JANUS: BREAKING LAYER-BOUND KV-CACHE LAYOUTS TO ACCELERATE LOCAL AGENT INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

As local agent conversations grow longer, the key-value (KV)-cache can exceed commodity GPU memory, making KV-cache offloading necessary but introducing substantial host-to-device (H2D) transfer latency. Sparse self-speculative decoding (SSP) is a promising way to reduce transfer latency because the sparse KV-cache can be small enough to fit on GPU during drafting, while the full KV-cache is fetched only to verify multiple candidate tokens together. However, the straightforward combination of SSP and offloading (naive SSP) incurs redundant H2D transfers and layout rematerialization. This is because existing layer-bound layouts cannot accommodate the different KV-cache requirements of draft and verification, preventing reuse of overlapping KV-pages. We propose JANUS, a local agent inference system whose layer-decoupled KV-page abstraction manages physical KV-pages in a unified GPU pool to enable cross-phase reuse. To keep large-scale metadata lookups and updates efficient, a coalesced metadata manager processes them through batched and fused primitives. To use the limited GPU page budget efficiently, a hierarchical GPU KV-page allocator first selects the verification staging depth and then calibrates the sparse draft budget offline. We implement JANUS on vLLM V1 and evaluate three LLMs on two commodity GPUs across three agent workloads. Compared with baselines, JANUS reduces time per output token (TPOT) by 26.2%–78.0% and end-to-end latency by 18.8%–61.8%.

1 INTRODUCTION

As LLMs become more capable, LLM agents are increasingly able to handle more complex tasks, such as document processing, file management, and information retrieval (Agashe et al., 2025; Bonatti et al., 2024; Xie et al., 2024). To use these capabilities while keeping personal data private, deploying these agents locally on personal computers with commodity GPUs is a promising approach (Wen et al., 2025; Erdogan et al., 2024). These local agents require low-latency inference to ensure a positive user experience (Hua et al., 2025; Ye et al., 2026a).

Meeting this latency requirement becomes challenging as agent conversations grow longer. LLM responses and tool outputs accumulate with each interaction, increasing the context length and the key-value (KV)-cache size required for attention (Zhou et al., 2026; Ye et al., 2026b; Gao et al., 2024). When the KV-cache exceeds GPU memory, KV-cache offloading stores it in CPU memory (Sheng et al., 2023; Lee et al., 2024), but repeated host-to-device (H2D) transfers of the full KV-cache slow autoregressive decoding. Sparse self-speculative decoding (SSP) offers a promising way to reduce this transfer latency while preserving the model output. It uses the same LLM to draft multiple candidate tokens with a sparse subset of the KV-cache and verify them together using full KV-cache (Sadhukhan et al., 2025; Sun et al., 2024b). This makes SSP a natural match for offloading, as sparse KV-cache can fit on GPU during drafting, while full KV-cache is fetched only for verification. Verifying multiple candidates together can therefore reduce the transfer latency per output token. We call this straightforward combination naive SSP.

However, naive SSP does not fully realize this potential due to a mismatch between its KV-cache requirements and existing layer-bound layouts. Drafting requires partial KV-cache across all layers, whereas verification stages full KV-cache for only a few layers at a time. Existing layouts allocate space separately to each layer and cannot directly reallocate it across layers to accommodate the different KV-cache requirements of draft and verification. Naive SSP therefore rematerializes layouts

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.

when switching phases. Although draft uses a subset of the full KV-cache required by verification, naive SSP reloads these overlapping KV-pages instead of reusing those already on GPU.

Based on the above analysis, we argue that **local agent inference under KV-cache offloading must break the existing layer-bound KV-cache layout to enable cross-phase reuse.** The key is to decouple physical KV-pages from a fixed organization across layers and tokens, allowing draft and verification to reuse pages already on GPU.

Building on this insight, we propose JANUS, an efficient local agent inference system with SSP under KV-cache offloading. Its layer-decoupled KV-page abstraction manages physical KV-pages in a unified GPU pool, enabling cross-phase reuse without materializing separate layouts. Supporting this reuse requires large-scale page lookups and mapping updates, which a coalesced metadata manager accelerates through batched and fused primitives. To use the limited GPU page budget efficiently, a hierarchical GPU KV-page allocator first selects the verification staging depth and then calibrates the sparse draft budget offline.

We summarize our contributions as follows.

- We identify how layer-bound layouts prevent naive SSP's cross-phase reuse of overlapping KV-pages, causing redundant H2D transfers and repeated layout rematerialization.
- We propose JANUS, a local agent inference system that enables cross-phase KV-page reuse for SSP under KV-cache offloading while keeping metadata overhead low and efficiently allocating the limited GPU page budget.
- We implement JANUS on the vLLM V1 inference engine and evaluate three LLMs on two commodity GPUs across three agent workloads. Compared with naive SSP, JANUS reduces time per output token (TPOT) by 26.2%–53.0% and end-to-end latency by 18.8%–46.7%. Compared with autoregressive inference with KV-cache offloading, it reduces TPOT by 38.0%–78.0% and end-to-end latency by 31.0%–61.8%.

2 PRELIMINARIES

2.1 LOCAL AGENT INFERENCE UNDER MEMORY CONSTRAINTS

We refer to agents whose LLM inference runs locally as local agents. A local agent usually serves one user request at a time. For a new request, the LLM first performs prefill, processing the input tokens and storing their keys and values in the key-value (KV)-cache. During decoding, the LLM generates new tokens in an autoregressive (AR) manner by reusing and extending the KV-cache.

As the conversation continues, LLM responses and tool outputs are appended to the conversation history and included in subsequent inputs (Yao et al., 2022), increasing the context length and KV-cache size (Kwon et al., 2023). To manage this KV-cache while reducing memory fragmentation, modern serving systems often organize it into fixed-size pages, each storing KV-cache entries for a block of tokens (Kwon et al., 2023; Ye et al., 2025; Tang et al., 2024). However, even with paged KV-cache management, commodity GPUs often cannot keep the KV-cache of a long-context agent conversation on device, making offloading necessary (Mao et al., 2026; Jie et al., 2025).

2.2 KV-CACHE OFFLOADED INFERENCE

Offloading stores inference states (*e.g.,* model weights and KV-cache) that exceed GPU memory in CPU memory and fetches them back to GPU through host-to-device (H2D) copies over the CPU-GPU interconnect (*e.g.,* PCIe) when needed (Sheng et al., 2023; Lee et al., 2024). For long-context LLM inference, the KV-cache often drives memory pressure because it grows with context length while model weights remain fixed (Ge et al., 2025; Li et al., 2025). We therefore focus on KV-cache offloading and treat weight offloading as orthogonal. Although KV-cache offloading solves the capacity problem, fetching the full KV-cache via H2D copies at each step slows inference.

2.3 SPARSE SELF-SPECULATION WITH KV-CACHE OFFLOADING

Whereas KV-cache offloading addresses memory capacity constraints, sparse self-speculative decoding (SSP) accelerates long-context inference. It uses the same LLM in two modes: a sparse draft

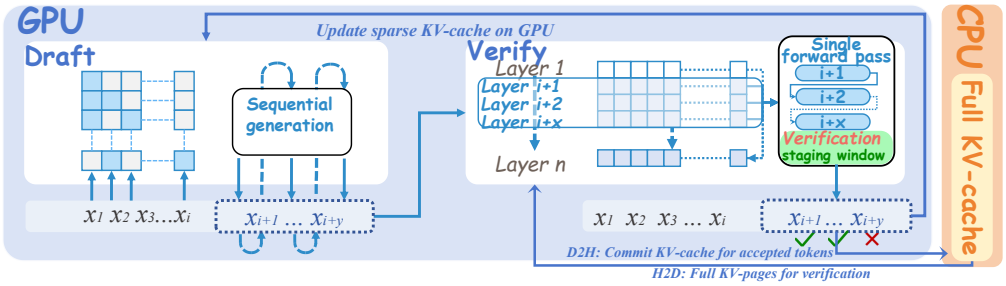
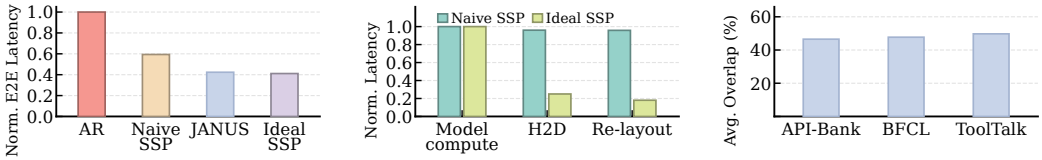


Figure 1: Sparse self-speculation combined with KV-cache offloading.



(a) Latency comparison (b) Latency breakdown (c) KV-page overlap ratio

Figure 2: (a) End-to-end latency under the same system configuration. (b) Decode latency breakdown. (c) Cross-phase KV-page overlap, averaged over draft and verification.

mode proposes multiple candidate tokens using partial KV-cache, while a full verify mode checks them using full KV-cache (Sadhukhan et al., 2025; Zhao et al., 2026). Since only verified tokens are accepted, the final output remains identical to standard inference (Leviathan et al., 2023). To maintain high acceptance as the context evolves, SSP reselects the sparse KV-cache after verification and reuses it for the following draft steps until the next verification (Zhao et al., 2026).

From the memory perspective, SSP is a natural match for KV-cache offloading because the sparse KV-cache can be small enough to fit on GPU during drafting, while the full KV-cache is fetched only during verification. As shown in figure 1, the draft phase uses the sparse KV-cache stored on GPU to generate candidate tokens, and the sparse KV-cache is reselected after verification. During verification, because GPU memory cannot hold the full verification KV-cache at once, the required full KV-cache pages are loaded from CPU memory into a rotating GPU staging window, which holds full KV-cache for only a few layers at a time and is refilled as verification proceeds layer by layer. We refer to the above straightforward combination as naive SSP.

3 Motivation

As discussed in section 2.3, SSP appears to be a promising match for KV-cache offloading to accelerate local agent inference. However, naive SSP does not fully realize this potential. We trace this limitation to a layout mismatch that prevents KV reuse across the draft and verification phases.

3.1 Naive Combination Fails

We evaluate naive SSP to assess the latency benefits of this combination. As shown in figure 2a, although naive SSP improves over autoregressive inference with KV-cache offloading, it remains 44.4% slower than ideal SSP. Ideal SSP performs the same computation as naive SSP but fetches only the required KV-cache pages that are not already on GPU. This gap shows that naive SSP remains far from the potential performance under KV-cache offloading.

3.2 Root Cause: Layout Mismatch

To explain this gap, we compare their decode latency breakdowns in figure 2b. Compared with ideal SSP, naive SSP incurs 283.7% higher H2D KV-cache transfer latency and 431.1% higher layout rematerialization latency, while their model computation time remains unchanged.

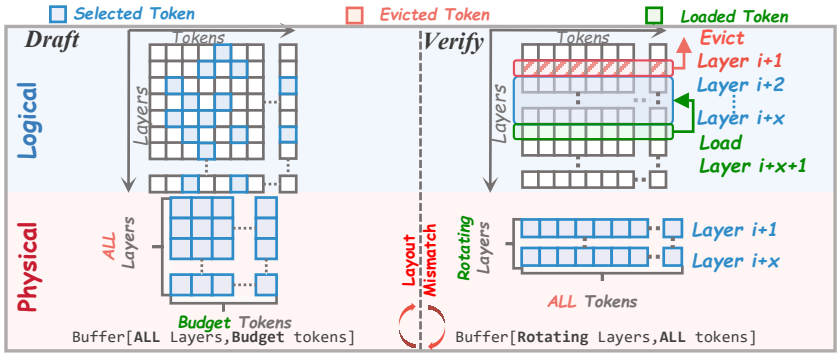


Figure 3: Illustration of layout mismatch between drafting and verification.

These additional costs arise from a mismatch between the KV-cache requirements of draft and verification and the existing layer-bound layout. As shown in figure 3, drafting uses an all-layer sparse layout that keeps partial KV-cache for every layer, whereas verification uses a layer-window dense layout that keeps full KV-cache for only a few layers at a time and rotates across layers.

Existing systems use layer-bound KV-cache layouts that allocate space separately to each layer at initialization, with a fixed layer count and maximum token count per layer. A layer therefore cannot directly use another layer's space. This suits autoregressive inference, where both counts remain fixed during inference. However, draft and verification require different KV-cache distributions across layers and tokens, which the layout cannot accommodate by directly reallocating space across layers. Naive SSP therefore repeatedly rematerializes the required layout when switching phases.

Although draft and verification have different KV-cache access patterns, they access overlapping KV-pages. The sparse KV-cache used by draft is a subset of the full KV-cache used by verification. As shown in figure 2c, the two phases have 46.5%–49.7% KV-page overlap, revealing a clear reuse opportunity. Yet naive SSP rematerializes these overlapping pages when switching between draft and verification, preventing the two phases from reusing the same physical KV-pages on GPU.

3.3 Implication: Break the Existing Layer-Bound KV-cache Layout

Motivated by this missed reuse opportunity, we argue that local agent inference under KV-cache offloading must break the existing layer-bound KV-cache layout to enable cross-phase reuse. The key is to decouple physical KV-pages from a fixed organization across layers and tokens, so that the same pages can be reused across both phases.

This implication exposes three challenges for system design. First, existing inference systems allocate KV-cache space separately to each layer. Supporting the different KV-cache access patterns while enabling cross-phase page reuse requires a new memory abstraction. Second, layer decoupling removes the fixed association between physical KV-pages and layers. The system therefore must explicitly track each page's logical position across all layers and tokens, making large-scale metadata lookups and updates a potential bottleneck. Third, draft and verification share physical KV-pages within a limited GPU page budget. Using more GPU pages for verification may leave fewer for reuse during drafting. The system should allocate this budget to preserve cross-phase reuse while balancing potential draft acceptance gains against fragmented-copy and metadata overhead.

4 Design

4.1 Overview

Building on this motivation, we propose Janus, an efficient local agent inference system with SSP under KV-cache offloading. Janus consists of three components: (1) a layer-decoupled KV-page abstraction that manages physical KV-pages in a unified GPU pool for cross-phase reuse; (2) a coalesced metadata manager that accelerates KV-page metadata lookups and updates through batched

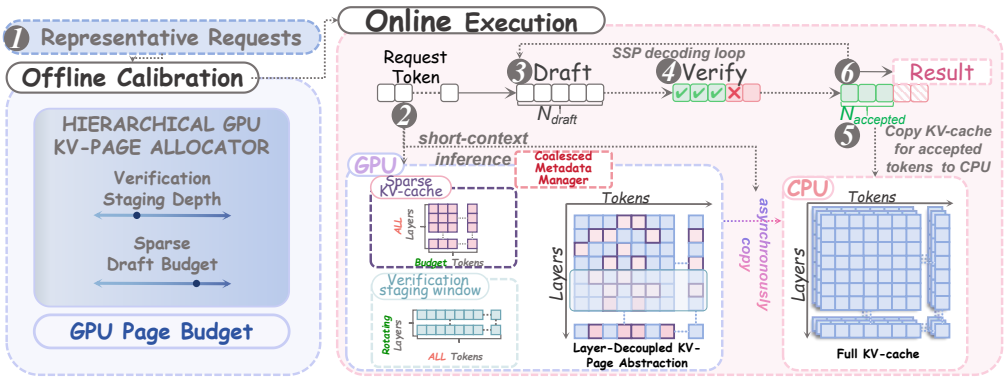


Figure 4: JANUS overview.

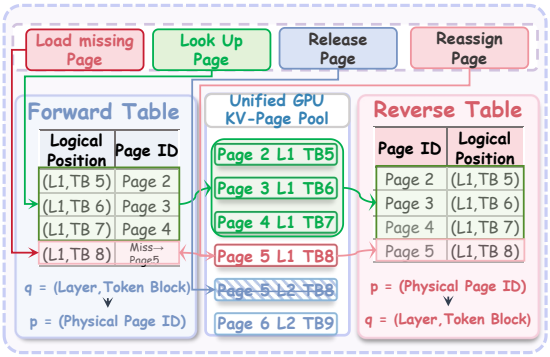


Figure 5: Layer-decoupled KV-page abstraction.

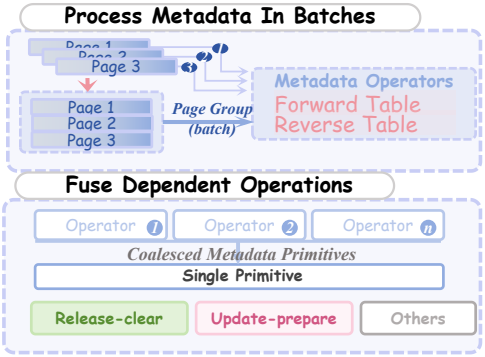


Figure 6: Coalesced metadata manager.

and fused primitives; (3) a hierarchical GPU KV-page allocator that determines the verification staging depth and sparse draft budget within the limited GPU page budget.

As shown in figure 4, JANUS operates across offline calibration and online execution. Before deployment, JANUS selects the verification staging depth based on the pipeline analysis and calibrates the sparse draft budget on representative requests ①. During online execution, JANUS keeps the full KV-cache on GPU for short-context inference and asynchronously copies it to CPU memory, so the KV-cache is already available in CPU memory when offloading is triggered ②. Once execution enters offloaded SSP, the draft phase queries the coalesced metadata manager to reuse the required KV-pages already on GPU or load missing pages, and then generates candidate tokens ③. The verification phase similarly queries the metadata manager for full KV-cache pages, and JANUS streams them through the GPU staging window while overlapping H2D transfers with computation ④. After verification, KV-cache updates for accepted tokens are asynchronously copied back to CPU memory ⑤. Draft and verification then alternate in the local agent loop until generation finishes ⑥.

4.2 LAYER-DECOUPLED KV-PAGE ABSTRACTION

As discussed in section 3.2, existing layer-bound layouts prevent naive SSP from directly reusing overlapping KV-pages across draft and verification. To address this problem, JANUS introduces a layer-decoupled KV-page abstraction that manages physical KV-pages in a unified GPU pool instead of allocating KV-cache space separately to each layer, as shown in figure 5. The KV-cache for a given layer and token block can then be placed in any available page in the pool. To maintain mappings between logical $(layer, token\ block)$ positions and physical KV-pages, JANUS uses (1) a forward table to locate requested physical KV-pages on GPU using their logical positions and (2) a reverse table to identify each physical KV-page's logical position. It maintains (3) mapping consistency between the two tables.

Forward table. During draft and verification, JANUS needs to determine whether the required KV-cache is already on GPU and where it is stored. It therefore introduces a forward table that maps each logical $(layer, token\ block)$ position to a physical KV-page in the unified GPU pool. If a requested KV-page is found on GPU, JANUS uses its physical id to reuse the page for attention computation; otherwise, it loads the required KV-cache from CPU memory.

Reverse table. Besides page lookup, JANUS releases pages no longer needed on GPU and reassigns occupied pages to make room for new KV-cache. For candidate physical pages, JANUS needs to identify the KV-cache they store to decide which pages can be released or reassigned. The forward table records these mappings, but is indexed by logical positions rather than physical page ids. A naive solution is to scan the forward table to find the logical position that maps to the target physical page, but this requires expensive metadata scans. Therefore, JANUS maintains a reverse table that maps physical page ids to logical positions, allowing direct mapping updates.

Mapping consistency. JANUS maintains consistency between the forward and reverse tables when KV-pages are loaded, released, or reassigned. For changes initiated from a logical position or a physical page, the corresponding table locates the matching entry in the other table. JANUS then clears or updates the mappings in both tables.

4.3 COALESCED METADATA MANAGER

JANUS uses KV-page metadata, including the forward and reverse tables, to identify reusable and missing pages and maintain mappings during loading, release, and reassignment. Draft and verification apply these operations to many pages across layers and tokens. Repeating lookup, update, and release page by page causes up to a 20.5× slowdown in section 5.3, making metadata management a bottleneck. As shown in figure 6, JANUS addresses it by first (1) identifying opportunities for batched and fused processing and then (2) introducing coalesced metadata primitives.

Opportunity for batched fusion. By examining metadata operations during draft and verification, we observe that they exhibit two useful structures. First, metadata requests are naturally batched. Specifically, JANUS queries one or more layers' KV-page metadata to find reusable and missing pages, then loads missing pages and updates their metadata. These lookups and updates therefore typically operate on page groups rather than individual pages. Second, key metadata operations often co-occur on the critical path. For example, releasing a physical KV-page requires a reverse table query followed by clearing the corresponding mappings, while loading missing pages requires updates to both tables. These structures offer opportunities to batch and fuse metadata operations.

Coalesced metadata primitives. Guided by these observations, JANUS coalesces dependent metadata operations over a page group into a single primitive. Each primitive is implemented as a fused kernel that combines metadata operations that usually occur together. Representative critical-path primitives include release-clear, which uses the reverse table to locate and clear old mappings when physical pages are released or reassigned; and update-prepare, which updates both mapping tables for newly allocated pages and prepares their physical page ids for attention. Other metadata operations follow the same coalesced execution style. Together, these primitives reduce metadata overhead by 32.4%–97.6% in section 5.3 through kernel fusion and parallel page-group processing.

4.4 HIERARCHICAL GPU KV-PAGE ALLOCATOR

Although the unified GPU pool enables cross-phase reuse, JANUS must allocate its limited GPU page budget between staging full KV-cache from CPU memory for verification and storing sparse KV-cache for drafting. JANUS allocates this shared budget hierarchically. It (1) determines the verification staging depth through pipeline analysis and (2) calibrates the sparse draft budget offline.

Verification staging allocation. Verification requires the full KV-cache, but GPU memory cannot hold it all at once for long contexts. JANUS therefore stages full KV-cache pages from CPU memory into a rotating GPU window. The staging depth S controls how many layers of full KV-cache can be kept on GPU at a time. A larger S may reduce H2D waits by prefetching more layers, but uses more GPU pages and can repeatedly flush KV-pages reusable during draft from the unified pool.

To determine the staging depth, we model verification as a pipeline between per-layer H2D transfer and computation. Let H and C denote the transfer and computation time of one layer, respectively,

and let L denote the number of layers. With $S = 1$, pages for the next layer cannot replace those of the current layer until its computation completes, giving

$$T_{\text{verify}}(1) \approx L(H + C). \tag{1}$$

With $S \geq 2$, the current layer can be computed while the next layer is loaded, giving

$$T_{\text{verify}}(S \geq 2) \approx H + (L - 1)\max(H, C) + C. \tag{2}$$

Comparing equation 1 and equation 2, increasing S from one to two enables transfer and computation overlap. Once this overlap is established, increasing S beyond two consumes more GPU pages without lowering the per-layer pipeline latency bound of $\max(H, C)$. JANUS therefore chooses $S = 2$, reaching the latency-budget Pareto point while leaving more GPU pages for draft KV-cache.

Sparse draft budget calibration. After determining the verification staging depth, JANUS calibrates the sparse draft budget B, which controls how much KV-cache the draft phase uses. A larger B may improve acceptance by providing more KV-cache context for drafting, but requires more pages in the unified GPU pool. As more of these pages occupy scattered positions, subsequent page loads become more fragmented and require more metadata operations.

This creates a trade-off in selecting B. An intuitive solution is to adjust B during online inference by probing different budget values. However, testing a new B requires loading a different set of KV-pages into GPU during drafting and may evict pages retained by the previous setting. Repeated probes therefore incur additional page transfers and metadata updates, which can outweigh the benefit of online adaptation. JANUS therefore calibrates B before deployment on a small set of representative long-context requests and uses the calibrated budget during inference. As shown in section 5.3, offline calibration reduces latency by 10.6% compared with online probing.

5 EVALUATION

5.1 EXPERIMENTAL SETUP

Implementation. We implemented JANUS on vLLM's V1 inference engine (Kwon et al., 2023), reusing its serving runtime and paged KV-cache infrastructure. We integrated sparse drafting and full verification into vLLM's speculative decoding pipeline and execute both phases over the layer-decoupled GPU KV-page pool. The coalesced metadata primitives are implemented as custom Triton kernels (Tillet et al., 2019). Dedicated CUDA streams overlap KV-cache transfers with model computation and asynchronously maintain a copy of the KV-cache in CPU memory. Our extensions comprise approximately 19K lines of runtime code, including about 0.8K lines of Triton kernels.

Hardware setup and models. We evaluate JANUS on two commodity GPUs, an NVIDIA RTX 4090 with 24GB VRAM and an NVIDIA RTX 5090 with 32GB VRAM. We evaluate Qwen3-8B (Yang et al., 2025) and Llama3.1-8B (Grattafiori et al., 2024) on the RTX 4090 and Qwen3-14B on the RTX 5090. These configurations represent practical settings for local agent inference.

Workload. We evaluate JANUS on three agent workloads, BFCL (Patil et al., 2025), API-Bank (Li et al., 2023), and ToolTalk (Farn & Shin, 2023). These benchmarks include common local agent tasks such as file management and information retrieval.

Baselines and key metrics. We compare JANUS against two baselines. AR performs autoregressive decoding with KV-cache offloading. Naive SSP combines the sparse self-speculation paradigm used in MagicDec (Sadhukhan et al., 2025) and SparseSpec (Zhao et al., 2026) with KV-cache offloading using the existing layer-bound KV-cache layout, with further details in Appendix A.1. We use a draft length of 6, as in prior SSP work (Sadhukhan et al., 2025; Sun et al., 2024b). For JANUS, we calibrate the sparse draft budget offline using 10 representative requests excluded from the evaluation set. For naive SSP, we tune the sparse draft budget through an offline parameter sweep. We use time per output token (TPOT) and end-to-end latency as the primary metrics and normalize both to AR.

5.2 END-TO-END EVALUATION

As shown in figure 7, across all evaluated configurations, JANUS reduces TPOT by 38.0%–78.0% compared with AR and by 26.2%–53.0% compared with naive SSP. It also reduces end-to-end latency by 31.0%–61.8% compared with AR and by 18.8%–46.7% compared with naive SSP. To understand these latency reductions, we conjecture that JANUS's advantage arises from two factors.

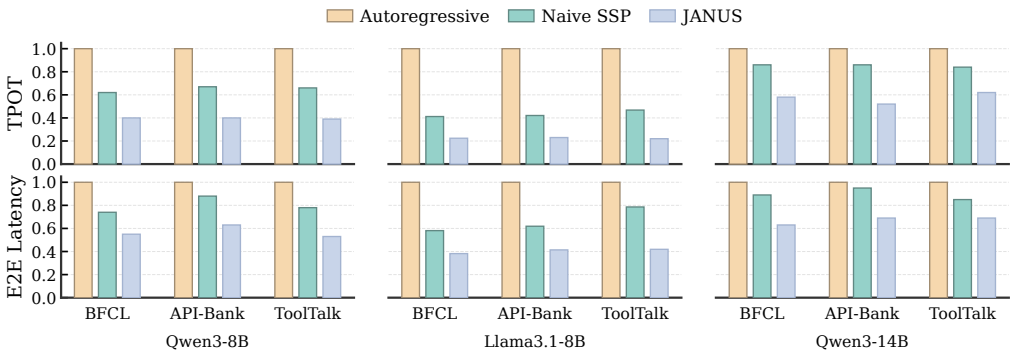


Figure 7: Normalized TPOT (top) and end-to-end latency (bottom).

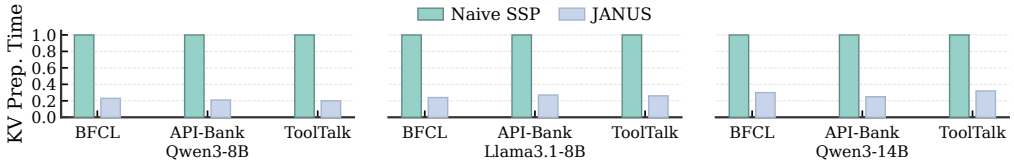


Figure 8: Normalized KV-page preparation time.

First, JANUS replaces the existing layer-bound KV-cache organization with a layer-decoupled KV-page abstraction. By managing all KV-pages on GPU in a unified pool, it enables drafting and verification to directly reuse the same physical pages without layout rematerialization.

Second, JANUS keeps the overhead of this reuse low through its coalesced metadata manager and hierarchical allocator. As shown in figure 8, the resulting design reduces KV-page preparation time by 68.0%–80.0% compared with naive SSP. The hierarchical allocator identifies an efficient combination of verification staging depth and sparse draft budget based on their measured latency trade-offs.

5.3 ABLATION STUDY

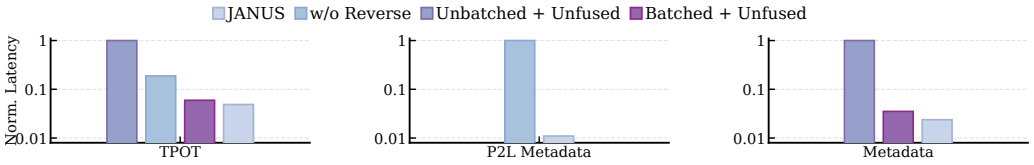
We ablate JANUS's components to quantify their latency benefits. Due to space constraints, we report results for Llama3.1-8B on ToolTalk. Other configurations show similar trends.

Efficiency of the KV-page abstraction. We first evaluate the benefit of cross-phase reuse. Enabling reuse provides a $1.40\times$ TPOT speedup, with further details provided in Appendix A.2. We then evaluate the reverse table by comparing JANUS with a variant that removes it while retaining the other components. As shown in figure 9a, JANUS reduces TPOT by 74.0% compared with the variant without the reverse table. This improvement is explained by a 98.9% reduction in physical-to-logical (P2L) metadata latency, as shown in figure 9b. These results confirm that the reverse table avoids expensive forward-table scans to identify each physical page's logical position.

Efficiency of coalesced metadata manager. To evaluate the efficiency of the coalesced metadata manager, we compare JANUS with an unbatched and unfused variant and a batched but unfused variant. All other components remain unchanged. As shown in figure 9a, coalesced metadata manager reduces TPOT by 18.0%–95.1%. To explain this improvement, we further measure metadata latency, which is reduced by 32.4%–97.6%, as shown in figure 9c. These results confirm that coalesced metadata manager substantially reduces the metadata overhead on the critical path.

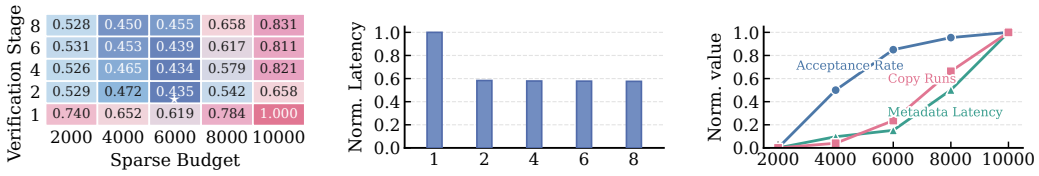
Efficiency of hierarchical KV-page allocator. We sweep the verification staging depth S and sparse draft budget B to evaluate the hierarchical KV-page allocator. As shown in figure 10a, JANUS selects $S = 2$ and $B = 6000$, achieving a $2.30\times$ speedup over the worst configuration and TPOT within 0.3% of the best configuration. The best configuration serves only as an oracle because identifying it requires exhaustively evaluating all combinations, making this search impractical.

The allocator first sets S through pipeline analysis. In figure 10b, verification forward latency drops by 41.7% as S increases from one to two, but by at most another 1.3% for larger S. Thus, $S = 2$



(a) Normalized TPOT. (b) P2L metadata latency. (c) Metadata latency.

Figure 9: Efficiency of reverse table and coalesced metadata processing.



(a) TPOT heatmap. (b) Verification forward latency. (c) Trade-offs with draft budget B.

Figure 10: Efficiency of hierarchical KV-page allocation. S is the verification staging depth and B the sparse draft budget in tokens per layer. The star marks JANUS's choice ($S = 2$, $B = 6000$).

reaches the latency plateau while leaving more GPU pages for cross-phase reuse. The allocator then calibrates B. In figure 10c, acceptance improves but gradually saturates as B grows, while copy runs and metadata latency rise. This supports calibrating B to balance acceptance against these costs.

We further compare offline calibration with online probing, which evaluates candidate B values during inference. Offline calibration reduces TPOT by 10.6%, KV-page preparation latency by 12.3%, and copy runs by 29.7%. Online probing repeatedly loads the pages required by different budgets and evicts pages retained under the previous budget, increasing page preparation and transfer fragmentation. These results validate calibrating B before deployment rather than probing it online.

6 Related Work

Local agent inference. TinyAgent (Erdogan et al., 2024) and AutoDroid-V2 (Wen et al., 2025) support local agent deployment, while PowerInfer (Song et al., 2024) and PowerInfer-2 (Xue et al., 2024) accelerate local LLM inference through sparse computation and efficient weight access. JANUS complements these efforts by targeting the growing KV-cache in local agent inference.

KV-cache management and offloading. vLLM (Kwon et al., 2023) improves GPU memory utilization through paging, while FlexGen (Sheng et al., 2023), InfiniGen (Lee et al., 2024), and ShadowKV (Sun et al., 2024a) use CPU memory for KV-cache beyond GPU capacity. JANUS builds on paging and offloading to manage GPU KV-pages in a unified pool for drafting and verification.

Sparse self-speculation. TriForce (Sun et al., 2024b) and MagicDec (Sadhukhan et al., 2025) combine sparse KV-cache drafting with full KV-cache verification. ViDA (Zheng et al., 2026) adapts sparse self-speculation to VideoQA, while VeriCache (Yao et al., 2026) overlaps KV-cache transfers with drafting across requests. JANUS adopts SSP and focuses on physical KV-page management under offloading. Its layer-decoupled KV-page pool enables cross-phase reuse, accelerating inference.

7 Conclusion

We propose JANUS, a local agent inference system addressing redundant H2D transfers and layout rematerialization in offloaded SSP. Its layer-decoupled KV-page abstraction breaks layer-bound layouts by managing physical KV-pages in a unified GPU pool for cross-phase reuse. To keep this reuse efficient, a coalesced metadata manager reduces metadata overhead, while a hierarchical GPU KV-page allocator allocates the limited page budget between verification staging and sparse drafting. Across three LLMs, two commodity GPUs, and three agent workloads, JANUS reduces TPOT by 26.2%–78.0% and end-to-end latency by 18.8%–61.8% compared with AR and naive SSP.

AI USE STATEMENT

We used large language models only to improve the manuscript's wording, grammar, and readability. The authors designed the study, conducted the experiments, analyzed the results, and formulated the conclusions. Every AI-assisted revision was checked by the authors, who retain full responsibility for the accuracy and integrity of the submitted work.

REFERENCES

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Wang. Agent s: An open agentic framework that uses computers like a human. In *International Conference on Learning Representations*, volume 2025, pp. 22924–22946, 2025.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, et al. Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*, 2024.

Lutfi Eren Erdogan, Nicholas Lee, Siddharth Jha, Sehoon Kim, Ryan Tabrizi, Suhong Moon, Coleman Richard Charles Hooper, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Tinyagent: Function calling at the edge. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 80–88, 2024.

Nicholas Farn and Richard Shin. Tooltalk: Evaluating tool-usage in a conversational setting. *arXiv preprint arXiv:2311.10775*, 2023.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. Cost-Efficient large language model serving for multi-turn conversations with CachedAttention. In *2024 USENIX annual technical conference (USENIX ATC 24)*, pp. 111–126, 2024.

Suyu Ge, Xihui Lin, Yunan Zhang, Jiawei Han, and Hao Peng. A little goes a long way: Efficient long context training and inference with partial contexts. In *International Conference on Learning Representations*, volume 2025, pp. 6470–6483, 2025.

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.

Wenyue Hua, Mengting Wan, Jagannath Vadrevu, Ryan Nadel, Yongfeng Zhang, and Chi Wang. Interactive speculative planning: Enhance agent efficiency through co-design of system and user interface. In *International Conference on Learning Representations*, volume 2025, pp. 14256–14283, 2025.

Shibo Jie, Yehui Tang, Kai Han, Zhi-Hong Deng, and Jing Han. Specache: Speculative key-value caching for efficient generation of llms. *arXiv preprint arXiv:2503.16163*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 155–172, 2024.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International conference on machine learning*, pp. 19274–19286. PMLR, 2023.

Junyan Li, Yang Zhang, Muhammad Yusuf Hassan, Talha Chafekar, Tianle Cai, Zhile Ren, Pengsheng Guo, Foroozan Karimzadeh, Chong Wang, and Chuang Gan. Commvq: Commutative vector quantization for kv cache compression. *arXiv preprint arXiv:2506.18879*, 2025.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 3102–3116, 2023.

Yuzhen Mao, Qitong Wang, Martin Ester, and Ke Li. Icecache: Memory-efficient kv-cache management for long-sequence llms. *arXiv preprint arXiv:2604.10539*, 2026.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian Yen, Avner May, Tianqi Chen, and Beidi Chen. Magicdec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In *International Conference on Learning Representations*, volume 2025, pp. 6835–6850, 2025.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

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*, pp. 590–606, 2024.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *arXiv preprint arXiv:2410.21465*, 2024a.

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*, 2024b.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019.

Hao Wen, Shizuo Tian, Borislav Pavlov, Wenjie Du, Yixuan Li, Ge Chang, Shanhui Zhao, Jiacheng Liu, Yunxin Liu, Ya-Qin Zhang, et al. Autodroid-v2: Boosting slm-based gui agents via code generation. In *Proceedings of the 23rd Annual International Conference on Mobile Systems, Applications and Services*, pp. 223–235, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

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

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

Jiayi Yao, Samuel Shen, Kuntai Du, Shaoting Feng, Dongjoo Seo, Rui Zhang, Yuyang Huang, Yuhan Liu, Shan Lu, and Junchen Jiang. Vericache: Turning lossy kv cache into lossless llm inference. *arXiv preprint arXiv:2605.17613*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Naimeng Ye, Arnav Ahuja, Georgios Liargkovas, Yunan Lu, Kostis Kaffes, and Tianyi Peng. Speculative actions: A lossless framework for faster ai agents. In *The Fourteenth International Conference on Learning Representations*, 2026a.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, et al. Agentfold: Long-horizon web agents with proactive context folding. In *The Fourteenth International Conference on Learning Representations*, 2026b.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

Yilong Zhao, Jiaming Tang, Kan Zhu, Zihao Ye, Chi-Chih Chang, Chaofan Lin, Jongseok Park, Guangxuan Xiao, Mohamed S Abdelfattah, Mingyu Gao, et al. Accelerating large-scale reasoning model inference with sparse self-speculative decoding. *Proceedings of Machine Learning and Systems*, 8:251–265, 2026.

Liang Zheng, Yitao Hu, Chen Shen, Zhixin Zhao, Bowen Shi, Yuhao Zhang, Laiping Zhao, Wenxin Li, and Keqiu Li. Vida: Lossless videoqa acceleration via selective sparse self-speculation with parallel computational load management. *IEEE Transactions on Parallel and Distributed Systems*, 2026.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *International Conference on Learning Representations*, volume 2026, pp. 58413–58438, 2026.

Table 1: Core SSP mechanisms included in naive SSP and the corresponding prior methods.

Core SSP mechanism	Representative methods
Sparse drafting and full KV-cache verification	MagicDec, SparseSpec, TriForce, VeriCache
Verification-guided sparse KV-cache selection	SparseSpec
GPU-resident sparse KV-cache with CPU-offloaded full KV-cache	TriForce, VeriCache

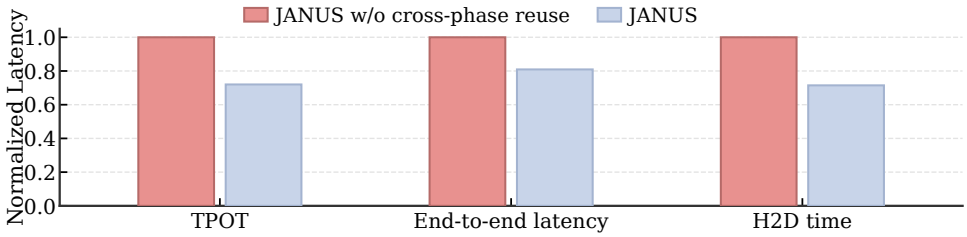


Figure 11: Normalized TPOT, end-to-end latency, and H2D time of JANUS with and without cross-phase KV-page reuse.

A APPENDIX

A.1 HOW NAIVE SSP COVERS EXISTING SSP MECHANISMS

Naive SSP incorporates the core SSP mechanisms of MagicDec (Sadhukhan et al., 2025), SparseSpec (Zhao et al., 2026), TriForce (Sun et al., 2024b), and VeriCache (Yao et al., 2026) into the existing layer-bound KV-cache management framework with offloading. Table 1 summarizes the shared mechanisms implemented in this baseline.

The baseline uses the same LLM for sparse drafting and full verification. For sparse KV-cache selection, it collects per-layer query–key scores during verification and selects the highest-scoring KV-pages within the sparse draft budget B. The selection is periodically refreshed as the context evolves. This implements verification-guided selection at the page granularity of the underlying paged KV-cache infrastructure.

Naive SSP is implemented in the same vLLM V1 inference engine as JANUS. It retains batched KV-page preparation and coalesced H2D transfers, and prefetches KV-cache for subsequent layers to overlap transfers with verification computation. We tune its sparse draft budget through an offline parameter sweep to account for the trade-off between draft acceptance and KV-page preparation cost. These optimizations operate under the existing layer-bound KV-cache layout, while JANUS introduces the layer-decoupled KV-page pool to enable cross-phase reuse.

A.2 DETAILS OF THE CROSS-PHASE KV-PAGE REUSE EXPERIMENT

The coalesced metadata manager and hierarchical GPU KV-page allocator both build on the layer-decoupled KV-page abstraction. Replacing this abstraction with naive SSP would therefore also change the mechanisms built on it, preventing an isolated comparison. We instead retain the unified pool and other components while disabling cross-phase reuse in both draft and verification. Unlike naive SSP, this variant still avoids reconstructing separate phase-specific layouts. This experiment therefore evaluates the H2D savings enabled by cross-phase reuse, rather than the full benefits of eliminating layout rematerialization and adopting the layer-decoupled abstraction.

As shown in figure 11, enabling cross-phase reuse provides a $1.40\times$ TPOT speedup and a $1.24\times$ end-to-end speedup, while reducing H2D time by 28.5%. These results show that cross-phase reuse

reduces repeated H2D loading and improves inference performance even when both variants already share the unified physical layout.