2024 IEEE International Conference on Metaverse Computing, Networking, and Applications (MetaCom) | 979-8-3315-1599-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/MetaCom62920.2024.00036

Trading Virtual Objects Quality for AI Performance in Mobile Augmented Reality Apps

Niloofar Didar
Wayne State University (Detroit, Michigan, USA)

Marco Brocanelli
The Ohio State University (Columbus, Ohio, USA)

***Abstract*—Typical Mobile Augmented Reality (MAR) apps include several compute-intensive tasks for rendering virtual objects (AR tasks) and analyzing the real environment through AI model inference (AI tasks). Unfortunately, most of the existing research work overlooks the computational concurrency of AR and AI tasks. In fact, increasing the total triangle count of virtual objects may improve virtual object quality but reduce AI inference performance (e.g., latency). In this paper, we design MIR, a framework for MAR apps that dynamically regulates the on-screen triangle count to trade off between the performance of AR and AI tasks, leveraging locally-trained linear performance models and an approximation algorithm. We have implemented MIR on Android and tested it on real smartphones and users against several baselines to prove it helps improve performance with minimal resource usage overhead.**

I. Introduction

MAR apps execute *AR tasks* to process the geometry of virtual objects (i.e., meshes of polygons such as triangles) for accurate rendering in the scene. In addition, they must execute at least some basic *AI tasks* for seamless integration of virtual and real objects, e.g., frame detection, tracking, pose estimation, and light estimation [1]. Some apps may include also extra AI tasks for specific features, such as *natural language processing* to translate text or *real object classification* to make a virtual dog jump on a real table.

Among various metrics used to monitor MAR app performance, *virtual object quality* and *AI inference throughput* (i.e., inferences per second and inversely proportional to inference latency) have recently attracted researchers' attention. For example, previous work [2], [3] have shown that the virtual objects quality and mobile device energy consumption mainly depend on the current user-object distance and object triangle count. Other studies focusing mainly on AI task performance propose to offload AI tasks to edge servers (e.g., [4]). However, offloading sensitive camera images may lead to privacy concerns and depends on whether the remote execution is beneficial in terms of energy and/or latency. Thus, AI tasks of MAR apps may still need to run on the mobile device concurrently with AR tasks.

In the attempt to address resource concurrency between AI and AR tasks, Yi and Lee [5] have proposed a scheduling method for a finer-grained GPU access control between AI and AR tasks, but they assume virtual objects are rendered with maximum triangle count, i.e., highest quality. In Section III we demonstrate that resource concurrency leads to *performance unbalance*, e.g., high virtual object quality but poor AI performance or vice versa. We also show that the performance balance[1] can be controlled by manipulating the total triangle count of virtual objects. *To our best knowledge, no existing work studies how to manage the concurrent mobile resources usage of AI and AR tasks to trade off their performance*.

In this paper, we design MIR, a framework for MAR apps that dynamically regulates the on-screen total triangle count to balance AR and AI task performance. MIR automatically trains linear models of performance balance based on user preferences for AI and AR task performance. This approach ensures adaptability to diverse device models, apps, AI tasks, and virtual objects. Then, it periodically analyzes the current AR/AI task performance and finds the total triangle count needed to maintain balance. Finally, it leverages an approximation algorithm to distribute this total triangle count across the virtual objects, thereby enhancing their quality.

Specifically, this paper has three main contributions:

- This is the first study to prove that MAR apps are affected by performance unbalance because they render virtual objects at their highest quality, reducing AI performance.
- We design MIR, which leverages runtime modeling and an approximation algorithm to adapt the triangle count for performance balance despite environmental changes.
- We have implemented and tested an Android prototype of MIR on real smartphones and users, proving its ability to maintain balance with minimal resource usage overhead.

The paper structure is as follows. Section II discusses the related work. Section III provides background and motivations. Section IV describes the MIR design. Section V presents experimental results, and Section VI concludes the paper.

II. Related Work

Improving rendering task energy efficiency and user experience has been widely studied in the last two decades (e.g., [6], [7]), but these studies focus on knobs such as screen brightness or dynamic resolution scaling. These approaches are orthogonal and complementary to manipulating virtual object quality in MAR apps. Some studies focus on virtual reality and mobile gaming apps (e.g., [8]). Most of these solutions pre-render the near-future field-of-view images in the edge/cloud to improve performance and lower GPU load.

This research was supported by the US National Science Foundation under Grant CNS-2142406.

[1]In the rest of the paper we use the keyword *balanced* to paraphrase *having a good trade off* between AR and AI task performance.

However, the camera frame in MAR apps constantly changes with user movement in the real-world environment, making pre-computation in edge servers impractical.

Some other studies [3], [2], [9] focus only on AR tasks and trade off virtual object quality for lower energy usage by dynamically manipulating virtual object triangle count. Specifically, the approach proposed in [9] focuses on trading off between quality of virtual objects and rendering latency by changing the virtual object triangle count while LPGL [2] focuses on headset devices, linking the user's focal angle with the choice of virtual object quality. Both these solutions do not take into consideration an actual estimation of the virtual object quality as perceived by the user. eAR [3] takes a step further by leveraging image quality assessment to estimate the virtual object quality score for energy optimization (see more details in Section III-A). While we consider a similar approach to estimate virtual object quality as eAR [3], none of them consider the potential interference of virtual object rendering tasks with AI task performance in MAR apps.

Several solutions [10], [4], [11] optimize accuracy and minimize energy consumption for AI tasks such as object/surface detection offloaded to edge servers by considering a desired maximum inference latency or throughput. While we also consider achieving a good inference throughput (or latency), these approaches do not consider virtual object triangle count as an additional knob to control AI performance. In addition, offloading camera images may raise privacy concerns for the user and may not always be feasible depending on environmental conditions (e.g., data transmission latency). The closest work addressing the resource concurrency of AR and AI task is Heimdall [5], which breaks down tasks into smaller computing units for finer-grained mobile GPU scheduling, aiming to improve system performance in terms of app framerate and AI latency. However, it assumes virtual objects are always rendered at their highest quality and does not address the performance unbalance between AI and AR tasks. Thus, MIR is orthogonal and complimentary to Heimdall. To our best knowledge, *this is the first work studying how to balance AI tasks performance and virtual object quality.*

III. Background and Motivation

A. Background on MAR Performance Metrics

AR rendering tasks. While classic metrics such as screen framerate or resolution have been widely studied, virtual object quality in augmented environments has received limited attention. *Approaches that statically reduce the triangle count at various distances (e.g., Level of Detail or LOD), may be difficult to tune due to the heterogeneity of virtual object quality sensitivity to variations in triangle count and distance (see next section) and do not balance with AI task performance.* To solve these issues, previous work [3] has shown the feasibility of using Image Quality Assessment (IQA) to characterize virtual object quality based on features (e.g., shape, size), triangle count, and distance. They model the normalized degradation error of a specific virtual object i at time period k as a factor of the decimation ratio $R_{k,i}$ (i.e., selected triangle

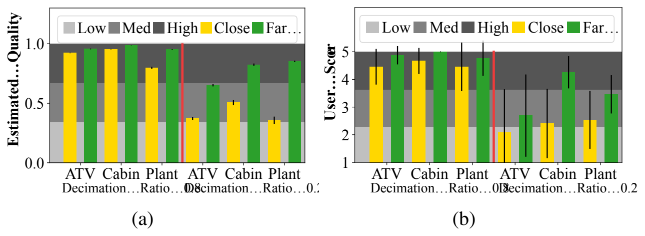


Fig. 1. Comparing virtual object quality: Equation 1 vs. real users

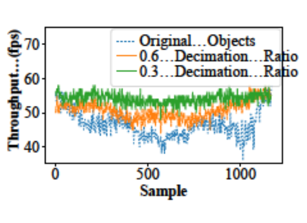
count over maximum count) and user-object distance $D_{k,i}$ as $D_{error_{k,i}} = (a_i R_{k,i}^2 + b_i R_{k,i} + c_i)/D_{k,i}^{d_i}$, where a_i, b_i, c_i, and d_i are parameters trained offline on a remote server [3]. We borrow this model to measure in period k the average quality Q_k across N_k virtual objects on screen as:

$$Q_k = \frac{1}{N_k} \sum_{i=1}^{N_k} \left(1 - D_{error_{k,i}}\right) \tag{1}$$

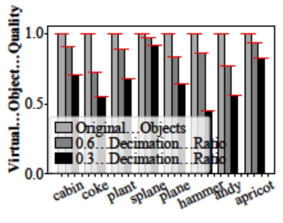
To validate this model's ability in characterizing virtual object quality as perceived by users, we have conducted an IRB-approved survey. To involve a large number of participants and speed up the study, we have created an online survey where we show high-resolution videos of the smartphone screen while using an MAR app we have developed (see details in Section V-A) with four virtual objects of varying complexity and size at close and far distances. Each video shows two versions of the same object side by side, one as the reference high-quality version (decimation ratio 1) and the other as the decimated version at ratios 0.8, 0.5, or 0.2, totaling 24 questions. Twenty-five participants have rated the decimated object quality compared to the reference on a 5-point *Likert scale*, where five means identical and one means much worse perception of the decimated object compared to the reference.

Figures 1a and 1b show the collected data of virtual object quality estimated using Equation 1 on the decimated object vs the average user scores, respectively. Due to space constraints, we show data for three objects and two decimation ratios. To provide a meaningful comparison, we categorize estimated and user scores into three quality classes, low, medium, and high (shaded gray areas in figures). Then, we assess the ability of the model to provide user-like virtual object quality classifications by calculating the accuracy and micro-average F1-score. Despite some variance in user scores, across all objects and distances the accuracy and F1-score are 88.8% and 83.3%, respectively. This proves a reasonable ability of Equation 1 to capture users perception of virtual object quality. Thus, we use it for runtime virtual object quality estimations.

AI tasks. Performance metrics for AI tasks are mainly dependent on accuracy, throughput, and latency. Multiple AI tasks may concurrently generate inference requests. However, here we do not allow a specific task to generate a new request before the previous one has been completed. Thus, the achieved per-task throughput is calculated as the inverse of its average latency (i.e., queuing plus inference time). Furthermore, given that AI inference accuracy relies on model design choices, which is beyond this paper's scope, we use TensorFlow Lite [12] pre-trained models, which are optimized



(a)



(b)

Fig. 2. (a) Decimating virtual objects can help improve AI performance but (b) may affect virtual object quality.

for accuracy and efficient mobile inference. Thus, we focus on AI task throughput (or latency equivalently) and leave as future work trading AI model accuracy for virtual object quality.

We also note that new top-tier smartphones can leverage multiple AI accelerators (e.g., GPU, NPU). Here, we mainly leverage GPU inference because it is the most common accelerator across users. In fact, the best-selling Android phone in 2023, the Samsung Galaxy A14 5G (MediaTek Dimensity 700), mainly relies on GPU for AI acceleration. Nevertheless, performance balance of AR/AI tasks remains necessary even with other accelerators. Similar to Android's NN-API [13], recent studies (e.g., [14]) minimize inference latency by distributing each AI operation of each task across CPU, GPU, and/or NPU according to the current system load. When the GPU load is low, an AI task execution can be divided across NPU and GPU to reduce latency. However, as GPU load increases due to larger triangle count, those AI operations on the GPU either experience a longer queuing latency or are moved to CPU/NPU, depending on compatibility, thereby deteriorating the latency of all AI tasks due to increased competition on fewer resources. Thus, AR tasks still affect AI tasks (up to three times with NN-API in our tests on Galaxy S22) even when more accelerators are used. We leave as future work to jointly manipulate virtual object quality and AI task allocation across accelerators for performance balance.

B. Motivation Study

To motivate our design, we use Samsung Galaxy S22, S10, Google Pixel 7, and Oneplus 5 on Android OS, but show only the S10 results due to trend similarity and space limitations.

Throughput vs Object Quality. In this experiment, we analyze how virtual object quality impacts AI task performance. We choose a representative pre-trained model for real object classification MobileNet V2 from TensorFlow Lite [12] (note, camera frames for AI inference do NOT include virtual objects). We conduct three experiment sets using the same sequence of on-screen virtual objects but at three different decimation ratios of 1.0 (highest quality), 0.6, and 0.3. We use the edge collapse algorithm to decimate objects to the desired ratio while preserving as much as possible the initial shape. Then, within each sampling period of two-seconds, we benchmark the performance trade off by collecting multiple inference throughput measurements as virtual objects are periodically added ($t = 0$ to $t = 600$) and removed ($t = 600$ to $t = 1200$) in the augmented environment.

As Figure 2a shows, highly decimated objects (0.3 decimation ratio) minimally impact throughput compared to the higher-quality objects. For example, 20 decimated objects at 0.3 ratio degrade throughput by only up to $4fps$, while using the highest-quality objects reduces throughput by up to $17fps$. Thus, (**Motivation 1**) *rendering virtual objects at a lower decimation ratio can help improve AI performance at the cost of a lower virtual object quality.*

Object Sensitivity. Figure 2b illustrates the variation in virtual object quality across the three case studies in this experiment (we show only a few to improve visual clarity but similar results were observed for the other objects). We estimate the quality of each virtual object as described in Section III-A. We can observe that, under the same decimation ratio, each object has a specific *sensitivity* to triangle count reduction due to different object characteristics (e.g., shape, size, max triangle count) and distance to the user. For example, when the *apricot* and the *hammer* are decimated to 0.3 ratio, the *apricot*'s quality is around 0.8 while *hammer*'s quality reduces to 0.4. This experiment highlights that (**Motivation 2**) *the sensitivity of the virtual object quality to decimation is heterogeneous across virtual objects and requires consideration during optimization.*

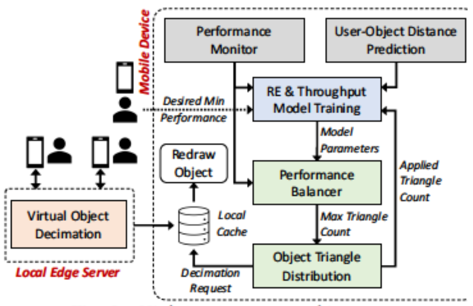


Fig. 3. High-level system architecture.

IV. MIR Design

A. System Architecture

As Figure 3 shows, MIR mainly includes *model training, performance balancer, and triangle distribution.* The first one trains system models used for control decisions based on runtime data from the *user-object distance prediction* and the *performance monitor*, ensuring runtime adaptability across various apps and devices. The user-object distance is estimated for the next period $k+1$ using the distance-prediction module from [3]. The *performance monitor* measures the AI tasks average throughput and estimates the average virtual object quality (Equation 1) over the last control period.

The *performance balancer* (Section IV-B) uses the trained model parameters to detect performance unbalance and re-balance the system by regulating the total triangle count (see **Motivation 1**) for the next control period. The calculated total triangle count is then distributed among the virtual objects on the screen by the *object triangle distribution* module (Section IV-C). It incorporates (*i*) a virtual-object priority assessment to capture how decimating each virtual object affects their quality (see **Motivation 2**) and (*ii*) an approximation algorithm to distribute the triangles across objects for improved quality. Finally, MIR accesses the app local cache to redraw the required decimated objects on the screen.

Cache Management. MIR employs both a local cache and an edge server to fetch the decimated objects. The server also runs virtual object decimation and quality profiling, which are generally too heavy to run on the mobile device [3]. While virtual objects are small in size (e.g., 1-15MB) and can be downloaded within a few tens of milliseconds at modern network speed, MIR mitigates network energy consumption and cache size by limiting the selectable variations in decimated objects for the *object triangle distribution* (Section IV-C).

B. Performance Balancer

Monitoring Performance Balance. The performance balancer finds the total triangle count on the screen that balances the system performance in the next decision period $k+1$ by defining the Relative Error (RE_k) as performance balance metric over the last period k. An $RE = 1$ indicates a balanced performance between average AI throughout and AR virtual object quality relative to their respective minimum desired performance. To account for the variance in user scores of virtual objects quality (see Figure 1b) and modeling errors, we use a tunable upper and lower bound of relative error to define a set of RE values where the system is considered to be balanced. Specifically, we calculate the relative error RE_k over the last control period k as follows:

$$RE_k = \frac{P_k^{AR}}{P_k^{AI}} = \frac{Q_k/Q^{min}}{H_k/H^{min}} \tag{2}$$

where P_k^{AR} and P_k^{AI} are the relative performance of AR and AI tasks in period k, respectively. P_k^{AR} is obtained from the estimated average quality of virtual objects in period k, i.e., Q_k in Equation 1, over the minimum desired quality Q^{min}. P_k^{AI} is the measured average AI inference throughput H_k in period k over the minimum desired throughput H^{min}.

We assume the user/app developer can easily decide the desired minimum quality $Q^{min} \in [0, 1]$ and throughput H^{min}, which normalize the two different metrics over a relative desired value and allow us to compare them with each other for performance balance evaluation. If the inference results should be displayed on screen within a certain delay, the developer can simply specify the desired latency to calculate H^{min}. In alternative, H^{min} could be chosen to be equal to $U^{pref} H^{max}$, where H^{max} is the average throughput across the AI models measured without virtual objects on the screen and $U^{pref} \in [0, 1]$ is the desired minimum performance factor. This approach is in line with previous research [10], [11].

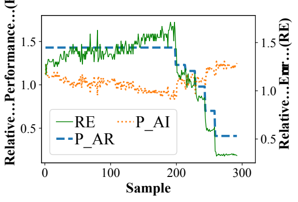


Fig. 4. Relative performance vs RE.

Figure 4 shows real-time measurements of RE, P^{AR}, and P^{AI} in our experiment (same setup as in Section III-B, with both Q^{min} and U^{perf} set to 0.7). From $t = 0$ to 200, 20 virtual objects are gradually placed on the screen, with a two-second sampling period for AI throughput data averaging. Then, an *auto decimate* function reduces object's decimation ratio by a factor of 0.2 every 20 seconds from $t = 200$ to $t = 250$. We can observe that before $t = 200$, P^{AI} decreases due to increased total triangle count, while P^{AR} remains constant with objects at the highest quality, leading to a rise in RE from 1.2 to 1.7. After $t = 200$, as objects are decimated, P^{AR} decreases due to lower average object quality, leading to an increase in P^{AI} and a significant reduction in RE. An RE value close to one (indicating balance) could be achieved for a specific triangle count on the screen at time period $240s$.

Triangle Count Control. MIR uses a *proactive* control to decide the triangle count for performance balance in next period $k+1$. This is to ensure that any required decimated objects are locally available when needed, aligning with predicted user-object distances as the user moves in the augmented environment. In order to implement this approach, we need to model what is the effect of the predicted distance and chosen total triangle count on the AI task throughput and relative error. To simplify control design, we choose to use two linear models that can be easily trained at runtime with little overhead. For the relative error model, we know from Equation 2 and Figure 4 that it depends on the specific AI models' load (i.e., encoded into the average throughput H), on the total triangle count, and user-object distance (due to OpenGL culling [15]). Thus, we model the relative error estimated for next period $\hat{RE}_{k+1}$ as:

$$\hat{RE}_{k+1} = \alpha_T T_{k+1}^{tot} + \alpha_D \hat{D}_{k+1} + \alpha_H \hat{H}_{k+1} + \zeta \tag{3}$$

where T_{k+1}^{tot} is the total triangle count in period $k+1$, $\hat{D}_{k+1}$ is the estimated average distance between user and all objects on the screen, $\alpha_T, \alpha_D, \alpha_H, \zeta$ are parameters trained at runtime. However, the average throughput $\hat{H}_{k+1}$ of the AI tasks in the taskset also depends on the triangle count (see Figure 2a) and predicted distance (OpenGL culling), thus we model it as:

$$\hat{H}_{k+1} = \rho_T T_{k+1}^{tot} + \rho_D \hat{D}_{k+1} + \delta \tag{4}$$

where ρ_T, ρ_D and δ are parameters dynamically estimated for each device. By keeping these two models separate, we find it easier to achieve a lower modeling error by pinpointing the specific parameters that need retrain at runtime. By substituting Equation 4 in Equation 3 and considering a desired relative error equal to one in the next period (i.e., $RE_{k+1} = 1$), we extrapolate the estimated total triangle count necessary to have performance balance as a function of predicted user-object distance and model parameters:

$$T_{k+1}^{tot} = \frac{1 - (\alpha_D + \alpha_H \rho_D)\hat{D}_{k+1} - (\zeta + \alpha_H \delta)}{\alpha_T + \alpha_H \rho_T} \tag{5}$$

If the evaluated RE_k is outside the desired region for a few consecutive times (e.g., five to eliminate false positives due to noise) and the modeling error of Equations 3 and 4 is sufficiently small, the performance balancer calculates the triangle count using Equation 5. However, if the modeling error is higher than a selected threshold (e.g., 10%), MIR locally re-trains the model parameters through linear regression before using Equation 5, which allows to achieve an average 5% error in our experiments. Finally, we guarantee solution existance for the triangle distribution by bounding T_{k+1}^{tot} between the maximum and minimum triangle count of all virtual objects on screen using the highest and lowest decimation ratio available for each object, respectively.

C. Object Triangle Distribution

Here, following Motivation 2 in Section III-B, we describe how MIR efficiently distributes the total triangle count from the balancer across the virtual objects on screen. For simplicity, in this section we omit k and refer to a generic period.

Triangle Distribution Problem Statement. Given n virtual objects characterized by triangle counts T_1, T_2,..., T_n, the maximum allowable total triangle T^{tot} set by the balancer, a coarse-grain decimation ratio R, e.g., $R = \{1, 0.8, ..., 0.1\}$, and the associated quality values $Q_{i,j} \in [0, 1]$, with 1 being highest quality, for each object $i \in [1, n]$ at each decimation ratio R_j, where $j \in [1, |R|]$. *The goal of the triangle distribution is to select a decimation ratio for each object i that maximizes the average virtual object quality while ensuring that the total triangle count is less than T^{tot}*. This multiple-choice Knapsack Problem is known to be NP-hard [16]. Thus, we design a polynomial-time approximation algorithm called Object Triangle Distribution (OTDA) to find a solution. Next, we introduce some concepts used in OTDA, and then discuss the algorithm and its approximation guarantees in Sections IV-C1 and IV-C2, respectively.

Object Prioritization. To prioritize triangle distribution across objects, we define *sensitivity* S_i and *triangle share* T_i^{share}. As shown in Section III-B, under the same decimation ratio each object i has a distinct sensitivity to triangle count reduction due to the object's characteristics and distance from the user. The triangle share is defined as the current contribution of object i to the total triangle count T^{tot}. Given that OTDA calculates the *priority of each virtual object to preserve higher quality*, higher priority should be given to a virtual object that is extremely sensitive to decimation and has a low triangle count compared to the total triangle count on screen. Thus, the virtual object priority P_i is calculated as:

$$P_i = \frac{S_i}{T_i^{share}} = \frac{[\bar{Q}_i - Q_{i,r}]/[T_i(1 - R_r)]}{T_i/T^{tot}} \tag{6}$$

where $\bar{Q}_i$ and $Q_{i,r}$ are the normalized quality of virtual object i with current triangle count T_i, and at a *reference* decimation ratio R_r, respectively, with r being a fixed index.

For example, if we calculate the priority of two similar objects with the same triangle count (i.e., same T_i^{share}) at two different distances, the farther object has a lower priority compared to the closer one since the larger distance makes quality reduction less perceivable. Thus, OTDA can reduce the triangle count of the farther object more than the closer one. For *triangle distribution* decisions we sort the n objects on screen in non-increasing order of P_i and assume that $P_{\alpha(1)}, P_{\alpha(2)}, \ldots, P_{\alpha(n)}$ is the order.

Triangle Distribution. MIR leverages a recursive function to find the object decimation ratio assignment given the heuristically-decided object's priority. We define a profit function $F_{\alpha(i),j}$ to be the cumulative maximum object quality for the first i highest-priority objects obtained by assigning the j^{th} decimation ratio in R to object $\alpha(i)$. This is calculated as the sum of $\max_s(F_{\alpha(i-1),s})$ and $Q_{\alpha(i),j}$, which are respectively, the maximum average quality of the first $i-1$ highest-priority objects and object $\alpha(i)$ quality with ratio R_j:

$$F_{\alpha(i),j} = \max_s(F_{\alpha(i-1),s}) + Q_{\alpha(i),j} \mid T^{rem}_{\alpha(i),j} \geq T^{min}_{\alpha(i)} \tag{7}$$

where $s, j \in [1 : |R|]$ are indexes of the array R, and j should be selected such that the remaining triangle count after assigning decimation ratio R_j to object $\alpha(i)$, i.e., $T^{rem}_{\alpha(i),j}$, is greater or equal to the minimum total triangle count needed for the $n-i$ remaining objects, i.e., $T^{min}_{\alpha(i)}$. This condition guarantees that there are enough triangles left to assign at least the lowest decimation ratio to the rest of un-assigned objects. Starting from T^{tot}, the remaining triangle count is calculated as:

$$T^{rem}_{\alpha(i),j} = T^{rem}_{\alpha(i-1),j} - (T^{max}_{\alpha(i)} \cdot R_j) \tag{8}$$

where $T^{rem}_{\alpha(i-1),j}$ is the remaining triangle count after assigning the first $i-1$ highest-priority objects, and $T^{max}_{\alpha(i)}$ is the maximum triangle count of object $\alpha(i)$ at the highest quality. The minimum total triangle count required to find a solution for the remaining $n-i$ objects using at least the minimum decimation ratio $R_{|R|}$ is calculated as:

$$T^{min}_{\alpha(i)} = \sum_{h=i+1}^{n} T_h^{max} \cdot R_{|R|} \tag{9}$$

1) OTDA Algorithm Design:

OTDA assigns triangles following the sorted priority order and the minimum triangle count for each object as defined by Equation 9. OTDA first selects the highest priority object $\alpha(1)$ and determines, for each decimation ratio j, its profit as $F_{\alpha(1),j} = Q_{\alpha(i),j}$ for those instances that leave enough triangles for the rest of the objects (see Equation 7). The profit remains zero if such condition is not verified. Then, OTDA selects the second virtual object in the order, i.e., $\alpha(2)$, and finds the highest ratio $j \in [1, |R|]$ for $\alpha(2)$, and maximum $s \in [1, |R|]$ for $\alpha(1)$, such that (a) the total profit Equation 7, i.e., $F_{\alpha(2),j} = F_{\alpha(1),s} + Q_{\alpha(i),j}$, is maximized and (b), the remaining triangles are enough for the rest of the objects. The algorithm then follows these steps for the rest of the objects, storing each time the combination of ratios for the selected object $\alpha(i)$ and the object with higher priority $\alpha(i-1)$ that lead to the highest combined profit.

When Equation 7 has been evaluated for each virtual object and each ratio in the order, OTDA finds the combination of decimations giving the maximum cumulative value for Equation 7. This combination is then used to redraw each object from the local cache, as described in Section IV-A. When all objects have been redrawn, OTDA returns the actual total triangle count applied to the model training module of MIR for local model training (see end of Section IV-B).

Complexity of OTDA. Objects are first sorted based on the priority factor with time complexity $O(n \log(n))$. Then, for each object, the algorithm executes $|R|^2$ times to find the sub-optimal decimation ratio. However, the number of $|R|$ elements is fixed, thus the algorithm complexity is $O(n \log(n))$.

Overhead. The CPU carries the majority of MIR's execution overhead. We experimentally find that setting the control period to $2s$ and the size of coarse grain decimation ratio list $|R|$ to 6 leads to a sufficiently short average execution time of 20.2ms and good performance. Additionally, the GPU overhead due to extra draw calls for decimated objects is largely outweighed by the long-term workload reduction (12% on average) due to the lower triangle count, which also leads to a substantial reduction (44% on average) in main memory usage.

2) Approximation Guarantees of OTDA: Theorem 1 defines the approximation ratio of OTDA, which finds a solution to the triangle distribution problem defined in Section IV-C. An algorithm has an *approximation ratio* η if, for all problem instances, it finds a solution whose value is within η from the optimal value, with $\eta < 1$ for maximization problems.

Theorem 1. *Considering $n > 2$ virtual objects, a reference decimation ratio $R_r = l$, and a quality difference $0 < \epsilon_1 < 1$ between the two highest-priority objects, the approximation ratio of OTDA with respect to total profit is $\eta = \frac{\pi_1}{\frac{\pi_2}{n}+1}$, where $\pi_1 = Q_{\alpha(1),r} + \epsilon_1$ and $\pi_2 = Q_{\alpha(1),r}$.*

Proof. We find the approximation ratio by analyzing the profit difference for the instance in which OTDA gives the worst-case solution against the optimal solution. The instance consists of a set O of $n > 2$ objects sorted based on priority (Equation 6) through permutation $\alpha(i)$, a quality matrix Q, a maximum triangle count $T^{max}_{\alpha(i)}$, and a set of coarse-grain decimation ratio $R = \{1, l\}$. Specifically, the instance has the following assumptions:

(i) Object $\alpha(1)$, i.e., $O_{\alpha(1)}$, has the largest max triangle count among all objects, i.e., $\forall i \in [2,n]$, $T^{max}_{\alpha(i)} < T^{max}_{\alpha(1)}$. In addition, the sum of total triangle count of all objects except $O_{\alpha(1)}$ is equal to $T^{max}_{\alpha(1)}$, i.e., $\sum_{i=2}^{n} T^{max}_{\alpha(i)} = T^{max}_{\alpha(1)}$.

(ii) $O_{\alpha(1)}$ at ratio $R_r = l$ has the minimum quality, i.e., $\forall i \in [2,n], Q_{\alpha(1),r} < Q_{\alpha(i),r}$. The other objects increase in quality by a factor ϵ_i, i.e., $\forall i \in [2,n], Q_{\alpha(i),r} = Q_{\alpha(1),r} + \epsilon_{i-1}$, such that:
   - (a) $0 < \epsilon_1 < \epsilon_2 < ... < \epsilon_{n-1} < 1$;
   - (b) $\epsilon_{n-1} < \frac{(1-Q_{\alpha(1),r})(n-2)}{n-1}$.
   - (c) The current objects' quality before executing OTDA is 1, i.e., $\bar{Q}_i$=1, the reference ratio is $R_r = l$, and $\forall i \in [2,n], Q_{\alpha(1),r} > 1 - \epsilon_{i-1} / \left[1 - \left(T^{max}_{\alpha(i)}/T^{max}_{\alpha(1)}\right)^2\right]$.

(iii) $O_{\alpha(1)}$ has the highest priority among all objects, i.e., $\forall i \in [2,n], P_{\alpha(i)} < P_{\alpha(i-1)}$.

(iv) The target triangle count T^{tot} is equal to $T^{max}_{\alpha(1)} + T^{max}_{\alpha(1)} l$, which is enough to either assign the highest decimation ratio 1 to the highest priority object $O_{\alpha(1)}$ and the lowest decimation ratio l to other objects, or to assign ratio 1 to all objects except $O_{\alpha(1)}$.

Under the above assumptions OTDA assigns the highest ratio to object $O_{\alpha(1)}$, which has the highest triangle share and lowest profit, leaving $T^{max}_{\alpha(1)} l$ remaining triangles to be distributed across the $n-1$ objects. Because of assumptions (i) and (iv), these objects are necessarily assigned to ratio l. It is relatively easy to show that such solution is feasible due to assumption (ii.c), so we omit it for space reasons. Thus, given assumption (ii) on object qualities relation, knowing that $\sum_{i=2}^{n} \epsilon_{i-1} > (n-1)\epsilon_1$, and that $1 > \epsilon_1 + Q_{\alpha(1),r}$, OTDA obtains a profit F as follows:

$$F = 1 + \sum_{i=2}^{n} (Q_{\alpha(1),r} + \epsilon_{i-1}) > nQ_{\alpha(1),r} + n\epsilon_1 \quad (10)$$

On the other hand, assumption (ii.b) guarantees that an optimal triangle allocation with profit $F^* > F$ would select ratio l for object $O_{\alpha(1)}$ and ratio 1 for all the other $n-1$ objects. The only way to further improve F^* would be to select ratio 1 also for object $O_{\alpha(1)}$. However, this is not possible because by assumptions (i) and (iv) there are not enough triangles left. Thus, the optimal allocation leads to the following profit:

$$F^* = F_{\alpha(1),r} + \sum_{i=2}^{n} F_{\alpha(i),1} = Q_{\alpha(1),r} + (n-1)(1) \quad (11)$$

We can now write the ratio F/F^* as:

$$\frac{F}{F^*} > \frac{nQ_{\alpha(1),r} + n\epsilon_1}{Q_{\alpha(1),r} + (n-1)} > \frac{Q_{\alpha(1),r} + \epsilon_1}{\frac{Q_{\alpha(1),r}}{n} + 1} \quad (12)$$

Let $\pi_1 = Q_{\alpha(1),r} + \epsilon_1$ and $\pi_2 = Q_{\alpha(1),r}$, then $F > \frac{\pi_1}{\frac{\pi_2}{n}+1} F^*$ and the approximation ratio of OTDA is $\eta = \frac{\pi_1}{\frac{\pi_2}{n}+1}$. □

V. Experimental Results

A. Experimental Setup

We tested MIR on several devices including Samsung Galaxy S22, S10, Oneplus 5, and Pixel 7. The trend results across these devices are similar, thus due to limited space we only show those of the S10 (middle performance device). Due to the lack of real open-source MAR apps with large virtual object datasets, related studies develop example apps with either a few (e.g., [2] uses only Stanford bunnies) or simplistic (e.g., [5] uses bounding boxes, emojis, body joints) virtual objects, which are not enough for our target scenarios. Thus, we develop an education/entertainment Android app that includes several virtual objects and AI features (see results for details).

To the best of our knowledge, there are no state-of-the-art works to balance AI throughput and virtual object quality[2]. Hence, we define two baselines for comparison as follows. **Quality Oriented (QO)** represents the current practice of Android where the virtual objects are always maintained at the highest quality level, which may lead to poor AI performance. **Throughput Oriented (TO)** has a similar target as [10], [11] since it maintains the AI throughput above a given threshold. However, they leverage offloading and do not consider AR tasks. To have a baseline that leverages triangle count for AI performance control, TO progressively reduces virtual objects quality until the throughput is above the threshold. Thus, it upholds AI performance without ensuring performance balance.

B. Benchmarking MIR Performance

We benchmark the performance of MIR for various combinations of virtual objects and AI models. Here, we show the results for one of the tested scenarios where we vary the number of virtual objects on screen from 0 to 21, each one of different size, location, and a triangle count between 2,324 and 146,803. We periodically add one virtual object at a time

[2]Two baselines could be [5] or [2], but the former does not provide source code while the latter focuses on energy and is only applicable to headsets.

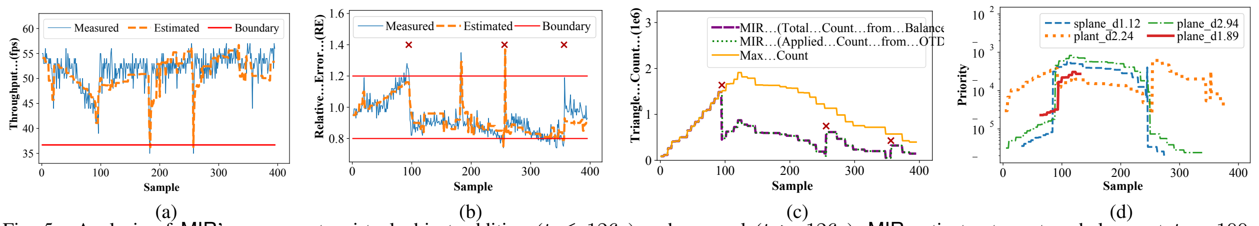


Fig. 5. Analysis of MIR's response to virtual object addition ($t \leq 126s$) and removal ($t > 126s$). MIR activates to restore balance at $t = 100$ (AR over-performing), $t = 270$ (AI over-performing), and $t = 370$ (AI over-performing).

reaching up to 2 million triangles. When all objects are placed, we then remove them one by one. Among the ten pre-trained AI tasks from the TensorFlow Lite repository [12] included in our app (e.g., for image segmentation, object detection, image classification, gesture detection), we also run an instance of mobilenet-v1 [17] for image classification. We set the desired minimum throughput to 65% (i.e., 37fps) of the maximum one measured without virtual objects on screen and the minimum average virtual object quality to 70% of the highest quality. Using a maximum modeling error threshold of 10% (see end of Section IV-B), we measure an average modeling error of 4.3% for throughput and 4.5% for relative error, which we consider satisfactory. The upper and lower bounds of desired relative error are 1.2 and 0.8, respectively.

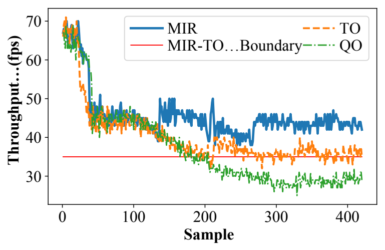
Performance Balance Analysis. Figures 5a to 5d show the measured and estimated AI model throughput, relative error, total triangle count on screen compared to the maximum triangle count with objects at maximum quality, and the virtual object priorities, respectively, as virtual objects are periodically added from $t = 0$ to $t = 126$ and removed after $t = 126$. There are three points in time (red crosses in Figure 5b) when the measured relative error stays out of boundaries, thus triggering the execution of the performance balancer and OTDA. For example, between time 0 and 100, the average quality of objects remains at its highest level, but the throughput drops significantly as more virtual objects are added. Around $t = 100$, the relative error exceeds the upper bound and the system is deemed unbalanced. Thus, the performance balancer decreases the total triangle count from 1.48 millions to about 453,000. The final applied triangle count by OTDA is around 450,000, with only a negligible left-over triangles caused by the limited set of decimation ratios considered. After the reduction in total triangle count at $t = 100$, the throughput raises from 38fps to 53fps while the average quality decreases from 1 to 0.84, which are both above the desired values. Thus, MIR brings the system back within the desired balanced region of relative error.

On the other hand, as virtual objects are removed starting at $t = 126$, the average throughput increases due to the lower triangle count, leading again to unbalance at $t = 270$, with the relative error dropping below the lower bound. Thus, MIR activates to increase the total triangle count and, subsequently, re-balance the relative error. A similar situation occurs at time $t = 370$. These results show that MIR maintains performance balance despite changes in the augmented environment by dynamically adjusting the objects' triangle count.

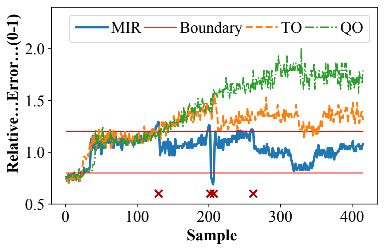
Triangle Distribution Analysis. Figure 5d shows the priority comparison among four virtual objects in the experiment (we omit the other 17 for clarity). Some objects' lines such as *plane_d1.89* (i.e., a *plane* virtual object at distance 1.89 meters from the user) might be shorter in the figures compared to others due to object addition and removal over time. The order of the object's priority follows that of the sensitivity and triangle share. For instance, *plant* with the lowest initial triangle count is more sensitive than other objects. Additionally, *splane_d1.12*, *plane_d1.89*, and *plane_d2.94* have the same triangle share but *splane* is smaller in size. Hence, despite being the closest, *splane*'s sensitivity changes less than the larger planes, even with the same reference decimation ratio (e.g., 50%). Thus, before $t = 100$, the *splane_d1.12* has the lowest priority due to its lowest sensitivity, *plane_d1.89* has higher priority than *plane_d2.94* because of its closer distance to the user, and *plant_2.24* holds the highest priority due to its higher sensitivity at a lower triangle count. Following this priority order at $t = 100$, MIR reduces the quality of low-priority objects *plane_d2.94* and *splane_d1.12*, which naturally increases their sensitivity and priority after $t = 100$. Similar considerations can be done at the other MIR activation points. These results show how OTDA efficiently distributes triangle count among objects based on their varying priority factor.

C. Performance Comparison

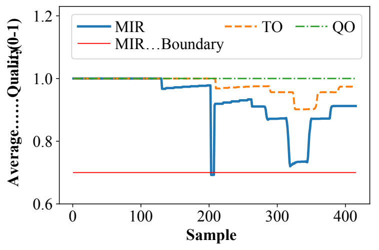
We have tested MIR with many different scenarios of AI models, virtual objects, and desired performance. Since we observe similar trend results across scenarios, for space reasons here we show the results for one of the scenarios where the user moves in the environment comparing MIR with TO and QO. The user adds 23 virtual objects from sampling time $t = 0$ to $t = 265$. Then, at $t = 265$ the user starts moving closer and at $t = 330$ starts moving away from the objects. Figures 6a to 6c compare the throughput, relative error, and virtual object quality of the three schemes, respectively. We set H^{min} for MIR and TO to be 35fps, which is shown with a bold horizontal line in the figure. We can observe that by t=265, the throughput of TO and QO decreases up to 32% and 43% compared to MIR. TO decimates objects uniformly to guarantee a throughput above the reference bound. Although it gives a better throughput compared to QO, which maintains the highest quality, it is not as efficient as MIR because finding the best throughput bound without uncontrollably deteriorating virtual-object quality remains challenging. For example, at time $t = 210$, TO reduces objects' decimation ratio to 0.8 to maintain throughput performance, which only slightly reduces



(a)



(b)



(c)

Fig. 6. Comparison of MIR with baselines in terms of performance (a-c). Virtual objects were added periodically until $t = 256$. The user moves closer to the objects from $t = 256$ to $t = 330$, and then moves farther.

the average quality. We also tested TO using H^{min} above and below 35fps, which uncontrollably lead to much lower and higher virtual object quality, respectively. Instead, MIR activates four times at $t = 130, 203, 208,$ and 260 and can find a better trade-off between throughput and virtual object quality, using the desired performance as guideline. In fact, it reduces the average quality without exceeding the $Q^{min} = 0.7$ desired bound, which results in higher AI performance compared to TO. As Figure 6b shows, MIR maintains balance 92.8% of the time while TO and QO only maintain balance for 27.5% and 26.3% of the time, respectively. These results show the usefulness of maintaining performance balance through MIR.

D. User Study

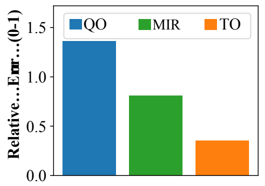


Fig. 7. Real Users Test.

We conduct a study to assess the system balance based on user scores of perceived object quality, by comparing the relative error of MIR with that of QO and TO. To speed up the study, we recorded the smartphone screen while a user moves around 14 virtual objects. The app also runs image segmentation inferences (devconv [18]) with $Q^{min} = 0.7$ and $H^{max} = 41fps$. For TO, we set as desired throughput the maximum achievable without virtual objects (i.e., 41fps) to see the effect of enforcing highest AI performance on virtual object quality. For MIR we set $U^{perf} = 0.7$ to allow for a better trade-off exploration. We then asked 25 anonymous participants to assess the virtual object quality with MIR and *TO* compared to *QO* (high-quality) on a 5-point *Likert scale*, with 1 to 5 indicating much worse to identical quality. Figure 7 presents the average relative error for each experiment calculated using the provided user scores. Notably, the average user score of MIR is 3.71 with an average throughput of $34.56fps$ and an average relative error of 0.81, falling within the defined boundary for system balance (0.8-1.2). However, *QO* and *TO* lead to a high (5 by definition) and low (1.81) user score of virtual object quality, respectively, resulting in an average relative errors of 1.36 and 0.35, respectively, which highlights high performance unbalance.

This experiment shows with real users the ability of MIR to trade off between AI throughput and virtual object quality.

VI. Conclusions

In this paper, we demonstrated that high quality virtual objects impact the AI inference throughput in MAR apps and cause performance unbalance. Thus, we proposed MIR, a framework for MAR apps to balance the performance of AI and AR tasks. MIR leverages accurate runtime linear model training of performance to manipulate the total triangle count and employs an approximation algorithm that assigns object triangle count across virtual objects for an enhanced object quality. We evaluated MIR with real users and against several baselines that either provide high virtual object quality or good throughput, proving its ability to dynamically keep a good trade-off between the two metrics.

References

[1] Android. (2023) ARCore Fundamental concepts. Android. [Online]. Available: https://developers.google.com/ar/develop/fundamentals

[2] J. Choi, H. Park, J. Paek, R. K. Balan, and J. Ko, "Lpgl: Low-power graphics library for mobile ar headsets," in *MobiSys*, 2019.

[3] N. Didar and M. Brocanelli, "ear: An edge-assisted and energy-efficient mobile augmented reality framework," *IEEE Transactions on Mobile Computing*, vol. 22, no. 7, pp. 3898–3909, 2023.

[4] Q. Liu, S. Huang, J. Opadere, and T. Han, "An edge network orchestrator for mobile augmented reality," in *INFOCOM*, 2018.

[5] J. Yi and Y. Lee, "Heimdall: Mobile gpu coordination platform for augmented reality applications," in *MobiCom*, 2020.

[6] V. Setlur, Y. Xu, X. Chen, and B. Gooch, "Retargeting vector animation for small displays," in *MUM*, 2005.

[7] S. He, Y. Liu, and H. Zhou, "Optimizing smartphone power consumption through dynamic resolution scaling," in *MobiCom*, 2015.

[8] S. Shi, V. Gupta, M. Hwang, and R. Jana, "Mobile VR on edge cloud: A latency-driven design," in *MMSys*, 2019.

[9] D. Narayanan and M. Satyanarayanan, "Predictive resource management for wearable computing," in *MobiSys*, 2003.

[10] J. Ahn, J. Lee, D. Niyato, and H. S. Park, "Novel qos-guaranteed orchestration scheme for energy-efficient mobile augmented reality applications in multi-access edge computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 11, pp. 13 631–13 645, 2020.

[11] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.

[12] TensorFlow Lite, "Hosted models for image classification," https://web.archive.org/web/20210225170007/https://www.tensorflow.org/lite/guide/hosted_models, 2021.

[13] Google Android, "Neural networks api," https://developer.android.com/ndk/guides/neuralnetworks, 2022.

[14] J. S. Jeong, J. Lee, D. Kim, C. Jeon, C. Jeong, Y. Lee, and B.-G. Chun, "Band: Coordinated multi-dnn inference on heterogeneous mobile processors," in *MobiSys*, 2022.

[15] learnopengl. (2023, Jan) Face culling. learnopengl. [Online]. Available: https://learnopengl.com/Advanced-OpenGL/Face-culling

[16] H. Kellerer, U. Pferschy, and D. Pisinger, *Multiple Knapsack Problems*. Springer Berlin Heidelberg, 2004, pp. 285–316.

[17] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," arXiv, 2017.

[18] TensorFlow Lite, "Portrait-segmentation," https://github.com/anilsathyan7/Portrait-Segmentation, 2023.