Characterizing Distributed Mobile Augmented Reality Applications at the Edge

Giovanni Bartolomeo
giovanni.bartolomeo@tum.de
Technical University of Munich

Jacky Cao
jacky.cao@oulu.fi
University of Oulu

Xiang Su
xiang.su@ntnu.no
Norwegian University of Science and Technology

Nitinder Mohan
mohan@in.tum.de
Technical University of Munich

ABSTRACT

Mobile Augmented Reality (AR) is gaining traction as a compelling application due to recent advancements in hardware and software. Previous studies have suggested that distributing AR services on an edge computing infrastructure can offer significant performance benefits, especially for consolidating concurrent clients. In this study, we shed light on several research challenges directly impacting the effective integration of distributed AR and edge computing. Specifically, we conduct extensive experiments by deploying our distributed stream processing-based AR pipeline, `scAtteR`, on a representative edge-cloud infrastructure managed by the `Oakestra` framework. We uncover several unapparent challenges that inhibit the effective marriage of distributed AR when deployed on edge and demonstrate the potential improvements through `scAtteR++`. We offer valuable insights and best practices to the growing AR research community, specifically those interested in leveraging edge and public cloud technologies for large-scale AR operations.

CCS CONCEPTS

• **Human-centered computing → Mixed / augmented reality**; • **Software and its engineering → Distributed systems organizing principles**.

KEYWORDS

AR, DSP, Edge Computing, Cloud, Containers.

ACM Reference Format:
Giovanni Bartolomeo, Jacky Cao, Xiang Su, and Nitinder Mohan. 2023. Characterizing Distributed Mobile Augmented Reality Applications at the Edge. In *19th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '23), December 5–8, 2023, Paris, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3624354.3630584

1 INTRODUCTION

The growing interest in mixed reality (XR), from both academia and industry, such as Apple [9] and Meta [87], has paved the way for new use cases leveraging immersive perception and interaction platforms [69, 84]. Augmented reality (AR), as one of the prominent pillars in XR, enables real-time interactive user experiences on mobile devices by superimposing digital content onto physical environments [22, 44, 53, 89]. Recent advancements in hardware [91] and software frameworks [4, 22, 23] further facilitate the widespread adoption of AR across several computing platforms [74].

Most existing AR applications are *monolithic* – a client captures a camera feed and offloads computation to a remote server [60, 61]. The remote execution is typically one-in-one-out, which takes AR client frame inputs and returns results over a network [13, 65]. To handle multiple clients, the backend must replicate, or employ a request queue. Even explorations that leverage edge computing to improve performance are typically limited to single edge servers, primarily due to strong dependencies between different pipeline stages [77, 97]. Such solutions effectively demonstrate AR-edge capabilities by utilizing low-latency edge servers but face significant challenges regarding scalability [43, 67] – limiting real-world adoption. While existing research suggests that distributing AR functions across multiple edge servers can improve performance [57, 68, 94], practical demonstrations for multi-client use remain limited. Specifically, solutions ignore effective ways to distribute and replicate AR services which consider resource contention. Applying the benefits of these studies in practice is difficult due to system complexity caused by heterogeneous clusters. Similarly, the interplay of virtualization (e.g., containers) and orchestration frameworks, like Kubernetes [10], to facilitate dynamic migrations and scaling of AR services has remained largely unexplored to date.

Our study marks one of the first attempts to examine the operational challenges of deploying distributed stream processing (DSP) [16, 46, 52] based AR applications on edge-cloud infrastructure. We aim to provide valuable recommendations and insights for AR developers and researchers. Our key contributions are:

1) We present `scAtteR`, an AR system composed of five containerized microservices (§3.1). Each service in the pipeline processes an input feed as a stream and can be independently deployed. We encapsulate real-world operations by employing a state-of-the-art edge orchestration framework, `Oakestra` [12, 15], to manage `scAtteR` deployment on our distributed edge-cloud infrastructure. We conduct extensive experiments deploying different configurations of `scAtteR` on local edge machines and public AWS cloud (§4). Our analysis reveals that stateful AR components can significantly limit pipeline scalability. Additionally, we demonstrate that traditional resource consumption metrics used by most orchestration systems do not accurately reflect AR quality-of-service (QoS) performance. Based on our findings, we highlight several recommendations for more effective edge-AR operations.

2) We systematically demonstrate the performance potential of our recommendations through the enhanced pipeline, `scAtteR++` (§4). Our experiments show significant performance improvements over `scAtteR`, i.e., ≈ 2.75× increase in concurrent client capacity and ≈ 4× improved framerate. Distributed AR performs notably better by introducing pipeline statelessness and effective request queue management strategies.

We release both `scAtteR` and `scAtteR++` as open-source [14].

2 RELATED WORK

Scalable Augmented Reality. Scalability is crucial when deploying AR on a large scale, as supporting concurrent users requires significant resources and can strain existing server capacity. Researchers have explored various approaches to address this challenge. Zhang *et al.* [98] propose a framework that utilizes local caching to scale AR as the number of users increases. By making dynamic edge decisions based on expected workload and job execution location, they optimize end-to-end latency. Jo *et al.* [48] enhance IoT objects with a metadata layer accessed directly by AR clients, bypassing the need for a server or gateway. This allows AR applications to dynamically obtain the most relevant data, supporting scalable AR experiences. Efforts have also been made to enhance computer vision performance in AR. Behzadan *et al.* [18] propose a scalable algorithm that addresses incorrect object occlusion in dynamic and real-time AR experiences. These studies collectively demonstrate that scaling AR systems is feasible by optimizing the AR pipeline and leveraging external resources for processing and connectivity.

Orchestrating AR Services. With the increasing interest in VR/AR glasses, mobile AR, and the Metaverse [8, 26], computation offloading of GPU-intensive tasks to edge servers results in several benefits. Most notably, reduced end-user device energy consumption and higher accuracy by utilizing larger ML models at minimal latency cost [25, 75]. However, there is limited research on orchestrated distributed AR systems. AR processing either relies on a single monolithic service, which is difficult to provision and scale at the edge [60], or on custom-built solutions, hard to distribute, monitor, and manage. Heo *et al.* [43] propose a distributed XR system where pipeline components are selectively offloaded. While their system offers flexibility, it lacks considerations for automated deployment and resource management. Ahn *et al.* [7] address energy efficiency in mobile AR applications through edge-based service orchestration schemes. They optimize the trade-off between energy consumption, latency, and AR accuracy for multiple clients. However, their evaluation does not encompass AR systems with multiple pipeline components. Ren *et al.* [79] focus on WebAR apps and employ distributed edge system orchestration with 5G, using WebAR latency and accuracy quality of service (QoS) metrics for service scheduling and migration. Their validation is based on simulations rather than real-world deployments. Wang *et al.* [88] present an optimal placement algorithm that considers server placement and maintenance criteria to determine offloading locations. In contrast, our work investigates the practical challenges while orchestrating AR pipeline services over a distributed edge infrastructure.

3 SYSTEM DESIGN & SETUP

In this section, we first present the design of our AR pipeline, followed by our experiment methodology details.

3.1 `scAtteR`: A Distributed AR Pipeline

A typical AR application has several components, e.g., frame preprocessing, object detection, recognition, tracking, etc., decouplable as distinct pipeline microservices [17, 97]. However, designing such applications is challenging due to intertwined ML models across several core functions along with dependencies in communication semantics and context (i.e., *state*) of upstream function blocks. For example, an object tracking service relies on *consistent* input from object detection and recognition functions simultaneously to follow objects across frames accurately (we explore these overheads later). Comparatively, a distributed AR pipeline is promising since each complex function block can be scaled out (instead of the entire application) to support concurrent client request load. Past research has demonstrated that pipeline parallelism could provide high throughput, low latency, and increased portability in distributed infrastructures [16, 46, 52].

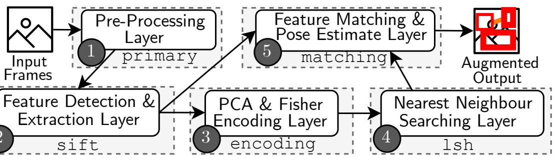
Input Frames
Pre-Processing Layer
1
primary
Feature Matching & Pose Estimate Layer
5
matching
Augmented Output
Feature Detection & Extraction Layer
2
sift
PCA & Fisher Encoding Layer
3
encoding
Nearest Neighbour Searching Layer
4
lsh

Figure 1: High level overview of `scAtteR` pipeline.

Design & Implementation. We draw inspiration from existing research to design and implement a representative DSP-based AR application, `scAtteR` (see Figure 1). We use independent containerized microservices for each pipeline function block to enable low overhead portability, isolation, and delivery of the services. This design approach is convenient for flexibly offloading in multi-tenant edge environments [60]. `scAtteR` captures the core real-world AR operation as it analyzes live video streams by (i) detecting and recognizing objects in-frame and (ii) tracking them across multiple frames[1]. `scAtteR` enables flexibility as there are *five* containerized functional blocks (shown as grey boxes) that can be independently deployed and scaled across different hardware platforms. `scAtteR`'s end-to-end operation is as follows. A client streams video from a (live/pre-recorded) camera source and feeds the frame data to the pipeline ingress – `primary` ➊. `primary` pre-processes frames (grayscaling and dimension reduction) and transforms input video to the requirements of the subsequent pipeline stages. The frame is passed to `sift` ➋, which performs object detection using SIFT [62], extracts recognized features from each frame as *descriptors* and forwards them to `encoding` ➌. `encoding` reduces/compresses feature descriptors through Principal Component Analysis (PCA) encoding and Fisher encoding [73]. `lsh` ➍ maps Fisher vectors from `encoding` into Local Sensitive Hashing (LSH) tables to find nearest neighbors – enabling `matching` ➎ to recognize object features to reference images in the training dataset. Correlating feature-matched results with the original from `sift`, `matching` also calculates object poses, allowing it to track objects across frames also using `sift`'s output data. The processed frame containing bounding boxes is delivered to the client, which renders it.

Note that all `scAtteR` services are GPU-dependent except for `primary`. With GPU offloading being one of the driving factors for distributed AR [97], we offload object detection, pose estimation and encoding workload as a pipelined replicable workflow [27, 61, 78]. The resulting architecture features non-linear component interactions alternating stateless and stateful services, a common practice

[1]Complex AR applications extend this core functionality by co-locating virtual objects within the physical scene [91].

in many AR systems [17, 43, 92]. sift is stateful as it stores data in-memory for matching (till timeout). Intermediary results transferred between services include client ID, frame number, client's IP address and port number, and the current pipeline step – allowing us to map multiple client inputs to the same service instance. To ensure real-time operation, we (i) use UDP for end-to-end communication, and (ii) each service only processes one frame at a time to avoid request queue build-up. Outstanding requests arriving at busy services are dropped (we explore the impact of this in scAtteR++).

3.2 Experiment Setup

Testbed Infrastructure. Our setup uses two edge servers, *Edge 1* (*E1*) and *Edge 2* (*E2*), and a cloud GPU instance in AWS. *E1* has an Intel i9 CPU, two NVIDIA RTX 2080 GPUs, and 128 GB memory. *E2* is a more capable rack server with two AMD EPYC 7302 CPUs, two NVIDIA A40 GPUs and 264 GB memory. We use local edge machines since existing edge offerings from public cloud operators do not include GPUs. We also virtualize our clients as containers and deploy them on Intel NUC NUC6i5SYB machines, allowing us to scale pipeline loads dynamically. The client NUCs are connected directly to *E1* via Ethernet with ≤ 1 ms RTT while *E2* is accessible to *E1* via LAN in 2–4 hops (RTT ≈ 3 ms). The cloud machine has four Intel Broadwell E5-2686 v4 vCPUs, NVIDIA Tesla V100 GPU, 64 GB memory, and has an RTT of ≈ 15 ms. Our infrastructure resembles a typical edge-cloud continuum [34] with representative local edge machine (*E1*), cellular-hosted edge (*E2*), and remote cloud.

Orchestration. We choose machines with different GPU architectures (*E1*: GeForce RTX, *E2*: Ampere, *C*: Tesla) to accommodate for expected heterogeneity in edge-cloud environments. While we ensure service portability across machines by installing the same CUDA drivers[2], we still need to manually map container images compiled for different architectures to different targets at runtime. We automate this using an edge-native orchestrator, Oakestra [15], for resource and service management. Oakestra is an open-source framework [12] and allows us to flexibly deploy scAtteR on our heterogeneous infrastructure by specifying high-level hardware constraints and each service's demands as service level agreement (SLA). Compared to popular solutions such as Kubernetes [10], KubeEdge [28], K3s [29], MicroK8s [21], Oakestra is significantly more lightweight - consuming fewer resources for core orchestration tasks. We also exploit Oakestra for inter-service communication, load balancing requests across multiple service replicas (see §4), and automatically re-deploying services upon failures.

Performance Metrics. A breadth of existing work evaluates the quality of experience (QoE) performance of AR applications. QoE is a highly subjective measure and requires extensive user studies [33, 81] and modeling of user preferences [72, 93]. While some works use implicit metrics, such as physiological biomarkers, to quantify AR QoE [51], there is a need for objective measurements that quantify and compare AR performance to deployed hardware operation.

As such, we collect both hardware consumption (from Oakestra) and end-to-end QoS (from scAtteR) statistics during experiments for a more holistic analysis. To ensure repeatability across runs, each client replays a pre-recorded 10 s, 30 FPS, 720p video captured from a smartphone. The video depicts a workplace environment with objects such as a monitor, keyboard, and table. Each experiment run lasts five minutes, and intermediary files are flushed between runs. We calculate the following metrics from experiment logs. (1) *Frame rate (FPS)* denotes the number of successfully analyzed frames per second by the AR pipeline. The metric encapsulates augmentation stability and, therefore, directly correlates to end-user experience [66]. (2) *End-to-end (E2E) latency* is the delta time between the input and the final processed frame and denotes the processing latency of the pipeline. To maintain smooth augmentation, the E2E latency should ideally be smaller than the inter-frame time of input FPS camera stream [96]. (3) *Service latency* is the processing time of each pipeline service. Finally, (4) we measure *memory, CPU and GPU usage*. We normalize the CPU and GPU utilization against the total number of available cores, which allows us to compare performance over edge-cloud machines with different capacities.

4 EVALUATION

Edge Deployment. We establish a baseline performance of scAtteR in our edge infrastructure in *four* deployment configurations: (a) *C1* – all scAtteR services are deployed on *E1*, (b) *C2* – all services are deployed on *E2*, (c) *C12* – primary and sift on E1 and encoding, lsh, and matching on E2, i.e. pipeline order $[E1, E1, E2, E2, E2]$, and (d) *C21* – similar to C12 but with *E1* and *E2* services swapped, i.e. $[E2, E2, E1, E1, E1]$. Both *C12* and *C21* decouple the heaviest and the only stateful service, sift, from the rest of the pipeline. Although, service distribution comes at the cost of additional latency (we explore the impact of variable network latency and packet loss in A.1.1) Figure 2 shows service QoS and hardware utilization metrics with increasing client load. With a single client, scAtteR can achieve ≥ 25 FPS ($\approx 85\%$ success rate – not shown) for all configurations with E2E latency of ≈ 40 ms.Compared to the best performer, *C21*, single machine deployments (*C1* and *C2*) only observe minor elevation in latency (≈ 2 ms and 4 ms respectively) but consume considerably more CPU and GPU. We observe reduced CPU/GPU utilization in *C2* compared to *C1*, explained by the hardware capabilities of the former, which lead to faster frame processing times.

C12 observed the highest service latency among all configurations, ≈ 4 ms slower that *E1*. We also observe that scAtteR's performance degrades significantly with increasing concurrent clients. After careful examination, we identify the bottleneck to be interdependence between sift and matching. Such dependency loops are known to amplify the *backpressure phenomena* in streaming pipelines. Recall that sift, after receiving input from primary, maintains the frame's state until matching requests the extracted features data for that frame. Therefore, sift observes 2× request load compared to others, increasing significantly with increasing clients. Consequently, matching starts discarding requests at increasing loads since it is busy waiting for sift's output (we see frame success rate drop to 80% with four clients). Backpressure mitigation strategies, as discussed in [42], may not be effective, as the bottleneck not only lies in the processing complexity of the service but in the dependency loop – common practice in pipelined AR systems [17, 43]. Interestingly, we also observe a counter-intuitive trend of declining CPU/GPU utilization with increasing clients. The high request drop due to congestion at earlier stages of the

[2]Binary compatibility is not always guaranteed and requires compiling the application with the correct sm code version [71].

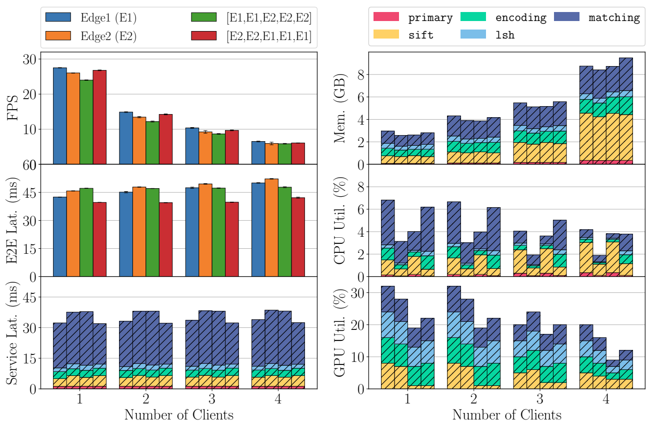

Figure 2: Baseline application performance on edge. FPS and E2E latency over edge servers E1 and E2 for different placement configurations ordered as [primary, sift, lsh, encoding, matching] with the increasing concurrent clients. Resources utilization of each service (stacked bars) ordering corresponds with the same placement configuration.

pipeline leads to this behaviour. The utilization decreases as services stall and do not process any requests due to the existing frame backlog. Contrarily, memory utilization increases several folds. This is due to `sift` storing intermediate results in-memory while it waits for `matching`'s request. The interaction also causes side-effects if `matching` drops incoming frames since `sift` will keep results in memory – which can limit its deployment over memory-constrained edge hardware. We posit that the correlation between application load and resource consumption is non-linear and application/resource-specific. For this reason, it is hard to predict QoS and, consequently, QoE metrics from resource usage, as also reported in [39, 85]. In the context of AR, application-awareness and context-specific scheduling in containerized workloads is an open research problem with limited attempts [82]. We also observe higher jitter (Δ inter-frame receive time) with increasing clients due to increased frame drops (see fig. 10 in Appendix A). Note that our results are not influenced by design artifacts since such complex interactions are commonplace in DSP pipelines [6, 31, 47, 95].

Insights and Recommendations:

(I) Hardware utilization alone does not reflect variations in end-to-end application performance.

(II) Congestion in the inbound service interface should be limited, e.g., by reducing internal service dependencies.

Service Scalability. A leading benefit of our orchestrated setup is that it flexibly allows services to scale up/down to handle request spikes without duplicating the pipeline. We now shed light on the impact of such replications in three different configurations. *First*, in `[2,2,1,1,1]` we replicate the ingress object detection services, i.e. `primary` and `sift`, on *E1* and *E2*. These services have a critical impact on end-to-end performance, more specifically for `sift` (see fig. 2), and may benefit from replication. In `[1,2,1,1,2]`, we scale `sift` and object tracking `matching` since they are the primary cause of bottleneck in multi-client experiments. Finally, in `[1,2,2,1,2]`, we extend the previous configuration and replicate object recognition ingress, `encoding`. We rely on `Oakestra`'s (round-robin) load

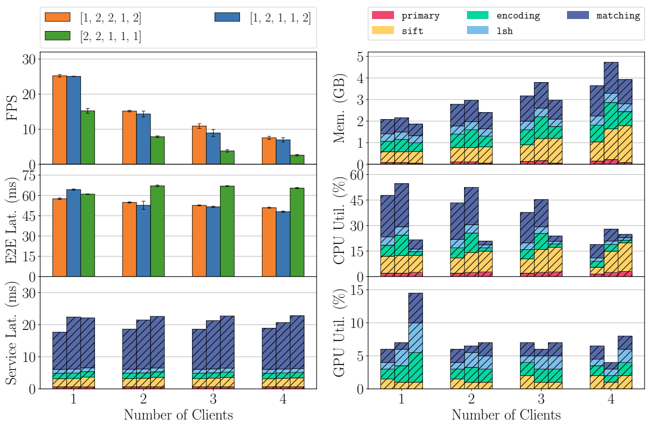

Figure 3: Impact of service scalability. QoS (FPS & E2E latency) over E2 with another replica on E1 for ordering [primary, sift, lsh, encoding, matching]. Resource utilization of each service (stacked bars) follows the same scaling configuration.

balancing to distribute requests across multiple service replicas. Note that frames balanced across `sift` instances remain tied to that replica due to state restrictions. Figure 3 shows our results.

Scaling services does not always result in performance improvements as configuration `[2,2,1,1,1]` observes 26% FPS reduction compared to baseline. Increased input FPS causes this, thanks to replicated ingress that congest the remaining single-instance services. Note that `Oakestra` remains unaware of such application-specific bottlenecks since it can only monitor underlying hardware utilization, which does not increase proportionally. On the other hand, the orchestrator observes $\approx$ 30% reduction in service latency, which might convey that the application is performing better than edge-only deployment. The results from `[1,2,1,1,2]` align with a single instance baseline, showing 20% FPS degradation with concurrent clients. Similar to the `[2,2,1,1,1]`, we observe increased congestion at `encoding` in this configuration (see memory in fig. 3). The cause, again, is state synchronization between `sift` and dependent services. Specifically, due to state tie-ins, the benefits of reducing pressure by balancing the traffic between `primary-sift` (in `[2,2,1,1,1]`) and `sift-matching` (in `[1,2,1,1,2]`) is limited. If a dependent service submits a request to a congested `sift` instance, the request cannot be load-balanced and will timeout. On the other hand, `[1,2,2,1,2]` is the best-performing configuration, achieving 15% and 10% FPS improvement for two and three concurrent clients, as it limits the pipeline ingress load and distributes requests across multiple replicas of detection, recognition and tracking services. Note that the improvement comes at the cost of $\approx$ 30% end-to-end latency elevation due to load balancing overhead.

Insights and Recommendations:

(III) Interdependence on stateful services in DSP can severely affect the scalability potential.

(IV) An application-aware orchestrator that incorporates internal application metrics alongside hardware utilization may prove more effective in the heterogeneous edge-cloud continuum.

Cloud Deployment. We now provide a contrasting viewpoint in Figure 4 which showcases `scAtteR`'s performance on the public AWS cloud. We only showcase results from single VM instance

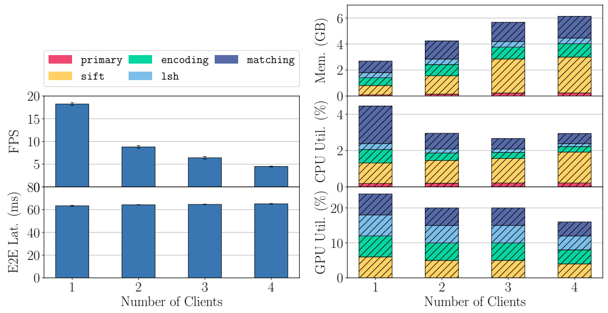

Figure 4: Cloud-only deployment. Left shows `scAtteR`'s QoS and right shows each service's hardware utilization.

configuration since (i) the cloud instance's capabilities far exceed `scAtteR`'s operational requirements and (ii) we observed little-to-no difference when the microservices were distributed over multiple instances due to homogeneous hardware interconnected by a consistent low-latency network.

Recall that the major differences between the edge and cloud setups are (i) ≈ 15 ms latency between clients and `scAtteR` ingress and (ii) virtualized hardware. Our cloud deployment achieves a lower 18.2 FPS (median), compared to 25 FPS in single edge configurations (*C1* or *C2*), along with lower frame success rate (64%). Note that the performance decrease is not due to hardware bottlenecks since `scAtteR` uses less than 5%, 25%, and 2% CPU, GPU, and memory, respectively. One reason could be that the virtualized application is not optimized for the Tesla GPU architecture and affects performance. Another reason could be the increased network latency between the clients and the cloud, delaying overall processing time (evident in fig. 4). We explore the impact of latency further through hybrid edge-cloud deployment, with `primary` on *E1* and the rest of the pipeline in the cloud (see fig. 11 in Appendix). We observe ≈ 2× increase in latency compared to a cloud-only with significant application performance degradation. While our preliminary examination reveals frame drops over the public Internet path as the primary contributor, we leave the thorough investigation to future work. In cloud-only deployment, the end-to-end latency sees a noticeable increase of ≈ 20 ms compared to the edge, corresponding to increased client-ingress access latency. We also observe a slightly higher jitter in received frames (compared to *C1* and *C2*). A closer investigation reveals the cause to be latency fluctuations between client(s) and the cloud machine.

Insights and Recommendations:
(V) While virtualization can help with portability, QoS can vary based on underlying GPU/CPU architecture.
(VI) Network latency and jitter affect real-time AR operation and require proactive measures within the application.

5 `scAtteR++`

Our experiments with `scAtteR` uncover design and implementation factors that can hinder the wide-scale deployment of multi-client AR. We now present `scAtteR++` to elucidate the performance improvement possible through our recommendations (see fig. 5). We strategically redesign `sift` to operate statelessly to remove the dependency on `matching`. We encode the frame's state and relevant data within the frame itself, packaging the required SIFT data at

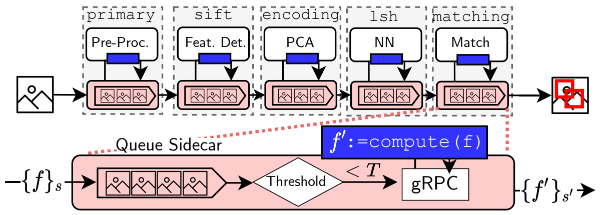

Figure 5: `scAtteR++` pipeline design.

the cost of increasing the output frame size from ≈ 180 KB to ≈ 480 KB. Stateless applications typically use the data stores or caches of the current application and not persistent storage to decouple state [35]. This technique is found in other services, such as networking for stateless network functionality where networks can be scaled efficiently [50]. Our approach to creating a statelessness pipeline is comparable by storing the state data in the forwarded data frames between services, avoiding unnecessary writing on storage. This allows us to create a more agile pipeline supporting easier application scaling, independent service deployment, and better fault tolerance [36]. Regarding recommendations (II) and (VI), we introduce a sidecar component attached to each service's ingress. This empowers `scAtteR++` to efficiently handle concurrent processing requests that surpass a service's processing rate. The sidecar performs queuing and filtering of the incoming requests and makes a gRPC call to the attached service for processing outstanding frames in filtered FIFO order. The sidecar also collects metrics (i.e., queueing and processing time or threshold ratio) that are attached to the data's state. The sidecar design is a well-known pattern in the microservices community [20] and is used in several projects [1–3]. In recent work, Lee *et al.* [56] propose a similar solution tailoring a sidecar process for an in-network acceleration of ML workload at the edge. This work is tightly coupled with Istio and Envoy, which can limit the solution's applicability in highly distributed or constrained environments due to high overhead [99]. The solution proposed in this work is built specifically for data streaming applications and uses Oakestra's semantic addressing to achieve transparent load balancing with minimal overhead [15].

We re-conduct our edge and scalability experiments with `scAtteR++`. We set the timing threshold to 100 ms, in line with the maximum tolerable latency in XR applications [43, 60, 67]. Figure 6 demonstrates significant performance improvements with `scAtteR++` compared to `scAtteR` in all edge deployment configurations. In single client configurations `scAtteR++` achieves a 9% FPS increase (+17.6% success rate), while with multiple concurrent clients, we record a substantial 2.5× frame rate increase. Specifically, `scAtteR++` consistently maintains 12 FPS with four clients (with *C12* achieving ≈ 20 FPS) where `scAtteR` struggled to maintain > 5 FPS (see fig. 2). The sidecar queue is crucial in this improvement as it buffers frame requests during service lag. Although `scAtteR++` incurs slightly higher per service latency, most evident in `primary`, it effectively reduces request drops and improves resource utilization, which scales proportionally with client load. Note that the performance decrease with increasing load in `scAtteR++` is due to throttling (see GPU utilization) instead of request drops in `scAtteR`. While `scAtteR` could not combat this by scaling out due to stateful `sift` restrictions, `scAtteR++` shows clear benefits with replications. As shown in fig. 7, `scAtteR++` achieves a 2.8× improvement by achieving a

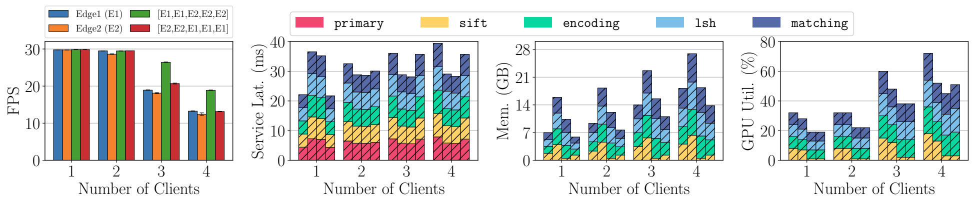

Figure 6: Baseline performance deployed on the edge using the sidecar component. Experiment methodology corresponds to that shown in fig. 2.

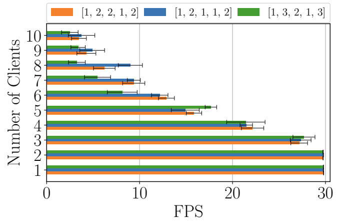

Figure 7: FPS when increasing scaled services and clients.

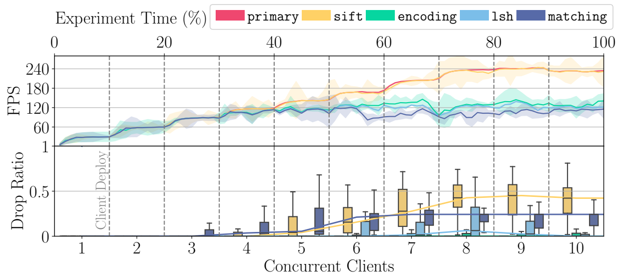

Figure 8: Sidecar analytics correlating each scAtteR++ service's FPS in fig. 7 (line denotes median) to queued request drops with increasing clients at fixed one-minute intervals.

similar framerate with eight clients compared to what scAtteR achieved with four on the same cluster.

Figure 8 presents the correlation between ingress FPS per service, the number of clients (from 1 to 10), and the drop rate of the queue enforced to maintain the latency threshold. We observe how the maximum ingress FPS starts plateauing at about 4 clients (≈ 90 ingress FPS) for the latest stages of the pipeline with drop rate increasing at matching starting from 3 clients. The delay introduced by the previous stages of the pipeline and the growing frames queue forces the threshold mechanism to drop 10% at first, then up to 40% of the frames. Sift's drop rate increases up to 50% in the range 8 − 10 clients, halving the ingress FPS of the latest stages of scAtteR++. Primary's pre-processing reached a max throughput of 240FPS, disregarding ingress UDP datagrams afterwards. The high drop ratio depicts the saturation of the pipeline max throughput with the available hardware, the build-up of the backpressure, and the need for further scaling out horizontally or vertically. We argue, however, that vertical scalability and model optimization help shift the saturation point of the model to a higher number of clients but require separate considerations. The former approach must deal with resource contention, which is critical especially for GPUs [45, 90], while the latter helps improve inference speed with faster models (e.g., substituting SIFT with [59]) but without a horizontally scalable design the application will incur in the same issues discussed in §4 but delayed to a higher number of clients.

6 CONCLUSION & FUTURE WORK

In this work, we investigated several challenges while deploying a distributed AR application, scAtteR, on a heterogeneous edge-cloud infrastructure. Our work builds upon the commonalities of the architectural design from related edge/cloud AR research to (i) provide the research community with a demonstrable prototype of distributed AR application released as an open-source project [14], (ii) highlight potential (non-apparent) bottlenecks in such designs and the trade-offs between flexibility and performance, and (iii) uncover future research directions and common approaches that can be applied from related works to this field. As such, our study serves as a foundation for further investigations, encouraging the integration of the following solutions.

Application-Aware Orchestration. Our study sheds light on the challenges hindering real-world deployments of XR applications. Most notably, we discover that application-level QoS is not proportionally reflected in hardware-level utilization and is highly dependent on implementation. This finding has significant implications since orchestration frameworks are oblivious to the internal operations of virtualized services and only rely on hardware-level metrics [11, 24, 55, 83]. Recent research from related fields also conceptually echoes our intuitions [37, 56, 63]. Our recommendations, (I) and (IV), propose the need for application-aware orchestrators that simultaneously correlate application and hardware metrics for optimizations. One potential approach is extending the sidecar in scAtteR++ to bridge across the virtualization boundary, providing predefined hooks for the orchestrator to access internal application metrics. However, this approach requires thorough investigation to address potential security concerns and vulnerabilities.

Real-World Relevance. We made several design decisions with scAtteR that may/may not encompass the state-of-the-art in AR development, given the rapidly growing scope of the field. Industry-driven XR platforms, such as [41, 70] allow the streaming of AR applications from the Cloud. However, to exploit the low latency of the edge environments, the user's application design requires thorough decomposition and flexibility considerations. There is a concerted effort from the community to optimize software development kits [4, 23] to enable AR functionality across platforms, including OSes [30, 38, 54], web browsers [49, 74], and more. Recent advancements explore model accuracy enhancement [86], footprint [77] reduction, as well as network performance improvements through 5G [40, 68], transport [5], and application layer protocols [19], which this study does not address. Additionally, the hardware configurations used in this paper can be extended further to explore the effects of vertical scalability and resource contention.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and the shepherd for their comments and insights during the review process. This work was partly supported by the Federal Ministry of Education and Research of Germany (BMBF) project 6G-Life (16KISK002) and the EU Health and Digital Executive Agency (HADEA) program under Grant Agreement No 101092950 (EDGELESS project).

REFERENCES

[1] 2022. Envoy. Web document. https://www.envoyproxy.io/
[2] 2022. Istio. Web document. https://istio.io/
[3] 2022. Linkerd. Web document. https://linkerd.io/
[4] A-Frame. 2023. A-Frame: Web Framework for Building Virtual Reality Experiences. Online. https://aframe.io/
[5] Maha Abdallah, Carsten Griwodz, Kuan-Ta Chen, Gwendal Simon, Pin-Chun Wang, and Cheng-Hsin Hsu. 2018. Delay-Sensitive Video Computing in the Cloud: A Survey. *ACM Trans. Multimedia Comput. Commun. Appl.* 14, 3s, Article 54 (jun 2018), 29 pages. https://doi.org/10.1145/3212804
[6] Neil Agarwal and Ravi Netravali. 2023. Boggart: Towards General-Purpose Acceleration of Retrospective Video Analytics. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 933–951. https://www.usenix.org/conference/nsdi23/presentation/agarwal-neil
[7] Jaewon Ahn, Joohyung Lee, Dusit Niyato, and Hong-Shik Park. 2020. Novel QoS-Guaranteed Orchestration Scheme for Energy-Efficient Mobile Augmented Reality Applications in Multi-Access Edge Computing. *IEEE Transactions on Vehicular Technology* 69, 11 (2020), 13631–13645. https://doi.org/10.1109/TVT.2020.3020982
[8] Tiago Andrade and Daniel Bastos. 2019. Extended reality in iot scenarios: Concepts, applications and future trends. In *2019 5th Experiment International Conference (exp. at'19)*. IEEE, 107–112.
[9] Apple Inc. Accessed 2023. Apple Vision Pro. https://www.apple.com/apple-vision-pro/.
[10] The Kubernetes Authors. 2023. Kubernetes. https://kubernetes.io/.
[11] The Kubernetes Authors. 2023. Scheduling and Eviction. https://kubernetes.io/docs/concepts/scheduling-eviction/.
[12] The Oakestra Authors. 2023. Oakestra: An Orchestration Framework for Edge Computing. https://www.oakestra.io/
[13] Haythem Bahri, David Krčmařík, and Jan Kočí. 2019. Accurate Object Detection System on HoloLens Using YOLO Algorithm. In *2019 International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO)*. 219–224. https://doi.org/10.1109/ICCAIRO47923.2019.00042
[14] Giovanni Bartolomeo, Jacky Cao, Xiang Su, and Mohan Nitinder. 2023. Artifact - Characterizing Distributed Mobile Augmented Reality Applications at the Edge. https://github.com/cao-jacky/characterizing_edge_mar
[15] Giovanni Bartolomeo, Mehdi Yosofie, Simon Bäurle, Oliver Haluszczynski, Nitinder Mohan, and Jörg Ott. 2023. Oakestra: A Lightweight Hierarchical Orchestration Framework for Edge Computing. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/atc23/presentation/bartolomeo
[16] Pablo Basanta-Val, Norberto Fernández-García, Luis Sánchez-Fernández, and Jesus Arias-Fisteus. 2017. Patterns for Distributed Real-Time Stream Processing. *IEEE Transactions on Parallel and Distributed Systems* 28, 11 (2017), 3243–3257. https://doi.org/10.1109/TPDS.2017.2716929
[17] Simon Bäurle and Nitinder Mohan. 2022. ComB: A Flexible, Application-Oriented Benchmark for Edge Computing. In *Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking* (Rennes, France) *(EdgeSys '22)*. Association for Computing Machinery, New York, NY, USA, 19–24. https://doi.org/10.1145/3517206.3526269
[18] Amir H. Behzadan and Vineet R. Kamat. 2010. Scalable Algorithm for Resolving Incorrect Occlusion in Dynamic Augmented Reality Engineering Environments. *Computer-Aided Civil and Infrastructure Engineering* 25, 1 (2010), 3–19. https://doi.org/10.1111/j.1467-8667.2009.00601.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8667.2009.00601.x
[19] Tristan Braud, Farshid Hassani Bijarbooneh, Dimitris Chatzopoulos, and Pan Hui. 2017. Future Networking Challenges: The Case of Mobile Augmented Reality. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 1796–1807. https://doi.org/10.1109/ICDCS.2017.48
[20] Brendan Burns and David Oppenheimer. 2016. Design patterns for container-based distributed systems. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*.
[21] Canonical. 2023. MicroK8s - ZERO-OPS Kubernetes for developers, edge And IoT: Microk8s. https://microk8s.io/
[22] Jacky Cao, Kit-Yung Lam, Lik-Hang Lee, Xiaoli Liu, Pan Hui, and Xiang Su. 2023. Mobile Augmented Reality: User Interfaces, Frameworks, and Intelligence. *ACM Comput. Surv.* 55, 9, Article 189 (jan 2023), 36 pages. https://doi.org/10.1145/3557999
[23] CareAR. 2023. CraftAR: Augmented Reality for the Future. Online. https://carear.com/craftar/
[24] Carmen Carrión. 2022. Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges. *ACM Comput. Surv.* 55, 7, Article 138 (dec 2022), 37 pages. https://doi.org/10.1145/3539606
[25] Koyela Chakrabarti. 2021. Deep learning based offloading for mobile augmented reality application in 6G. *Computers and Electrical Engineering* 95 (2021), 107381.
[26] Luyi Chang, Zhe Zhang, Pei Li, Shan Xi, Wei Guo, Yukang Shen, Zehui Xiong, Jiawen Kang, Dusit Niyato, Xiuquan Qiao, et al. 2022. 6G-enabled edge AI for Metaverse: Challenges, methods, and future research directions. *Journal of Communications and Information Networks* 7, 2 (2022), 107–121.
[27] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, Mark Holderbaugh, Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Jerry Peng, et al. 2016. Benchmarking streaming computation engines: Storm, flink and spark streaming. In *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*. IEEE, 1789–1792.
[28] CNCF. 2017. KubeEdge. https://github.com/kubeedge/kubeedge. Retrieved May 24, 2022 from https://kubeedge.io/en/
[29] CNCF. 2019. Lightweight Kubernetes | K3S. https://k3s.io. Retrieved May 24, 2022 from https://k3s.io
[30] Microsoft Corporation. 2023. Microsoft HoloLens. Web document. https://www.microsoft.com/en-us/hololens Retrieved from https://www.microsoft.com/en-us/hololens.
[31] Vittorio Cozzolino, Leonardo Tonetto, Nitinder Mohan, Aaron Yi Ding, and Jorg Ott. 2022. Nimbus: Towards Latency-Energy Efficient Task Offloading for AR Services. *IEEE Transactions on Cloud Computing* (2022), 1–1. https://doi.org/10.1109/TCC.2022.3146615
[32] The Khang Dang, Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Jörg Ott, and Jussi Kangasharju. 2021. Cloudy with a Chance of Short RTTs: Analyzing Cloud Connectivity in the Internet. In *Proceedings of the 21st ACM Internet Measurement Conference* (Virtual Event) *(IMC '21)*. Association for Computing Machinery, New York, NY, USA, 62–79. https://doi.org/10.1145/3487552.3487854
[33] Elijs Dima, Kjell Brunnström, Mårten Sjöström, Mattias Andersson, Joakim Edlund, Mathias Johanson, and Tahir Qureshi. 2020. Joint effects of depth-aiding augmentations and viewing positions on the quality of experience in augmented telepresence. *Quality and User Experience* 5, 1 (10 Feb 2020), 2. https://doi.org/10.1007/s41233-020-0031-7
[34] P. J. Escamilla-Ambrosio, A. Rodríguez-Mota, E. Aguirre-Anaya, R. Acosta-Bermejo, and M. Salinas-Rosales. 2018. *Distributing Computing in the Internet of Things: Cloud, Fog and Edge Computing Overview*. Springer International Publishing, Cham, 87–115. https://doi.org/10.1007/978-3-319-64063-1_4
[35] I. Farris, T. Taleb, H. Flinck, and A. Iera. 2018. Providing ultra-short latency to user-centric 5G applications at the mobile network edge. *Transactions on Emerging Telecommunications Technologies* 29, 4 (2018), e3169. https://doi.org/10.1002/ett.3169 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/ett.3169 e3169 ett.3169.
[36] Christoph Fehling, Frank Leymann, Ralph Retter, Walter Schupeck, and Peter Arbitter. 2014. *Cloud computing patterns: fundamentals to design, build, and manage cloud applications*. Springer.
[37] Xenofon Foukas and Bozidar Radunovic. 2021. Concordia: Teaching the 5G VRAN to Share Compute. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (Virtual Event, USA) *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 580–596. https://doi.org/10.1145/3452296.3472894
[38] Victor Fragoso, Steffen Gauglitz, Shane Zamora, Jim Kleban, and Matthew Turk. 2011. TranslatAR: A mobile augmented reality translator. In *2011 IEEE Workshop on Applications of Computer Vision (WACV)*. 497–502. https://doi.org/10.1109/WACV.2011.5711545
[39] Ziyan Fu, Ju Ren, Deyu Zhang, Yuezhi Zhou, and Yaoxue Zhang. 2022. Kalmia: A heterogeneous QoS-aware scheduling framework for DNN tasks on edge servers. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 780–789.
[40] Moinak Ghoshal, Pranab Dash, Zhaoning Kong, Qiang Xu, Y Charlie Hu, Dimitrios Koutsonikolas, and Yuanjie Li. 2022. Can 5G mmWave Support Multi-user AR?. In *Passive and Active Measurement: 23rd International Conference, PAM 2022, Virtual Event, March 28–30, 2022, Proceedings*. Springer, 180–196.
[41] Google. 2023. Immersive Stream AR. Web document. https://cloud.google.com/immersive-stream/xr
[42] Muhammad Hanif, Hyeongdeok Yoon, and Choonhwa Lee. 2020. A Backpressure Mitigation Scheme in Distributed Stream Processing Engines. In *2020 International Conference on Information Networking (ICOIN)*. IEEE, 713–716.
[43] Jin Heo, Ketan Bhardwaj, and Ada Gavrilovska. 2023. FleXR: A System Enabling Flexibly Distributed Extended Reality. In *Proceedings of the 14th Conference on ACM Multimedia Systems* (Vancouver, BC, Canada) *(MMSys '23)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3587819.3590966
[44] Jaylin Herskovitz, Jason Wu, Samuel White, Amy Pavel, Gabriel Reyes, Anhong Guo, and Jeffrey P. Bigham. 2020. Making Mobile Augmented Reality Applications Accessible. In *Proceedings of the 22nd International ACM SIGACCESS Conference on Computers and Accessibility* (Virtual Event, Greece) *(ASSETS '20)*. Association for Computing Machinery, New York, NY, USA, Article 3, 14 pages. https://doi.org/10.1145/3373625.3417006
[45] Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and KyoungSoo Park. 2021. Elastic resource sharing for distributed deep learning. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 721–739.
[46] Anand Jayarajan, Kimberly Hau, Andrew Goodwin, and Gennady Pekhimenko. 2021. LifeStream: A High-Performance Stream Processing Engine for Periodic Streams. In *Proceedings of the 26th ACM International Conference on Architectural

Support for Programming Languages and Operating Systems (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 107–122. https://doi.org/10.1145/3445814.3446725

[47] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: Scalable Adaptation of Video Analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 253–266. https://doi.org/10.1145/3230543.3230574

[48] Dongsik Jo and Gerard Jounghyun Kim. 2016. ARIoT: scalable augmented reality framework for interacting with Internet of Things appliances everywhere. *IEEE Transactions on Consumer Electronics* 62, 3 (2016), 334–340. https://doi.org/10.1109/TCE.2016.7613201

[49] Brandon Jones, Manish Goregaokar, and Rik Cabanier. 2023. WebXR Device API. Web document. https://www.w3.org/TR/webxr/

[50] Murad Kablan, Blake Caldwell, Richard Han, Hani Jamjoom, and Eric Keller. 2015. Stateless Network Functions. In *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization* (London, United Kingdom) *(HotMiddlebox '15)*. Association for Computing Machinery, New York, NY, USA, 49–54. https://doi.org/10.1145/2785989.2785993

[51] Conor Keighrey, Ronan Flynn, Siobhan Murray, and Niall Murray. 2021. A Physiology-Based QoE Comparison of Interactive Augmented Reality, Virtual Reality and Tablet-Based Applications. *IEEE Transactions on Multimedia* 23 (2021), 333–341. https://doi.org/10.1109/TMM.2020.2982046

[52] Shweta Khare, Hongyang Sun, Julien Gascon-Samson, Kaiwen Zhang, Aniruddha Gokhale, Yogesh Barve, Anirban Bhattacharjee, and Xenofon Koutsoukos. 2019. Linearize, Predict and Place: Minimizing the Makespan for Edge-Based Stream Processing of Directed Acyclic Graphs. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing* (Arlington, Virginia) *(SEC '19)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3318216.3363315

[53] Iris Kico and Fotis Liarokapis. 2019. A Mobile Augmented Reality Interface for Teaching Folk Dances. In *Proceedings of the 25th ACM Symposium on Virtual Reality Software and Technology* (Parramatta, NSW, Australia) *(VRST '19)*. Association for Computing Machinery, New York, NY, USA, Article 47, 2 pages. https://doi.org/10.1145/3359996.3364752

[54] Sung Lae Kim, Hae Jung Suk, Jeong Hwa Kang, Jun Mo Jung, Teemu H. Laine, and Joonas Westlin. 2014. Using Unity 3D to facilitate mobile augmented reality game development. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*. 21–26. https://doi.org/10.1109/WF-IoT.2014.6803110

[55] Kubernetes. 2023. Controller Manager. Online. https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-manager/

[56] HyunJong Lee, Shadi Noghabi, Brian Noble, Matthew Furlong, and Landon P Cox. 2022. BumbleBee: Application-aware adaptation for edge-cloud orchestration. In *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*. IEEE, 122–135.

[57] Lik-Hang Lee, Tristan Braud, Pengyuan Zhou, Lin Wang, Dianlei Xu, Zijun Lin, Abhishek Kumar, Carlos Bermejo, and Pan Hui. 2021. All one needs to know about metaverse: A complete survey on technological singularity, virtual ecosystem, and research agenda. *arXiv preprint arXiv:2110.05352* (2021).

[58] Linux. 2023. Traffic control manual. https://man7.org/linux/man-pages/man8/tc.8.html

[59] Bingqiang Liu, Zehua Yin, Xvpeng Zhang, Yi Zhan, Xiaofeng Hu, Guoyi Yu, Yuanjin Zheng, Chao Wang, and Xuecheng Zou. 2022. An Energy-Efficient SIFT Based Feature Extraction Accelerator for High Frame-Rate Video Applications. *IEEE Transactions on Circuits and Systems I: Regular Papers* 69, 12 (2022), 4930–4943.

[60] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge Assisted Real-Time Object Detection for Mobile Augmented Reality. In *The 25th Annual International Conference on Mobile Computing and Networking* (Los Cabos, Mexico) *(MobiCom '19)*. Association for Computing Machinery, New York, NY, USA, Article 25, 16 pages. https://doi.org/10.1145/3300061.3300116

[61] Luyang Liu, Ruiguang Zhong, Wuyang Zhang, Yunxin Liu, Jiansong Zhang, Lintao Zhang, and Marco Gruteser. 2018. Cutting the Cord: Designing a High-Quality Untethered VR System with Low Latency Remote Rendering. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services* (Munich, Germany) *(MobiSys '18)*. Association for Computing Machinery, New York, NY, USA, 68–80. https://doi.org/10.1145/3210240.3210313

[62] David G Lowe. 2004. Distinctive image features from scale-invariant keypoints. *International journal of computer vision* 60 (2004), 91–110.

[63] Ashraf Mahgoub, Karthick Shankar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2021. SONIC: Application-aware Data Passing for Chained Serverless Applications. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 285–301. https://www.usenix.org/conference/atc21/presentation/mahgoub

[64] Roberto Maldonado, Anders Karstensen, Guillermo Pocovi, Ali A Esswie, Claudio Rosa, Olli Alanen, Mika Kasslin, and Troels Kolding. 2021. Comparing Wi-Fi 6 and 5G downlink performance for industrial IoT. *IEEE Access* 9 (2021), 86928–86937.

[65] Ana Malta, Mateus Mendes, and Torres Farinha. 2021. Augmented Reality Maintenance Assistant Using YOLOv5. *Applied Sciences* 11, 11 (2021). https://doi.org/10.3390/app11114758

[66] Microsoft. 2023. App Quality Criteria - mixed reality. https://learn.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/app-quality-criteria-overview

[67] Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. 2020. Pruning Edge Research with Latency Shears. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks* (Virtual Event, USA) *(HotNets '20)*. Association for Computing Machinery, New York, NY, USA, 182–189. https://doi.org/10.1145/3422604.3425943

[68] Diego González Morín, Pablo Pérez, and Ana García Armada. 2022. Toward the Distributed Implementation of Immersive Augmented Reality Architectures on 5G Networks. *IEEE Communications Magazine* 60, 2 (2022), 46–52. https://doi.org/10.1109/MCOM.001.2100225

[69] Stylianos Mystakidis. 2022. Metaverse. *Encyclopedia* 2, 1 (2022), 486–497. https://doi.org/10.3390/encyclopedia2010031

[70] Nvidia. 2023. CloudXR. Web document. https://developer.nvidia.com/blog/deploying-xr-applications-in-private-networks-on-a-server-platform/

[71] NVIDIA. 2023. Nvidia - GPU Compilation. Web document. https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html#gpu-compilation

[72] Rui Pascoal, Ana De Almeida, and Rute C. Sofia. 2020. Mobile Pervasive Augmented Reality Systems—MPARS: The Role of User Preferences in the Perceived Quality of Experience in Outdoor Applications. *ACM Trans. Internet Technol.* 20, 1, Article 7 (feb 2020), 17 pages. https://doi.org/10.1145/3375458

[73] Florent Perronnin, Yan Liu, Jorge Sánchez, and Hervé Poirier. 2010. Large-scale image retrieval with compressed Fisher vectors. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 3384–3391. https://doi.org/10.1109/CVPR.2010.5540009

[74] Xiuquan Qiao, Pei Ren, Schahram Dustdar, Ling Liu, Huadong Ma, and Junliang Chen. 2019. Web AR: A Promising Future for Mobile Augmented Reality—State of the Art, Challenges, and Insights. *Proc. IEEE* 107, 4 (2019), 651–666. https://doi.org/10.1109/JPROC.2019.2895105

[75] Xiuquan Qiao, Pei Ren, Guoshun Nan, Ling Liu, Schahram Dustdar, and Junliang Chen. 2019. Mobile web augmented reality in 5G and beyond: Challenges, opportunities, and future directions. *China Communications* 16, 9 (2019), 141–154.

[76] Qualcomm. 2023. Snapdragon XR2 Platform. https://www.qualcomm.com/news/releases/2022/05/qualcomm-cuts-cord-new-wireless-ar-smart-viewer-reference-design-powered

[77] Xukan Ran, Carter Slocum, Maria Gorlatova, and Jiasi Chen. 2019. ShareAR: Communication-Efficient Multi-User Mobile Augmented Reality. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks* (Princeton, NJ, USA) *(HotNets '19)*. Association for Computing Machinery, New York, NY, USA, 109–116. https://doi.org/10.1145/3365609.3365867

[78] Jie Ren, Ling Gao, Xiaoming Wang, Miao Ma, Guoyong Qiu, Hai Wang, Jie Zheng, and Zheng Wang. 2021. Adaptive computation offloading for mobile augmented reality. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 4 (2021), 1–30.

[79] Pei Ren, Ling Liu, Xiuquan Qiao, and Junliang Chen. 2022. Distributed Edge System Orchestration for Web-based Mobile Augmented Reality Services. *IEEE Transactions on Services Computing* (2022), 1–15. https://doi.org/10.1109/TSC.2022.3190375

[80] Justus Rischke, Peter Sossalla, Sebastian Itting, Frank HP Fitzek, and Martin Reisslein. 2021. 5G campus networks: A first measurement study. *IEEE Access* 9 (2021), 121786–121803.

[81] Thiago Braga Rodrigues, Ciarán Ó Catháin, Noel E. O'Connor, and Niall Murray. 2020. A Quality of Experience assessment of haptic and augmented reality feedback modalities in a gait analysis system. *PLOS ONE* 15, 3 (mar 2020), e0230570. https://doi.org/10.1371/journal.pone.0230570

[82] Khaldoun Senjab, Sohail Abbas, Naveed Ahmed, and Atta ur Rehman Khan. 2023. A Survey of Kubernetes Scheduling Algorithms. 12, 1 (2023). https://doi.org/10.1186/s13677-023-00471-1

[83] Arjun Singhvi, Arjun Balasubramanian, Kevin Houck, Mohammed Danish Shaikh, Shivaram Venkataraman, and Aditya Akella. 2021. Atoll: A Scalable Low-Latency Serverless Platform. In *Proceedings of the ACM Symposium on Cloud Computing* (Seattle, WA, USA) *(SoCC '21)*. Association for Computing Machinery, New York, NY, USA, 138–152. https://doi.org/10.1145/3472883.3486981

[84] Yushan Siriwardhana, Pawani Porambage, Madhusanka Liyanage, and Mika Ylianttila. 2021. A survey on mobile augmented reality with 5G mobile edge computing: architectures, applications, and technical aspects. *IEEE Communications Surveys & Tutorials* 23, 2 (2021), 1160–1192.

[85] Huaiying Sun, Huiqun Yu, Guisheng Fan, and Liqiong Chen. 2020. QoS-aware task placement with fault-tolerance in the edge-cloud. *IEEE Access* 8 (2020), 77987–78003.

[86] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. 2017. Visual SLAM algorithms: A survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications* 9, 1 (2017), 1–11.

[87] The Verge. 2023. Meta Quest 3 VR headset price details. https://www.theverge.com/2023/6/1/23744576/meta-quest-3-vr-headset-price-details. Accessed June 27, 2023.

[88] Lin Wang, Lei Jiao, Ting He, Jun Li, and Max Mühlhäuser. 2018. Service Entity Placement for Social Virtual Reality Applications in Edge Computing. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 468–476. https://doi.org/10.1109/INFOCOM.2018.8486411

[89] Gesa Wiegand, Christian Mai, Kai Holländer, and Heinrich Hussmann. 2019. InCarAR: A Design Space Towards 3D Augmented Reality Applications in Vehicles. In *Proceedings of the 11th International Conference on Automotive User Interfaces and Interactive Vehicular Applications* (Utrecht, Netherlands) *(AutomotiveUI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3342197.3344539

[90] Bingyang Wu, Zili Zhang, Zhihao Bai, Xuanzhe Liu, and Xin Jin. 2023. Transparent {GPU} Sharing in Container Clouds for Deep Learning Workloads. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 69–85.

[91] Jianghao Xiong, En-Lin Hsiang, Ziqian He, Tao Zhan, and Shin-Tson Wu. 2021. Augmented reality and virtual reality displays: emerging technologies and future perspectives. *Light: Science & Applications* 10, 1 (2021), 216.

[92] Jingao Xu, Hao Cao, Zheng Yang, Longfei Shangguan, Jialin Zhang, Xiaowu He, and Yunhao Liu. 2022. {SwarmMap}: Scaling up real-time collaborative visual {SLAM} at the edge. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 977–993.

[93] Longyu Zhang, Haiwei Dong, and Abdulmotaleb El Saddik. 2018. Towards a QoE model to evaluate holographic augmented reality devices. *IEEE MultiMedia* 26, 2 (2018), 21–32.

[94] Lei Zhang, Andy Sun, Ryan Shea, Jiangchuan Liu, and Miao Zhang. 2019. Rendering Multi-Party Mobile Augmented Reality from Edge. In *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video* (Amherst, Massachusetts) *(NOSSDAV '19)*. Association for Computing Machinery, New York, NY, USA, 67–72. https://doi.org/10.1145/3304112.3325612

[95] Qingyang Zhang, Hui Sun, Xiaopei Wu, and Hong Zhong. 2019. Edge Video Analytics for Public Safety: A Review. *Proc. IEEE* 107, 8 (2019), 1675–1696. https://doi.org/10.1109/JPROC.2019.2925910

[96] Wenxiao Zhang, Bo Han, and Pan Hui. 2017. On the Networking Challenges of Mobile Augmented Reality. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network* (Los Angeles, CA, USA) *(VR/AR Network '17)*. Association for Computing Machinery, New York, NY, USA, 24–29. https://doi.org/10.1145/3097895.3097900

[97] Wenxiao Zhang, Bo Han, and Pan Hui. 2018. Jaguar: Low Latency Mobile Augmented Reality with Flexible Tracking. In *Proceedings of the 26th ACM International Conference on Multimedia* (Seoul, Republic of Korea) *(MM '18)*. Association for Computing Machinery, New York, NY, USA, 355–363. https://doi.org/10.1145/3240508.3240561

[98] Wenxiao Zhang, Bo Han, and Pan Hui. 2022. SEAR: Scaling Experiences in Multi-user Augmented Reality. *IEEE Transactions on Visualization and Computer Graphics* 28, 5 (2022), 1982–1992. https://doi.org/10.1109/TVCG.2022.3150467

[99] Xiangfeng Zhu, Guozhen She, Bowen Xue, Yu Zhang, Yongsu Zhang, Xuan Kelvin Zou, Xiongchun Duan, Peng He, Arvind Krishnamurthy, Matthew Lentz, et al. 2022. Dissecting Service Mesh Overheads. *arXiv preprint arXiv:2207.00592* (2022).

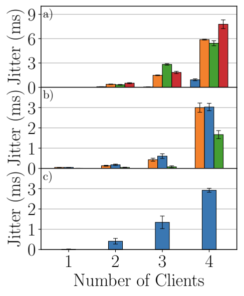

Figure 10: Jitter results from a) baseline edge, b) service scalability, and c) cloud deployment.

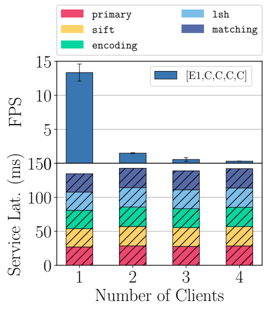

Figure 11: Impact of distributing to the edge and cloud in a hybrid setup.

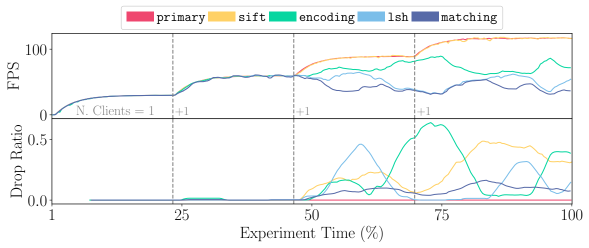

Figure 12: Sidecar analytics correlating each scAtteR++ service's framerate to request drops from the queue. All services are deployed on E1.

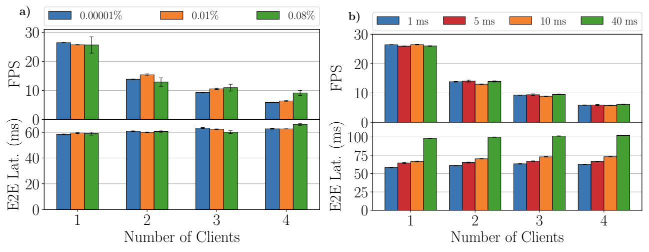

Figure 9: Impact of varying network conditions, specifically packet loss (a) and latency (b), on scAtteR with increasing concurrent clients.

A APPENDIX

A.1 Additional Experiments

A.1.1 Mobile Connectivity. The vision for AR hardware manufacturers and application developers is to enable use cases wherein clients with wireless AR glasses interact with objects in the environment, enabling usecases like augmented tourism, augmented navigation, etc. [76]. To emulate such setups in our infrastructure, we deploy the pipeline on *E2* and use `tc` [58] to introduce packet loss artificially and latency in the link connecting the clients to the pipeline ingress `primary`. Our latency and loss values are inspired by recent measurement studies. Specifically, we emulate LTE (40 ms RTT and 0.08% loss) [32], 5G (10 ms RTT and 0.00001 - 0.01% loss) [80], and WiFi-6 (5 ms RTT and 0.00001 - 0.01% loss) [64]. To emulate mobility, we add 10 ms delay oscillation with 20% probability, and we perform loss measurements with 1 ms delay and latency measurements with 0.00001% loss.

Figure 9a shows that packet loss variations do not drastically impact end-to-end performance but limit the framerate throughput due to frame drops. However, packet loss seems to affect the success rate of the frame transmission and, consequently, the FPS. Interestingly, higher network loss shows minor performance improvement at higher concurrent clients (see fig. 2) due to reduced load at congested services. Similar to packet loss, we do not observe a significantly notable impact of latency on application performance (see fig. 9b). Note that the experiments were conducted with `scAtteR`, which does not drop packets if it exceeds a timing threshold (unlike `scAtteR++`). As a result, frames that were no longer within the maximum tolerable 100 ms end-to-end latency with 10 ms and 40 ms settings were not dropped – resulting in a consistent framerate.

A.1.2 Hybrid Edge-Cloud Deployment. Figure 11 shows the preliminary performance of `scAtteR` in hybrid edge-cloud deployment. We deploy the ingress `primary` service on *E1* while the rest of the pipeline is on the cloud. We also tested other configurations that decoupled the pipeline across *E1*, *E2* and cloud but found significant artifacts due to state dependencies between `sift` and other functions. We find that `scAtteR`'s performance severely degrades in hybrid settings, largely due to increased frame drops across services deployed on edge and cloud machines. Note that improved network protocols [19, 40, 68] instead of UDP may help alleviate this, which we plan to explore in future extensions.

A.2 Sidecar Queue Application Analytics

We extend the sidecar in `scAtteR++` to access service QoS metrics, allowing us to correlate queue drops with per-service framerate (see fig. 12) as we periodically increase the client load (vertical blue lines). We observe that all services keep up with the increasing load until we introduce the third client. At 90 FPS input rate, each service, except `primary` and `sift`, show reduced framerate, with `encoding` service dropping almost 50% of packets from the queue. This is because while `sift` is able to process frames at line rate, the requests have already spent significant time in the queue waiting to be serviced. Note when `sift` drop ratio is at its apex, `encoding` receives only ≈ 60FPS. Our results highlight the possibility of a better holistic understanding of virtualized AR application performance by correlating QoS with hardware-level metrics.