High-density Mobile Cloud Gaming on Edge SoC Clusters

Li Zhang, Shangguang Wang, Mengwei Xu
Beijing University of Posts and Telecommunications

Abstract

System-on-Chip (SoC) Clusters, i.e., servers consisting of many stacked mobile SoCs, have emerged as a popular platform for serving mobile cloud gaming. Sharing the underlying hardware and OS, these SoC Clusters enable native mobile games to be executed and rendered efficiently without modification. However, the number of deployed game sessions is limited due to conservative deployment strategies and high GPU utilization in current game offloading methods. To address these challenges, we introduce *SFG*, the first system that enables high-density mobile cloud gaming on SoC Clusters with two novel techniques: (1) It employs a resource-efficient game partitioning and cross-SoC offloading design that maximally preserves GPU optimization intents in the standard graphics rendering pipeline; (2) It proposes an NPU-enhanced game partition coordination strategy to adjust game performance when co-locating partitioned and complete game sessions. Our evaluation of five Unity games shows that *SFG* achieves up to 4.5× higher game density than existing methods with trivial performance loss. Equally important, *SFG* extends the lifespan of SoC Clusters, enabling outdated SoC Clusters to serve new games that are unfeasible on a single SoC due to GPU resource shortages.

1 Introduction

Mobile gaming, a popular and portable entertainment form, is expected to generate approximately $100 billion globally in 2024 [24]. The growing complexity of game logic and graphics has led to the emergence of edge cloud-based mobile gaming [10, 14]. This approach offloads game execution and rendering to nearby edge servers, allowing users to experience high-quality, low-latency gaming on more affordable devices with reduced battery and disk usage.

Generally, there are two approaches to support mobile gaming. The first one is running mobile operating system on commodity edge servers with x86/ARM CPUs and NVIDIA GPUs [3, 5, 11, 16]. However, due to hardware discrepancies

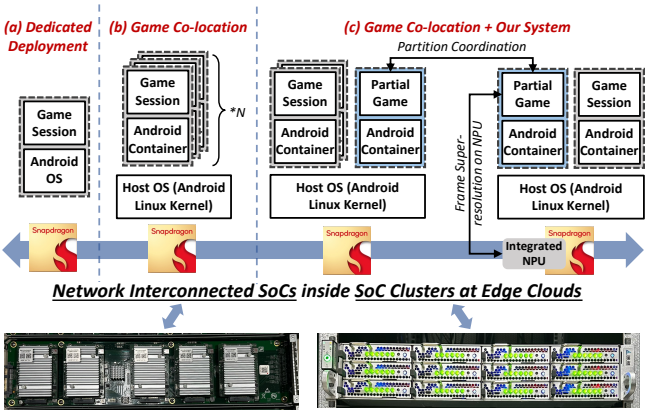


Figure 1: Hardware/software architecture and different deployment strategies of mobile gaming on SoC Clusters.

with mobile devices, this method requires either mobile OS simulation [3, 8, 27, 33, 41] or extensive re-engineering for compatibility and performance [5, 15]. These alterations often compromise flexibility and game fidelity. Our work focuses on the second approach that leverages mobile System-on-Chip (SoC) Clusters [9, 12, 40, 43]. These SoC Clusters host intensive mobile SoCs, each of which is capable of independently running game sessions. Due to their inherent compatibility and high fidelity, SoC Clusters have emerged as a popular platform for mobile cloud gaming.

Low game deployment density on SoC Clusters. Serving a large number of game sessions can substantially reduce costs for edge operators. However, current deployment strategies result in low game deployment density due to several factors. (1) Games often exhibit dynamic resource usage affected by unpredictable user actions and scene changes. (2) To prevent resource contention and preserve game performance, edge operators often use a dedicated deployment strategy that assigns a separate SoC for each game session [43, 44], as shown in Figure 1(a). This approach leads to severe resource waste on SoC Clusters, with an average of 66% for CPU and 38% for GPU (§2). Simply scaling out SoC Clusters is not cost-efficient with limited space and electricity sup-

ply at the edge [39]. Furthermore, the evolving capacities of mobile SoCs and mobile gaming market demand more computation resources for immersive gaming experiences (§2). SoC Clusters equipped with outdated SoC models will become inadequate for new games, similar to the obsolescence seen in older smartphones [38]. This leads to further decline in game deployment density.

To improve game deployment density, a natural idea is to co-locate multiple game sessions on a single SoC, as shown in Figure 1(b). However, this is challenging due to mobile GPU's limited capacity, which may struggle to simultaneously support a few or only two game sessions. For example, Genshin Impact [13] uses 52% of GPU resources on its own. Hosting two such sessions on one SoC significantly reduces performance – from 45 frames per second (FPS) to 38 FPS as Figure 2 illustrates. Inspired by previous cloud gaming systems that address resource shortages on user devices [26,31,32,34,36], we explore partitioning and offloading graphics rendering workloads to use distributed, underutilized GPU resources across SoCs. A well-designed game partition strategy would enable multiple SoCs to collaboratively serve more game sessions. However, prior systems are geared towards device-to-cloud scenarios, where cloud GPUs have almost unlimited computing power, and the bottleneck resides at the network latency. No prior systems have examined how multiple resource-limited nodes/SoCs can collaboratively support additional game sessions.

***SFG*'s approach.** We present *SFG*, the first system for high-density mobile cloud gaming on SoC Clusters. As shown in Figure 1(c), *SFG* operates by partitioning a complete game instance, and co-locating partitioned sessions with others on different SoCs. *SFG* incorporates two key components to achieve high game density while preserving performance. (1) A straightforward yet effective game offloading strategy that enables partitioned game sessions to render only part of the original game view. Through adapting the game camera's projection matrix before rendering, it maintains the intentions of the default graphics rendering pipeline and conserves GPU usage more than existing partitioning methods. We further abstracted the partition decision on game's screen coordinates system, facilitating flexible balancing of rendering workloads. (2) An neural network accelerator (NPU)-enhanced game partition coordinator. It allows two partitioned game sessions to swiftly coordinate and shift graphics rendering workloads to NPUs for better game performance. It exploits a unique opportunity in SoC Clusters: SoCs are equipped with NPUs that are left totally idle during gaming. Thereby, *SFG* selectively downgrades frame rendering resolution and applies lightweight frame super-resolution to partitioned game sessions that underperform the performance target. This approach addresses GPU shortages without involving external hardware, while preserving game graphical integrity.

Evaluation. We implemented *SFG* atop Unity and tested it with five open-source Unity games. We utilized a commercial SoC Cluster comprising 60 Qualcomm SM8250 SoCs [2]. The results indicate that *SFG* achieves up to 4.5× higher game density compared to dedicated deployment and 1.5× higher than game co-location baselines. It only incurs a maximum performance loss of 3.3% and negligible frame quality degradation in a few game sessions. *SFG* also exhibits capabilities in serving graphics-intensive games, which outperforms dedicated deployment strategy and game partitioning approaches that typically cause high GPU usage overhead.

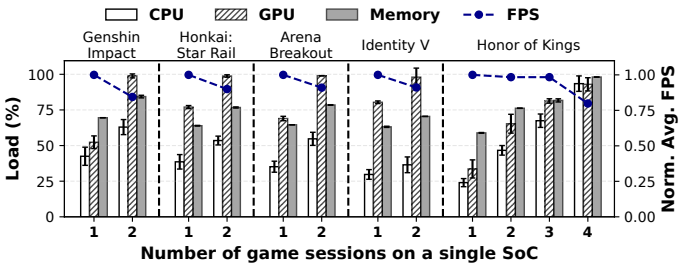


Figure 2: Average hardware load and normalized average FPS of five commercial mobile games. Hardware: Qualcomm Snapdragon SM8250 SoC.

Contributions are summarized below.

- We first disclosed the status quo of mobile cloud gaming on SoC Clusters and their limitations in low game density.
- We proposed *SFG*, the first system on SoC Clusters to achieve high-density mobile cloud gaming. It integrates a resource-efficient game partitioning design and an NPU-enhanced partition coordination strategy to improve game density while maintaining game performance.
- We evaluated *SFG* on a commercial SoC Cluster with 60 mobile SoCs. The results show *SFG* can remarkably improve game density with trivial performance loss.

2 Preliminary Measurements

We selected five representative commercial mobile games from 2023 for a preliminary exploration of existing game deployment strategies to uncover their limitations.

Dedicated deployment allocates a dedicated hardware for every game session. Major cloud gaming platforms, such as Xbox Cloud, use this strategy [44]. Despite ensuring game performance, this approach leaves over 50% of CPU/GPU resources underutilized on average [44]. SoC Clusters also commonly adopt this strategy, as mobile games are originally designed to operate on individual SoCs. To demystify its effectiveness, we analyzed the CPU, GPU, and memory load of five commercial mobile games on a commercial off-the-shelf

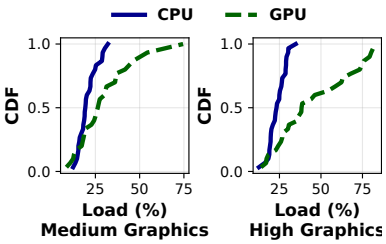


Figure 3: Average hardware load of the top 30 games from the Google Play Store. Each game is tested under both medium and high graphics qualities.

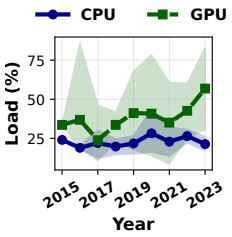


Figure 4: Average hardware load of 30 games aggregated by year. Shaded areas: min-max load.

SoC Cluster equipped with Qualcomm Snapdragon SM8250 SoCs [2]. The results displayed in Figure 2 indicate that deploying a single game session per SoC indeed preserves game performance, however leading to *substantial underutilization of hardware resources.* Specifically, the average CPU and memory loads across the five games are 34% and 64%, respectively. GPU utilization varies between 34% and 80%.

Game co-location on single hardware units has been explored by academia to improve game deployment density [28, 35, 42]. However, these methods have seen limited application in the industry mainly because they either require dynamic adjustments to graphics quality during gameplay [28], or rely on error-prone performance prediction before co-location [35]. In some cases, these approaches might negatively impact user experience and fail to meet the stringent service level agreements of cloud gaming services. Currently, some mobile cloud gaming providers adopt a conservative, containerization-based co-location strategy. Given a performance target, they pre-profile and record the maximum number of game sessions a single SoC can support. During the deployment phase, game sessions are scheduled based on these offline records. For example, Figure 2 shows a single SoC can accommodate three concurrent sessions of "Honor of Kings" without performance loss. However, the other four games tested could only support one session per SoC, primarily due to GPU resource limitations. This observation has led us to focus on GPU workloads as the primary target for optimization.

Evolution of mobile gaming. We extended our experiments to include the top 30 mobile games on the Google Play Store. We observed a consistent hardware usage pattern: mobile games generally consume more GPU than CPU resources. Figure 3 shows that the GPU is the primary computing resource being utilized compared to CPU. By categorizing these games according to their release year, we observed a nearly steady increase in average GPU usage over time in Figure 4. This trend suggests that future mobile games will likely demand even more GPU resources, potentially surpassing the capabilities of current SoCs. This would disable deploying new games on old SoC Clusters.

3 *SFG* Design

3.1 Resource-efficient Game Partition

Improving game density on SoC Clusters requires an efficient game partitioning and offloading design that enables partitioned game sessions to effectively utilize distributed yet limited hardware resources. To achieve this goal, we first assess the effectiveness of previous game partitioning methods to evaluate whether they can aptly partition GPU workloads. After that, we detail *SFG*'s game partitioning approach and its underlying rationales.

3.1.1 Revisiting Prior Game Partitioning Designs

In traditional cloud gaming systems, there are usually two types of game partition strategies. The first one runs a full copy of the original game session [26, 32, 34], which is impractical in our context as it doesn't reduce graphics rendering workloads. The second strategy partitions graphics rendering workloads to different game sessions. A recent example is the distance-based game partitioning [36], which divides graphics rendering workloads in the game world based on a distance away from the game camera[1]. As demonstrated in Figure 5(a), this method partitions the original game world into a near part (Figure 5(b)) and a remote part (Figure 5(c)). To assess its effectiveness in separating GPU workloads, we applied it to an open-source Unity game [18] using three different split distances. Results displayed in Figure 6 reveal average GPU usage for complete, near, and remote game sessions. Our main observation is that distance-based partitioning results in a disproportionately skewed reduction in GPU usage, with the remote part only showing a marginal decrease of 1% or 2%. This indicates inefficiency in the partitioning design. Moreover, the summarized GPU load of the near and remote parts significantly exceeds that of the original game session, indicating an inefficient partitioning design that could lead to minimal or no benefits.

Insights. We found that the excessive GPU usage primarily stems from rendering unnecessary game objects. For example, the shaded areas in Figure 5(c) are unnecessarily rendered but not visible in the final game scene due to occlusion by the rendering results in the near part shown in Figure 5(b). On the contrary, in complete game instances, these unnecessary rendered areas would be optimized by the default rendering pipeline [21]. We deduced that the unintended GPU usage overhead arises from indiscriminate partition decisions that disrupt the optimization of the default graphics rendering pipeline. This insight guided us toward a game partition design that maintains the intentions and optimizations of the default rendering pipeline.

[1]The functionality of a game camera is to capture game objects within its view frustum [22] and determine what game objects should be displayed on the screen by undergoing complex computations like occlusion culling [21].

(a) Original game view before partition (b) Near-part game view after partition (c) Remote-part game view after partition

Figure 5: Three different game views in distance-based game partitioning. GPU usage overhead mainly comes from the occluded objects that are rendered in the remote game view (c) while not displayed in the original game view (a).

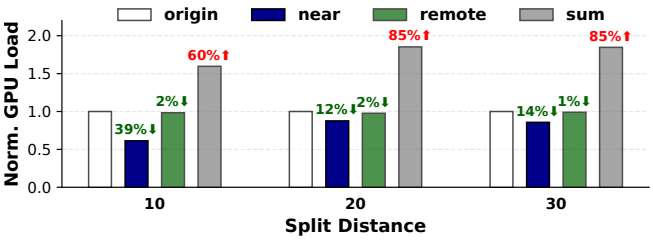
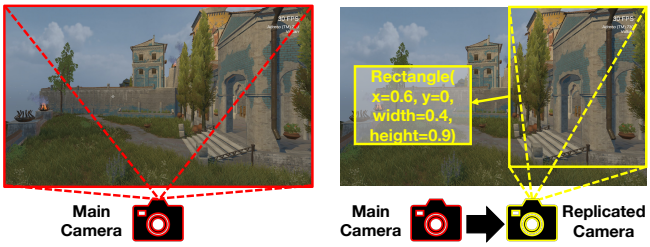


Figure 6: Average GPU load of different game sessions when utilizing distance-based game partition. "sum": accumulated GPU usage of the near part and remote part.



(a) Full game view rendered by the original main camera (b) Partial game view rendered by the replicated camera

Figure 7: The original game view captured by the default game camera (a), and the partial game view captured by the replicated game camera (b).

3.1.2 *SFG*'s Game Partition

To preserve optimizations provided by the standard graphics rendering pipeline, we borrow the idea from sort-first rendering [37], and set partitioned game sessions to focus solely on rendering a specific portion of the original game view before the rendering pipeline begins. We demonstrate *SFG*'s partition design in Figure 7.

Workflow. In typical game setups, there is a main camera that captures game objects within its view frustum, processes them through the rendering pipeline, and finally produces rendered frames. As shown in Figure 7(a), the main camera observes the entire area in front of it and projects it onto a 2D screen. *SFG* injects this process by first replicating the main camera, and then modifying the projection matrix of the replicated camera to focus on a subarea on the screen, e.g., the upper right area where dense game objects are occluded behind, as shown in Figure 7(b). Frames generated by the replicated camera are used as the final rendering outputs.

Abstracted projection matrix adaptation. The projection matrix of a game camera is responsible for transforming the game's 3D world into a 2D screen representation [1]. During the process, the coordinates of all game objects in the 3D game world are converted into normalized 2D screen coordinates. We simplify the projection matrix adaptation by abstracting a $Rectangle(x, y, width, height)$ data structure to represent the specific screen area targeted for rendering. An example is illustrated in Figure 7(b). The $x \in [0,1]$ and $y \in [0,1]$ indicate the normalized starting position on the x-axis and y-axis from the top-left corner of the screen; *width* and *height* represent the normalized dimensions of the rectangle along the x-axis and y-axis, respectively. Based on a user-defined *Rectangle*, we can flexibly adjust the projection matrix of the replicated camera to render a partial view identical to the one produced by the default main game camera. Our evaluation in §4.2 shows that *SFG*'s partition design effectively reduces GPU usage overhead, thereby enhancing resource efficiency.

3.2 NPU-enhanced Partition Coordination

To effectively co-locate partitioned game sessions with complete ones on the same SoC, it's crucial to balance rendering workloads to achieve optimal game performance. To address this challenge, we introduce an NPU-enhanced game partition coordination strategy. It allows partitioned game sessions to adjust the partitioning decision based on real-time game performance, and utilizes frame super-resolution on SoC NPU to compensate for GPU shortages when necessary.

Assumptions. We base our strategy on the following practical assumptions. (1) A complete game session is ideally partitioned into two parts. Increasing the number of partitions tends to escalate resource overhead while offering diminishing returns in game density. (2) Game sessions should ideally render frames at their native resolution to maintain graphical fidelity. Frame super-resolution is employed only when there are insufficient resources left. (3) Complete, unpartitioned game sessions are given priority over partitioned ones, as they are more prevalent in mobile cloud gaming systems. Thereby, *SFG* only applies frame super-resolution to partitioned game sessions. This helps minimize compromises in graphics fi-

delity across the entire system. Under these assumptions, our algorithm operates in two stages.

Stage #1: Coordinating and shifting GPU workloads. The ultimate objective of the first stage is to ensure the performance of complete game sessions on at least one of the two SoCs. This stage deals with the performance of complete game sessions when partitioned game sessions are collocated with them. The performance of these complete game sessions can fall into one of three states. (1) All complete game sessions meet performance targets. If complete game sessions on both SoCs meet the performance targets, *SFG* moves directly to *Stage #2* for further processing. (2) All complete game sessions underperform relative to performance targets. *SFG* will coordinate the partition decision between two game sessions by shifting more graphics rendering workloads from one session to another. The shifting process typically adjusts the partition *Rectangle* incrementally on either the x-axis or y-axis (e.g., by 0.1) every set interval (e.g., 500 ms). This adjustment continues until the complete game sessions on one of the two SoCs meet the performance target, at which point *SFG* transitions to the third state. (3) Complete game sessions on a certain SoC fail to meet the performance target. Under this situation, *SFG* proceeds to *Stage #2* for further performance adjustments.

Stage #2: Eliminating GPU shortages through frame super-resolution on SoC NPUs. The second stage is critical for ensuring the performance of all game sessions. It is entered from two specific states outlined in *Stage #1*. First, if all complete game sessions meet the performance target, partitioned game sessions will too. This is based on observations from our experiments and attributed to the simple mobile GPU execution model without priority and preemption [30]. The second state transformed from *Stage #1*, where only game sessions deployed on a certain SoC fail to meet the performance target, indicates a shortage of GPU resources on that specific SoC. To address this problem, *SFG* first reduces the target resolution of rendered frames, which decreases GPU usage while ensuring the performance of all game sessions. It then utilizes the unique opportunity in SoC Clusters that massive integrated but idle SoC NPUs are experts in executing deep learning models. These rendered low-resolution frames are upscaled to their target resolution using a lightweight frame super-resolution model on SoC NPUs, only when there is enough time budget between frames for injecting this process into the graphics rendering pipeline. In our end-to-end experiments (§4.3), only a small portion of partitioned game sessions (two out of five evaluated games, 16% of all game sessions) involve frame super-resolution with minor frame quality loss.

4 Evaluation

We implemented a prototype of *SFG* on top of Unity, which functions as a plugin that can be integrated seamlessly into existing Unity-based games. We leveraged Unity's Camera API to implement *SFG*'s game partition design. We incorporated WebRTC into each partitioned game session, utilizing its media channel for streaming rendered frames and its data channel for game state synchronization and partition coordination. We used TFLite [19] to implement frame super-resolution inference on NPU with Hexagon delegation support [20].

4.1 Setup

Hardware. We used a commercial SoC Cluster (demonstrated in Figure 1) consisting of 60 Qualcomm Snapdragon SM8250 SoCs released in 2020 [2]. All SoCs are interconnected with 1 Gbps Ethernet using network switches. Each SoC runs an Android 10 operating system.

Software. We selected five open-sourced Unity games with varied graphics settings: Sun Temple (1920x1080, 30 FPS) [18], Corridor (1280x720, 30 FPS) [4], Sewer-Mid (1920x1080, 60 FPS) [6], Sewer-High (2560x1440, 60 FPS) [6], and Viking Village (1920x1080, 30 FPS) [7]. The selected games include those with high dynamics and rapid scene changes (e.g., Viking Village), and those with fewer scene changes, such as Sun Temple. To simulate human gameplay during evaluation and ensure consistent game behaviors across game sessions, we employed a record-and-replay method using Unity's animation system. We first recorded interactive scripts for each game, then these scripts are automatically replayed after the game session starts. We used TFLite [19] to run the frame super-resolution model, ETDS-T [25], with int8 quantization on SoC NPUs.

Metrics. Game deployment density is measured by the maximum number of game sessions that can be deployed on an SoC Cluster with 60 SoCs. For each SoC, we recorded (1) CPU load by accessing `/proc/stat`; (2) GPU load through Qualcomm's Adreno GPU driver file in sysfs: `/sys/class/kgsl/kgsl-3d0/gpu_busy_percentage`. We measured FPS for each game session by instrumenting game runtime and tallying rendered frames over a period. These metrics were sampled every 500 ms.

4.2 GPU Load Reduction

In this section, we compare the GPU usage of partitioned game sessions under different strategies. For *SFG*'s partition, we vertically split a complete game session into equal left and right parts (using $Rectangle(0,0,0.5,1)$). We first assess the GPU usage of the two partitions. Then, with the distance-based partition, we aim to match the near-part session's GPU usage with *SFG*'s left part. This approach ensures a fair GPU usage comparison. Table 1 demonstrates that *SFG* incurs lower GPU usage overhead than the distance-based partition. Specifically, *SFG* facilitates running resource-intensive games like Sewer-High and Viking Village on two separate SoCs, which a single SoC can't support. In contrast, the distance-

Game	GPU Load: Origin	Partition Method	GPU Load: Partition			
			P1	P2	P1+P2	Co(P1+P2)
Sun Temple	76.1	Distance	57.0	75.4	132.4	92.4 (21.4%↑)
		Ours	55.3	73.8	129.1	74.2 (2.50%↓)
Corridor	48.5	Distance	30.0	41.1	71.1	60.0 (23.7%↑)
		Ours	29.5	36.1	65.6	56.1 (15.7%↑)
Sewer-Mid	72.4	Distance	59.8	72.7	132.5	85.7 (18.4%↑)
		Ours	58.5	56.0	114.5	75.9 (4.83%↑)
Sewer-High	✗	Distance	73.8	✗	✗	✗
		Ours	71.3	70.2	141.5	✗
Viking Village	✗	Distance	80.8	✗	✗	✗
		Ours	82.5	79.1	161.6	✗

Table 1: Average GPU load of game sessions when utilizing distance-based game partitioning and our approach. P1 and P2 represent two partitioned game sessions. P1+P2 indicates the accumulated GPU load of the two partitioned game sessions, where Co(P1+P2) is the GPU load measured by co-locating the two parts on the same SoC. ✗ means GPU usage exceeds SoC's capacity.

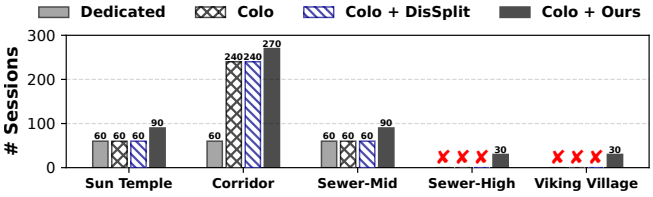


Figure 8: Game deployment density comparison between games and deployment strategies.

based partition fails due to high GPU usage. For the first three games, *SFG* reduces GPU usage by 14.8% on average when co-locating the partitioned game sessions on the same SoC (Co(P1+P2) in Table 1). Although the reduction in absolute GPU usage is modest, it significantly impacts given the limited, fragmented resources on each SoC. We will next reveal the effectiveness of *SFG*'s game partitioning strategy in the end-to-end game density evaluation.

4.3 End-to-end Game Deployment

Baselines. We selected the following baselines for comparing game deployment density.

• *Dedicated* deployment strategy: Assigning each game session to a dedicated SoC, a common practice in existing mobile cloud gaming systems [40, 44].

• *Colo (Co-location)*: An approach that allows multiple complete game sessions to co-locate on a single SoC if they achieve the performance target. This approach is used by edge operators to improve game density on SoC Clusters (§2).

• *Colo + DisSplit* co-locates partitioned game sessions using distance-based partition strategy [36] with complete game sessions. It ensures the SoC serving the near part achieves a nearly full GPU load while meeting the performance target. Then the remote game session is scheduled on another SoC. If the remote part fails to meet the performance target, the game session will be ejected.

• *Colo + Ours*: Integrating *SFG* with game co-location to boost game density. Initially, it vertically partitions a game into equal left and right parts. In the partition coordination stage, the x-axis step is 0.1 and the time interval is 500 ms.

Overall, *SFG* remarkably improves game density with minimal impact on performance and CPU usage.

Game density. As shown in Figure 8, *SFG* significantly enhances game density over previous baselines. (1) For resource-demanding games like Sewer-High and Viking Village, which a single SoC cannot support, *SFG*'s partitioned sessions can be served by two separate SoCs. This approach enables support for 30 game sessions on 60 SoCs. The distance-based partition, however, fails to achieve similar outcomes. (2) For games that can be accommodated into a single SoC, *SFG* efficiently utilizes limited remaining resources to host an additional 30 game sessions. In cases like Sun Temple and Sewer-Mid, *SFG* attains 1.5× higher density than that of co-location baselines.

Game performance. Figure 9 presents the normalized 99th percentage FPS for all games under different deployment strategies. Compared to baselines, *SFG* consistently maintains game performance, with a minor FPS reduction observed in Sewer-Mid. Specifically, compared to the dedicated deployment, employing *SFG* results in a 3.3% decrease in FPS (from 54 to 52).

Hardware load. Figure 10 shows the results of hardware load. On average, *SFG* increases GPU load by 22.4% compared to dedicated deployment and by 7.5% compared to the game co-location method. The average maximum GPU load reaches 97.3%, indicating near-full utilization of GPU resources on a

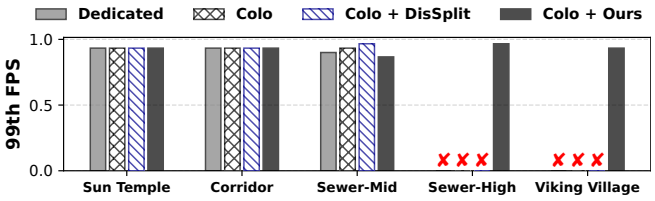


Figure 9: Normalized 99th FPS of different games with different deployment strategies. FPS values are averaged across multiple game sessions on the same hardware.

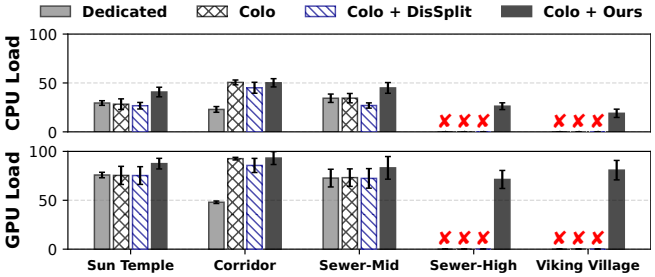


Figure 10: Average SoC CPU/GPU load. Hardware load are averaged if one game is partitioned to different SoCs.

Game	SR Conf	Time Budget	SR Time	Frame Time	Total Time	Frame Quality
Corridor	640x360 x2	33.3 (30 FPS)	16.0	8.9	24.9	33.4
Sun Temple	640x360 x3	33.3 (30 FPS)	18.9	4.76	23.7	29.8

Table 2: Frame super-resolution (SR) metrics breakdown. SR Conf indicates the resolution of the source frame and the scaling factor of the SR model. Time budget refers to the available time for graphics rendering and frame super-resolution given an FPS requirement. Time is measured in milliseconds. Frame quality is measured by PSNR [29].

single SoC. Additionally, *SFG* incurs an average CPU load increase of 12.4%, attributable to redundant game logic execution and *SFG*'s operational cost.

***SFG*'s partition coordination and frame super-resolution.** Besides the end-to-end experiments, we conducted additional experiments to demonstrate the functionality of *SFG*'s game partitioning and frame super-resolution. *SFG* began by vertically partitioning each complete game session with a left part ratio of 0.5 ($Rectangle(0, 0, 0.5, 1)$). Specifically, Sewer-Mid, Sewer-High, and Viking Village maintained this ratio to achieve the performance target, without employing frame super-resolution. Conversely, both Corridor and Sun Temple utilized a 0.4 partition ratio, where their left parts (0.4) rendered frames at native resolution, and their right parts employed frame super-resolution. Detailed configurations and simulated metrics are provided in Table 2. Overall, both partitioned sessions completed graphics rendering and frame super-resolution on SoC NPU within the allocated time budget, with only negligible frame quality loss measured by PSNR [29] (values above 30 indicate satisfactory frame quality). Furthermore, the ongoing advancements in mobile NPU technology within recent mobile SoC models are expected to significantly enhance the implementation of frame super-resolution on mobile NPUs [17, 23].

5 Conclusion

This paper proposed *SFG*, the first system for high-density mobile cloud gaming on edge SoC Clusters. It employs two key techniques (1) to reduce the high GPU resource usage overhead of previous game partitioning systems; and (2) to guarantee the performance of co-located game sessions. We developed a prototype of *SFG* and conducted tests on five open-source Unity games. The results demonstrate its effectiveness in reducing resource usage overhead, enabling high-density gaming deployment on SoC Clusters with trivial performance loss.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback on this work. This work was supported by National Key R&D Program of China (No.2021ZD0113001), NSFC (62102045, 62032003, 61921003), and China Institute of IoT (Wuxi). Mengwei Xu is the corresponding author of this work.

References

[1] Opengl projection matrix. http://www.songho.ca/opengl/gl_projectionmatrix.html, 2019.

[2] Hardware | qualcomm snapdragon 865 5g mobile platform | 5g mobile processor | qualcomm. https://www.qualcomm.com/products/application/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-865-5g-mobile-platform, 2020.

[3] Android-x86. https://www.android-x86.org/, 2022.

[4] Corridor lightning example | unity asset store. https://assetstore.unity.com/packages/essentials/tutorial-projects/corridor-lighting-example-33630, 2022.

[5] Nvidia and ampere computing create arm-based server platform to stream mobile games. https://amperecomputing.com/press/nvidia-and-ampere-computing-create-arm-based-server-platform-to-stream-mobile-games, 2022.

[6] Sewer/underground | unity asset store. https://assetstore.unity.com/packages/3d/environments/sewer-underground-modular-pack-v4-0-112692, 2022.

[7] Viking village urp | unity asset store. https://assetstore.unity.com/packages/essentials/tutorial-projects/viking-village-urp-29140, 2022.

[8] Anbox cloud. https://anbox-cloud.io/, 2023.

[9] Aws device farm. https://aws.amazon.com/device-farm/, 2023.

[10] Caregame | mobile cloud gaming. https://www.caregame.com/, 2023.

[11] Cuttlefish virtual android devices. https://source.android.com/docs/setup/create/cuttlefish, 2023.

[12] Firebase test lab | test in the lab, not on your users. https://firebase.google.com/products/test-lab, 2023.

[13] Genshin impact. https://genshin.hoyoverse.com/en/, 2023.

[14] Genshin impact cloud. https://cloudgenshin.hoyoverse.com/en-us, 2023.

[15] Intel bridge technology. https://www.intel.com/content/www/us/en/developer/topic-technology/bridge-technology.html, 2023.

[16] Run apps on the android emulator. https://developer.android.com/studio/run/emulator, 2023.

[17] Snapdragon 8 gen 3 mobile platform. https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-8-gen-3-mobile-platform, 2023.

[18] Sun temple - unity asset store. https://assetstore.unity.com/packages/3d/environments/sun-temple-115417, 2023.

[19] Tensorflow Lite. https://www.tensorflow.org/lite, 2023.

[20] Tensorflow lite hexagon delegate. https://www.tensorflow.org/lite/android/delegates/hexagon, 2023.

[21] Unity - manual: Occlusion culling. https://docs.unity3d.com/Manual/OcclusionCulling.html, 2023.

[22] Viewing frustum - wikipedia. https://en.wikipedia.org/wiki/Viewing_frustum, 2023.

[23] World's first on-device demonstration of stable diffusion on android. https://www.qualcomm.com/news/onq/2023/02/worlds-first-on-device-demonstration-of-stable-diffusion-on-android, 2023.

[24] Mobile games - worldwide | statista market forecast. https://www.statista.com/outlook/dmo/digital-media/video-games/mobile-games/worldwide, 2024.

[25] Jiahao Chao, Zhou Zhou, Hongfan Gao, Jiali Gong, Zhengfeng Yang, Zhenbing Zeng, and Lydia Dehbi. Equivalent transformation and dual stream network construction for mobile image super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14102–14111, 2023.

[26] Eduardo Cuervo, Alec Wolman, Landon P. Cox, Kiron Lebeck, Ali Razeen, Madan Musuvathi, and Stefan Saroiu. Kahawai: High-quality mobile gaming using gpu offload. In *MobiSys'15*. ACM – Association for Computing Machinery, May 2015.

[27] Di Gao, Hao Lin, Zhenhua Li, Chengen Huang, Liangyi Gong, Feng Qian, Yunhao Liu, and Tianyin Xu. Trinity: High-performance mobile emulation through graphics projection. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, 2022.

[28] Sergey Grizan, David Chu, Alec Wolman, and Roger Wattenhofer. djay: Enabling high-density multi-tenancy for cloud gaming servers with dynamic cost-benefit gpu load balancing. In *Proceedings of the sixth ACM symposium on cloud computing*, pages 58–70, 2015.

[29] Alain Horé and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369, 2010.

[30] Gang Huang, Mengwei Xu, Felix Xiaozhu Lin, Yunxin Liu, Yun Ma, Saumay Pushp, and Xuanzhe Liu. Shuffledog: characterizing and adapting user-perceived latency of android apps. *IEEE Transactions on Mobile Computing*, 16(10):2913–2926, 2017.

[31] Zeqi Lai, Y Charlie Hu, Yong Cui, Linhui Sun, and Ningwei Dai. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, pages 409–421, 2017.

[32] Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman, and Jason Flinn. Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming. In *MobiSys 2015*. ACM - Association for Computing Machinery, June 2015.

[33] Linsheng Li, Bin Yang, Cathy Bao, Shuo Liu, Randy Xu, Yong Yao, Mohammad R Haghighat, Jerry W Hu, Shoumeng Yan, and Zhengwei Qi. Droidcloud: Scalable high density androidtm cloud rendering. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 3348–3356, 2020.

[34] Yong Li and Wei Gao. Muvr: Supporting multi-user mobile virtual reality with resource constrained edge cloud. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 1–16. IEEE, 2018.

[35] Yusen Li, Chuxu Shan, Ruobing Chen, Xueyan Tang, Wentong Cai, Shanjiang Tang, Xiaoguang Liu, Gang Wang, Xiaoli Gong, and Ying Zhang. Gaugur: Quantifying performance interference of colocated games for improving resource utilization in cloud gaming. In *Proceedings of the 28th international symposium on high-performance parallel and distributed computing*, pages 231–242, 2019.

[36] Jiayi Meng, Sibendu Paul, and Y Charlie Hu. Coterie: Exploiting frame similarity to enable high-quality multiplayer vr on commodity mobile devices. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 923–937, 2020.

[37] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A sorting classification of parallel rendering. *IEEE computer graphics and applications*, 14(4):23–32, 1994.

[38] Jennifer Switzer, Gabriel Marcano, Ryan Kastner, and Pat Pannuto. Junkyard computing: Repurposing discarded smartphones to minimize carbon. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 400–412, 2023.

[39] Mengwei Xu, Zhe Fu, Xiao Ma, Li Zhang, Yanan Li, Feng Qian, Shangguang Wang, Ke Li, Jingyu Yang, and Xuanzhe Liu. From cloud to edge: a first look at public edge platforms. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 37–53, 2021.

[40] Mengwei Xu, Li Zhang, and Shangguang Wang. Position paper: Renovating edge servers with arm socs. In *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, pages 216–223. IEEE, 2022.

[41] Qifan Yang, Zhenhua Li, Yunhao Liu, Hai Long, Yuanchao Huang, Jiaming He, Tianyin Xu, and Ennan Zhai. Mobile gaming on personal computers with direct android emulation. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–15, 2019.

[42] Chao Zhang, Jianguo Yao, Zhengwei Qi, Miao Yu, and Haibing Guan. vgasa: Adaptive scheduling algorithm of virtualized gpu resource in cloud gaming. *IEEE Transactions on Parallel and Distributed Systems*, 25(11):3036–3045, 2013.

[43] Li Zhang, Zhe Fu, Boqing Shi, Xiang Li, Rujin Lai, Chenyang Chen, Ao Zhou, Xiao Ma, Shangguang Wang, and Mengwei Xu. Soc-cluster as an edge server: an application-driven measurement study. *arXiv preprint arXiv:2212.12842*, 2022.

[44] Wei Zhang, Binghao Chen, Zhenhua Han, Quan Chen, Peng Cheng, Fan Yang, Ran Shu, Yuqing Yang, and Minyi Guo. Pilotfish: Harvesting free cycles of cloud gaming with deep learning training. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 217–232, 2022.