IDLESERVE: A SYSTEM LAYER FOR IDLE RESOURCE UTILIZATION IN LLM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM serving systems retain capacity for peak demand, leaving opportunities to prepare useful information between requests. However, background preparation can delay live traffic, and its outputs may be irrelevant or incorrect. We present IdleServe, a serving system that coordinates idle capacity with the preparation of reusable knowledge for future agentic requests. IdleSensor estimates preparation budgets from serving telemetry and controls background admission and cancellation. IdleTaskBuilder schedules preparation tasks, checks generated artifacts on past requests, and uses outcome feedback to guide subsequent allocation. Applications supply task definitions and adapters; IdleServe manages execution and the resulting knowledge. Implemented on vLLM, the system supports procedures, worked examples, and application references without updating model weights. Across four open-weight models and three agentic benchmarks, evaluations using prepared knowledge improve task score by 4.9 points on average, with a maximum gain of 18.5 points. Separate live-server replays of recorded model-call sequences reduce p95 episode latency by 8.7% on average and increase SLO attainment by 1.8 points. Component experiments show that relevant content and verification contribute to quality. Without admission and cancellation, background work increases p95 latency by up to 238% relative to serving without background work; IdleServe's controls keep p95 latency near that baseline. Together, these results support a serving-system approach to turning idle capacity into useful preparation for later requests.

1 INTRODUCTION

Agentic requests repeatedly inspect the same database schemas, API conventions, and task procedures. This discovery consumes model and tool calls on the user's critical path, even when earlier requests needed the same information. Meanwhile, LLM services retain capacity to meet latency service-level objectives (SLOs) during peak demand. Bursty traffic leaves that capacity underused between peaks (Stojkovic et al., 2025; Wang et al., 2025a; Xiang et al., 2026). In our replay of a production traffic day, the GPU is idle about 43% of the time (Section 2). These recurring information needs create a use for spare capacity: prepare knowledge before the next request needs it. For example, a database agent can prepare a reusable schema reference between requests and supply it to later queries over that database. Without such preparation, the service repeatedly pays for discovery while available compute passes unused.

Turning this opportunity into a serving capability requires two decisions. **When can preparation run without delaying live requests?** A quiet interval is not a reservation: demand can return before background work finishes. A forecast can guide a compute budget, but execution must also respond to current demand. An agent waiting for a tool does not imply that its shared serving GPU is idle. **Which prepared knowledge is worth serving?** Future requests are unknown, and completing more preparation does not ensure better answers. Irrelevant context can distract an agent, while an incorrect procedure can turn a solvable request into a failure. Useful preparation therefore requires both resource control and checks on the knowledge supplied to later requests.

Prior work establishes ways to create reusable information. Skills, workflow memory, and sleep-time compute carry useful context across requests (Wang et al., 2024; 2025b; Lin et al., 2025); proactive methods anticipate future needs or prepare an agent's next step (Hu et al., 2026; Choi

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.

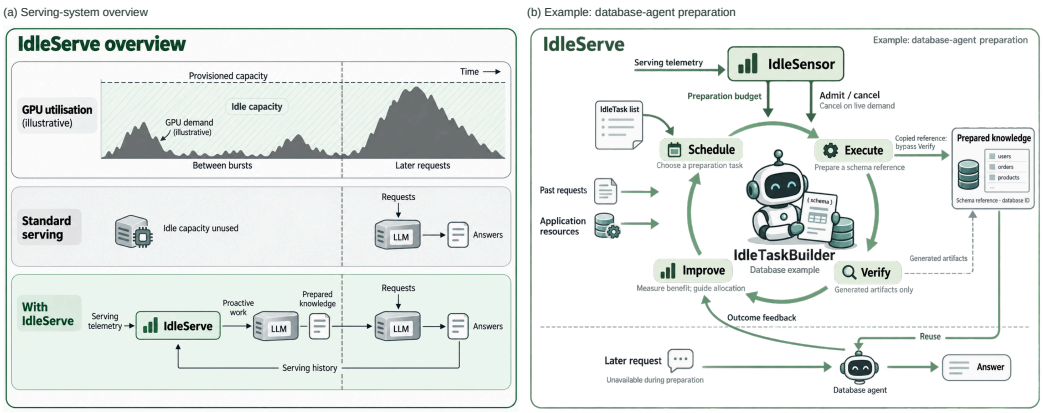

Figure 1: IdleServe overview and an illustrative application. (a) Capacity retained for peak demand can prepare knowledge between requests. (b) A database-agent example: IdleSensor budgets and controls preparation; IdleTaskBuilder prepares a schema reference for reuse by database ID. Copied references bypass Verify; generated artifacts are checked on past requests. Future requests are unavailable during preparation. Their later outcomes guide reuse and allocation, while live demand triggers background cancellation.

et al., 2026). Co-serving systems manage a different part of the problem: scheduling supplied offline workloads alongside live requests (Qiao et al., 2026; Sun et al., 2025). Connecting these capabilities requires a runtime that accounts for preparation cost, regulates execution, and tracks whether its outputs help the service's later requests. **IdleServe coordinates two decisions: when background preparation may run, and which prepared knowledge is useful enough to serve to later requests.** Its contribution is a common serving interface and lifecycle through which applications can run preparation tasks under resource control and measure their downstream value.

We present IdleServe, a serving system with two components (Figure 1). IdleSensor forecasts spare capacity, estimates preparation budgets, and admits background work when the serving engine is idle. It initiates cancellation when live demand returns. IdleTaskBuilder manages an IdleTask list through Schedule, Execute, Verify, and Improve. It prepares knowledge from serving history and application resources, checks generated artifacts on past requests, and uses outcome feedback to guide subsequent allocation. Applications supply task definitions and adapters; the system manages execution and the resulting knowledge. Procedures, worked examples, and application references share this interface. References copied from the environment bypass generated-artifact verification. The implementation runs on vLLM without updating model weights.

Our evaluation distinguishes preparation quality, serving performance, and component behavior. Across four open-weight models and three agentic benchmarks, evaluations using prepared knowledge improve task score by 4.9 points on average, with a maximum gain of 18.5 points. Separate live-server replays of recorded model-call sequences reduce p95 episode latency by 8.7% and increase SLO attainment by 1.8 points on average across the twelve configurations. Component experiments test the mechanisms behind these results. IdleSensor releases $1.94\times$ the usable capacity of a fixed reservation at matched overcommit risk. Without admission and cancellation, background work raises p95 latency by up to 238% relative to serving without background work; IdleServe's controls keep p95 near that baseline. Content and verification controls show that relevant knowledge and checks on generated artifacts contribute to quality. Together, these experiments establish the benefits of prepared knowledge and the controls needed to execute preparation alongside live traffic.

2 Motivation: Idle Capacity as a Serving Resource

Capacity retained for responsiveness has a second use. Peak provisioning and bursty demand create a persistent gap between available capacity and average load. DynamoLLM documents this gap in production LLM serving, while HyGen shows the opportunity for colocating offline

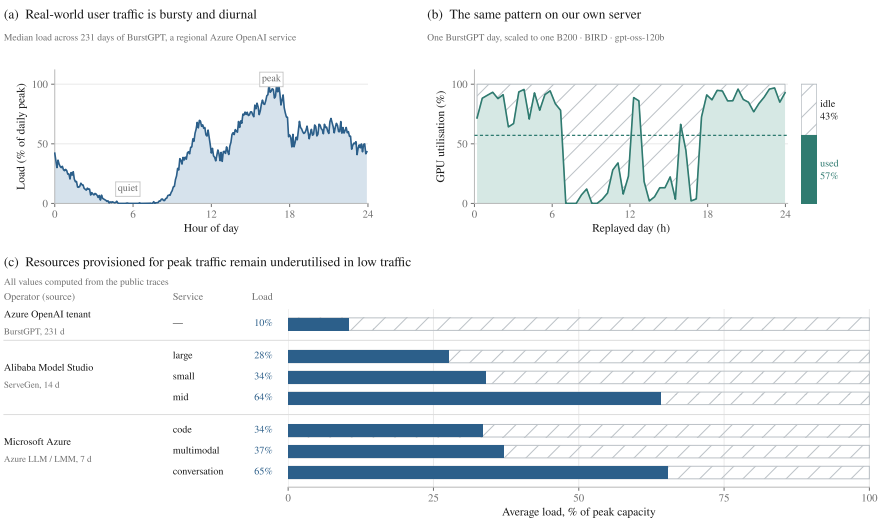

Figure 2: Evidence of resource underutilization. (a) BurstGPT traffic exhibits extended quiet periods between daily peaks. (b) A replay on our server leaves approximately 43% GPU idle time. (c) Across public serving traces, average estimated demand falls well below capacity provisioned for busy periods. For (c), capacity is the 99th percentile of estimated token-processing demand within each trace; provider GPU allocations are not observed. Panel (b) uses BIRD requests with gpt-oss-120b on one B200, replaying a BurstGPT day scaled to the server. Sources: BurstGPT, ServeGen, and Azure LLM/LMM (Wang et al., 2025a; Xiang et al., 2026; Stojkovic et al., 2025; Qiu et al., 2026). Trace durations in (c) are the days our binning covers in each release, not the span each source reports.

work (Stojkovic et al., 2025; Sun et al., 2025). IdleServe targets deployments that retain serving capacity through these quieter intervals. The question is how to put that capacity to work for the service's own future requests.

Figure 2 quantifies the opportunity. BurstGPT exhibits long quiet periods between daily peaks. Across the public traces in panel (c), average estimated demand occupies 10–65% of capacity sized to each trace's 99th-percentile demand. This is a common provisioning model for comparison across traces. Panel (b) measures the opportunity on hardware: replaying a BurstGPT day with BIRD requests leaves the GPU idle about 43% of the time. Across the twelve serving replays used in our evaluation, model-alone GPU utilization is 28–67%; with preparation load admitted into the gaps, it is 87–98%.

Idle time is available, but it is not owned by the background task. A GPU can have unused space in a running batch or intervals with no work executing. IdleServe targets the latter. These intervals can be harvested in a layer above the engine, while systems such as ConServe manage finer-grained colocation within it (Qiao et al., 2026). The difficulty is that a background task cannot know when the next live request will arrive. A budget based on expected spare capacity must therefore be paired with a mechanism that yields when demand returns. This is the role of IdleSensor.

Reusable knowledge makes spare capacity valuable to the service. Consider a database agent. Different questions can require the same tables, keys, and value conventions. Preparing that reference once can avoid rediscovering it during every episode. The same pattern appears in application APIs and recurring task procedures (Patil et al., 2024; Wang et al., 2025b). Useful preparation can improve decisions and remove model calls from the critical path at the same time. Its benefit depends on what is prepared, which requests reuse it, and whether it is correct. IdleTaskBuilder manages these decisions through the deployment's serving history and outcome feedback. Together, the two components turn idle time into a resource for improving the service, with live requests retaining priority.

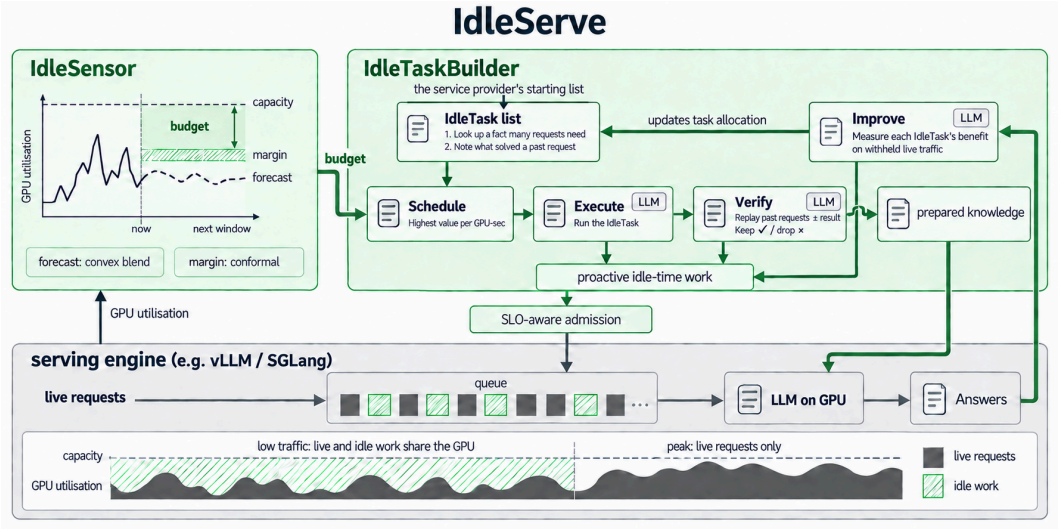

Figure 3: IdleServe architecture. IdleSensor supplies a preparation budget; IdleTaskBuilder manages the application-supplied IdleTask list through Schedule, Execute, Verify, and Improve. Generated knowledge passes Verify before reuse; copied references follow the bypass in Figure 1(b). Outcome feedback updates allocation and reuse within the supplied list. The engine strip illustrates resource use over time: background work is admitted only on an empty engine and cancelled when live demand returns.

3 IdleServe Design

IdleServe is a serving layer with two components (Figure 3). IdleSensor determines how much background work can be planned and when it may execute. IdleTaskBuilder manages the tasks that consume this capacity and the knowledge they produce. The application supplies an initial IdleTask list and an adapter; the system supplies budgeting, execution control, verification, and feedback. This separates the definition of a proactive task from the serving decisions needed to run it.

3.1 IdleSensor: budget preparation and protect live requests

Estimate capacity from serving telemetry. IdleSensor converts processed prompt and completion tokens into estimated GPU time. Separate prefill and decode rates are fitted from the lower latency envelope observed by the server. For a bin t with estimated demand d_t and capacity C in the same units, the spare fraction is $s_t = \mathrm{clip}(1 - d_t/C, 0, 1)$. The estimate gives a common resource unit for preparation across tasks with different prompt and output lengths.

Turn a forecast into a compute budget. The target is average spare capacity over the next H bins. IdleSensor combines recent-load persistence, an exponentially weighted level, and daily profiles in a convex forecast $\hat{s}_{t,H} = \sum_j w_j f_j(t, H)$, with $w_j \geq 0$ and $\sum_j w_j = 1$. Weights are refitted daily from observed history. A point forecast alone is too optimistic for committing work. IdleSensor adds an adaptive lower margin (Gibbs & Candès, 2021) using the signed residuals $e_{t,H} = s_{t,H} - \hat{s}_{t,H}$ from elapsed horizons. Let $q_{t,H}$ be their empirical α_t-quantile. The resulting budget is

$$B_{t,H} = HC_{\mathrm{GPU}}\ \mathrm{clip}(\hat{s}_{t,H} + q_{t,H}, 0, 1), \tag{1}$$

where C_{GPU} is capacity in GPU-seconds per bin. The lower quantile is normally negative and reduces the committed allowance. To adapt the margin, IdleSensor measures the mean miss rate $\widehat{m}$ once per non-overlapping block of $H + 1$ forecast origins and updates

$$\alpha_{t+1} = \mathrm{clip}\big(\alpha_t + \gamma(\alpha - \widehat{m}),\ 10^{-3},\ 0.5\big)\,. \tag{2}$$

We use target non-coverage $\alpha = 0.10$ and step $\gamma = 0.02$. Too many misses lower the quantile level and make the allowance more conservative. The budget is distributed across the horizon's bins.

Admit work only when the engine can give way. The budget controls total preparation; SLO-aware admission controls immediate interference. Background execution starts only when no live request is in flight and the engine has no running or queued requests. When live demand returns, the layer closes the background stream at its next cancellation check. Live requests thus retain priority even when arrivals depart from the forecast. This separation is central: a forecast plans useful work, while admission responds to demand at execution time. Both operate above an unmodified serving engine.

3.2 IDLETASKBUILDER: MANAGE THE LIFECYCLE OF PROACTIVE TASKS

A common task interface. Each IdleTask specifies a preparation instruction, a scope, and an optional execution tool. The adapter exposes context and recurring keys, such as a database, application, or task category. It also attaches prepared knowledge to matching requests and measures their outcomes. The same system can therefore manage model-written procedures, harvested examples, and extracted references through four stages.

Schedule and Execute. Schedule allocates the interval's budget according to estimated benefit per GPU-second. Benefit is measured by request class and weighted by its share of traffic. Untried tasks receive priority; built but unmeasured tasks receive an optimistic prior that decays with preparation spend. Measured benefit then guides allocation. Execute runs the selected instruction through the model or its tool and stores the result under the relevant key. Preparation, verification, and failed attempts are charged to the same budget. The task list defines available work; serving history determines which keys need it and provides the material from which it is built.

Verify before serving generated knowledge. A plausible procedure can still harm an agent. Verify tests a candidate on K past requests, comparing outcomes with and without it. These paired checks produce differences δ_i. The default rule admits the candidate when

$$\bar{\delta} - z\ \mathrm{SE}(\delta) > 0, \qquad \mathrm{SE}(\delta) = s_\delta/\sqrt{K}, \tag{3}$$

with $z = 1$ and $K = 4$. Verification requests come from the learning history and are disjoint from evaluation requests. References copied directly from the environment, such as API documentation or database schemas, bypass model-output verification. This lets the system spend its checking budget on knowledge whose correctness depends on generation.

Improve through measured reuse. IdleTaskBuilder withholds each task's knowledge from a random 25% of eligible requests and estimates benefit from the served-minus-withheld outcomes. Estimates are tracked by task and request class, then fed back to Schedule. Once sufficient observations show negative benefit, the knowledge is withdrawn from most traffic. A small served fraction continues to measure whether it becomes useful again. Thus the system manages knowledge after creation as well as before deployment. The four stages connect preparation cost, reusable output, and downstream outcomes without updating model weights.

4 EXPERIMENTAL SETUP

We evaluate the system along three dimensions: the quality of requests served with prepared knowledge, serving performance under background load, and the contribution of each component.

Models, workloads, and proactive tasks. Our vLLM-based prototype serves four open-weight models: gpt-oss-120b, Llama-3.3-70B-Instruct, Llama-3.1-8B-Instruct, and Gemma-3-27B (Kwon et al., 2023; Agarwal et al., 2025; Grattafiori et al., 2024; Team, 2025). All use greedy decoding. The workloads span household task completion (ALFWorld), application APIs (AppWorld), and agentic text-to-SQL (BIRD) (Shridhar et al., 2021; Trivedi et al., 2024; Li et al., 2023). ALFWorld tasks prepare procedures, object-location notes, and worked examples from solved requests. AppWorld and BIRD tasks extract API references and database descriptions from the environment. These are application-specific IdleTask lists managed by the same two system components.

Table 1: Task score, p95 episode latency and SLO attainment for all twelve configurations, for the model alone and with IdleServe. Higher score and SLO, and lower p95, are better. The SLO target is three times the model-alone median latency. *: paired 95% bootstrap interval excludes zero.

	Model			IdleServe + Model		
Model	Score (%)	p95 (s)	SLO (%)	Score (%)	p95 (s)	SLO (%)
ALFWorld						
gpt-oss-120b	66.5	155.4	83.3	72.5*	142.0	84.9
Llama-3.3-70B	52.0	121.4	95.4	70.5*	87.6*	100.0*
Gemma-3-27B	73.2	97.8	91.8	74.2	99.7	90.0
Llama-3.1-8B	30.8	51.9	91.6	32.0	56.4	92.1
AppWorld						
gpt-oss-120b	37.1	106.8	97.6	45.5*	136.5	94.7
Llama-3.3-70B	41.1	155.1	100.0	44.1	134.4*	100.0
Gemma-3-27B	34.8	246.8	87.3	35.6	175.1	96.4*
Llama-3.1-8B	20.2	68.7	89.8	21.8*	75.8	82.0
BIRD						
gpt-oss-120b	45.5	50.3	87.8	51.9*	49.9	89.6
Llama-3.3-70B	47.7	27.1	96.2	51.1	17.5*	99.5*
Gemma-3-27B	41.7	68.0	79.5	47.7*	46.6*	90.2*
Llama-3.1-8B	35.6	15.9	85.1	37.9	14.9	87.2*

Quality protocol. A learning stream precedes an evaluation stream. Each evaluation request is served with and without prepared knowledge in the same process, providing a paired model-alone baseline. The evaluation streams contain 400 requests on ALFWorld, 130 on AppWorld, and 264 on BIRD. Preparation is charged against the trace day's observed spare capacity; forecasting is evaluated separately (Appendix A.2). Verification uses only learning requests. ALFWorld uses the default rule except Llama-3.3-70B, which uses an earlier positive-gain threshold. Environment references are served without replay verification. AppWorld/Llama-3.1-8B retains references for the first eight model calls; other configurations retain them throughout the episode. BIRD agents can inspect schemas through tools (3,000 characters per response), while preparation supplies up to 6,000 characters ahead of the query. Outcome feedback and 25% withholding remain enabled during evaluation, so scores include the cost of withholding.

Serving protocol. We replay recorded model-call sequences, prompt lengths, and completion limits against a live vLLM server. Arrivals follow a time-scaled BurstGPT day (Wang et al., 2025a), calibrated toward 90% peak server load. Background requests replay preparation-call shapes under IdleSensor's admission. Episode latency runs from scheduled arrival to the last model-call completion, including queueing; benchmark tools are not executed in this replay. The SLO target is three times the model-alone median episode latency, fixed across arms. Admission controls hold foreground requests and arrivals fixed, varying only the background policy. Hardware and system constants appear in Appendix A.1.

Metrics and uncertainty. We report task score, p95 episode latency, and SLO attainment. Task score is success on ALFWorld, execution accuracy on BIRD, and the fraction of unit tests passed on AppWorld, all on a 0–100 scale. Paired 95% bootstrap intervals resample games on ALFWorld, scenarios on AppWorld, and questions on BIRD; serving intervals resample paired episodes within a replayed day. Matrix averages weight the twelve configurations equally. Score and SLO gains are percentage points; latency changes are relative unless marked in seconds.

5 RESULTS

Table 1 compares the model alone with IdleServe across all twelve model–benchmark configurations. The system improves task quality across the matrix and reduces latency on average in the separate serving replays.

Table 2: The two components, isolated with gpt-oss-120b. (A) The same live requests under each background policy; the interpolated row prices a fixed worker cap at the background work IdleServe completes. (B) Each arm against its own paired baseline on the same evaluation requests. *: 95% interval excludes zero.

(A) Without IdleSensor: admission policies on fixed foreground requests.

	ALFWorld		AppWorld		BIRD	
Background policy	p95 (s)	SLO (%)	p95 (s)	SLO (%)	p95 (s)	SLO (%)
None (Model)	161.0	83.0	120.2	96.8	60.8	87.0
Throttle, 1 worker	176.3	78.7	127.9	95.0	65.7	84.4
Throttle at equal work (interp.)	193.4	–	132.9	–	66.7	–
IdleSensor's admission (IdleServe)	161.7	83.5	116.1	97.3	58.9	87.8

(B) Without IdleTaskBuilder's checks: content and verification controls.

Gain vs. own baseline	ALFWorld (400 req.)	AppWorld (130)	BIRD (264)
Real prepared knowledge	$+6.75\,[+0.26, +13.53]^*$	$+8.44\,[+1.60, +15.05]^*$	$+6.44\,[+2.65, +10.23]^*$
same knowledge, wrong key	$-1.75\,[-8.46, +5.05]$	$+3.66\,[+0.02, +7.48]^*$	$-2.65\,[-6.44, +1.14]$
same length, no content	$-9.50\,[-15.29, -3.72]^*$	$+0.01\,[-5.14, +5.41]$	$-4.17\,[-7.58, -1.14]^*$
model-written, not verified	$+1.25\,[-6.14, +8.60]$	–	–

Idle serving capacity supports better task outcomes. Task score increases in all twelve configurations, by 4.90 points on average $[+2.61, +7.90]$. Six individual gains are significant under the paired intervals. The largest gain is 18.5 points on ALFWorld with Llama-3.3-70B, raising success from 52.0% to 70.5%. With gpt-oss-120b, IdleServe adds 6.0 points on ALFWorld, 8.4 on AppWorld, and 6.4 on BIRD. The same serving system supports these gains across different kinds of proactive work: learning task procedures, retaining solved examples, and preparing application references.

Preparation can also shorten the request's critical path. In the live-server replays, p95 episode latency falls in eight of twelve configurations, with an average reduction of 8.7%. SLO attainment rises in eight, by 1.8 points on average. On BIRD with Gemma-3-27B, p95 falls from 68.0 s to 46.6 s while SLO attainment rises from 79.5% to 90.2%. On ALFWorld with Llama-3.3-70B, p95 falls from 121.4 s to 87.6 s, with 100% SLO attainment in the replay. The mechanism is fewer steps to complete a request. With gpt-oss-120b, ALFWorld episodes average 26.8 turns with prepared knowledge versus 29.9 without it; AppWorld averages 19.1 model calls versus 26.1. The serving benefit depends on how preparation changes the episode, while background interference is controlled by IdleSensor. We isolate that distinction next.

6 Component Analysis

The two components address different failure modes. IdleSensor must turn spare capacity into executable work while preserving foreground performance. IdleTaskBuilder must turn that work into useful knowledge. The following experiments isolate these responsibilities.

6.1 IdleSensor: usable capacity with foreground priority

Adaptive budgets release more capacity at matched risk. At the same overcommit risk, IdleSensor releases $1.94\times$ the usable idle capacity of a fixed reservation across 60 trace settings. It releases $1.48\times$ across 36 settings from a second trace family. On BurstGPT, lower-bound coverage is 89.8%, 89.9%, and 91.2% at the three forecast horizons, against a 90% target (Appendix A.2). These results establish the resource-planning benefit of adapting to demand.

Admission and cancellation prevent costly interference. Holding foreground requests fixed, background work that never pauses raises p95 latency by 238%, 72%, and 82% on ALFWorld, AppWorld, and BIRD (Table 3). With admission and cancellation, p95 remains near its no-background level while 3,767, 2,819, and 1,070 background requests complete. ALFWorld shows why both execution controls matter: admission alone yields 117.8 s p95 versus 108.7 s without background work; adding cancellation brings it to 106.8 s.

Table 3: Background policies on the same live requests and arrival schedule: 110 replayed episodes on ALFWorld (Gemma-3-27B), 217 on AppWorld and 588 on BIRD (gpt-oss-120b). With IdleSensor's admission and cancellation, background work completes 3,767, 2,819 and 1,070 requests.

	ALFWorld / Gemma-3-27B		AppWorld / gpt-oss-120b		BIRD / gpt-oss-120b	
Background policy	p95 (s)	SLO (%)	p95 (s)	SLO (%)	p95 (s)	SLO (%)
None (Model)	108.7	90.0	113.2	96.8	53.0	85.7
Never pauses	367.1	39.1	194.8	76.5	96.2	69.9
Admission only	117.8	84.5	106.2	98.2	52.5	86.7
Admission + cancellation	106.8	90.9	103.4	98.2	52.4	87.1

A fixed worker cap also leaves interference unresolved (Table 2(A)). Even one background worker raises p95 on all three workloads. Interpolating the fixed-cap frontier to match IdleServe's completed background work gives 31.7 s, 16.8 s, and 7.8 s higher p95. The system must respond to live demand, rather than merely limit the number of background workers.

6.2 IdleTaskBuilder: relevant knowledge, checked before use

The gain comes from useful content. Table 2(B) compares real knowledge with same-length padding and knowledge assigned to the wrong key, using gpt-oss-120b. Real knowledge gives the largest score gain on every benchmark. On ALFWorld it gains $+6.75$ points, versus -1.75 for the wrong key and -9.50 for padding. On BIRD the correct schema gains $+6.44$, versus -2.65 and -4.17. On AppWorld the correct API reference gains $+8.44$, versus $+3.66$ and $+0.01$. The runtime's useful output is knowledge matched to recurring requests, not simply additional prompt text.

Verification improves the value of generated knowledge. With the same ALFWorld IdleTask list, verified knowledge gains $+6.75$ points and unchecked knowledge gains $+1.25$. The verification control uses the earlier acceptance rule; the default-rule result in Table 1 is $+6.00$. This experiment supports checking generated outputs before they enter live prompts. AppWorld and BIRD demonstrate the other execution path: preparing references directly from the environment. Together, these tasks show how one serving system can manage different sources of reusable knowledge.

Prepared knowledge complements engine-level reuse. With prefix caching enabled in both arms, prepared knowledge still lowers p95 by 22–25% on the Gemma-3-27B configurations (Appendix A.3). IdleServe prepares useful context; the engine's cache reduces the cost of processing it. Both can benefit the same request.

7 Related Work

Proactive computation and reusable experience. Sleep-time compute processes context before a query arrives; ProAct anticipates future needs from dialogue and memory; IdleSpec plans while an agent waits for tool results (Lin et al., 2025; Hu et al., 2026; Choi et al., 2026). These works establish useful forms of proactive computation. IdleServe addresses their system requirements on a shared serving engine: capacity estimation, execution control, and the management of outputs used by later requests. In particular, an agent's idle interval and its server's idle interval are different resources.

Experience-based methods provide further forms of preparation. Reflexion retains verbal feedback, Voyager builds verified skills, Agent Workflow Memory induces reusable workflows, and Agentic Context Engineering maintains structured context through generation, reflection, and curation (Shinn et al., 2023; Wang et al., 2024; 2025b; Zhang et al., 2026). These methods focus on how experience becomes useful context. IdleServe provides a task interface for running preparation within a serving budget and measuring its downstream value. Our procedures and examples exercise this interface; the system contribution is the common execution and knowledge lifecycle around them.

Knowledge access and prefix reuse. Text-to-SQL and tool-use research demonstrates the importance of schema and API information (Pourreza & Rafiei, 2023; Patil et al., 2024). IdleTaskBuilder uses recurring request keys to prepare such references ahead of demand. Prefix caching, SGLang's RadixAttention, and Prompt Cache reuse attention state for shared prompt content (Kwon et al., 2023; Zheng et al., 2024; Gim et al., 2024). IdleServe determines which knowledge to prepare and serve, so its contribution composes with the engine's reuse of that context (Appendix A.3).

Resource management for LLM serving. ConServe harvests spare capacity through fine-grained scheduling and preemption; HyGen uses interference-aware online–offline colocation; Valve bounds preemption latency and frequency (Qiao et al., 2026; Sun et al., 2025; Liu et al., 2026). Llumnix dynamically reschedules requests across serving instances (Sun et al., 2024). These systems improve execution of supplied workloads. IdleServe connects resource management to the preparation of knowledge for the service's own future requests. IdleSensor controls when preparation can run, while IdleTaskBuilder controls what is built, served, and measured. Engine-level co-serving can provide a complementary execution substrate for this system.

8 Scope and Limitations

IdleServe targets services with retained capacity, recurring request context, and measurable outcomes. The evaluation covers a fixed set of application adapters and preparation tasks. Quality runs and serving replays establish preparation benefits and execution behavior separately. Extending this system to long-running deployments with changing task lists and traffic is a natural next step. Tool waiting time and engine-level preemption offer further opportunities for harvesting capacity within the same two-component design.

9 Conclusion

IdleServe makes proactive work a serving-system responsibility. Capacity retained for peak demand can prepare knowledge that later requests would otherwise have to discover for themselves. IdleSensor budgets that work and controls its execution around live demand. IdleTaskBuilder manages its outputs through Schedule, Execute, Verify, and Improve. Across four models and three agentic benchmarks, evaluations using prepared knowledge raise task score by 4.9 points on average and up to 18.5 points. Separate live-server replays of recorded model-call sequences show 8.7% lower p95 latency and 1.8 points higher SLO attainment on average. These results demonstrate a practical role for idle capacity: improving the requests a serving system will handle next.

REPRODUCIBILITY STATEMENT

Section 3 states the decision rules of both components: the budget and its conformal update, admission, allocation, verification and withholding. Appendix A.1 gives their constants and the serving configuration of each model. Section 4 defines the quality and serving protocols, the SLO target and the resampling units. All three benchmarks and all traces are public. We will release the code and the run records.

ETHICS STATEMENT

Prepared knowledge should inherit the access controls, retention policies, and tenant boundaries of its source requests. Verification assesses utility on available history; deployment also requires the application's own safety controls. Proactive computation consumes energy, so operators should weigh its benefit against reducing power or scaling down during quiet periods. Our experiments use public benchmarks and traces, with no live production users or human-subject experiments.

AI USE STATEMENT

We used generative AI tools to aid and polish writing, including copy-editing and improving clarity, and to draft sections of the paper. We also used image-generation tools to refine the overview and methodology illustrations. We used generative AI for retrieval and discovery, including finding related work and checking citation details against source records. For research execution, we used AI coding assistance to implement and debug parts of the experimental harness and analysis scripts. The authors are responsible for the research decisions and for verifying all AI-assisted content, including the accuracy of citations, code, results and claims in the final paper.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Daewon Choi, Kyunghyun Park, Woomin Song, Saket Dingliwal, Sai Muralidhar Jayanthi, Jinwoo Shin, and Aram Galstyan. Idlespec: Exploiting idle time via speculative planning for llm agents. *arXiv preprint arXiv:2605.22154*, 2026.

Isaac Gibbs and Emmanuel J. Candès. Adaptive conformal inference under distribution shift. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In P. Gibbons, G. Pekhimenko, and C. De Sa (eds.), *Proceedings of Machine Learning and Systems*, volume 6, pp. 325–338, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/file/a66caa1703fe34705a4368c3014c1966-Paper-Conference.pdf.

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

Haoyi Hu, Qirong Lyu, Xianghan Kong, Weiwen Liu, Jianghao Lin, Zixuan Guo, Yan Xu, Yasheng Wang, Weinan Zhang, and Yong Yu. Anticipate and learn: Unleashing idle-time compute in proactive agents. *arXiv preprint arXiv:2605.25971*, 2026.

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

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357, 2023.

Kevin Lin, Charlie Snell, Yu Wang, Charles Packer, Sarah Wooders, Ion Stoica, and Joseph E Gonzalez. Sleep-time compute: Beyond inference scaling at test-time. *arXiv preprint arXiv:2504.13171*, 2025.

Fangyue Liu, Hua Liu, Xinyuan Lyu, Shuo Ai, Hao Liang, Lingpeng Chen, Ziqian Hu, Chong Zha, Xin Jin, Hanmei Luo, et al. Valve: Production online-offline inference colocation with jointly-bounded preemption latency and rate. *arXiv preprint arXiv:2604.07874*, 2026.

NVIDIA Corporation. NVIDIA system management interface (nvidia-smi). https://docs.nvidia.com/deploy/nvidia-smi/index.html, 2026. Accessed September 18, 2026.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 126544–126565. Curran Associates, Inc., 2024. doi: 10.52202/079017-4020. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/e4c61f578ff07830f5c37378dd3ecb0d-Paper-Conference.pdf.

Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 36339–36348. Curran Associates, Inc., 2023. doi: 10.52202/075280-1577. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/72223cc66f63ca1aa59edaec1b3670e6-Paper-Conference.pdf.

Yifan Qiao, Shan Yu, Shu Anzai, Haoran Ma, Shuo Yang, Yang Wang, Miryung Kim, Yongji Wu, Yang Zhou, Jiarong Xing, Joseph E. Gonzalez, Ion Stoica, and Harry Xu. Conserve: Fine-grained GPU harvesting for LLM online and offline co-serving. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=eKfWG67mZB.

Haoran Qiu, Anish Biswas, Zihan Zhao, Jayashree Mohan, Alind Khare, Esha Choukse, Íñigo Goiri, Zeyu Zhang, Haiying Shen, Chetan Bansal, Ram Ramjee, and Rodrigo Fonseca. Modserve: Modality- and stage-aware resource disaggregation for scalable multimodal model serving. In *Proceedings of the 2025 ACM Symposium on Cloud Computing*, SoCC '25, pp. 817–830, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400722769. doi: 10.1145/3772052.3772254. URL https://doi.org/10.1145/3772052.3772254.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2010.03768.

Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. Dynamollm: Designing llm inference clusters for performance and energy efficiency. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1348–1362. IEEE, 2025.

Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 173–191, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/sun-biao.

Ting Sun, Penghan Wang, and Fan Lai. Hygen: Efficient llm serving via elastic online-offline request co-location. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 14910–14937. Curran Associates, Inc., 2025. doi: 10.52202/085713-0502. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/15e92839359df6d22e80042e572538d4-Paper-Conference.pdf.

Gemma Team. Gemma 3. 2025. URL https://arxiv.org/abs/2503.19786.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models. *Transactions on Machine Learning Research*, 2024. URL https://mlanthology.org/tmlr/2024/wang2024tmlr-voyager/.

Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, et al. Burstgpt: A real-world workload dataset to optimize llm serving systems. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 5831–5841, 2025a.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 13–19 Jul 2025b. URL https://proceedings.mlr.press/v267/wang25bx.html.

Yuxing Xiang, Xue Li, Kun Qian, Yan Zhang, Wenyuan Yu, Ennan Zhai, Xin Jin, and Jingren Zhou. ServeGen: Workload characterization and generation of large language model serving in production. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pp. 1845–1859, Renton, WA, May 2026. USENIX Association. ISBN 978-1-939133-54-0. URL https://www.usenix.org/conference/nsdi26/presentation/xiang-servegen.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatgpt interaction logs in the wild. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 34590–34605, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/9421261e06f1a63a352b068f1ac90609-Paper-Conference.pdf.

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 *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

A Implementation Details

A.1 Method and implementation

Interface. A provider supplies each IdleTask's instruction, its scope and an optional tool. A workload adapter exposes the recurring request keys and their context, installs prepared knowledge, serves requests with it, and returns outcome measurements. Adapters that can replay past requests also support Verify.

Constants. Schedule gives an untried IdleTask priority, a built but unmeasured IdleTask the value $0.05\, e^{-g_j/600}/c_j$, and a measured one $\max(\hat{b}_j, 0)/c_j$, where c_j is its mean GPU-seconds per build, g_j the GPU-seconds spent on it, and $\hat{b}_j$ its traffic-weighted benefit. Verify defaults to $K = 4$ replays and $z = 1$. Improve withholds 25% of eligible requests and withdraws an IdleTask once it has 30 served and 30 withheld observations and $\hat{b}_j + \mathrm{SE}(\hat{b}_j) < 0$. IdleSensor uses five-minute bins and blends four predictors: the trailing mean of the last six bins, an exponentially weighted level with a one-day half-life, and the same time of day one and seven days earlier. Its weights are refitted daily under a Huber loss, and its margin uses $\alpha = 0.10$ and $\gamma = 0.02$.

Table 4: Serving configuration. vLLM 0.17.1, maximum model length 32,768, GPU memory utilization 0.85, one server per run, arms run one at a time on an otherwise idle server. Decoding is greedy (temperature 0). The harnesses call the completions API through a shim that forwards each call as one chat turn through the model's own chat template; gpt-oss-120b runs at reasoning effort "low". BIRD declares no tools and uses no tool-call parser.

Model	GPUs	Tool-call parser	Reference
gpt-oss-120b	1× B200 (MXFP4 release weights)	openai	Agarwal et al. (2025)
Llama-3.3-70B-Instruct	2× RTX PRO 6000, TP=2	llama3_json	Grattafiori et al. (2024)
Gemma-3-27B-it	1× RTX PRO 6000	hermes	Team (2025)
Llama-3.1-8B-Instruct	1× RTX PRO 6000	llama3_json	Grattafiori et al. (2024)

A.2 IdleSensor capacity evaluation

Table 5 gives each budget the spare capacity it would commit at the same risk (at most 1% of claimed spare capacity not delivered). IdleSensor releases more usable idle capacity than a fixed reservation in every setting, and its lower-bound coverage is near the 90% target on held-out traces.

Table 5: IdleSensor on public traces (Wang et al., 2025a; Zhao et al., 2024; Xiang et al., 2026). Settings combine trace partitions, load levels and horizons of 30 minutes, two hours and six hours on five-minute bins (ServeGen: ten-minute bins, 14 days each, the first seven building the daily profile).

Usable idle fraction at matched overcommit risk			
Trace settings	Fixed reservation	IdleSensor	IdleSensor / fixed
BurstGPT + WildChat (60)	0.273	0.531	1.94× (60 of 60)
ServeGen (36)	0.249	0.369	1.48× (36 of 36)
Lower-bound coverage (%) against a 90% target			
Trace settings	30 min	2 h	6 h
BurstGPT (48)	89.8	89.9	91.2
ServeGen (18)	90.3	90.9	88.4

A.3 Prefix caching

Table 6 enables the engine's prefix cache in both arms. Prepared knowledge keeps its latency benefit with the cache on.

Table 6: Engine prefix caching enabled in both arms: the cached model alone against the cached model with IdleServe, under the same replay schedule.

		p95 (s)		SLO (%)	
Workload	Model	APC	APC + IdleServe	APC	APC + IdleServe
BIRD	Gemma-3-27B	46.0	35.7	77.5	87.9
AppWorld	Gemma-3-27B	113.5	85.1	96.3	96.3
BIRD	gpt-oss-120b	45.9	46.0	87.2	91.4

A.4 MOTIVATION FIGURE MEASUREMENTS

Panel (a) normalizes each BurstGPT day by its mean load and shows the median profile across days. Panel (b) is one replayed day on our server; the idle fraction is the time with no executing kernel, as reported by the NVIDIA utilization counter (NVIDIA Corporation, 2026). Panel (c) weights each bin's prompt and completion tokens by rates measured on our server, and plots mean demand over its 99th percentile for each trace.