticeTTS records

$$\Phi(\tau_r) = (z_r^G, z_r^V, z_r^K) \in \mathcal{L}, \qquad \mathcal{L} = \mathcal{Z}_1 \times \cdots \times \mathcal{Z}_d. \tag{1}$$

where Φ maps completed trace τ_r to generator regime z_r^G, verifier regime z_r^V, and post-round KV state z_r^K. Their ordered coordinate sets $\mathcal{Z}_j$ form $\mathcal{L}$, whose product order keeps generator-heavy and verifier-heavy states distinct.

Each cell stores stage cost, verified yield, and its observed successor. A proposed allocation maps to the smallest compatible profiled cell ℓ, whose empirical cost envelope is

$$\bar{c}(\ell) = \max_{\ell' \preceq \ell} Q_\rho(c \mid \ell'), \tag{2}$$

where $\ell' \preceq \ell$ ranges over preceding service states and $Q_\rho(c \mid \ell')$ is the calibrated ρ-quantile of stage cost c. This monotone envelope supports unseen states while retaining discrete boundaries: choices within one envelope are compared by verified yield, whereas crossing a boundary incurs the next batch configuration.

3.2 COMPILING NESTED CHILD COUNTS

To preserve the policy's decision, every choice begins with one child per selected parent. Additional slots follow the policy's shares, and the next goes to

$$i^* = \arg\min_i \frac{e_i + 1}{q_{r,i}}, \tag{3}$$

where m_i is the current count, $e_i = m_i - 1$ its added children, and $q_{r,i}$ is parent i's target share. Stable identifiers resolve ties. At total width B,

$$m_i(B) \geq 1, \qquad \sum_i m_i(B) = B, \qquad \mathbf{m}(B+1) = \mathbf{m}(B) + \mathbf{e}_{i_B}. \tag{4}$$

where $m_i(B)$ is parent i's count and $\mathbf{e}_{i_B}$ identifies the parent receiving the next child. Each richer allocation therefore adds one child without dropping a selected parent. Beam Search uses one global group, DVTS one per subtree, and Dynamic Branching the positive-target parents. Stable sample identities keep each smaller choice inside the next, so the planner can defer width selection without replacing retained work.

Many adjacent widths share a service envelope. The compiler retains the minimum allocation, boundary neighborhoods, choices that change the predicted successor, and nondominated cost–yield points. When the envelope and successor match, the richer choice represents the region if its extra children still complete verification.

3.3 Updating the plan from observed cost

The compiled family sets the available choices; observed sequence length, termination, and cache state then refine their cost. Each completed round updates the two stages separately:

$$\alpha_r^x = \frac{\epsilon + \sum_{j<r} t_{j,\mathrm{obs}}^x}{\epsilon + \sum_{j<r} t_{j,\mathrm{base}}^x}, \qquad x \in \{G, V\}, \tag{5}$$

where x is the generator or verifier, $t_{j,\mathrm{obs}}^x$ and $t_{j,\mathrm{base}}^x$ are its measured and profiled times in round j, and $\epsilon > 0$ stabilizes early rounds. The resulting α_r^x prices action a as

$$\widetilde{c}_r(a) = \alpha_r^G c_{r,G}^{\mathrm{med}}(a) + \alpha_r^V c_{r,V}^{\mathrm{med}}(a) + c_{r,\mathrm{ctrl}}(a), \tag{6}$$

where $c_{r,G}^{\mathrm{med}}$ and $c_{r,V}^{\mathrm{med}}$ are profiled median stage costs and $c_{r,\mathrm{ctrl}}$ is control-plane work. Separate α_r^G and α_r^V retain stage-specific drift.

The planner then evaluates suffix π of at most H rounds from current state x_r on the empirical state graph:

$$\max_{\pi} \quad \sum_{a \in \pi} \widehat{y}(a) \tag{7}$$

$$\text{s.t.} \quad \widehat{C}_\rho(\pi \mid x_r) \le D_r, \qquad x_{j+1} = \widehat{T}(x_j, a_j). \tag{8}$$

Here $\widehat{y}(a)$ is expected verified yield, $\widehat{T}$ predicts the state after future round j, and $\widehat{C}_\rho$ adds a calibrated request-level reserve to corrected suffix cost. A smaller frontier can reserve time for a later round, while a richer one pays off inside the same service envelope. The planner executes only the first choice, then updates its state, stage corrections, and deadline from the trace. If the suffix no longer fits, it shortens the horizon and stops admitting work that cannot finish verification. The objective values both current evidence and the successor paths it makes available.

4 Experiments

4.1 Experimental setup

System, Models and Workloads. The prototype adds frontier compilation and round-level cost feedback to a vLLM-based generator–verifier stack (Kwon et al., 2023), running BF16 inference on an RTX 4090 with 24 GB of memory. Qwen2.5-Math generators are paired with process reward models (PRMs) at three size combinations: 1.5B–1.5B, 1.5B–7B, and 7B–1.5B (Yang et al., 2024; Skywork Team, 2024). The workloads draw mathematical reasoning problems from AIME, non-diagram AMC12, and MATH-500 (Yang et al., 2024; Hendrycks et al., 2021; Lightman et al., 2024), covering both short and long reasoning traces with different rates of early path completion.

Search policies. Beam Search is the default policy. DVTS and Dynamic Branching supply different parent sets and expansion shares (Snell et al., 2025; Chen et al., 2026a).

Baselines. vLLM is the general-purpose serving baseline (Kwon et al., 2023); fixed-width FastTTS is the state-of-the-art serving baseline specialized for verifier-guided TTS on constrained hardware (Chen et al., 2026a). Comparisons share the model pair, search policy, and sampling rule.

Table 1: Median end-to-end model time (s) and geometric-mean paired-request speedup at D_{50}. F/L and v/L denote FastTTS/LatticeTTS and vLLM/LatticeTTS.

Model pair	Metric		16	32	64	128	256	512
1.5B–1.5B	E2E	vLLM	10.8	18.2	26.5	40.7	82.9	169.9
		FastTTS	7.0	10.4	15.6	26.6	53.0	116.5
		LatticeTTS	4.2	5.0	6.7	9.5	19.8	33.2
	Speedup	F/L	**2.23**×	**2.56**×	**2.82**×	**3.22**×	**2.85**×	**3.43**×
		v/L	**3.78**×	**4.47**×	**4.74**×	**4.78**×	**4.25**×	**5.28**×
1.5B–7B	E2E	vLLM	22.1	38.7	73.6	148.7	325.8	648.1
		FastTTS	15.0	25.5	47.5	86.5	163.0	325.2
		LatticeTTS	6.6	8.5	11.4	28.2	51.4	76.6
	Speedup	F/L	**2.81**×	**3.73**×	**4.64**×	**3.42**×	**3.19**×	**4.38**×
		v/L	**4.09**×	**5.75**×	**7.56**×	**6.08**×	**6.32**×	**8.76**×
7B–1.5B	E2E	vLLM	33.2	44.9	74.8	168.6	384.9	892.4
		FastTTS	20.6	26.2	37.1	62.5	99.8	204.3
		LatticeTTS	14.6	14.4	19.3	28.8	42.1	55.9
	Speedup	F/L	**1.85**×	**2.38**×	**1.84**×	**2.72**×	**3.22**×	**5.14**×
		v/L	**3.32**×	**4.47**×	**4.46**×	**7.12**×	**10.98**×	**19.19**×

Metrics. End-to-end model time (E2E) covers generator and verifier execution; deadline and concurrency results use request wall time. We also report fixed-deadline accuracy, PRM-top and oracle goodput, verified candidates, throughput, and P95 wall time (Zhong et al., 2024; Chen et al., 2026a).

Deadline budgets. For each model pair and initial frontier, D_q sets the deadline to $q\%$ of a held-out FastTTS P95. Main comparisons use D_{50}; the deadline study varies q to measure how much verified work each system returns as the available time changes.

4.2 End-to-end efficiency and quality

Table 1 compares end-to-end efficiency at D_{50}. LatticeTTS reduces median model time for every model pair and initial width while retaining the parents selected by the same search policy. The geometric-mean speedup is $2.96\times$ over FastTTS and $5.75\times$ over vLLM. Because a fixed frontier pays for the full batch opened by another child, that work remains costly even when its scores arrive late. By contrast, LatticeTTS stops in the preceding service cell without removing a parent. For 7B–1.5B, the FastTTS-relative speedup rises from $1.85\times$ at width 16 to $5.14\times$ at 512 as another generator batch becomes costly. The 1.5B–1.5B pair shows the same steps at lower absolute times, while swapping the larger model shifts the dominant stage boundary; the joint service state captures how these boundaries shape the cost of each frontier.

Figure 4 relates the latency reduction to PRM-top and oracle goodput and to the amount of work avoided. Fixed frontiers spend time on expansions that cross a batch boundary without returning proportionally more useful scores, but LatticeTTS preserves parent coverage while withholding those late children. It therefore produces more correct verified evidence per unit time even while executing fewer candidates. The gains vary non-monotonically with width because they follow the discrete service boundaries in Figure 2.

4.3 Deadline-aware allocation

Figure 5 follows the executed frontier as more time becomes available. The two initial widths represent different search budgets; within each one, the planner selects a nested allocation that can complete generation and verification before the cutoff.

The benefit is largest under tight budgets, where fixed-deadline accuracy rises by up to 16.7 percentage points. LatticeTTS prices a complete generator–verifier cycle from observed stage costs and admits children whose scores can return before the cutoff. As the budget grows from D_{25} to D_{100}, verified candidates rise from 10.8 to 32.3 at width 128 and from 23.0 to 145.9 at width 512; PRM

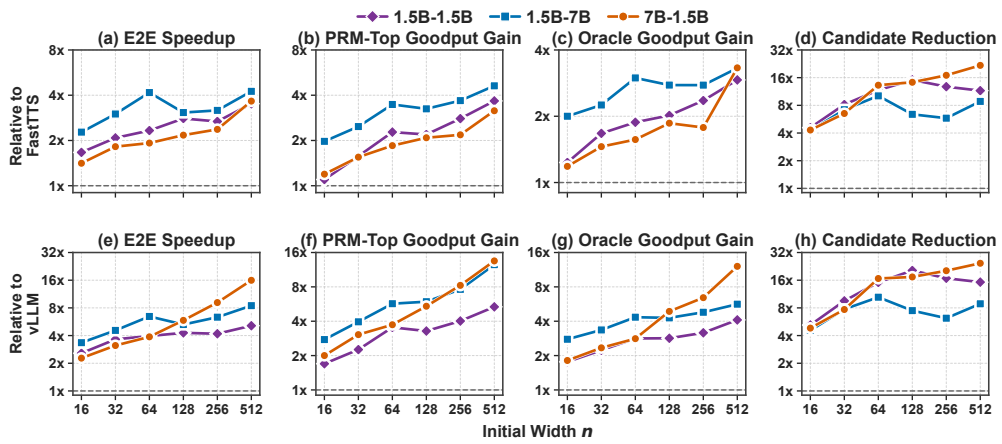

Figure 4: Relative gains of LatticeTTS over FastTTS (top) and vLLM (bottom) across initial widths.

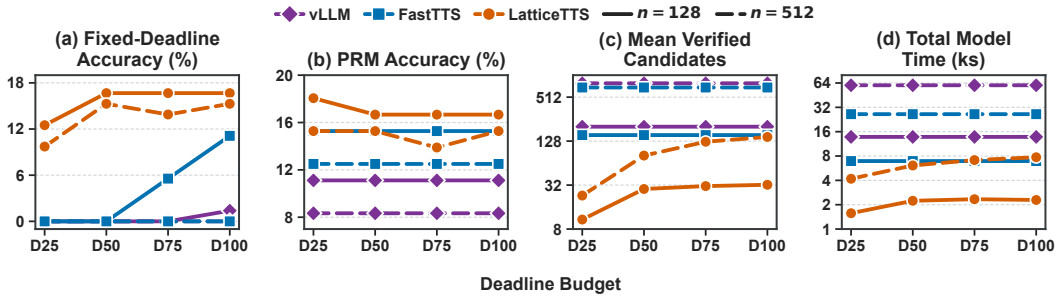

Figure 5: Deadline response at initial widths 128 (solid curves) and 512 (dashed curves); coincident curves overlap.

accuracy varies less because ranking is unchanged. FastTTS and vLLM instead retain their original frontiers, leaving admitted work fixed and sometimes spending a costly final cycle whose scores arrive late. Thus, the added time becomes completed evidence under LatticeTTS but yields little additional deadline accuracy for the fixed-width baselines.

4.4 Synchronized-window concurrency

Synchronized arrivals test whether shorter executable frontiers release useful capacity when several requests compete for both stages. Table 2 reports completed-request throughput and P95 wall time; C is the number of requests released together.

The advantage grows as more requests share both stages. At width 512 and $C = 8$, LatticeTTS delivers $9.65\times$ FastTTS throughput with a $9.05\times$ P95 reduction; the vLLM gains are $18.36\times$ and $20.93\times$. This widening occurs because fixed frontiers hold overlapping batches whose boundary costs accumulate across requests, whereas compiled frontiers return capacity sooner. At width 128, LatticeTTS throughput rises from 0.0313 to 0.0581 requests/s between $C = 1$ and $C = 8$, while both baselines remain nearly flat.

Each completed frontier therefore releases a batching opportunity for the remaining requests; fixed-width execution cannot recycle that time until every child has left both stages.

4.5 Ablation Study

Figure 6 separates the decisions needed to turn saved time into verified progress. The one-round variant ignores later expansions, the no-exit variant admits work after a complete cycle no longer fits, and the static variant never revises its initial plan.

Table 2: Throughput and P95 wall time under synchronized arrivals. C is the concurrent request count, and v/F/L lists vLLM, FastTTS, and LatticeTTS. L/F and L/v are throughput ratios; F/L and v/L are P95 wall-time reductions.

Initial width	C	Throughput (req/s)			P95 wall time (s)		
		$v/F/L$	L/F	L/v	$v/F/L$	F/L	v/L
128	1	0.0049/0.0089/0.0313	**3.53**×	**6.39**×	385.9/160.1/46.7	**3.43**×	**8.26**×
	4	0.0049/0.0107/0.0558	**5.23**×	**11.39**×	1076.5/422.6/96.8	**4.37**×	**11.12**×
	8	0.0047/0.0096/0.0581	**6.03**×	**12.36**×	2173.2/905.4/155.0	**5.84**×	**14.02**×
512	1	0.0011/0.0025/0.0115	**4.65**×	**10.45**×	1618.7/549.8/107.8	**5.10**×	**15.02**×
	4	0.0011/0.0022/0.0191	**8.77**×	**17.36**×	4668.7/2082.0/258.7	**8.05**×	**18.05**×
	8	0.0011/0.0021/0.0202	**9.65**×	**18.36**×	9465.0/4093.0/452.3	**9.05**×	**20.93**×

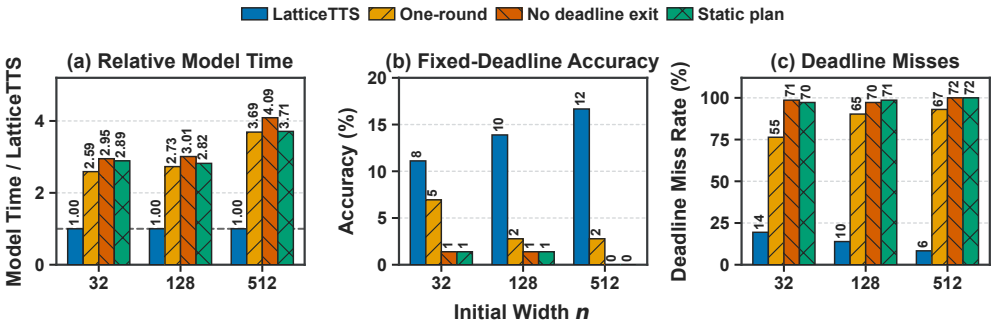

Figure 6: Effect of suffix planning, deadline exit, and online replanning at D_{50}.

Each removal exposes a different role. One-round valuation overlooks the verified descendants displaced by the current expansion; no-exit admits cycles whose scores cannot return; a static plan retains stale stage costs. At width 512, these variants use 3.71×–4.09× the full controller's model time and miss 97–100% of deadlines, compared with 6% for LatticeTTS. Effective boundary-aware allocation therefore needs both future-round value and feedback from the current request.

4.6 Transfer across search policies and local sensitivity

DVTS and Dynamic Branching alter parent groups and expansion shares. Table 3 tests whether the same execution choice transfers beyond Beam Search.

Across both policies, LatticeTTS lowers end-to-end model time by 2.74×–16.60× and turns FastTTS's 100% deadline-miss rate into 0–81.9%, so fixed-deadline accuracy rises from zero at every point. PRM-top accuracy falls in three settings and rises for Dynamic Branching at width 512 because the unchanged ranker sees fewer, timelier candidates. The compiler preserves DVTS subtree partitions and Dynamic Branching targets while changing only child multiplicity. DVTS offers less freedom within each subtree, whereas uneven Dynamic Branching targets expose more children that can be withheld; this difference produces the largest speedup at width 512.

Table 3: Transfer beyond Beam Search. F/L denotes FastTTS/LatticeTTS.

Search	Initial width	PRM acc. F/L (%)	Deadline acc. F/L (%)	Misses F/L (%)	E2E F/L (s)	Speedup
DVTS	128	15.3/12.5	0.0/9.7	100.0/43.1	8149.8/2999.0	**2.74**×
	512	18.1/13.9	0.0/8.3	100.0/81.9	29960.0/8966.2	**3.37**×
Dynamic Branching	128	16.7/12.5	0.0/12.5	100.0/1.4	6707.2/1385.8	**5.13**×
	512	12.5/18.1	0.0/18.1	100.0/0.0	30012.3/1776.9	**16.60**×

Local parameter sensitivity. The complete parameter-sensitivity results appear in Appendix B.7. Planning horizon has the clearest latency effect: short lookahead favors the current expansion, while extending the default from four rounds to eight changes little. Minimum width and completion priority stay near the default latency and cause smaller, less consistent accuracy changes, so the gain does not depend on a narrow parameter choice.

5 Related Work

5.1 Test-time reasoning and verification

Sampling and self-consistency aggregate solutions (Wang et al., 2023; Brown et al., 2024), while tree search carries partial paths forward and directs later computation toward promising branches (Yao et al., 2023; Snell et al., 2025; Guan et al., 2025). Outcome verifiers rank complete answers, whereas process verifiers provide intermediate supervision (Cobbe et al., 2021; Uesato et al., 2022; Lightman et al., 2024; Setlur et al., 2025). These methods determine which paths receive more computation. LatticeTTS leaves that choice and its ranking rule unchanged, adapting child counts so that the resulting evidence can still finish verification.

5.2 Resource-efficient inference near the edge

Offloading, selective activation, and quantization reduce memory demand (Sheng et al., 2023; Song et al., 2024; Lin et al., 2024); attention, KV compression, and prefix reuse lower sequence cost (Ye et al., 2025; Liu et al., 2024a;b; Hooper et al., 2024; Li et al., 2024; Cai et al., 2025; Tang et al., 2024; Zhang et al., 2023; Xiao et al., 2024; Juravsky et al., 2024; Yao et al., 2025; Liu et al., 2025b). Speculative and self-speculative decoding parallelize verification or adapt layer use (Miao et al., 2024; Cai et al., 2024; Chen et al., 2025; Cha et al., 2026; Xia et al., 2025; Kim et al., 2025; Song et al., 2026). FastTTS specializes generator–verifier execution, while SpecEdge and Cassandra study resource-limited reasoning (Chen et al., 2026a; Park et al., 2025; Choi et al., 2026). These methods reduce the cost of a chosen shape, which still varies with length, prefixes, and cache state; LatticeTTS instead chooses among policy-equivalent child allocations.

5.3 Adaptive LLM serving

Orca, vLLM, and vAttention coordinate iterations and KV memory, while SGLang and InferCept reuse program or prefix state (Yu et al., 2022; Kwon et al., 2023; Prabhu et al., 2025; Zheng et al., 2024; Abhyankar et al., 2024). Disaggregation and chunked scheduling separate phase costs (Zhong et al., 2024; Agrawal et al., 2024; Patel et al., 2024; Du et al., 2025), while DREX and FASER adapt batches (Liu et al., 2025a; Chen et al., 2026b); other systems adapt placement, loading, cached state, admission, and model selection (Li et al., 2023; Fu et al., 2024; Gao et al., 2025; Li et al., 2026; Crankshaw et al., 2017; Gujarati et al., 2020; Romero et al., 2021; Yu et al., 2025); DistServe measures goodput under service targets (Zhong et al., 2024). These systems operate on known shapes. By contrast, LatticeTTS acts after scoring and before batch formation, while child counts and complete-cycle cost remain adjustable.

6 Conclusion

Verifier-guided TTS improves reasoning when its additional computation produces evidence that can reach the search before the deadline. LatticeTTS preserves selected parents while adapting child multiplicities to coupled generator and verifier costs. Its service lattice captures discontinuous execution costs, and its planner uses completed-round feedback to reserve time for verification and the later search rounds that build on returned scores. Across model pairs, deadlines, request loads, and search policies, this approach reduces latency and improves throughput and fixed-deadline accuracy. By separating which paths to retain from how much work to execute beneath them, LatticeTTS makes limited inference resources support more completed reasoning while preserving the search policy's control over exploration.

AI USE STATEMENT

Generative AI tools assisted with code generation and review, mathematical derivations and proof development, literature search for related work, and suggestions for improving the clarity of the manuscript. The authors reviewed all AI-assisted material and take responsibility for the final paper, code, and 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

Reproduction code for 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*, 2024.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation*, 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.

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*, 2025.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

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*, 2024.

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. In *Second Conference on Language Modeling*, 2025.

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*, 2026.

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*, 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*, 2025.

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.

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)*, 2026.

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*, 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*, 2025.

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*, 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*, 2025.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. rStar-Math: Small LLMs can master math reasoning with self-evolved deep thinking. In *Proceedings of the 42nd International Conference on Machine Learning*, 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*, 2020.

Daya Guo, Dejian Yang, Haowei Zhang, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 2025.

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*, 2024.

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.

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*, 2025.

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

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of Machine Learning and Systems*, 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*, 2025a.

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*, 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*, 2025b.

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*, 2024b.

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*, 2024.

Jinwoo Park, Seunggeun Cho, and Dongsu Han. SpecEdge: Scalable edge-assisted serving framework for interactive LLMs. In *Advances in Neural Information Processing 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 *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture*, 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*, 2025.

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

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for LLM reasoning. In *International Conference on Learning Representations*, 2025.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y. Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E. Gonzalez, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *International Conference on Machine Learning*, 2023.

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.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025.

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*, 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*, 2024.

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*, 2024.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

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 *International Conference on Learning Representations*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

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.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2.5-Math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

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*, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, 2023.

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*, 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*, 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*, 2025.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. 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*, 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*, 2024.

Table 4: Core notation for frontier-to-service compilation.

Symbol	Meaning
G, V, S	Generator, verifier, and original search policy
r, H	Current round and planning horizon
P_r, K_r	Selected parent set and its cardinality
$q_{r,i}$	Target expansion share assigned by S to parent i
$\mathbf{m}_r, B_r$	Child-count vector and executed frontier width
$\mathcal{F}_r, \mathcal{C}_r$	Complete and retained compiled action sets
u_r, D_r	Revealed pre-action state and remaining deadline
$\Phi, \widehat{\Phi}$	Observed trace map and pre-action profile query
$\ell^{\uparrow}(u, a)$	Minimal profiled cell dominating a pre-action query
$c_r^{\text{phase}}, c_r^{\text{wall}}$	Phase service cost and full wall cost
α_r^G, α_r^V	Online generator and verifier cost corrections
y_r	Verified-candidate yield
$x_h, \widehat{T}_h$	Discrete planning state and estimated transition
R_ρ	Request-level reserve for a predicted suffix

A Derivations and Properties

The method relies on two connected objects: a nested action family that keeps the search decision intact, and a receding-horizon controller that admits work in complete generator–verifier cycles. This section derives the service order and dominance rules that keep the action family compact, then gives both algorithms in full.

A.1 Feasible frontiers and complete-cycle objective

The symbols used in the derivations below are summarized in Table 4. For a selected parent set P_r, the legal action space is

$$\mathcal{A}_r(u_r) = \left\{ \mathbf{m} \in \mathbb{Z}_{\geq 1}^{K_r} : \sum_i m_i \leq B_{\max},\ \mathbf{m} \in \Gamma_S(P_r) \right\}, \tag{9}$$

where $\Gamma_S(P_r)$ contains the policy-specific structural constraints. The positive integer domain keeps every parent selected in the current round; different actions may still produce different children and hence different future parent sets. The shares are positive and normalized within each active policy group. A zero-target parent is excluded from that group's expansion set before compilation, so every heap priority is finite.

Two clocks enter the evaluation. End-to-end model time is

$$c_r^{\text{phase}} = t_r^G + t_r^V, \tag{10}$$

whereas deadline feasibility uses complete wall time

$$c_r^{\text{wall}} = c_r^{\text{phase}} + t_r^{\text{tokenize}} + t_r^{\text{IPC}} + t_r^{\text{select}} + t_r^{\text{plan}}. \tag{11}$$

For action a, $\widehat{y}(a)$ is the number of candidates predicted to finish verification and enter the next policy decision. For a suffix $\pi = (a_r, \ldots, a_{r+h})$, the planning problem is

$$\max_{\pi} \sum_{a \in \pi} \widehat{y}(a) \quad \text{subject to} \quad \widehat{C}_\rho(\pi \mid x_r) \leq D_r, \qquad x_{j+1} = \widehat{T}(x_j, a_j). \tag{12}$$

The graph state x_j combines the revealed request state with its lattice cell. Semantic ranking remains inside S. The objective prices verified candidates that can reach the next ranking step within the wall-time budget.

A.2 Service-state order and monotone envelope

For $\ell = (z_1, \ldots, z_d)$ and $\ell' = (z'_1, \ldots, z'_d)$, the product order is

$$\ell \preceq \ell' \quad \Longleftrightarrow \quad z_j \preceq_j z'_j, \quad \forall j. \tag{13}$$

Its coordinate-wise meet and join are

$$\ell \wedge \ell' = (\min\{z_1, z_1'\}, \ldots, \min\{z_d, z_d'\}), \qquad \ell \vee \ell' = (\max\{z_1, z_1'\}, \ldots, \max\{z_d, z_d'\}). \quad (14)$$

Every $\mathcal{Z}_j$ is a finite chain induced by an ordered service boundary, so both operations remain in $\mathcal{Z}_j$. Absorption, commutativity, and associativity hold coordinate by coordinate. Min and max distribute over one another on a total order, which makes $\mathcal{L} = \prod_j \mathcal{Z}_j$ a finite distributive lattice under Equation 13. The order records stage execution states, so a generator-heavy state and a verifier-heavy state can remain incomparable even when their measured latencies are similar.

The coordinates follow observed service transitions. The generator coordinate changes with stable shifts in decoding shape or sequence service, the verifier coordinate follows scoring batches and text length, and the KV coordinate captures residency that affects the following round. The product order records service commitment along each axis without forcing incomparable stage states into a scalar ranking.

Before execution, the request exposes a query $\widehat{\Phi}(u, a)$ rather than a completed trace. Let $\mathcal{L}_{\text{prof}}$ be the cells with measured profiles and define

$$\mathcal{D}(u, a) = \left\{ \ell \in \mathcal{L}_{\text{prof}} : \widehat{\Phi}(u, a) \preceq \ell \right\}. \quad (15)$$

The query uses

$$\ell^{\uparrow}(u, a) \in \operatorname{Min}_{\preceq} \mathcal{D}(u, a), \quad (16)$$

where stable coordinate order resolves multiple minimal cells. The lookup moves only upward in service commitment; it never substitutes a cheaper cell below the requested generator, verifier, or KV state.

Let $\downarrow \ell = \{\ell' : \ell' \preceq \ell\}$ be the lower set of ℓ. The empirical envelope in Equation 2 can be written as

$$\overline{c}(\ell) = \max_{\ell' \in \downarrow \ell} Q_\rho(c \mid \ell'). \quad (17)$$

The construction is evaluated separately for generator and verifier time. We write $\overline{c}^G(\ell)$ and $\overline{c}^V(\ell)$ when the stage matters, and use $\overline{c}(\ell)$ for the same argument applied to either stage.

The envelope is monotone on this order. If $\ell_1 \preceq \ell_2$, then $\downarrow \ell_1 \subseteq \downarrow \ell_2$, and hence

$$\overline{c}(\ell_1) = \max_{\ell' \in \downarrow \ell_1} Q_\rho(c \mid \ell') \le \max_{\ell' \in \downarrow \ell_2} Q_\rho(c \mid \ell') = \overline{c}(\ell_2). \quad (18)$$

Equations 16 and 18 preserve the direction of the service order for an unprofiled combination of boundaries. Taking the maximum over the lower set also prevents a sparsely observed richer cell from appearing cheaper merely because its expensive predecessor was sampled elsewhere. Boundary neighbors can therefore be compared within one ordered state even when their generator and verifier contributions differ.

A.3 Policy-preserving nested allocation

For a requested width B, let $\mathbf{m}(B)$ be the child-count vector after $B - K_r$ applications of Equation 3. Initialization sets

$$\mathbf{m}(K_r) = \mathbf{1}, \qquad \sum_{i=1}^{K_r} m_i(K_r) = K_r. \quad (19)$$

At an expansion step, the selected index i_B receives one unit:

$$\mathbf{m}(B+1) = \mathbf{m}(B) + \mathbf{e}_{i_B}, \qquad i_B = \arg\min_i \frac{m_i(B)}{q_{r,i}}, \quad (20)$$

which is equivalent to Equation 3 because $e_i = m_i - 1$. Stable identifiers resolve ties and make the sequence deterministic.

Let $s_{i,j}$ be the stable identity of the jth child of parent i. The executed sample set at width B is

$$\mathcal{U}(B) = \bigcup_{i=1}^{K_r} \{s_{i,j} : 1 \le j \le m_i(B)\}. \quad (21)$$

Algorithm 1 Parent-preserving frontier compilation and screening.

Require: Parents P_r, shares $\mathbf{q}_r$, legal structure Γ_S, maximum width $B_{\max}$, state (u_r, x_r)
Ensure: Ordered retained choices $\mathcal{C}_r$
$\mathbf{m} \leftarrow \mathbf{1}$; initialize stable samples $\mathcal{U} \leftarrow \{s_{i,1}\}_{i=1}^{K_r}$
Initialize a heap with key $(m_i/q_{r,i}, \mathrm{id}(p_{r,i}))$
for $B = K_r, \ldots, B_{\max}$ **do**
  $a_B \leftarrow (\mathbf{m}, \mathcal{U})$ and $\ell_B \leftarrow \ell^{\uparrow}(u_r, a_B)$
  Record $\widehat{y}(a_B)$, $\widehat{T}(x_r, a_B)$, and stage envelopes; append a_B to $\mathcal{F}_r$
  if $B < B_{\max}$ **then**
    Pop the minimum-key parent i whose increment remains in $\Gamma_S(P_r)$
    $m_i \leftarrow m_i + 1$; $\mathcal{U} \leftarrow \mathcal{U} \cup \{s_{i,m_i}\}$; update its heap key
  end if
end for
$\mathcal{C}_r \leftarrow \{a_{K_r}\}$
for adjacent $a_{B-1}, a_B \in \mathcal{F}_r$ **do**
  if $\chi_r(a_{B-1}) \neq \chi_r(a_B)$ **then**
    Add both actions to $\mathcal{C}_r$
  end if
end for
Within each predicted successor, add its cost–yield nondominated actions to $\mathcal{C}_r$
return $\mathcal{C}_r$ in increasing-width order

The action $a_B = (\mathbf{m}(B), \mathcal{U}(B))$ therefore fixes both child counts and sample identities. For screening, each action is profiled at $\ell_B = \ell^{\uparrow}(u_r, a_B)$ and carries the signature

$$\chi_r(a_B) = \left(\ell_B, \overline{c}^G(\ell_B), \overline{c}^V(\ell_B), \widehat{T}(x_r, a_B)\right). \tag{22}$$

Changes in this signature mark a service boundary or a different predicted successor. Algorithm 1 constructs the complete nested family $\mathcal{F}_r$ and retains the choices that can change planning.

Induction on $u = B - K_r$ establishes parent-set preservation and nestedness. For $u = 0$, Equation 19 gives every selected parent one child, so the parent set is preserved. Assume $\mathbf{m}(B)$ has positive coordinates and sums to B. Equation 20 changes exactly one coordinate by $+1$ and leaves all others unchanged. Hence every coordinate remains positive, the new sum is $B + 1$, and

$$\mathbf{m}(B+1) - \mathbf{m}(B) = \mathbf{e}_{i_B}. \tag{23}$$

Stable child identifiers couple the samples across widths, so the executed units satisfy

$$\mathcal{U}(B) \subset \mathcal{U}(B+1), \qquad |\mathcal{U}(B+1) \setminus \mathcal{U}(B)| = 1. \tag{24}$$

Applying the update inside each policy-defined group preserves $\Gamma_S(P_r)$: Beam Search has one global group, DVTS has one group per subtree, and Dynamic Branching includes parents with positive target counts. Thus every compiled width preserves the selected parent set, remains legal under its policy, and extends the preceding width.

The heap priority also tracks the shares requested by the policy. A parent whose cumulative count falls behind its target ratio reaches the top first and receives the next slot. The same construction therefore maintains parent coverage, proportional expansion, and sample nesting.

A heap stores the current priority $m_i(B)/q_{r,i}$. Initialization costs $O(K_r)$, and every additional unit performs one extract-min and one insertion. Constructing the complete sequence through $B_{\max}$ consequently costs

$$O(K_r + (B_{\max} - K_r)\log K_r) \quad \text{time and} \quad O(K_r) \text{ memory}. \tag{25}$$

A.4 Removing envelope-equivalent choices

Screening compares actions only after preserving the minimum allocation and both sides of every signature change. Within a common predicted successor, a' dominates a when

$$\widetilde{c}_r(a') \leq \widetilde{c}_r(a), \qquad \widehat{y}(a') \geq \widehat{y}(a), \tag{26}$$

with at least one strict inequality. Actions with different successors remain available because they lead to different suffixes, even when their immediate cost and yield coincide. The adjacent-pair test in Algorithm 1 also retains the last action before a service transition and the first action after it.

Let a_1 and a_2 denote the compiled actions at $B_1 < B_2$, and consider an arbitrary feasible predicted suffix σ after a_1. Suppose a_1 and a_2 have the same corrected action cost and predicted successor state. The same suffix σ is therefore available after a_2, and replacing a_1 with a_2 does not change the predicted cost or residual term:

$$\widehat{C}_\rho((a_1, \sigma) \mid x_r) = \widehat{C}_\rho((a_2, \sigma) \mid x_r) \le D_r. \tag{27}$$

Nestedness retains every unit of a_1 in a_2. When the additional units are predicted to finish verification, their yield is nonnegative, so

$$\widehat{y}(a_2) + \sum_{a \in \sigma} \widehat{y}(a) \ge \widehat{y}(a_1) + \sum_{a \in \sigma} \widehat{y}(a). \tag{28}$$

Every feasible plan beginning with a_1 has a counterpart beginning with a_2 and no smaller objective. Under equal corrected cost and predicted successor state, a_2 weakly dominates a_1. The compiler prunes such redundant points from the nested sequence, while retaining actions at a boundary or with a different predicted successor. Keeping both sides of every observed boundary preserves the decision to stop before an additional batch or cross it when the verified yield justifies the cost.

A.5 Observed stage cost and suffix pricing

For stage $x \in \{G, V\}$, let O_r^x and P_r^x accumulate observed and profiled time through the rounds preceding r. A completed round updates

$$O_{r+1}^x = O_r^x + t_{r,\mathrm{obs}}^x, \qquad P_{r+1}^x = P_r^x + t_{r,\mathrm{base}}^x, \qquad \alpha_{r+1}^x = \frac{\epsilon + O_{r+1}^x}{\epsilon + P_{r+1}^x}. \tag{29}$$

With $O_0^x = P_0^x = 0$, the initial correction is one. Equal cumulative times keep $\alpha_r^x = 1$, while sustained underprediction gives $\alpha_r^x > 1$. Separate accumulators preserve the generator–verifier asymmetry exposed by the lattice; their corrected sum is the action cost in Equation 6.

For suffix π, write its corrected action sum as

$$S_r(\pi) = \sum_{a \in \pi} \widetilde{c}_r(a). \tag{30}$$

The planner adds a nonnegative residual reserve indexed by the current state, horizon length, and corrected-cost bucket:

$$\widehat{C}_\rho(\pi \mid x_r) = S_r(\pi) + R_\rho(x_r, |\pi|, b(S_r(\pi)))\,. \tag{31}$$

The reserve is charged once to the complete suffix. This treats prediction error at the request level and keeps costs comparable when candidate suffixes contain different numbers of rounds.

A.6 Online control across verified rounds

Let $\Pi_r^{(h)}$ contain predicted paths of h actions from state x_r, with each edge drawn from the retained choices at its source state. The feasible suffix set is

$$\Pi_r^{\mathrm{fit}} = \bigcup_{h=1}^{H} \left\{ \pi \in \Pi_r^{(h)} : \widehat{C}_\rho(\pi \mid x_r) \le D_r \right\}. \tag{32}$$

Longer paths are evaluated first; shorter paths remain available when the remaining deadline cannot support the full horizon. Algorithm 2 connects compilation, suffix selection, execution, and feedback at verification boundaries.

After executing the first action, the controller observes complete wall time and updates

$$D_{r+1} = \max\{0, D_r - c_r^{\mathrm{wall}}\}. \tag{33}$$

Algorithm 2 Deadline-aware control over verified search rounds.

Require: Search policy S, deadline D, horizon H, maximum width $B_{\max}$, service profiles
Ensure: Final answer selected from completed verification
$r \leftarrow 0$, $D_0 \leftarrow D$; initialize x_0 and $O_0^x = P_0^x = 0$ for $x \in \{G, V\}$
while $D_r > 0$ and S has an active frontier **do**
  $(P_r, \mathbf{q}_r) \leftarrow S$ applied to the latest verified scores
  Compile retained choices with Algorithm 1 at each reachable predicted state
  Build Π_r^{fit} from those choices through horizon H
  if $\Pi_r^{\mathrm{fit}} = \varnothing$ **then**
    break
  end if
  $\pi_r^\star \leftarrow \arg\max_{\pi \in \Pi_r^{\mathrm{fit}}} \sum_{a \in \pi} \widehat{y}(a)$
  Execute only the first action a_r of $\pi_r^\star$ through G and V
  Observe trace τ_r, stage times, wall time, and verifier scores
  Update S from the returned scores and update $O_{r+1}^x, P_{r+1}^x, \alpha_{r+1}^x$ using Equation 29
  $x_{r+1} \leftarrow \Phi(\tau_r)$; $D_{r+1} \leftarrow \max\{0, D_r - c_r^{\mathrm{wall}}\}$; $r \leftarrow r + 1$
end while
return the answer selected by S from completed candidates

The realized trace replaces the predicted state, updates both correction factors, and defines the next graph root. Resolving Equation 12 from this root gives a receding-horizon policy. At round $r + 1$, all candidate suffixes are compared against the same updated D_{r+1} and the latest service cell; only their first actions are committed before the calculation repeats. The suffix objective avoids a locally attractive expansion that consumes the deadline before its descendants can be scored. Committing one action is enough: the resulting trace reveals the next stage cost and successor state, and the remaining actions are valued again from that evidence.

The horizon-layered state graph is acyclic. If the explored graph contains N_H states and E_H retained action edges, suffix evaluation takes $O(E_H)$ time and $O(N_H)$ memory once action costs and transitions are cached. Frontier construction contributes the separate cost in Equation 25. No predicted action after a_r is committed, so the next decision always uses the realized stage times, KV state, and verifier output from the current round.

B Service Behavior and Additional Results

Round traces expose the generator and verifier transitions represented by the service lattice and show where they change the amount of verified work returned to search. The results below follow that connection through boundary isolation, choice-set pruning, control cost, and longer deadlines.

B.1 Tracing complete rounds

Service traces. Each completed round records generator and verifier shapes and times, post-round KV residency, verified candidates, the realized successor state, and complete wall time. Stable changes in generator service, verifier service, and cache residency define the coordinates of $\mathcal{L}$ within a model-pair and length profile. An unseen tuple uses the minimal profiled cell that dominates it before the envelope in Equation 17 is evaluated.

Search-policy binding. The compiler receives selected parent identities and their target child shares. Beam Search uses one allocation group. DVTS retains the policy's subtree partition and grows child counts within each subtree. Dynamic Branching admits parents with positive target counts and normalizes those counts into shares. Stable child identifiers make each richer allocation extend the samples in the preceding one.

Metrics and deadlines. For request i, end-to-end model time sums generator and verifier time over its executed rounds,

$$T_i^{\mathrm{phase}} = \sum_r (t_{i,r}^G + t_{i,r}^V). \tag{34}$$

Table 5: Latency multiplier when one child crosses a service boundary.

Model pair	Boundary	Generator	Verifier	Generator + verifier
1.5B–7B	Sequence	1.197×	1.004×	1.048×
	Verifier	1.029×	1.097×	1.060×
7B–1.5B	Sequence	1.261×	1.004×	1.210×
	Verifier	1.002×	1.162×	1.011×

Table 6: Effect of retaining the active boundary. B/U/A denotes beneficial, unchanged, and adverse outcomes; Δ is the change in on-time verified candidates.

Model pair	Boundary retained (%)	Decision change (%)	Replay outcome B/U/A (%)	Problem outcome B/U/A (%)	Mean replay Δ	Mean problem Δ
1.5B–7B	100.0	5.59	93.2/1.4/5.4	93.8/0.0/6.3	+1.60	+1.71
7B–1.5B	100.0	1.82	94.3/0.0/5.7	68.8/31.3/0.0	+2.94	+1.33

Complete wall time also includes tokenization, communication, policy selection, planning, and finalization. PRM accuracy uses the answer selected by the verifier rule; oracle accuracy records whether any completed, verified candidate contains the correct answer. Their time-normalized goodputs are

$$G_{\mathrm{PRM}} = \frac{\sum_i \mathbf{1}[\widehat{a}_i = a_i]}{\sum_i T_i^{\mathrm{phase}}}, \qquad G_{\mathrm{oracle}} = \frac{\sum_i \mathbf{1}[\exists c \in \mathcal{C}_i : c = a_i]}{\sum_i T_i^{\mathrm{phase}}}. \tag{35}$$

For deadline D, fixed-deadline accuracy is

$$A_D = \frac{1}{N} \sum_i \mathbf{1}[\widehat{a}_i = a_i \ \wedge \ T_i^{\mathrm{wall}} \le D]. \tag{36}$$

For each model pair and initial frontier, D_q sets D to $q\%$ of the held-out FastTTS P95 for that operating point.

B.2 Discrete service boundaries across stages

A one-unit change in logical width can move either model stage into a new service regime. To isolate that transition, the replay fixes generated work and verifier text and compares b with $b + 1$ around a profiled sequence or verifier boundary. The two stage times are measured separately. The measured multipliers are summarized in Table 5.

At sequence boundaries, generator latency rises by 19.7% and 26.1%, while verifier latency changes by 0.4%. Verifier boundaries instead raise verifier latency by 9.7% and 16.2%. Logical width hides which stage crossed a costly transition; separate lattice coordinates retain that information for frontier selection.

B.3 Boundary information guides useful work

Two planners start from the same recorded state and remaining deadline. One can choose the neighborhood around the active boundary; the other cannot. Their outcomes reveal whether the local service transition changes the evidence that reaches search. Table 6 summarizes these outcomes.

Boundary information changes 5.59% and 1.82% of decisions for the two model pairs. More than 93% of changed replays favor the boundary-aware choice. At the problem level, its effect is beneficial or unchanged in 93.8% and 100.0% of cases. The active boundary affects a small part of the choice space, but those local changes return more work before the next search decision.

B.4 Retaining the useful child counts

Most legal integer widths share a cost envelope, so online planning need not retain them all. The compact set keeps the minimum parent set, boundary neighbors, successor-state changes, and non-dominated cost–yield choices. Replay compares this set with every legal width; exhaustive feasible-

Table 7: Choice-set compression and decision quality.

Model pair	Choice set	Median choices	Reduction	Regret	Δ regret
1.5B–7B	All widths	497	1.0×	0.396	–
	Retained	6	82.8×	0.405	+0.009
7B–1.5B	All widths	498	1.0×	0.441	–
	Retained	6	83.0×	0.446	+0.005

Table 8: Planner latency across planning horizons (ms), using all legal widths or the retained choices.

Horizon	All widths P50	All widths P95	Retained P50	Retained P95
1	6.355	7.191	0.070	0.096
2	6.545	7.460	0.299	0.437
4	6.921	8.631	1.233	1.884
8	7.380	12.793	2.527	7.996

path enumeration provides the corresponding two-step planning decision. Table 7 summarizes the resulting retained choice set.

Regret is normalized against the best action after every legal width has been replayed. The planner chooses before execution from profiled cost and expected yield, whereas this reference uses the realized outcome of each replay. Restricting the median choice set from roughly 500 widths to six adds only 0.009 and 0.005 regret, an $82.8\times$–$83.0\times$ reduction with little additional decision loss.

B.5 Planner and telemetry cost

Online replanning runs between model phases. Recorded planning states expose how its cost changes with horizon and maximum width. To isolate telemetry cost, trace-on and trace-off runs execute the same fixed-shape generator and verifier choices.

Table 8 shows that the retained choice set keeps planning latency low as the look-ahead horizon grows. At the default horizon of four, P95 latency is 1.884 ms, compared with 8.631 ms when every legal width is enumerated. Even at horizon eight, retaining boundary and nondominated choices keeps the P50 at 2.527 ms. The compression therefore leaves room to value later verification rounds without putting exhaustive search on the request path.

The maximum frontier affects exhaustive planning more directly because each new width adds another candidate action. Table 9 measures this scaling at the default horizon.

As Table 9 shows, exhaustive P95 latency increases from 8.615 ms at maximum width 4 to 875.712 ms at width 512. The retained planner reaches 48.631 ms at the same endpoint. Its growth follows the small set of service-boundary and successor-state changes rather than every integer width, which preserves online replanning at the largest frontier.

Table 10 separates action construction and profile lookup from the search over candidate suffixes.

The stage breakdown in Table 10 places most control cost in suffix search rather than state collection. State description, profile query, and child construction each remain below 0.6 ms at P99. Pruning lowers the search P95 from 9.592 to 1.943 ms while using essentially the same peak memory, so the benefit comes from comparing fewer meaningful choices rather than simplifying the request state.

Table 9: Planner latency as maximum width grows at horizon 4 (ms).

Maximum width	4	8	16	32	64	128	256	512
All widths P50	6.566	13.016	25.286	50.768	99.021	201.413	403.576	784.373
All widths P95	8.615	15.332	28.568	58.906	108.830	227.878	455.773	875.712
Retained P50	1.856	2.056	2.544	4.950	2.674	11.224	16.502	25.897
Retained P95	2.929	3.008	4.158	7.467	4.480	20.865	34.378	48.631

Table 10: Planner-stage latency and memory at horizon 4.

Stage	Latency (ms)			Memory (KiB)		
	P50	P95	P99	P50	P95	Peak
State descriptor	0.069	0.152	0.214	8.03	8.53	8.59
Profile query	0.178	0.319	0.427	5.72	5.81	5.84
Child construction	0.264	0.350	0.528	17.05	17.05	17.05
Pruned planner	1.233	1.943	2.694	77.80	117.45	144.62
Exhaustive planner	6.999	9.592	19.487	66.65	125.05	145.74

Table 11: End-to-end control-plane overhead.

Model pair	Planner/wall P50 (%)	Planner/wall P95 (%)	Planner/stage P95 (%)	Maximum wall share (%)	Telemetry delta (%)
1.5B–7B	0.1961	0.4260	0.4415	0.8882	-0.0155
7B–1.5B	0.1648	0.4441	0.4587	1.5159	-0.1588

The request-level effect is reported in Table 11.

Across complete requests, Table 11 places planner time below 0.45% of wall time at P95 for both model pairs. Its maximum observed share is 1.52%, and the telemetry delta, $(T_{\text{on}} - T_{\text{off}})/T_{\text{off}}$, stays within 0.16% of zero. The controller thus preserves the model-time savings measured in the main experiments after its own planning and trace collection are included.

B.6 Extended deadline response

Table 12 follows the planner beyond the D_{25}–D_{100} range. Each deadline produces an independent execution: an early allocation changes the scores returned to the policy, which can retain different parents and send the subsequent search along a different trajectory. The complete trajectories are therefore not nested across deadlines, even though the allocations available at each decision are nested. Accuracy and verified counts may consequently vary rather than increase monotonically. Accuracy and on-time columns are percentages.

All requests finish within these extended budgets. Because each deadline follows its own search trajectory, additional time does not simply append candidates to the shorter-budget run. End-to-end model time and mean verified candidates nevertheless vary over a much narrower range than in the D_{25}–D_{100} region of Figure 5. The selected choices have reached a stable service region, so a longer deadline no longer produces sustained frontier growth.

B.7 Concurrency mechanics and local sensitivity

Synchronized-window concurrency. When a synchronized window releases C requests, each keeps its own search state, lattice cell, correction factors, and remaining deadline. The backend

Table 12: Extended deadline response for LatticeTTS. E2E is median model time.

Initial width	Budget	PRM (%)	Majority (%)	Oracle (%)	On-time requests (%)	E2E (s)	Mean verified candidates
128	D_{150}	15.3	19.4	27.8	100.0	33.31	32.2
	D_{200}	16.7	13.9	25.0	100.0	34.17	33.8
	D_{300}	13.9	16.7	26.4	100.0	35.50	35.7
512	D_{150}	12.5	18.1	38.9	100.0	99.56	130.2
	D_{200}	13.9	15.3	30.6	100.0	101.57	142.7
	D_{300}	13.9	16.7	34.7	100.0	101.33	138.5

Table 13: Local sensitivity around the default controller. Time ratio is default/variant; Δ PRM is variant minus default. Bold rows mark the default value.

Parameter	Value	Time ratio	Δ PRM (pp)
Planning horizon	1	0.840	0.00
	2	0.890	−1.39
	4	**1.000**	**0.00**
	8	0.962	1.39
Minimum width	2	0.955	5.56
	4	**1.000**	**0.00**
	8	0.998	2.78
	16	0.957	1.39
Completion priority	1	0.960	2.78
	2	0.956	2.78
	4	**1.000**	**0.00**
	8	1.023	2.78

batches compatible generator or verifier phases while each planner selects its request-local frontier. Throughput counts completed requests per window time; P95 uses complete per-request wall time.

Ablation and local sensitivity. The one-round variant values the current choice without a future suffix, the no-exit variant omits deadline-conditioned termination, and the static variant keeps its initial frontier plan. These changes isolate suffix evaluation, deadline-aware completion, and online replanning. For local sensitivity, each run changes one controller parameter while keeping the other two at their default value of four. Table 13 reports the resulting wall-time ratio and change in PRM-top accuracy at D_{50}.

Table 13 identifies planning horizon as the clearest control on latency: horizons one and two yield time ratios of 0.840 and 0.890, because they assign too much budget to the immediate expansion. Once the horizon reaches four, looking farther ahead changes little. Minimum width and completion priority remain close to the default time across the local sweep, while several choices increase PRM-top accuracy. Together with the component ablation, these results show that preserving time for a later verified round is the central planning decision; the nearby allocation parameters leave a broad operating region around it.