

# ACES: A Chiplet Architecture with Resource Partition and Dynamic Scheduling for Agentic LLMs

Hongou Li<sup>1,2</sup>, Mingxuan Li<sup>1</sup>, Zhantong Zhu<sup>1</sup> and Tianyu Jia<sup>1†</sup>

<sup>1</sup>School of Integrated Circuits, Peking University, Beijing, China

<sup>2</sup>Yuanpei College, Peking University, Beijing, China

<sup>†</sup>Corresponding Author: tianyu.jia@pku.edu.cn

**Abstract**—Agentic LLM is an emerging working paradigm leveraging large language models (LLMs) as assistant agents for complex, multi-step tasks. However, the operations of LLM agents also create unique workload characteristics with highly dynamic resource demands. In this work, we propose ACES, a co-designed solution leveraging scalable chiplet architecture together with dynamic workload scheduling for agentic LLMs. At hardware level, the chiplet architecture is designed to support a zoned fabric with flexible Swing Zones. Upon this, at software level, a conversation-centric dynamic scheduling approach is adopted, which includes topology-aware homing, proactive caching, and adaptive resource conversion to accommodate to data locality and resource balance. We evaluate our architecture using Llama-3 8B models on representative agentic-RAG-derived tasks. The system evaluations demonstrate that our design achieves  $2.33\times$  throughput improvement and an average 58% conversation latency reduction compared to state-of-the-art DistServe-like chiplet baselines, showcasing superior performance and scalability.

**Index Terms**—Chiplet Architecture, Dynamic Scheduling, Agentic Workloads, Large Language Models (LLMs)

## I. INTRODUCTION

The deployment of large language models (LLMs) is rapidly shifting from single-step stateless LLMs to continuous stateful collaboration between external tools and LLM agents, i.e. agentic LLMs [1]. This new paradigm is increasingly promising for complex tasks such as scientific discovery [2], where various external tools enhance LLM capability. In contrast to traditional standalone LLMs, agentic LLM involves tool calling in the response generation, leading to complex multi-step operations.

As shown in Fig. 1, a conversation between the user and agent includes repetitive inference iterations to solve the problems. Each iteration can be classified into three phases: 1) Prefill phase to understand user request and feedback from tool calling, 2) Decode phase to generate response and action plan, and 3) Tool-call phase to invoke external tools through API to interact with more information. The tool-call result is appended to the context for next iteration, where the incremental prefill reuses the cache and only processes the new inputs.

The unique workflow of agentic LLM poses new challenges for KV cache memory and workload scheduling. First, the memory footprint of KV cache grows rapidly since each processed token from either user prompts or tool-call feedback generates KV cache storage for subsequent decoding. For example, with GPT-3 175B at a sequence length over 8192, the KV cache occupies more than 36 GB of memory [3], exceeding the 32GB memory in commercial GPUs, e.g. RTX5090 [4]. Second, the latency of each iteration is unpredictable, as the

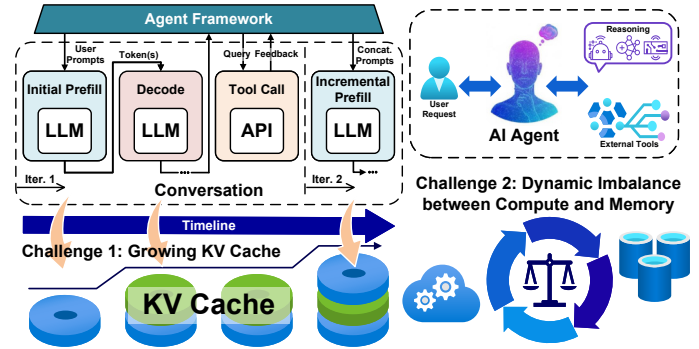


Fig. 1. The workflow of agentic LLMs and the deployment challenges.

agentic LLM might jump to tool-calling phase at any time of the decoding phase and result in unbalanced computation and memory characteristics in different iterations. Such unbalanced workload will lead to frequent GPU conflicts, e.g. up to 18.4% of agent requests preempted and 14.2% of memory wasted [5].

Although agentic LLMs greatly extend real-world problem-solving capability, the multi-iteration workflow requires architectural trade-off exploration for existing hardware. GPU architectures [6] with fixed on-device memory encounter a notable memory bandwidth bottleneck, as they need to repeatedly fetch the growing KV cache of each long-running agentic conversation. Chiplet [7], which integrates numerous dies onto a single package, is a promising solution to provide scalable performance and aggregated memory with high on-package bandwidth. However, static workload mapping on chiplet architecture leads to suboptimal bandwidth utilization, as traffic is unevenly distributed across inter-instance links.

In this work, we propose ACES (Agent-centric Chiplet Engine with adaptive Scheduling), which is a chiplet architecture with dynamic resource partition and fine-grained workload scheduling to achieve high-throughput agentic LLM serving. From the hardware aspect, we split the chiplet fabric into three different zones: Prefill Zone (P-zone), Decode Zone (D-zone) and Swing Zone (S-zone), each containing its corresponding instances (e.g. S-instances). S-instance can be converted to either P-instance or D-instance depending on the runtime workload requirement, achieving a dynamic resource reallocation. From the software aspect, a fine-grained dynamic workload scheduling is introduced to manage the mapping of different conversations from different users. Our scheduler minimizes and hides the transmission latency of large KV cache via P-

instance and D-instance pairing. The experiment results show that our design achieves up to  $2.33\times$  throughput improvement and 58% end-to-end latency reduction compared to DistServe-like [8] chiplet baseline for agentic LLM workloads.

The contributions of this work are summarized as follows:

- We profile and analyze agentic LLM workloads, identifying key bottlenecks on growing KV cache and unpredictable multi-iteration operations.
- A resource zone partition approach is developed on chiplet architecture to achieve dynamic resource reallocation during agentic LLM serving.
- A fine-grained dynamic workload scheduling is introduced considering the demands from prefill and decode phases for flexible KV cache management.
- Evaluation results show up to  $2.33\times$  throughput improvement and 58% end-to-end latency reduction is obtained compared to DistServe-like chiplet baseline.

## II. BACKGROUND

### A. Agentic LLM and the Serving

Establishing agentic LLM serving applications requires leveraging both basic frameworks, e.g. LangChain [9], LlamaIndex [10], and agent paradigms, e.g. ReAct [11], Reflexion [12], LATS [13], to define the agent interaction with external tools and users. Different user prompts can be served for diverse applications ranging from personalized assistants [14] to integrated circuit design [15]. KV cache management is the central challenge in agentic LLM serving. Current research focuses mainly on two optimization techniques, i.e. reducing the cache size through quantization [16] or sparsity [17], and improving memory utilization with methods such as PagedAttention [18]. However, these optimizations cannot be directly leveraged for agentic LLMs, where KV cache is accumulative and continuously expands with each iteration. Beyond memory challenge, agentic LLMs impose dynamic and unpredictable computational characteristics compared to conventional LLM inference. In agentic LLM serving, multi-iteration interactions repeatedly switch between prefill phase and decode phase. This frequent switching requires the KV cache to be shared between two phases, introducing significant communication overhead and becoming a primary performance bottleneck.

Existing LLM serving systems are optimized for a disaggregated inference paradigm, where prefill and decode phases are processed by specialized resource pools. For example, on GPU clusters, this paradigm is realized by system schemes such as DistServe [8] or SplitWise [19], which split tasks across separate GPU nodes to enable phase-specific parallelisms. Although improving per-phase efficiency, this approach is constrained by the high cost of transferring KV cache over coarse-grained, off-package interconnects. Targeting agentic LLMs, our work is exploring a hardware-software co-designed solution to adapt to agentic LLM characteristics.

### B. Chiplet Architecture for Scale-Up

The chiplet architecture is adopted to overcome the physical and economic barriers of large-size monolithic chip manufacturing, e.g.  $>600\text{ mm}^2$  chip [20]. By replacing a large

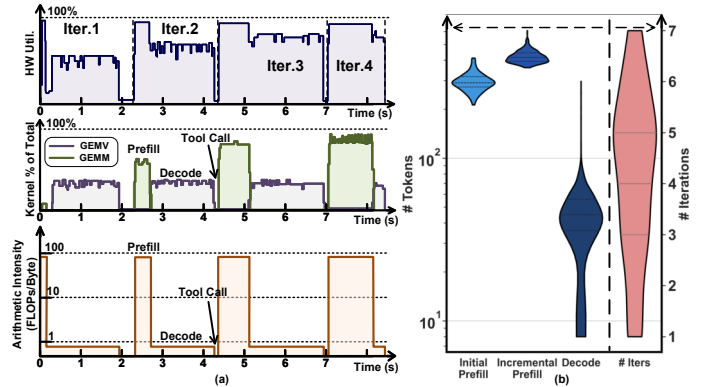


Fig. 2. (a) Temporal profiling of operation and utilization for a few agentic iterations, (b) statistical analysis of the characteristics of these agentic iterations.

monolithic chip with a few small-scale chips in one package, this “disaggregate-and-reintegrate” approach has become a key strategy for scaling up high-performance computing. For example, modern GPUs such as AMD MI300 [21] and NVIDIA B200 [22] leverage chiplet integration to combine vast computational resources. Beyond increasing resource density at package level, the chiplet architecture also provides a high-bandwidth on-package fabric that interconnects chiplet dies. This fabric, with standards such as UCIE [23], offers orders-of-magnitude higher bandwidth than traditional off-package networks, alleviating the memory-intensive computation in agentic LLM serving. This fundamental benefit on the communication-to-compute cost ratio unlocks a new hardware design space.

The physically disaggregated chiplets also widely facilitate non-uniform memory access (NUMA) systems, where each chip can be tightly coupled with local high-bandwidth memory (HBM) and creates a low-latency access path. However, accessing data from a remote HBM requires traversal of the network-on-package (NoP), incurring significantly high latency. Current scheduling policies mainly target GPU clusters [24], while scheduling policies that account for such non-uniform access latency on chiplet architectures remain under-explored.

## III. PROFILING ANALYSIS AND MOTIVATIONS

To understand the characteristics of agentic LLMs, we run the open-source Search-R1 [25] agentic RAG workflow, which is a canonical example of an agent autonomously interacting with external tool, i.e. a search engine. The Qwen2.5-7B model [26] is used as LLM backbone and the experiments are run on an NVIDIA H20 GPU [27].

Fig. 2(a) visualizes the Search-R1 execution trace over four iterations of prefill, decode, and a tool-calling phase. Prefill phase, dominated by GEMM kernels, reaches an arithmetic intensity (AI) above 70 FLOPs/Byte, indicating compute-bound workloads. Decode phase, dominated by GEMV kernels, has an AI of about 1 FLOP/Byte, showing memory-bound behavior. Between them, short but irregular tool-calling intervals leave the GPU underutilized, further complicating resource scheduling. The frequent switching across these phases, as seen in the trace, leads to inefficiencies for statically configured hardware.

Fig. 2(a) also indicates that each conversation alternates dynamically between the compute-bound prefill phases and the memory-bound decode phases, with prefill gradually taking a larger share of total latency. These frequent phase switches force the hardware to shift rapidly between workloads with opposed bottlenecks: high arithmetic throughput vs. high memory bandwidth. As a result, any statically configured system inevitably suffers from sustained underutilization of either compute units or memory bandwidth. Achieving high efficiency requires fine-grained, run-time dynamic resource reallocation.

**Observation 1: Agentic workload induces highly dynamic resource demands on both prefill and decode phases.**

Building on Fig. 2(a), Fig. 2(b) summarizes the token distribution of Search-R1 running on an MTRAG-derived [28] workload. All conversations are multi-iteration and involve 3 to 6 retrieval cycles, meaning the context including user prompts, agent-generated tokens and retrieved information from tool calling is repeatedly expanded across iterations. The violin plots show the token distribution for initial prefill, incremental prefill, and decode phases. It is observed that the prefill phases often integrate substantial retrieved context, while the decode phases produce far fewer tokens. Most prefill lengths fall within a narrow range but exhibit a pronounced long-tail distribution. Since each generated token needs to be stored in the KV cache, the cache size grows steadily across iterations, and the variability in prefill token counts makes this growth highly unpredictable. Together, these effects make runtime memory demand difficult to predict and may also create a potential bandwidth bottleneck due to KV cache transfers across the hardware.

**Observation 2: KV cache growth creates unpredictable memory pressure.**

Based on the above observations, agentic LLM serving systems should adaptively manage resource and data locality rather than rely on static allocation strategies. In this work, we propose ACES, a co-designed chiplet-based architecture with conversation-centric dynamic resource partition and scheduling to meet the above design demands.

## IV. CHIPLET ARCHITECTURE FOR AGENTIC LLMs

### A. Chiplet Architecture

To address the KV cache challenge of agentic LLM serving, we present chiplet-based architecture solution for agentic LLMs, as shown in Fig. 3. ACES comprises a multi-chiplet module (MCM) with  $8 \times 12$  homogeneous chiplets in mesh topology, forming an approximately square package. For memory access, ACES adopts a NUMA system similar to [29] where each chiplet has a dedicated HBM. The local HBM access latency is extremely low, whereas the remote HBM access latency depends on the 2D-mesh NoP bandwidth.

The microarchitecture of each chiplet is shown in Fig. 3(b). It integrates 16 compute cores, one host CPU, 16MiB L2 cache, one NoP router and  $4 \times 500\text{GB/s}$  die-to-die (D2D) interface. Each core contains a  $128 \times 128$  systolic array matrix engine similar to that in TPUs [30] and capable of executing both GEMM operations for prefill phase or batched vector-matrix multiplications for decode phase. A 1MiB local SRAM scratchpad is equipped for intermediate data buffering generated by

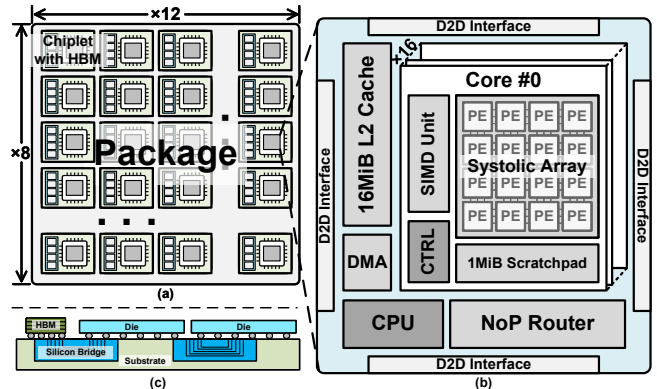


Fig. 3. (a) Package view with chiplets and integrated HBM. (b) Chiplet organization with compute cores, caches, and interconnect. (c) Packaging with chiplets and HBM linked by silicon bridges.

GEMM or GEMV. SIMD unit is used to perform non-linear operations such as normalization and softmax. To implement a unified data transfer mechanism for the system, a 1TB/s DMA engine controls the data movement across different memory hierarchy, i.e. local SRAM scratchpads, L2 cache and HBMs.

### B. Zoned Resource Partition

Fig. 4(a) demonstrates our resource zone partition of ACES, i.e. Prefill Zone (P-zone), Decode Zone (D-zone) and Swing Zone (S-zone). Each zone is composed of multiple instances (e.g. S-instances), all built from several chiplets. The role of each zone is explained as follows:

- **Prefill (P) zones** are dedicated to compute-intensive prefill phase by aggregating more chiplets into one instance.
- **Decode (D) zones** are utilized for latency-sensitive decode phase with compact instance scale for flexible scheduling.
- **Swing (S) zones** are positioned between above two zones and capable of dynamically switching between prefill and decode workloads for run-time resource reallocation.

To minimize KV cache transfer latency, D- and P-zones are arranged adjacently in a D–P–S–P stripe sequence, as shown in Fig. 4(a). The duplicated P-zone reflects the workload profile in Fig. 2, where prefill sequences are much longer than decode and gradually dominate total latency. This topology leverages the short, high-bandwidth links between adjacent chiplets in the 2D mesh, ensuring efficient data exchange across phases. To achieve high throughput, tensor parallelism (TP) is applied to each zone, which is widely used for large-scale chiplets [31].

Design space exploration (DSE) identifies the optimal TP configurations for each zone, as shown in Fig. 4(b). Under fixed  $8 \times 12$  mesh, our DSE finds that the optimal TP size is 8 for the P-zone. This is due to the prefill phase being dominated by large GEMM operations, where computation demand far outweighs communication. Here, using a larger TP size increases parallelism and improves compute throughput, while the associated all-reduce overhead remains negligible. Conversely, the D-zone processes multiple short, latency-sensitive decoding tasks concurrently. Therefore, a smaller TP size of 4 is preferable, as it reduces synchronization overhead and alleviates communication bottlenecks, enabling higher concurrency. Finally,

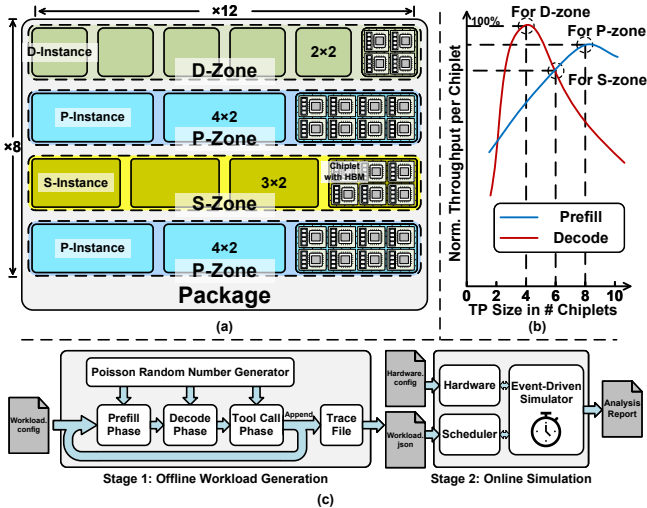


Fig. 4. (a) Zoned fabric with Prefill, Decode, and Swing instances arranged in alternating stripes, (b) throughput per chiplet across TP sizes for Prefill and Decode, (c) two-stage workload with offline generation and online simulation.

for the S-zone, we select a TP size of 6 as a compromise: it sustains efficiency when switching its role to prefill, while keeping communication manageable during decode. For unified scheduling among three zones, a specific number of chiplets are grouped together as one instance and the value equals to the corresponding optimal TP size. For example, the D-zone comprises 6 D-instances ( $2 \times 2$  chiplets each), while P-instance and S-instance consist of  $4 \times 2$  and  $3 \times 2$  chiplets respectively.

### C. Agentic LLM Workloads

To evaluate ACES architecture under agentic LLM serving workload, we develop a two-phase workload modeling including an offline workload generator and an online event-driven simulator, as shown in Fig. 4(c). To synthesize real-world-like workload, the generator first takes as input the token counts distribution of different phases in one conversation as shown in Fig. 2(b). Second, random conversations are produced with detailed multi-iteration events (typically 3-6 iterations) and each iteration includes prefill event ( $E_P$ ), decode event ( $E_D$ ), and a subsequent tool-call wait. The token counts for  $E_P$  and  $E_D$  are sampled from measured long-tail distributions, while tool-call waits correspond to idle periods the external APIs required. This combination of short computation bursts and unpredictable idle intervals creates the “burst-and-wait” workload pattern that dominates agentic LLM service. Finally, the arrival time of all conversations will follow a Poisson process [32] to represent the randomness of user requests. The synthesized workloads are then sent to the online event-driven simulator, which is triggered by  $E_P$  and  $E_D$  to accurately model each event latency and KV cache size growth across iterations. Our two-phase modeling framework provides a rigorous and reproducible basis for evaluating system under agentic LLM serving workloads.

## V. DYNAMIC WORKLOAD SCHEDULING

### A. Conversation-Centric Scheduling

To alleviate massive KV cache transmission and improve overall throughput, we introduce a runtime conversation-centric

scheduling approach, which we term as **Homing**. As illustrated in Fig. 5(a), when a new conversation arrives, the scheduler assigns this request to a less loaded D-instance ( $D_{home}$ ) coupled with a physically proximate P-instance ( $P_{paired}$ ), and they become the “home” of this conversation, responsible for its computation and KV cache storage. The scheduler maps decode and prefill computation to  $D_{home}$  and  $P_{paired}$  in priority, while arranging the KV cache in the local HBM of  $D_{home}$ . Such conversation-centric mapping creates long-term data locality as  $D_{home}$  and  $P_{paired}$  are physically adjacent to each other, reducing repeated inter-instance transfers and improving overall latency. By scheduling at the granularity of both individual events and hardware instances, the scheduler achieves fine-grained control over workload and hardware resources.

Our scheduler adopts a lightweight and topology-aware algorithm to select the optimal “home” for the upcoming conversation. Considering both communication efficiency and resource availability, the scheduler scores each candidate “home” pairs ( $P_i, D_j$ ) based on two dimensions: the topology distance between ( $P_i, D_j$ ) and the free memory capacity of ( $P_i, D_j$ ) local HBM, explicitly incorporating topology. Further, to alleviate local traffic congestion caused by skewed workload, the scheduler also supports **Re-Homing**. When the home pair of a conversation becomes congested, the KV cache is migrated to a less-loaded pair. This adaptive reassignment mitigates local congestion and prevents throughput degradation.

After homing, we maintain synchronized KV cache management between  $D_{home}$  and  $P_{paired}$ , as illustrated in Fig. 5(b). The original KV cache remains on  $D_{home}$ , while  $P_{paired}$  holds a local replica to accelerate future incremental prefill events. This is feasible because prefill requires less memory than decode, allowing the replica to use idle capacity without slowing the P-instance. To keep both KV cache in  $D_{home}$  and  $P_{paired}$  up-to-date and hide synchronization latency, the scheduler performs a bidirectional proactive streaming cache scheme on  $D_{home}$  and  $P_{paired}$ . When  $P_{paired}$  completes the computation for each layer, the corresponding KV cache is immediately streamed to  $D_{home}$  in the background in parallel with the computation of next layer. Conversely,  $D_{home}$  streams incremental KV updates back to  $P_{paired}$ , keeping the KV cache in  $P_{paired}$  up-to-date and avoiding expensive on-demand fetches from  $D_{home}$  to  $P_{paired}$ . This bidirectional streaming is different from conventional LLM inference where synchronization flows only from prefill to decode, enabling tight and low-latency coupling required by agentic LLM serving workloads.

### B. Resource Adaptivity by Scheduling

While Conversation Homing optimizes for individual conversations, the scheduler also manages the aggregate resource demands across all workloads. Two primary challenges arise under high concurrency for agentic LLM serving. First, memory saturation happens during numerous long-running agentic conversations due to incremental KV cache memory. Second, unpredictable phase alternation between prefill and decode leads to uneven resource requirement. To address these issues, we introduce two adaptive scheduling mechanisms: Spill-to-Peer and S-instance-switch.

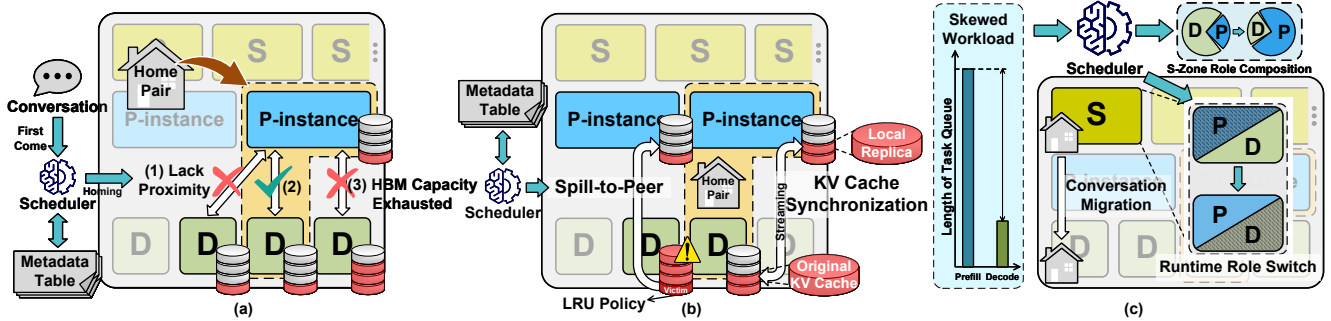


Fig. 5. (a) Upon conversation arrival, the Conversation Homing mechanism performs a topology- and load-aware assignment, binding the conversation to a proximate P-D instance pair. (b) During its lifecycle, the KV cache of conversation is managed via proactive bidirectional synchronization between the paired instances, while memory pressure is alleviated by a Spill-to-Peer mechanism. (c) At system level, the scheduler monitors aggregate queue lengths and triggers the role conversion of S-instances to rebalance hardware resources.

The first mechanism, Spill-to-Peer, as shown in Fig. 5(b), addresses memory capacity saturation by “spilling”, i.e. offloading KV cache from overloaded D-instances to adjacent P-instances. In a system with many long-running agentic conversations, the cumulative KV cache footprint can quickly exceed the available memory in specialized zones, such as the D-zone. Normally, when the HBM of an instance is full, the system evicts older KV cache blocks using an LRU policy. However, this eviction causes a problem: when a new event for the conversation arrives, it triggers a time-consuming prefill of all previous context to restore the KV cache. To solve this, Spill-to-Peer offloads part of the KV cache from the overloaded  $D_{home}$  to an underutilized physically adjacent P-instance, effectively turning the P-zone into a flexible “swap space”. This on-package memory offloading is efficient because D2D interconnect bandwidth is significantly higher than that of a single HBM, allowing the system to handle memory oversubscription without significant delays. Conceptually, this mechanism treats the aggregate HBM capacity of the entire chiplet fabric as a globally addressable memory pool rather than isolated per-instance storage.

The second mechanism provides hardware resource flexibility by enabling Swing instances in S-zones to switch between prefill and decode roles based on workload imbalances, as depicted in Fig. 5(c). When demand for either prefill or decode severely exceeds the other, the scheduler triggers a role conversion for an S-instance, assigning it to the more demanding task to improve resource utilization. This involves completing any pending tasks in its local queue, flushing its memory, and fetching the required model weights from an adjacent instance. A D-to-P conversion requires relocating all conversations currently homed to the S-instance. For each conversation, the scheduler transfers its KV cache to a new D-instance and updates the associated metadata. Only after all active conversations are securely migrated does the S-instance switch roles, ensuring uninterrupted service. This smooth transition allows the system to adaptively balance resources and respond to shifting demands between prefill and decode tasks.

## VI. EVALUATION

### A. Experimental Setup

We evaluate ACES performance by establishing a multi-level simulation environment. The performance of individual

instances is modeled using ASTRA-sim [33] integrated with SCALE-Sim [34], while system-level dynamics are captured by our in-house discrete-event simulator. We model two hardware platforms for comparison. The first is our proposed  $8 \times 12$  chiplet architecture designed for a TSMC 22nm process. Each chiplet occupies approximately  $400 \text{ mm}^2$ , delivering a peak performance of 262 TFLOPS at FP16, and is configured with 32 GB of HBM and a high-bandwidth D2D interconnect. The second is a baseline GPU cluster with NVIDIA H100 GPUs [35], comparable in compute and memory resources and equipped with a two-level interconnect for intra- and inter-node communication. For evaluations, we use a Llama 3 8B-class [36] model and a workload generated from the stochastic model derived from our real-world agentic RAG analysis.

We evaluate four primary configurations, with names corresponding to those in Fig. 6. DS-GPU represents a state-of-the-art disaggregation policy inspired by DistServe [8] running on the GPU cluster. The other three systems run on our chiplet platform: DS-Chiplet adapts the same DistServe policy to the on-package fabric; WSC-Chiplet is inspired by the static, offline-optimized co-design methodology of WSC-LLM [37]; and ACES is our full system with fine-grained adaptive scheduling. The following sections present detailed comparisons of these systems under agentic LLM workload.

### B. Overall Performance Comparison

Fig. 6 compares system capacity, congestion, user-perceived latency, and traffic balance across the four configurations. The first observation is the large gap between GPU- and chiplet-based systems. Under the same disaggregation policy, moving from DS-GPU to DS-Chiplet improves goodput [38] from 11.8 to 28.9 RPS ( $2.45 \times$ ). This reflects the advantage of the chiplet architecture, where low-latency on-package transfers avoid contention and minimize queuing delays. Consistently, DS-GPU also shows the most unbalanced communication, with a traffic variance more than twice that of chiplet-based designs.

Within the chiplet platform, the progression across scheduling strategies shows the limits of static policies and the gains from dynamic adaptation. WSC-Chiplet, with its topology-aware zoning, improves goodput to 51.1 RPS and reduces congestion relative to DS-Chiplet, demonstrating the value of static partitioning. Our system reaches the best: by combining

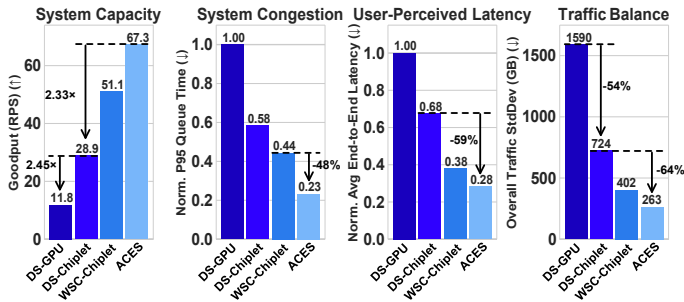


Fig. 6. Performance comparison with different hardware architectures.

homing, proactive cache replication, and adaptive S-zone reconfiguration, it halves queueing time again (48% lower than WSC-Chiplet) and cuts normalized latency to 0.28. These improvements result in 67.3 RPS, 2.33 $\times$  higher than DS-Chiplet and 5.70 $\times$  higher than DS-GPU, and the lowest traffic imbalance (263 GB vs. 1590 GB), while incurring negligible scheduling overhead. Together, the results show that architectural efficiency is foundational, topology-aware zoning adds benefit, and only dynamic adaptation fully unlocks the chiplet fabric.

### C. Ablation of Scheduling Mechanisms

Fig. 7 breaks down the performance impact of the individual scheduling mechanisms. Starting from the Static baseline, which achieves only 25.4 RPS while suffering the worst congestion and responsiveness, we progressively add layers of scheduling to expose their marginal contributions. The first enhancement, Homing, incorporates homing to enforce locality. This step alone reduces network contention, bringing a 13% throughput improvement and a 22% reduction in TTFT (Time-to-First-Token) [38]. The modest gain highlights that static locality management is necessary, but cannot by itself resolve the inefficiencies of agentic serving.

A more substantial jump comes from Caching, which introduces bidirectional cache synchronization and spill-to-peer transfer. By ensuring that subsequent prefill tasks can access the required context without waiting for fetching the entire cache, it raises goodput by over 50% to 43.8 RPS reduces queueing time by nearly half. This shift reveals that, once locality is secured, the main bottleneck is no longer communication hops but the ability to serve dependent requests without stalling on cache availability. Finally, the full ACES adds dynamic S-zone role conversion to balance global compute demand. This unlocks the highest goodput of 67.3 RPS and drives congestion down to 36% of the baseline, a 2.8 $\times$  improvement. These gains outweigh the cost of each role switch, which requires only a brief flush and a weight reload. Interestingly, responsiveness regresses slightly, as the system deliberately prioritizes stability and throughput under memory-bound decode pressure. This small trade-off is outweighed by the overall gain of 53.7% in throughput, showing that robust serving requires not just data-level optimizations but also flexibility at the resource level.

### D. Throughput Scalability

Fig. 8 examines how each system scales as the offered load increases. The baseline DS-GPU shows limited growth,

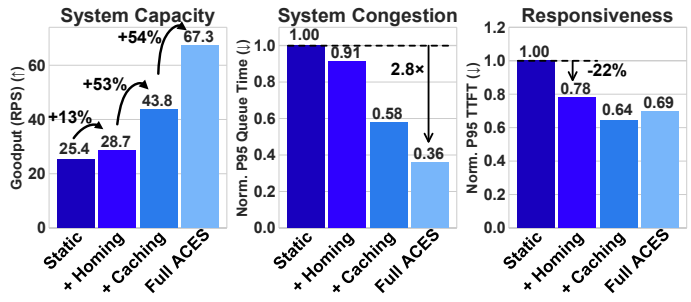


Fig. 7. Ablation study of scheduling strategies on key performance metrics.

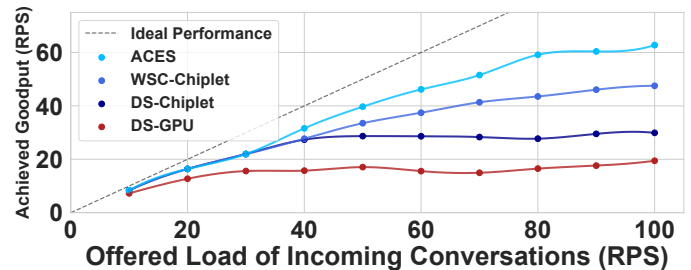


Fig. 8. Throughput scalability with increased loading.

saturation around 17-20 RPS, a consequence of expensive off-package transfers that quickly congest the inter-node fabric. DS-Chiplet initially scales better for the low-latency, high-bandwidth interconnect of the chiplet fabric on the package, but plateaus at  $\sim$ 30 RPS once static zoning begins to bottleneck traffic, leaving resources underutilized. WSC-Chiplet extends scalability further, reaching nearly 50 RPS by leveraging topology-aware partitioning, yet still diverging from the ideal slope as static boundaries fail to adapt under shifting load. In contrast, our dynamic scheduling is able to closely track the ideal performance curve, sustaining about linear growth up to 60+ RPS before signs of saturation appear. This result reinforces the earlier observations: architectural efficiency and topology awareness are necessary, while only adaptive reconfiguration can maintain high throughput under sustained pressure.

## VII. CONCLUSION

In this work, we propose ACES, a hardware-software co-designed solution to efficiently serve agentic LLMs on a scalable chiplet architecture. We first analyze agentic LLM workloads, identifying their unique characteristics and challenges for existing serving paradigms. A novel chiplet architecture is introduced featuring a zoned resource for flexible partition. Upon this architecture, we develop a dynamic, conversation-centric scheduling with topology-aware homing, proactive caching, and adaptive role conversion mechanisms. Evaluations demonstrate that ACES improves goodput by 2.33 $\times$  and reduces latency by 58% compared to the state-of-the-art DistServe-like chiplet baseline, underscoring the importance of coupling flexible hardware with dynamic scheduling for future agentic workloads.

## ACKNOWLEDGMENT

This work was supported in part by NSFC Grant No. 92464202.

## REFERENCES

- [1] T. Zhang, "System Architecture for Agentic Large Language Models," technical report no. eecs-2025-5, University of California, Berkeley, EECS Department, Berkeley, CA, Jan. 2025. Available via Berkeley Library Digital Collections.
- [2] Google, "Gemini Deep Research — your personal research assistant," 2025. [Online]. Available: <https://gemini.google/overview/deep-research/>.
- [3] L. Ye, Z. Tao, Y. Huang, and Y. Li, "Chunkattention: Efficient self-attention with prefix-aware kv cache and two-phase partition," *arXiv preprint arXiv:2402.15220*, 2024.
- [4] NVIDIA, "GeForce RTX 5090 Graphics Card," 2025. [Online]. Available: <https://www.nvidia.com/en-us/geforce/graphics-cards/50-series/rtx-5090/>.
- [5] J. Chen, J. Shi, Q. Chen, and M. Guo, "Kairos: Low-latency Multi-Agent Serving with Shared LLMs and Excessive Loads in the Public Cloud," *arXiv preprint arXiv:2508.06948*, 2025.
- [6] NVIDIA, "NVIDIA GH200 Superchip Accelerates Inference by 2x in Multiturn Interactions with LLaMA Models," 2024. [Online]. Available: <https://developer.nvidia.com/blog/nvidia-gh200-superchip-accelerates-inference-by-2x-in-multiturn-interactions-with-llama-models/>.
- [7] R. Kaplan, "Intel Gaudi 3 AI Accelerator: Architected for Gen AI Training and Inference," in *2024 IEEE Hot Chips 36 Symposium (HCS)*, pp. 1–16, 2024.
- [8] Y. Zhong, S. Liu, J. Chen, J. Hu, Y. Zhu, X. Liu, X. Jin, and H. Zhang, "Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210, 2024.
- [9] H. Chase, "LangChain," Oct. 2022.
- [10] J. Liu, "LlamaIndex," 11 2022.
- [11] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafraan, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," in *International Conference on Learning Representations (ICLR)*, 2023.
- [12] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. 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), Curran Associates Inc., 2023.
- [13] A. Zhou, K. Yan, M. Shlapentokh-Rothman, H. Wang, and Y.-X. Wang, "Language agent tree search unifies reasoning, acting, and planning in language models," in *Proceedings of the 41st International Conference on Machine Learning, ICML'24*, JMLR.org, 2024.
- [14] Microsoft, "Enjoy AI Assistance Anywhere with Copilot for PC, Mac, Mobile, and More," 2025. [Online]. Available: <https://www.microsoft.com/en-us/copilot>.
- [15] P. Yan, Q. Zhi, L. Liu, and T. Jia, "GenSoC: A Multi-Agent-Assisted SoC Generation Methodology Leveraging Open-Source Hardware," in *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, (Iceland), Aug. 2025.
- [16] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, and A. Gholami, "Kvquant: Towards 10 million context length llm inference with kv cache quantization," *Advances in Neural Information Processing Systems*, vol. 37, pp. 1270–1303, 2024.
- [17] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," *arXiv preprint arXiv:2309.17453*, 2023.
- [18] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient Memory Management for Large Language Model Serving with PagedAttention," in *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP '23*, (New York, NY, USA), p. 611–626, Association for Computing Machinery, 2023.
- [19] P. Patel, E. Choukse, C. Zhang, A. Shah, Í. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative llm inference using phase splitting," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132, IEEE, 2024.
- [20] S. Li, M.-S. Lin, W.-C. Chen, and C.-C. Tsai, "High-bandwidth chiplet interconnects for advanced packaging technologies in AI/ML applications: Challenges and solutions," *IEEE Open Journal of the Solid-State Circuits Society*, 2024.
- [21] A. Smith, E. Chapman, C. Patel, R. Swaminathan, J. Wu, T. Huang, W. Jung, A. Kaganov, H. McIntyre, and R. Mangaser, "11.1 AMD Instinct™ MI300 Series Modular Chiplet Package – HPC and AI Accelerator for Exa-Class Systems," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, pp. 490–492, 2024.
- [22] NVIDIA, "DGX B200: The Foundation for Your AI Factory," 2025. [Online]. Available: <https://www.nvidia.com/en-us/data-center/dgx-b200/>.
- [23] D. Das Sharma, G. Pasdast, Z. Qian, and K. Aygun, "Universal Chiplet Interconnect Express (UCIe): An Open Industry Standard for Innovations With Chiplets at Package Level," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 12, no. 9, pp. 1423–1431, 2022.
- [24] Y. Wang, B. Li, M. T. I. Ziad, L. Eeckhout, J. Yang, A. Jaleel, and X. Tang, "OASIS: Object-Aware Page Management for Multi-GPU Systems," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1678–1692, 2025.
- [25] B. Jin, H. Zeng, Z. Yue, J. Yoon, S. Arik, D. Wang, H. Zamani, and J. Han, "Search-r1: Training llms to reason and leverage search engines with reinforcement learning," *arXiv preprint arXiv:2503.09516*, 2025.
- [26] Q. Team, "Qwen2.5: A party of foundation models," September 2024.
- [27] NVIDIA, "NVIDIA CEO Jensen Huang Promotes AI in Washington, DC and China: NVIDIA to resume H20 sales to China, announces new, fully compliant GPU for China," jul 2025. [Online]. Available: <https://blogs.nvidia.com/blog/nvidia-ceo-promotes-ai-in-dc-and-china/>.
- [28] Y. Katsis, S. Rosenthal, K. Fadnis, C. Gunasekara, Y.-S. Lee, L. Popa, V. Shah, H. Zhu, D. Contractor, and M. Danilevsky, "MTRAG: A Multi-Turn Conversational Benchmark for Evaluating Retrieval-Augmented Generation Systems," *Transactions of the Association for Computational Linguistics*, vol. 13, pp. 784–808, 2025.
- [29] N. B. Mallya, B. Goel, and I. Sourdis, "MEMPLEX: A Memory System with Replication and Migration of Data for Multi-Chiplet NUMA Architectures," in *Proceedings of the 39th ACM International Conference on Supercomputing*, pp. 1219–1233, 2025.
- [30] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, *et al.*, "Ten lessons from three generations shaped google's tpuv4i: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–14, IEEE, 2021.
- [31] Z. Huang, S. Fan, C. Tang, X. Lin, S. Deng, and Y. Liu, "Hecaton: Training Large Language Models with Scalable Chiplet Systems," *arXiv preprint arXiv:2407.05784*, 2024.
- [32] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A Distributed Serving System for Transformer-Based Generative Models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, (Carlsbad, CA), pp. 521–538, USENIX Association, July 2022.
- [33] W. Won, T. Heo, S. Rashidi, S. Sridharan, S. Srinivasan, and T. Krishna, "ASTRA-sim2.0: Modeling Hierarchical Networks and Disaggregated Systems for Large-model Training at Scale," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 283–294, 2023.
- [34] R. Raj, S. Banerjee, N. Chandra, Z. Wan, J. Tong, A. Samajdhar, and T. Krishna, "SCALE-Sim v3: A modular cycle-accurate systolic accelerator simulator for end-to-end system analysis," in *2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 186–200, IEEE, 2025.
- [35] NVIDIA, "NVIDIA H100 Tensor Core GPU," sep 2024. [Online]. Available: <https://resources.nvidia.com/en-us-hopper-architecture/nvidia-tensor-core-gpu-datasheet?ncid=no-ncid>.
- [36] Meta, "Introducing Meta Llama 3: The most capable openly available LLM to date," 2025. [Online]. Available: <https://ai.meta.com/blog/met-a-llama-3/>.
- [37] Z. Xu, D. Kong, J. Liu, J. Li, J. Hou, X. Dai, C. Li, S. Wei, Y. Hu, and S. Yin, "WSC-LLM: Efficient LLM Service and Architecture Co-exploration for Wafer-scale Chips," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pp. 1–17, 2025.
- [38] Z. Wang, S. Li, Y. Zhou, X. Li, R. Gu, N. Cam-Tu, C. Tian, and S. Zhong, "Revisiting SLO and Goodput Metrics in LLM Serving," *arXiv preprint arXiv:2410.14257*, 2024.