Joint AI Task Allocation and Virtual Object Quality Manipulation for Improved MAR App Performance

Niloofar Didar
Department of Computer Science
Wayne State University
Detroit (MI), USA
niloofar_didar@wayne.edu

Marco Brocanelli
Department of Electrical and Computer Engineering
The Ohio State University
Columbus (OH), USA
brocanelli.1@osu.edu

2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS) | 979-8-3503-8605-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDCS60910.2024.00110

***Abstract*—The emergence of modern mobile System on Chips (SoCs), featuring robust neural network accelerators such as GPUs, DSPs, and NPUs, has made on-device inference a compelling alternative to cloud-assisted inference. Typical mobile augmented reality (MAR) applications enable users to interact with virtual objects, leveraging diverse Artificial Intelligence (AI) capabilities facilitated by a range of deep learning models. Several studies seek to improve the performance of MAR apps. However, they often overlook the computational concurrency between the AR tasks necessary to render virtual objects and the AI tasks, which can influence virtual object quality and AI inference response time. In this paper, we present HBO, a framework for MAR apps that trades off between AR and AI task performance. HBO leverages Bayesian optimization and heuristic algorithms to jointly manipulate the virtual objects' triangle count and AI task allocation for optimized MAR app performance. We have implemented HBO on Android and tested it on real smartphones and with real users. Our results show that HBO helps reduce the average AI task latency by up to 3.5x and increase the average virtual object quality by up to 38.7% compared to several state-of-the-art baselines.**

***Index Terms*—Augmented Reality, Artificial Intelligence, Bayesian Optimization**

I. Introduction

Modern mobile devices feature AI hardware accelerators such as GPUs, DSPs, and NPUs/TPUs to accelerate mathematical computations and facilitate on-device deep learning inferences. As Figure 1 shows, MAR apps often involve a variety of *AI tasks* (e.g., object detection, pose estimation) necessary to scan the real environment using any of the computing units available on the SoC, alongside *AR tasks*, which are necessary to render virtual objects (i.e., meshes of polygons such as triangles) into the augmented environment using CPU and GPU. The inevitable task concurrency to access the computing resources requires designing new strategies that trade off between the AI task latency and the virtual object quality.

Previous studies have shown that AI task latency is not only affected by the specific AI model architecture but also by the allocated AI accelerator [1]–[8]. Hence, assigning AI models to the appropriate computing resources can improve AI tasks' performance and mitigate contention with AR tasks, specially in scenarios with high triangle count from virtual objects. In fact, as we show in this paper, manipulating the virtual objects quality [9]–[11] can further enhance the response time of AI tasks. To the best of our knowledge, no existing solutions jointly leverage these two techniques at same time to find the optimal choice of AI task allocation and virtual object triangle count that maximizes virtual object quality while minimizing AI task latency. On the other hand, finding such a solution presents many challenges, including the difficulty to (*i*) create a mathematical model that can help map the *manipulated variables*, i.e., allocation choice of each AI task and triangle count of each virtual object, to the *controlled variables*, i.e., AI task latency and virtual object quality, (*ii*) explore a large search space, and (*iii*) adapt to different SoC, AI tasks, and objects.

To overcome all these challenges, we introduce a framework called HBO, which activates as an event-based system and leverages Bayesian optimization to find the virtual objects' triangle count and the AI task allocation that best trade off between virtual object quality and AI latency. Specifically, this paper has three main contributions:

- We investigate the challenges and opportunities of jointly manipulating AI task allocation and virtual objects triangle count to optimize the performance of both AI and AR tasks in MAR apps.
- We design HBO to trade off between AI and AR task performance and overcome the above challenges by leveraging Bayesian optimization and heuristic algorithms.
- We have developed and evaluated an Android prototype of HBO on modern smartphones and real users against state-of-the-art baselines to demonstrate its effectiveness in enhancing the performance of MAR apps.

The rest of this paper is organized as follows. Section II discusses the related work. Section III provides background and design motivations. Section IV describes HBO. Section V shows the experimental results. Section VI discusses limitations and future work, and Section VII concludes the paper.

II. Related Work

Previous studies have extensively explored methods to improve the response time or quality of experience in mobile apps. Several studies (e.g., [12]–[15]) offload compute-intensive tasks (e.g., machine learning) to the edge for better energy consumption, AI model accuracy, and response time.

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

2575-8411/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDCS60910.2024.00110

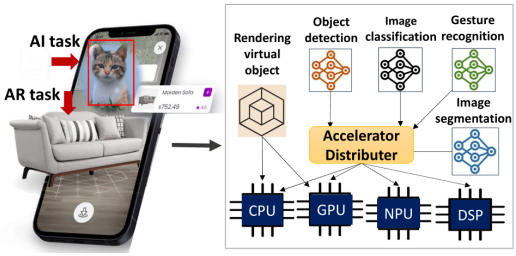


Fig. 1: AI and AR tasks on heterogeneous processors.

While offloading represents a valid solution in general, it may lead to privacy concerns in MAR apps due to the necessity to offload camera images. In addition, it may not always be useful due to variable network latency, specially for lighter AI tasks specifically designed to run efficiently on mobile devices [16]. Hence, in this paper, we assume all operations execute locally on the mobile devices.

For MAR apps, some studies [9]–[11] trade-off virtual object quality, energy, and/or performance by manipulating the virtual objects' triangle count. Specifically, Narayanan et al. [9] trade off between virtual objects triangle count and rendering latency. LPGL [10] focuses on headsets, adjusting object quality based on the user's focal angle. These approaches do not consider the actual virtual object quality. Thus, eAR [11] uses image quality assessment to estimate virtual object quality based on user-object distance and triangle count for energy optimization. However, none of them study the effect of virtual object quality on AI latency.

In order to reduce the on-device AI task latency, some prior works focus on model compression techniques (e.g., [17]) or on splitting their execution among the available hardware resources [2]–[8]. For example, BAND [8] coordinates the allocation of each AI task operation across the heterogeneous processors through subgraphs scheduling. However, these approaches assume virtual objects in MAR apps are rendered with their highest triangle count and do not trade off between virtual object quality and AI task latency. In addition, most of them rely on estimations of each operation's execution time on each processor, which is not easily predictable when AI allocation and triangle count are jointly manipulated.

In our work, rather than allocating each AI operation (fine-grain), we choose a coarser-grained solution to allocate each AI task either on the CPU or on one of the available *delegates* (e.g., GPU, NNAPI[1]) for two reasons. First, similar model slicing techniques are already embedded in the available NNAPI delegate of Android [18]. However, due to inter-processor communication delays and inefficiencies, the delegate/CPU allocation choice that maximizes the AI performance still highly depends on the specific AI model and SoC (see test results in [1]) as well as AI taskset and triangle count (see Section III-B). Second, finding the allocation for each one of the AI tasks' operations jointly to triangle count manipulation makes the problem too complex to solve rapidly and efficiently. Thus, the above studies are orthogonal and complementary to our solution since new slicing techniques could be simply added on the list of available allocation strategies in our solution. To the best of our knowledge, *this is the first work studying how to jointly manipulate triangle count and AI task allocation for MAR app performance optimization.*

[1]The GPU delegate allocates all operations on GPU while NNAPI splits them across all the available accelerators.

III. Background and Motivation

A. Background

Virtual Object Quality in MAR apps. Different from general performance metrics such as screen resolution or framerate, virtual object quality is an important metric unique to MAR apps. Thus, we focus on this metric and leave the inclusion of the others for future work. Some previous work [11] has shown the feasibility of using an image quality assessment method [19] (both tested with real users) to characterize virtual object quality based on specific object features (e.g., shape), triangle count, and distance to the user. Specifically, they model the normalized degradation error $D_{error_{t,i}}$ of a specific virtual object i at time period t as a factor of decimation ratio $R_{t,i}$ (i.e., selected triangle count over maximum count) and user-object distance $D_{t,i}$:

$$D_{error_{t,i}} = \frac{a_i R_{t,i}^2 + b_i R_{t,i} + c_i}{D_{t,i}^{d_i}} \tag{1}$$

where a_i, b_i, c_i, and d_i are parameters trained offline as detailed in [11]. We borrow this model to estimate in period t the average quality Q_t across L_t virtual objects on screen as:

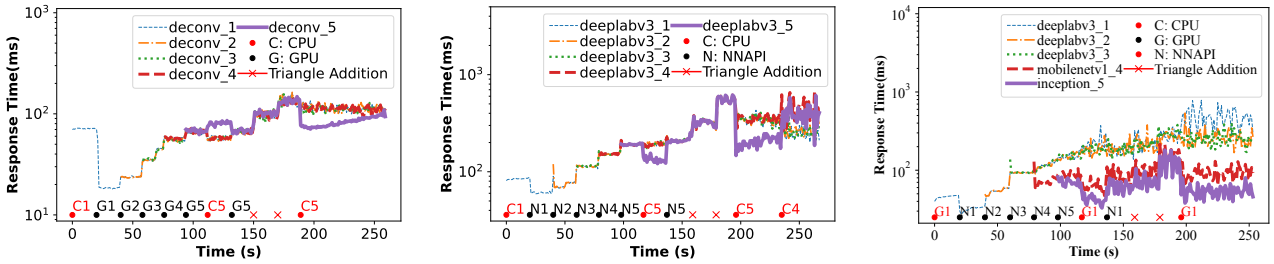
$$Q_t = \frac{1}{L_t} \sum_{i=1}^{L_t} \left(1 - D_{error_{t,i}}\right) \tag{2}$$

Leveraging this model, we capture real users perception of virtual object quality and show the results with a real user study in Section V-E. Nevertheless, other models could be used in its place for enhanced performance.

AI Task Performance. The performance of MAR apps for AI tasks is mainly dependent on inference accuracy and response time. Given that AI inference accuracy relies on model design choices beyond this paper's scope, we use TensorFlow Lite [16] pre-trained models optimized for accuracy and efficient on-device inference. Hence, we focus on AI task response time and leave as future work the possibility of trading AI model accuracy for virtual object quality.

B. Motivation Study

Building upon the findings of a previous study [1] regarding the heterogeneity of an AI model performance across different delegates and SoCs, we collected AI model response time data using the two available delegates GPU/NNAPI and the CPU on a Samsung Galaxy S22 and a Google Pixel 7. Table I shows the average response times in milliseconds. Note that some models are not compatible with some delegate (i.e., NA). For example, on the S22 Deeplabv3 takes 45ms on the GPU, 27ms on NNAPI, and 46ms on the CPU, suggesting it has a higher *affinity* with NNAPI. On the other hand, some other models (e.g., model-metadata and deconv-munet) show better affinity, i.e., lower response time, with the GPU delegate. While these *static* profiles are fairly stable when these tasks



(a) CPU vs GPU (b) CPU vs NNAPI (c) GPU vs NNAPI

Fig. 2: Taskset and virtual object triangle count can highly influence the performance of AI task latency for various allocations.

TABLE I: Baseline response time (ms) of TensorFlow-lite [16] models on Galaxy S22 and Pixel 7 (IS: Image Segmentation, OD: Object Detection, IC: Image Classification, GD: Gesture Detection)

AI Model	Task	Galaxy S22			Google Pixel 7		
		GPU	NNAPI	CPU	GPU	NNAPI	CPU
deconv-munet	IS	**18**	33	58	**17.9**	NA	65.9
deeplabv3	IS	45	**27**	46	136.6	NA	**110.1**
efficientdet-lite	OD	72	NA	**68**	109.8	NA	**97.3**
mobilenetDetv1	OD	38	**13**	38	56.5	**18.1**	48.9
efficient-litev0	IC	28	**10**	29	43.37	**18.3**	41.5
inception-v1-q	IC	28	**8**	36	60.8	**8.7**	63.2
mobilenet-v1	IC	26	**9.5**	28	37.1	**10.2**	40.5
model-metadata	GD	**12.7**	18	14	**24.6**	40.7	25.5

execute without contention (i.e., no other AI tasks and no virtual objects), as we will show in the following experiments, *the best delegate choice for each task changes when other AI/AR tasks execute concurrently.*

To prove this point, we create various tasksets and allocation scenarios using the Galaxy S22 (similar trends on the Pixel 7) and several instances of image segmentation (deconv and deeplabv3) and classification (mobilenetv1 and inception). Figure 2 shows the results. For space reason, we explain Figure 2b in detail but similar behaviors can be observed in to other two experiments of Figures 2a and 2c. Here we use up to five instances of deeplabv3 that, according to the static profiling in Table I, in isolation performs best on the NNAPI delegate and similarly worse on the CPU and GPU delegate. In the figures, the dots at the bottom indicate the time of a change in allocation and the specific allocation choice (C for CPU, G for GPU, and N for NNAPI). For example, in Figure 2b N1 means that the first instance of deeplabv3, i.e, deeplabv3_1, is allocated to the NNAPI delegate at the indicated time point. The first instance of the deeplabv3 task initially runs on the CPU (i.e., C1) but at $t = 25s$ is switched to the NNAPI delegate (i.e., N1), improving its performance.

From $t = 40s$ to $t = 95s$, we progressively add AI tasks to the same NNAPI delegate, which leads to a gradual increase in the response time for all tasks due to the higher resource competition. At $t = 120s$, we relocate the fifth instance of deeplabv3 to the CPU (results do not change choosing a different instance since they use the same model). Different from the response time comparison between C1 and N1, this relocation C5 now leads to (*i*) an improvement in the response time of deeplabv3_5 rather than an increase and (*ii*) unaffected performance for the other instances. This underscores the advantage of dynamic AI reallocation, particularly when the load becomes significant. Next, we move deeplabv3_5 back to NNAPI (N5 at time 140s in Figure 2b) and progressively add some virtual objects in the scene (the red cross signs) around $t = 150s$ and $t = 180s$. *This change leads to a significant increase in all AI tasks response times despite the usage of both the available accelerators through the NNAPI delegate, i.e., the GPU and the Neural Processing Unit (NPU).* At $t = 200s$ we then reallocate deeplabv3_5 back to the CPU (i.e., C5). Different from the same action at time 120s, this time we observe a significant *reduction* in response time not only for the reallocated task but also for the others. On the other hand, moving one more instance on the CPU (deeplabv3_4 at C4) leads to a further improvement in performance for the tasks remaining on the NNAPI but much worse performance for those on the CPU. Similar unpredictable behaviors can be observed in the other two figures where we experiment with deconv on CPU/GPU (Figure 2a) and a mix of AI tasks on GPU/NNAPI (Figure 2c).

These experiments allow us to point out that *while jointly manipulating the total triangle count and the AI task allocation can lead to substantial performance gains, it is challenging to profile the expected AI task performance in MAR apps because it highly depends on the specific taskset used and on the total triangle count from the virtual objects.*

IV. HBO Design

A. Overview

In order to overcome the challenges depicted in previous section, here we describe the design of our proposed solution HBO. Figure 3 shows its high-level architecture, which consists of four main component types: *dynamic optimization, control, object quality, and evaluation*. The *dynamic optimization* (displayed in gray) is event driven and periodically monitors the system performance to detect when it is worthwhile to execute a Bayesian Optimization (BO) algorithm. When triggered, BO operates over several iterations to find the best performance trade off using both the total

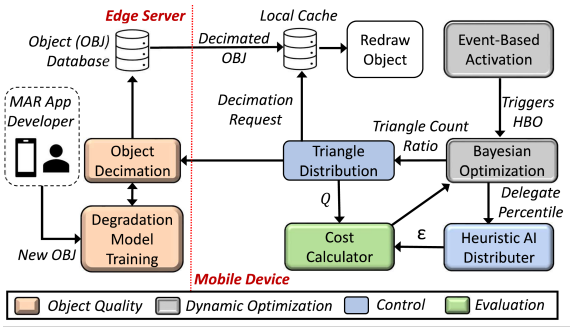


Fig. 3: HBO Architecture.

triangle count ratio, i.e., selected total triangles over the maximum triangle count across objects, and the proportion of AI tasks to allocate to each available resource. Leveraging a Bayesian approach allows us to optimize system performance efficiently and without need of profiling the expected system performance ahead of time. The chosen configuration in the current BO iteration is sent to the *control* components that help enforce it. Specifically, HBO decides how to divide the total triangle count from BO across all virtual objects (i.e., object decimation) for maximized average quality. Each decimated version can either be found in the local cache or downloaded from a server executing a virtual object decimation algorithm and virtual object parameter training [11] (see Equation 1). In addition, HBO examines the proportion of AI tasks that must be allocated to each resource from the chosen BO configuration and decides the specific task allocation. Then, HBO starts measuring the resulting performance in terms of average AI latency ϵ and average virtual object quality Q. These measurements are forwarded to the *evaluation* components to recalculate the value of the cost function used for the Bayesian optimization. The calculated cost is used by BO to decide the next configuration to explore for convergence to a solution over a limited number of iterations.

B. HBO Problem Statement

Consider an MAR app scenario with L virtual objects on screen accounting for T^{max} total triangle count, M AI tasks executing in background, and N allocatable resources. *The HBO problem is to find the best triangle count for each object and allocation for each AI task that maximize the average virtual object quality while minimizing the AI task latency.* This problem is challenging to solve due to the large search space, i.e., $N^M \cdot T^{max}$, and the consequent difficulty to create an offline system model, which would require to profile ahead of time the performance resulting from each possible solution in many different scenarios. To overcome this challenge, our approach divides this complex problem into two steps. The first step uses BO to find the best performance trade off using high-level optimization variables, which are then translated into actual task allocations and per-object triangle count in the second step.

In the rest of this section we will describe the first step optimization problem through BO, then the heuristics used for the second step, and finally the HBO activation policy.

C. HBO Optimization

Problem Formulation. When triggered (see Section IV-E), HBO runs an algorithm that maximizes at time period t the function B_t, which is calculated as the difference between the average quality of virtual objects Q_t and the average normalized AI task execution latency ϵ_t:

$$B_t(c_t, x_t) = Q_t - w \cdot \epsilon_t, \quad \forall t \in \mathcal{T} \tag{3}$$

The weight w controls the importance of AI task performance in comparison to the average quality of virtual objects. The average quality is estimated using Equation 2 while the average latency ϵ_t is measured at runtime as:

$$\epsilon_t = \frac{1}{M} \sum_{m=1}^{M} \frac{\tau^a_{m,t} - \tau^e_m}{\tau^e_m} \tag{4}$$

where $\tau^a_{m,t}$ is the latency of AI task m measured at time t while τ^e_m is the expected latency of AI task m measured on the *most suitable resource*, i.e., the resource where task m shows the lowest latency when running in isolation (no other AI tasks running and no virtual objects on screen), which can be determined offline directly on the user device for each AI task and allocatable resource. This offline profiling is a *one-time* operation, thus incurring little inconvenience to the user.

The optimization variables are considered to be continuous. x_t represents the total triangle count ratio, e.g., if $x_t = 0.3$ only 30% of the total triangles is used across objects. $c_t = [c_{t,1}, ..., c_{t,N}]$, where each $c_{t,i}$ is the proportion of AI tasks allocated to the i^{th} resource out of the N resources (e.g., CPU, GPU delegate, NNAPI delegate). For example, $c_{t,2} = 0.2$ denotes that 20% of the AI tasks should be allocated to the 2^{nd} resource in period t. Compared to using directly the per-task allocation, this variables choice allows to create a meaningful relationship between the effect of varying c_t by a certain amount on the function to optimize. For example, it is easier to *learn* that allocating more tasks to the GPU leads to an improvement/deterioration of B_t. To obtain the actual per-task allocations (and per-object triangle count), we then leverage a heuristic described in Section IV-D. Thus, HBO learns the factors influencing B_t based on the defined heuristic and efficiently finds a near-optimal solution over a few iterations.

Formulation Challenges. There are two major challenges with Equation 3. First, as mentioned in previous sections, it is difficult to find a mathematical model that can relate the performance Q_t and ϵ_t to the optimization variables, which makes the function to maximize unknown. Second, the function value can only be obtained after testing a candidate solution (c_t, x_t) for a whole time period t. To find a solution to this bandit continuous variable optimization problem we employ Bayesian Optimization (BO). In general, BO is a sample-efficient method to find an input point that minimizes an expensive black-box function $f : Z \subset R^d \rightarrow R$, specifically to find $z^* = \text{argmin} f(z), z \in Z$ through a limited number of evaluations $z_1, ..., z_T$ of function f guided by an acquisition function. Each input value z_i is represented as a vector of dimension d, and T denotes the number of evaluations. In our tests we have found that Expected Improvement (EI) [20] is a

well-suited acquisition function for our problem compared to other commonly-used acquisition functions such as *probability of improvement*, which is too conservative during exploration, and *lower confidence bound*, which requires tuning a dedicated exploration/exploitation parameter.

Bayesian Formulation of HBO. The BO's cost function value depends on the two optimization variables x_t and c_t. Equation 3 can be reformulated as a cost function, denoted as $\varphi(c_t, x_t) := -B_t(c_t, x_t)$ and aimed at finding the minimum cost φ over $t \in \mathcal{T}$ iterations:

$$min_{\{c_t, x_t\}} \varphi_t(c_t, x_t) \quad \forall t \in \mathcal{T} \tag{5}$$

Given the continuous joint variables (c_t, x_t) defined as z_t, BO seeks to minimize the black-box $\varphi(z_t)$ with $z_t := [c_t^\top, x_t^\top]^\top$ by sequentially acquiring function observations using a Gaussian process $\mathcal{GP}$ surrogate model [21]. For an unknown cost function φ, a $\mathcal{GP}$ defines the probability distribution of the possible values $\varphi(z)$ for each point z. These probability distributions conform to a Gaussian distribution and are thus defined by a mean function μ and a standard deviation σ. On the other hand, μ and σ may vary after testing a specific input z. Therefore, BO establishes a probability distribution over the function φ as follows:

$$p(\varphi(z_t)|\mathcal{D}_t) = \mathcal{N}(\mu_t(z_t), \sigma_t^2(z_t)) \tag{6}$$

where $\mathcal{D}_t := \{(z_\tau, \varphi_\tau)\}_{\tau=1}^t$ is the dataset BO creates and updates based on the evaluated solutions $z_1, ..., z_t$, and their corresponding observation of objective function $\varphi(z_\tau)$ with values $\varphi(z_1), ..., \varphi(z_t)$, respectively. Each BO iteration consists of (a) obtaining the posterior probability density function $p(\varphi(z_t)|\mathcal{D}_t)$ based on the chosen surrogate model and dataset $\mathcal{D}_t$, and (b) selecting z_{t+1} to evaluate at the beginning of slot $t+1$, whose cost is evaluated at the end of slot $t+1$. $\mu_t(z_t)$ and $\sigma_t^2(z_t)$ are the posterior function's mean and variance calculated based on a kernel function. In this paper, we use Matérn [22], which is a class of flexible and commonly used kernels in BO [23]–[25]. It uses a parameter v that controls the smoothness of the learning function. The smaller v is, the less smooth the sought function is assumed to be. Based on extensive testing we use $v = 5/2$. The resulting kernel function is thus calculated as:

$$k(z_t, z') = \sigma_\varphi^2 \left(1 + \frac{\sqrt{5}r}{l} + \frac{5r^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}r}{l}\right) \tag{7}$$

where $r = \sqrt{(z_t - z')^\top (z_t - z')}$ is the Euclidean distance between z_t and configuration z' in the current dataset $\mathcal{D}$, and l is the length scale parameter that controls the width of the kernel (we set to 1 in our experiments).

Constraints. Our system imposes three *known* constraints on the selection of c_t and x_t:

$$0 \leq c_{t,i} \leq 1 \quad \forall i \in [1, N], t \in \mathcal{T} \tag{8}$$

$$\sum_{i=1}^{N} c_{t,i} = 1 \quad \forall t \in \mathcal{T} \tag{9}$$

$$R^{min} \leq x_t \leq 1 \quad \forall t \in \mathcal{T} \tag{10}$$

Algorithm 1 Heuristic Bayesian Optimization (HBO)

Input: M: number of AI tasks; N: number of resources available; P: PriorityQueue of latencies for every AI task on each resource; L: number of virtual objects, $\mathcal{D}$: Bayesian optimization database; w: latency/quality weight.

```
c, x ← BO(D)
C ← []
for i ← 1 to N do
    C_i ← ⌊c_i · M⌋
r ← M − Σ_{i=1}^{|C|} C_i
if r > 0 then
    Sort resource usage c in non-increasing order. Let c_α(1), c_α(2), ..., c_α(N) be the order.
    for i ← 1 to N do
        C_α(i) ← C_α(i) + 1
        r ← r − 1
        if r ≤ 0 then
            break
k ← 0
while k ≠ M do
    (i*, j*) ← P.poll()
    if C_j* ≠ 0 then
        Allocate AI task i* on resource j*
        C_j* ← C_j* − 1
        k ← k + 1
        Remove N entries with same index of i* from P
    else
        Remove M entries with same index of j* from P
Execute TD(x, L) and redraw decimated virtual objects
ϵ, Q ← Measure average latency (Equation 4) and estimate the average quality (Equation 2) over the control period.
φ ← −(Q − w.ϵ)
D ← D ∪ {(c, x, φ)}
```

Constraint 8 restricts the resource usage to a value between 0 and 1, Constraint 9 ensures that the sum of resource usages is exactly 1 for consistency, while Constraint 10 maintains the triangle count ratio between a minimum R^{min} and the maximum 1 (i.e., render objects at highest quality).

D. HBO Control

Here, we describe the HBO algorithm, which leverages the above Bayesian optimization formulation guided by heuristics to find a solution to the HBO problem in polynomial time.

HBO Algorithm Design. The HBO algorithm is triggered by the event-based activation policy (see Section IV-E) and executes for a limited number of iterations to determine the best configuration to use in the current scenario to improve overall performance. In each iteration, the pseudo-code shown in Algorithm 1 is executed. It takes as inputs the number of AI tasks M and the number of available resources N. In addition, it takes the priority queue P of latencies for every AI task on each resource (profiled one time in isolation with no virtual objects or other AI tasks running) sorted in non-decreasing order. Let (i^*, j^*) be the head element of the queue referring

to AI task i^* (e.g., task 1, 2, 3,...) on resource j^* (e.g., 1 for CPU, 2 for GPU, 3 for NNAPI, ...). Finally, it takes the number of virtual objects L, and the database of previous BO iterations $\mathcal{D}$. The main idea is to first obtain a high-level solution guided by BO about the proportion of AI tasks on each resource and total triangle count ratio (Line 1), and then translate them into actual allocation for each task (Lines 2-22) and triangle count for each virtual object (Line 23). Using the new configuration, HBO then measures the obtained performance to calculate the corresponding cost value (Lines 24-25), which is used to update the BO database $\mathcal{D}$ for the next iteration (Line 26). After the last iteration, the configuration that obtained the lowest cost value is selected to be used until the next activation. We describe the details of the algorithm and its complexity analysis in the rest of this section.

Bayesian Optimization (Line 1). In each iteration, the algorithm first executes the BO with cost function expressed by Equation 5, the EI acquisition function, a $\mathcal{GP}$ surrogate model, Constraints 8-10, and the performance database $\mathcal{D}$ of exploration/exploitation in previous iterations of the current activation. At every execution it returns the AI resource usage c and triangle count ratio x to test in the current period.

Heuristic AI Allocation (Lines 2-22). Given that BO provides fractional values of resource usages, the algorithm then needs to translate how many of the M tasks to allocate on each resource according to c. For instance, omitting the time index t for simplicity, $c_1 = 0.4$, $c_2 = 0.1$, and $c_3 = 0.5$ mean that 40%, 10%, and 50% of the M tasks should be allocated on the resources 1, 2, and 3, respectively. Thus, the algorithm in Lines 2-12 maps each fractional value to an integer portion of the M tasks to allocate on each resource. This information is stored in the array C. Specifically, Lines 3-4 round down the resource usage values into integer numbers while Line 5 determines the remaining tasks r that must be added to vector C (due to rounding). In Lines 6-12, if $r > 0$, it then determines on which resource to add the remaining r tasks, giving priority to the resource that has the highest usage. This is because a higher resource usage choice from BO may indicate the need to explore/exploit how the taskset behaves when more tasks are allocated to certain resources. Thus, in Line 7 the algorithm sorts vector c and determines the indexes $\alpha(i)$ in non-increasing capacity order. The algorithm then iterates in Lines 8-12 to add one task to each resource following the sorted order until all remaining tasks are allocated (Line 12). For example, if $c = [0.4, 0.1, 0.5]$ with three AI tasks, then Lines 2-12 would populate vector C as $[1, 0, 2]$, i.e., one task to resource one, and two tasks on resource three.

The algorithm then starts a while loop to allocate each of the M tasks on one of the available resources (Lines 14-22). In each iteration, to greedily achieve low latency, HBO starts the resource assignment from the highest-priority task, which corresponds to the task with the lowest profiled latency. To do this, HBO retrieves the head element of the priority queue P and stores its task/resource indexes in (i^*, j^*). Then, it checks the array C to determine whether the required resource j^* for this task is available (Line 16). If there is enough space, it finalizes the task i^* allocation to resource j^*, updates the availability of C_{j^*}, and increments the count of assigned tasks k (Lines 17-19). Additionally, in Line 20, the algorithm removes all the N instances of the same task ID i^* from the priority queue P (since this task has already been allocated). However, if the high-priority resource $j*$ is not available, the algorithm narrows down the search space by removing all tasks requiring the same AI allocation $j*$ from P in Line 22. It then proceeds to the next task in P to find another available resource for the remaining high-priority tasks.

Triangle Distribution (Line 23). In Line 23, the TD function is invoked to distribute the chosen total triangle count $x{\cdot}T^{max}$, among the virtual objects in the augmented environment. To do so, we utilize the degradation model in Equation 1 and distribute the triangles leveraging as weight the *sensitivity of virtual objects' degradation to triangle variations*, which is evaluated, for each object, based on the difference between its degradation at a reference decimation ratio (same for all virtual objects) and the current object degradation. This choice, allows to enhance the overall average quality by giving more triangles to the most sensitive objects, e.g., the ones closer to the user or with particular shapes. The new decimated versions are then redrawn on screen.

Bayesian Database Update (Lines 24-26). After applying the configuration decision of the Bayesian optimization to test in the current iteration, the algorithm starts collecting performance data in terms of average latency and quality for the remaining time of the period t (Line 24). At the end, it then calculates the cost function $\varphi_t = -B_t$ obtained (see Equation 3) and, in Line 26, it updates the dataset $\mathcal{D}$ with the configuration (c_t, x_t) and their corresponding cost value φ_t. Finally, the algorithm waits for the next iteration to start.

HBO Complexity. The complexity of the proposed algorithm is influenced by several factors. The first factor is the complexity of BO (Bayesian optimization with *known* constraints), which at each iteration is dominated by $O(K^3)$, where K represents the number of evaluations of candidate solutions [23]. The second factor is the complexity of Lines 3-12, which is bounded by the sort function in Line 7 that runs no more than M times, requiring $O(MN\log(N))$. The third factor is the AI resource distribution in Lines 14-22, which includes priority queue P operations and while loop iterations. The priority queue P implemented by a binary heap requires $O(\log(MN))$ time for each removal operation. Every iteration of the while loop removes N or M elements from the priority queue P. Thus, in the worst case, the while loop complexity is bounded by $O(MN\log(MN))$. The complexity of triangle count distribution in Line 23 is $O(L\log(L))$, where L is the number of virtual objects, due to sorting objects based on their sensitivity order. As a result, the total complexity of the proposed algotrithm is $O(K^3 + MN\log(MN) + L\log(L))$.

E. HBO Activation

Rather than activating the optimization periodically, to limit the system overhead and impact of exploration during iterations we design an event-based activation policy for the Bayesian optimization. In particular, we found that two main

TABLE II: Example scenarios used in our experiments.

Virtual Objects (SC1)	Count	Triangles
apricot	1	86,016
bike	1	178,552
plane	4	146,803
splane	1	146,803
Cocacola	2	94,080
Virtual Objects (SC2)	Count	Triangles
cabin	1	2,324
andy	2	2,304
ATV	2	4,907
hammer	2	6,250
AI Models (CF1)	Count	Task
mnist	1	Digit Classifier
mobilenetDetv1	1	Object detection
model-metadata	2	Gesture Detection
mobilenetv1	1	Image Classification
efficientclass-lite0	1	Image Classification
AI Models (CF2)	Count	Task
mnist	1	Digit Classifier
mobilenetDetv1	1	Object detection
efficientclass-lite0	1	Image Classification

factors that can affect performance of AR or AI tasks are change in the number of object count and user-object distance. The former impacts the load of rendering virtual objects on the processing units (i.e., CPU, GPU), thus impacting the average AI latency. The latter, can change the average AI task latency due to the backface culling of the OpenGL library [26]. Our event-based activation policy initially runs HBO after the first object placement. This is to record a reference for reward value B_t (Equation 3). Then, it monitors B_t periodically for any changes due to variations in distance or total triangle count. When the current reward value changes by a minimum tunable fraction from the reference, Algorithm 1 is executed over a fixed number of iterations to find a new configuration that can improve system performance. The new obtained reward is then used as new reference for future activations. We evaluate the benefits of this policy in Section V-D while in Section VI we discuss on the real-life scenarios where our approach is most suitable and those that require further research.

V. Experimental Results

A. Experimental Setup

We implemented the proposed framework and tested it on Android 13 and 14 (source code will be available after publication). We leveraged the *skopt* package of the scikit-optimize library to implement the Bayesian optimization in Line 1 of Algorithm 1 and tested it on several devices, including Google Pixel 7 and Samsung Galaxy S22. Due to space limitation and similarity, in this section we show the results with the Pixel 7 (Google Tensor G2 SoC, Octa-core CPU, 128 GB Storage, 8 GB RAM, Mali G710 GPU, and Tensor Processing Unit). Due to the lack of open-source MAR apps with a large virtual objects dataset and the unavailability of source codes from previous potential baselines (e.g., [7], [27]), we created an example MAR app as well as some scenarios of AI tasksets and virtual objects (see Table II) to test our solution against the following baselines:

Static Match Quality (SMQ) uses the same triangle count distribution as HBO to obtain similar average quality but uses a static task allocation policy that assigns each AI task on the resource that has shown the lowest latency when profiled in isolation (see examples in Table I). This baseline helps us quantify the impact of the dynamic resource allocation on the task latency driven by our Bayesian formulation and heuristics.

Static Match Latency (SML) uses the same static AI allocation strategy as SMQ but the total triangle count of virtual objects is gradually reduced until the average latency is similar to that of HBO. This baseline helps us quantify the impact of the jointly manipulating triangle count and resource allocation focusing on the virtual object quality metric.

Bayesian No Triangle (BNT) is similar to HBO in that it employs the same heuristic for AI task relocation, but it does not regulate the triangle ratio. Its BO's cost function solely incorporates the average latency. This baseline helps us show that modifying AI task allocation alone, without regulating the quality of objects, cannot lead to similar AI performance.

All NNAPI (AllN) leverages the state-of-the-art NN-API of Android [18] available on all Android devices running Android 8.1 or later. We tested AllN on Android 13 (API 28). Similar to prior studies [2], [8], NNAPI distributes each operation of deep learning workloads across CPU, GPU, and NPU to reduce inference latency and leaves virtual objects at the highest quality level. Thus, it does not optimize the load due to AR tasks and can incur latency overhead when an operation becomes too slow on the GPU due to heavy virtual object rendering.

In the next sections, we first present an analysis of HBO's behavior across four distinct scenarios. Then, in Section V-C we compare its performance with the above baselines. In Section V-D we provide an in-depth analysis of HBO's properties and in Section V-E we report the results of our user study.

B. HBO Behavior Across Scenarios

We evaluate HBO's performance using as example weight $w = 2.5$ in Equation 3. Here, we present results of HBO in four different experiments that encompass various AI tasks and virtual object combinations (see Table II). The goal is to prove its ability to adapt its choices on the specific combination of virtual object and AI tasksets. The first set of virtual objects (SC1) includes objects with high triangle count while the second one (SC2) features virtual objects with low triangle count. Similarly, the first taskset (CF1) includes six AI tasks. Based on our offline profiling in isolation, three of these tasks are optimized for better performance on the GPU delegate, while the remaining exhibit a lower latency when using the NNAPI delegate. On the other hand, the second taskset (CF2) includes three AI tasks, with one favoring GPU and the other two favoring NNAPI. All the models are pre-trained and available in the TensorFlow Lite repository [16].

In our experiments, in each activation we initialize dataset $\mathcal{D}$ through 5 random configurations of allocation proportions and triangle ratio, and then execute HBO for 15 iterations to ensure convergence to a solution. The results we present here are based on HBO's activation after all objects have been added in the augmented environment, with all AI tasks concurrently performing inferences. We show the results over mul-

TABLE III: AI allocation and triangle ratio in four scenarios.

AI Model/Scenario	SC1-CF1	SC2-CF1	SC1-CF2	SC2-CF2
mobilenetDetv1	NNAPI	NNAPI	NNAPI	NNAPI
efficientclass-lite0	NNAPI	NNAPI	NNAPI	NNAPI
mobilenetv1	NNAPI	NNAPI	-	-
mnist	CPU	GPU	CPU	NNAPI
model-metadata_1	CPU	NNAPI	-	-
model-metadata_2	CPU	CPU	-	-
Triangle Count Ratio	0.72	1	0.85	0.94

tiple activations in Section V-D. Next, we explore how HBO converges to a lower-cost solution across four distinct combinations of the virtual objects and AI tasks shown in Table II.

AI Allocation Analysis. Figures 4a and 4b show HBO's solution in terms of AI task allocation and triangle count ratios, respectively, across the four experiments. For space reasons, here we analyze the configurations chosen by HBO in various scenarios and leave for the next section the comparison with the baselines in terms of the obtained performance. Table III shows the detailed breakdown of each experiment's individual task assignments and triangle ratio choices. We can observe in Figure 4a that for the first and third experiments (SC1-CF1 and SC1-CF2), HBO relocates all the tasks with high GPU-delegate affinity to the CPU. This migration is primarily a response to avoid losing too much object quality in these two particular scenarios and reach slightly better latency. Indeed, these two SC1 scenarios with higher number of virtual objects reduce the triangle count by ratio 0.72 and 0.85, respectively, to further enhance AI latency. On the other hand, the second and fourth experiments (SC2-CF1 and SC2-CF2), feature lighter-weight objects, which reduce the demand for the GPU and enable more AI tasks to operate either through the GPU delegate or through the NNAPI delegate[2]. This is why HBO can keep tasks of SC2-CF2, with lower number of AI tasks, on their preferred allocation, NNAPI. Similarly, SC2-CF1 retains the task mnist, which has the lowest latency overall, on the GPU, maintains the NNAPI-preferred tasks on the same resource, and allocates the other two GPU-preferred AI tasks one to the CPU and the other to the NNAPI delegate for optimized resource utilization.

Cross-Scenario Convergence Study. Figure 4c shows the best cost value obtained while HBO runs iterations to progressively explore the search space for lower-cost, ultimately improving performance for the various AI allocation and triangle count ratio configurations explored. When comparing the best cost across the four experiments, the last scenario (SC2-CF2) exhibits the lowest value. This can be mainly attributed to minimal competition between AI inference and AR tasks for computational resources. Hence, it leads to a more favorable cost and improved system performance in comparison to the other scenarios. Notably, HBO shows its ability to converge to a lower-cost solution in the best-case after just 7 iterations and on average 13 (including the 5 initial explorations to populate the BO database). We will study the convergence properties more in detail in Section V-D.

[2]For tasks running on NNAPI, certain operators not supported on NPU or TPU may run on GPU, further increasing GPU's demand

TABLE IV: AI allocation and triangle ratio comparison.

AI Model/Experiment	HBO	SMQ, SML	BNT	AllN
mobilenetDetv1	NNAPI	NNAPI	NNAPI	NNAPI
efficientclass-lite0	NNAPI	NNAPI	CPU	NNAPI
mobilenetv1	NNAPI	NNAPI	NNAPI	NNAPI
mnist	CPU	GPU	CPU	NNAPI
model-metadata_1	CPU	GPU	CPU	NNAPI
model-metadata_2	CPU	GPU	CPU	NNAPI
Triangle Count Ratio	0.72	0.72, 0.5	1	1

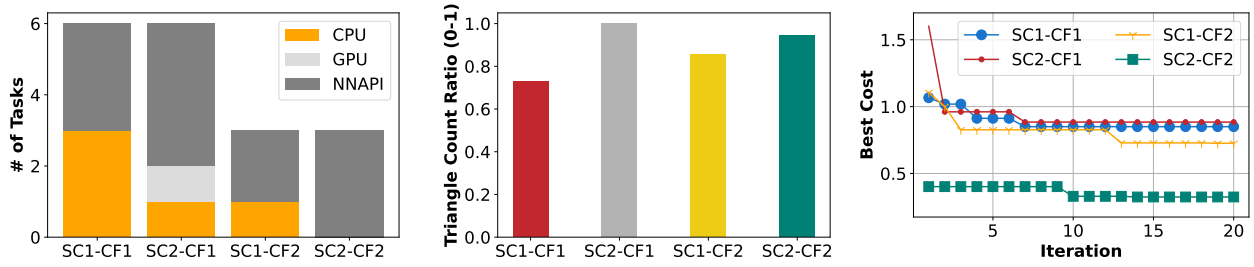
These results show that HBO can automatically adapt to different scenarios of virtual objects and tasksets with little information prior execution.

C. HBO Performance Comparison

Performance Comparison. We evaluate the performance of HBO in comparison to SMQ, SML, BNT, and AllN under various scenarios of virtual object and AI task configuration. In this discussion, we present one of our results, specifically for the SC1-CF1 scenario, which is more challenging in terms of the number of virtual objects and AI tasks (see Table II for more details). Figures 5a to 5c compare the AI task allocation, triangle count selection, average quality of virtual objects, and latency ratio of baselines and HBO. As we discussed in Section V-B, HBO makes the decision to relocate three GPU-preferred tasks to CPU and reduces the total triangle count ratio to 72% of maximum triangle count. In the following, we will explore how this decision can enhance system performance compared to the defined baselines.

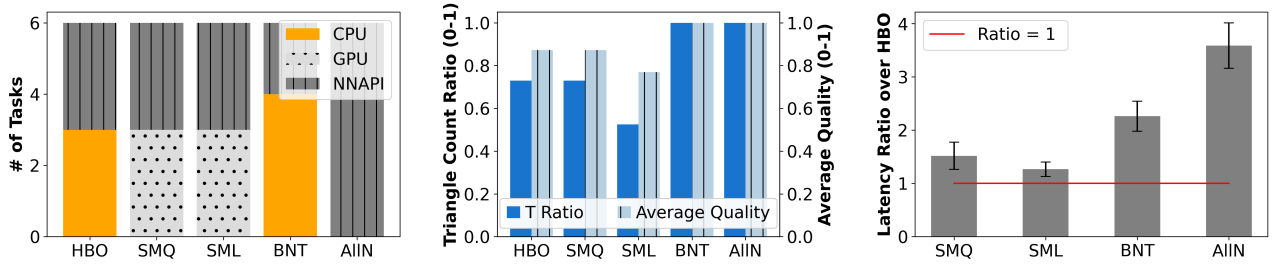
In Figure 5a, we observe that SMQ and SML both rely on selecting specific delegates based on static data analysis, as shown in Table IV, without considering the impact of each task assignment or triangle count ratio on overall system performance. Specifically, we use the same triangle ratio for SMQ to achieve a similar average quality as HBO but this comes at the cost of 1.5x higher average latency compared to our solution (see Figure 5c). On the other hand, we could manage to approach HBO's average latency using static task allocation through simply reducing total triangle count of virtual objects as SML does. However, we cannot guarantee the same high average quality of virtual object as achieved by HBO. Indeed, HBO achieves 14.5% better average quality than SML under comparable average latency. These results demonstrate HBO's better performance over the static baselines due to jointly reallocating AI tasks other than reducing virtual object quality.

In contrast to the static allocation baselines, BNT and AllN do not reduce objects quality but perform dynamic allocation to optimize system performance. As we can see from the results, BNT, which uses a simplified version of our Bayesian formulation, chooses to completely relocate some AI tasks to the CPU at run-time, thus enhancing the performance of AI inference tasks compared to AllN. This helps reducing the competition among tasks using NNAPI and generally alleviates high communication overhead. In fact, AllN automatically splits inference execution across CPU and AI accelerators, including NPU and GPU, to reduce latency. The high triangle count of virtual objects in these two baselines leads to a substantial GPU load. This can potentially extend either the queuing time of AI tasks' operators relying on



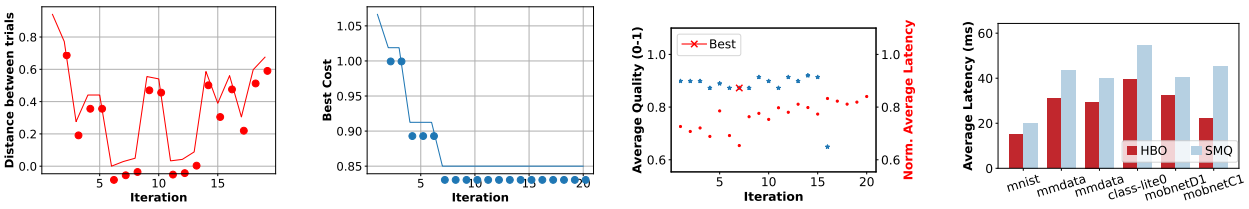
(a) (b) (c)

Fig. 4: HBO behavior for (a) AI task allocation, (b) triangle count ratio, and (c) best cost convergence through iterations.



(a) (b) (c)

Fig. 5: HBO performance vs baselines: (a) task allocation, (b) average quality vs triangle count ratio, (c) average latency ratio.



(a) (b) (c) (d)

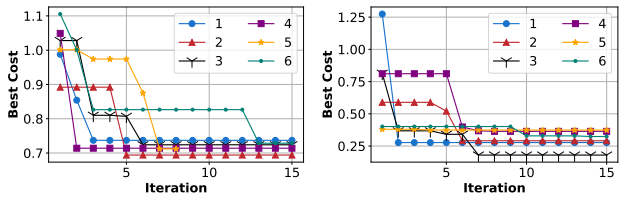
Fig. 6: Detailed analysis of HBO: (a) Distance between consecutive input values, (b) best-cost selected samples through iterations, (c) average quality and latency over iteration, (d) individual AI model latency compared to SMQ.

the GPU for inference or the concurrency of the tasks on the NPU, leading to performance degradation. Including the total triangle count manipulation into the decision variables of the Bayesian optimization, as HBO does, can help further improve the overall performance. We can observe that HBO results in 2.2x and 3.5x better latency on average compared to BNT and AllN, while reducing the virtual object quality of only 13% (i.e., 1.15x). These results show that operator-level solutions such as NNAPI may not necessarily enhance AI latency in MAR apps. Our approach to AI allocation, combining various coarse-grain allocation methods (e.g., CPU inference, GPU delegate, or per-operator NNAPI delegate) and triangle count manipulation, proves more effective in improving system performance.

D. In Dept Analysis of HBO Convergence and Activation

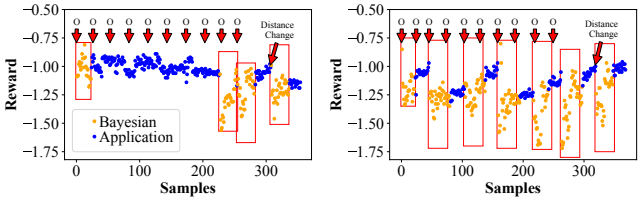
Here, we analyze in detail how HBO executes, its in-depth convergence properties, and activation policy performance.

Figure 6 provides in-detail analysis of HBO execution for SC1-CF1. In Figure 6a we calculate the Euclidean distance between consecutive configurations selected by the Bayesian optimization of HBO, which consists of three resource usage values and one triangle count ratio over multiple iterations. This distance shows HBO's acquisition function exploration (large distances) and exploitation (small distances) as it converges to a low-cost solution. In fact, as Figure 6b shows, HBO converges to the minimum cost after 7 iterations and remains robust to various explorations thereafter. Increasing the maximum iteration count could yield a more cost-effective solution at the expense of increased exploration time, which may not align with user expectations and satisfaction. Figure 6c shows the average quality of virtual objects and AI latency, the two metrics used to calculate the BO's cost, over the iterations. The selected point with the lowest cost across 20 iterations is the 7^{th} point, marked by a red cross signs in the figure. In this iteration, the average quality is 0.87, and the normalized average latency is 0.69, reflecting a better configuration for system performance trade off. Specifically, using the best configuration found, Figure 6d shows a comparison of each AI task latency in milliseconds between HBO and SMQ, which has shown best results across the other baselines (Figures 5b



(a) (b)

Fig. 7: HBO best-cost convergence through iterations in different runs of scenarios (a) SC1-CF2 and (b) SC2-CF2.



(a) HBO (b) Periodic

Fig. 8: HBO activation through (a) our policy and (b) periodic.

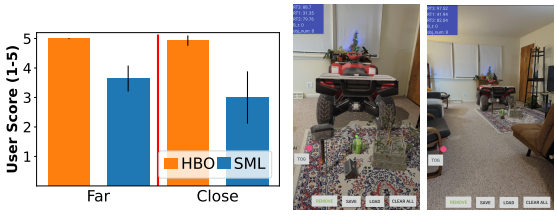
and 5c). Under the same total triangle count ratio (i.e., same AR task performance), HBO's decision to move the first three tasks (mnist and two mmdata) from their static-preferred GPU to the CPU not only improves the performance of these tasks but also alleviates the burden on the resources used by the other three AI tasks running on NNAPI, thus decreasing their latency. In the best case HBO can improve their latency by 103% (MobileNet classification, mobnetC1) and, in the worst case, by 23.8% (MobileNet detection, mobnetD1), both running on the NNAPI delegate.

HBO Convergence Robustness Study. To study HBO's convergence, we monitor the best cost for two scenarios, i.e., SC1-CF2 and SC2-CF2, across six runs as shown in Figures 7a-7b. The 6^{th} run of each figure corresponds to the same experimental run in Figure 4c. Through each run we observe the same scenario could lead to a slightly different best solution in terms of the AI allocation and/or the total triangle count ratio. This variability arises from the initialization phase upon an activation, where a few random configurations (5 in our setting) are explored before the algorithm starts the exploring/exploiting phase according to the chosen EI acquisition function. However, it is noteworthy that, despite the Bayesian decision process, all runs converge to a similar-cost solution at the end of each run, indicating a certain robustness to the initial datapoints collected in the BO database. For example, the first and sixth run of Figure 7a result in a triangle ratio of 0.85 and an AI allocation proportion vector equal to [0.03, 0.24, 0.72] and [0.29, 0.23, 0.46], respectively, both indicating a higher preference to keep more tasks on the NNAPI (third entry in vector). The only difference is that the 6^{th} run allocates the mnist inference on the CPU. However, this model has similar latencies across all resources, thus leading to similar performance between the two runs.

HBO Activation Study. Here, we conduct a comparative analysis of HBO activation strategies outlined in Section IV-E and a periodic activation approach. Figure 8a shows the results. The set boundaries for reward increase and decrease in HBO activation are empirically determined at 5% and 10%, respectively, which allow to balance the obtained performance and the activation overhead. The experiment involves the automated addition of 10 virtual objects to the augmented environment (depicted as O signs) from the sampling time $t = 0$ to $t = 255$ and the user distance change around $t = 320$. We monitor the reward value B_t for any changes at 2-second intervals. Figure 8a shows HBO activation triggers with the first activation occurring after the placement of the first object. Subsequent activations correspond to the addition of the 9th and 10th objects and a change in user-object distance. The boxes in figures highlight the reward values during Bayesian iterations while the blue points show the reward value during normal app usage. This underscores that not all object additions significantly impact AI task performance, warranting necessity to a selective HBO activation. Indeed, the design of HBO aims to enhance performance while minimizing activation frequency to reduce computation costs. However, there might be instances such as the placement of a heavy virtual object (e.g., the 10th object with $150k$ triangle count) that prompt HBO activation to address increased average AI response times. To evaluate HBO activation concerning changes in distance, we move farther away from virtual objects around $t = 320$ where we observe HBO activation due to an improvement in average quality of virtual objects. In Figure 8b, we show the effect of a periodic HBO activation, which takes place seven times, potentially imposing unnecessary burdens on the system. This periodic approach may not effectively discern the system's actual need for performance improvement, risking a higher computational cost and a lower overall user experience due to frequent explorations.

E. HBO Performance in User Study

We conduct a small-scale user study to evaluate the performance of HBO in comparison to the SML baseline, which has similar AI task performance and helps us focus on the user-perceived quality of virtual objects. We use a scenario involving a mix of heavy and lightweight objects with six AI tasks (CF1). The experimental setup involves placing virtual objects on the screen at their maximum quality as a reference for the participants. Subsequently, we individually activate HBO and the SML baseline at both close distances (Figure 9b) and far distances (Figure 9c). In each instance, seven students were asked to evaluate the perceived quality of virtual objects on a scale of 1-5, with 5 indicating a perceived virtual object quality *equivalent* to that of the reference (max quality), and 1 indicating *much worse* quality. Figure 9a shows the average scores for HBO at close (4.9) and far (5) distances, which outperform SML at close (3) and far (3.6) distances translating in up to 38.7% quality improvement. Notably, HBO maintains a triangle ratio of 0.52, while SML requires a substantial reduction to 0.2 triangle ratio to achieve a comparable AI task



(a) User Scores (b) Close (c) Far

Fig. 9: User study (a) results and (b-c) screenshot.

latency. These results underscore its ability to optimize performance while preserving virtual object quality as perceived by real users.

VI. Limitations and Future Work

Dynamic Environment. In our experimental assessments, we have found that HBO is an effective solution for optimizing MAR performance by jointly manipulating triangle count of virtual objects and AI task allocation. Given the high problem complexity, the proposed algorithm leverages Bayesian Optimization to find a solution in polynomial time and within a few iterations at each activation. This strategy is valuable for many AR educational and professional applications enabled by recent technologies such as Animal Safari AR [28], JigSpace [29], ARvid [30], and Adobe Aero [31] that allow to create engaging and interactive experiences for users. In such scenarios, AR is used to increase the level of engagement of students in classrooms or during professional (AR-enabled) presentations where participants tend to focus on and interact with objects for extended periods with lower variation in either distance or the number of virtual objects on the screen. However, we recognize that this solution may not be suitable in other scenarios where users tend to frequently move in the augmented environment (e.g., gaming) or in more fast-paced activities. In such cases, HBO may lead to too many activations, which could hinder user experience due to frequent explorations.

More research is necessary to alleviate this problem in fast-paced scenarios. For example, we could construct a lookup table that stores environmental conditions, including maximum triangle count, average distances, and task configurations. This table could serve as a reference for assessing the similarity of the current environment to past conditions. Over time, when the user's interaction approaches conditions that closely resemble those stored in the table, the framework could choose to simply apply the solution from the lookup table instead of initiating a new and potentially unnecessary HBO activation. This approach can minimize unnecessary computations and enhance the system efficiency. We leave as future work devising new strategies for such cases.

Scalability and Overhead. HBO is designed to be scalable and to limit the overhead of running Bayesian Optimization algorithms at each activation. In fact, the optimization variables are continuous, the triangle count is cumulative thus eliminating scalability issues for increasing number of virtual objects, and the number of resources is limited by the actual computing units available on modern devices, which is usually no more than two or three delegates. On the other hand, we recognize that running such optimization may still be heavy on some devices. In such cases, the Bayesian Optimization algorithm can be executed on a local edge server to eliminate its overhead from local computations, e.g., the same running the object decimation algorithm in Figure 3, by uploading the obtained performance from the *cost calculator* to the server and downloading the next configuration to test through the Wi-Fi or 5G network interface. The payload for exchanging such information is in the order of a few Bytes, which limits the network energy overhead. The execution time of the remaining HBO components is around 50 milliseconds in our tests across various devices, thus making HBO's overhead relatively small.

VII. Conclusion

In this paper, we have shown that the concurrent execution of augmented reality (AR) rendering tasks and Artificial Intelligence (AI) inferences can significantly affect system performance, leading to increased AI task latency and compromise user experience. Jointly managing virtual object triangle count and AI task allocation is a necessary, yet complex, strategy to effectively trade off between virtual object quality and AI task latency. The proposed HBO framework provides an efficient solution to this problem. First, it leverages Bayesian optimization (BO) to identify a solution for the continuous joint optimization variables related to AI resource usage and virtual objects triangle count ratio. This optimization process aims to minimize a black-box cost function for improved performance within a few exploratory steps. In the second stage, HBO employs heuristics to incorporate the candidate solutions into the system for cost evaluation, by adjusting AI allocation of each task and the triangle count of each virtual object. We evaluate HBO with real smartphones and users against four different state-of-the-art baselines that provide static and dynamic AI allocation methods. Our results have shown that HBO helps reduce the average AI task latency by up to 3.5x and increase the average virtual object quality by up to 38.7% compared to the baselines.

References

[1] A. Pouget, S. Ramesh, M. Giang, R. Chandrapalan, T. Tanner, M. Prussing, R. Timofte, and A. Ignatov, "Fast and accurate camera scene detection on smartphones," in *Proceedings of the IEEE/CVF CVPR Workshops*, 2021.

[2] Y. Kim, J. Kim, D. Chae, D. Kim, and J. Kim, "mulayer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization," in *Proceedings of EuroSys*, 2019.

[3] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in *Proceedings of MobiSys*, 2016.

[4] M. Han, J. Hyun, S. Park, J. Park, and W. Baek, "Mosaic: Heterogeneity, communication, and constraint-aware model slicing and execution for accurate and efficient inference," in *Proceedings of PACT*, 2019.

[5] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Proceedings of IPSN*, 2016.

[6] W. Seo, S. Cha, Y. Kim, J. Huh, and J. Park, "Slo-aware inference scheduler for heterogeneous processors in edge platforms," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 4, 2021.

[7] J. Yi and Y. Lee, "Heimdall: mobile gpu coordination platform for augmented reality applications," in *Proceedings of MobiCom*, 2020.
[8] 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 *Proceedings of MobiSys*, 2022.
[9] D. Narayanan and M. Satyanarayanan, "Predictive resource management for wearable computing," in *Proceedings of MobiSys*, 2003.
[10] J. Choi, H. Park, J. Paek, R. K. Balan, and J. Ko, "Lpgl: Low-power graphics library for mobile ar headsets," in *MobiSys*, 2019.
[11] 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.
[12] K. Chen, T. Li, H.-S. Kim, D. E. Culler, and R. H. Katz, "Marvel: enabling mobile augmented reality with low energy and low latency," in *Proceedings of SenSys*, 2018.
[13] Y.-j. Seo, J. Lee, J. Hwang, D. Niyato, H.-S. Park, and J. K. Choi, "A novel joint mobile cache and power management scheme for energy-efficient mobile augmented reality service in mobile edge computing," *IEEE Wireless Communications Letters*, 2021.
[14] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proceedings of MobiCom*, 2019.
[15] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, N. Dai, and H.-S. Lee, "Furion: Engineering high-quality immersive virtual reality on today's mobile devices," *IEEE Transactions on Mobile Computing*, vol. 19, no. 7, pp. 1586–1602, 2019.
[16] TensorFlow Lite, "AI models," https://web.archive.org/web/20210225170007/https://www.tensorflow.org/lite/guide/hosted_models.
[17] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proceedings of SenSys*, 2017.
[18] Google Android, "Neural networks api," https://developer.android.com/ndk/guides/neuralnetworks, 2022.
[19] W. Xue, L. Zhang, X. Mou, and A. C. Bovik, "Gradient magnitude similarity deviation: A highly efficient perceptual image quality index," *IEEE Trans. on Image Processing*, vol. 23, no. 2, pp. 684–695, 2013.
[20] V. Nguyen, S. Gupta, S. Rana, C. Li, and S. Venkatesh, "Regret for expected improvement over the best-observed value and stopping condition," in *Proceedings of MLR*, 2017.
[21] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning.*, ser. Adaptive computation and ML. MIT Press, 2006.
[22] M. Stein, *Interpolation of Spatial Data: Some Theory for Kriging*, ser. Springer Series in Statistics. Springer New York, 2012.
[23] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.
[24] J. Yan, Q. Lu, and G. B. Giannakis, "Bayesian optimization for online management in dynamic mobile edge computing," *IEEE Transactions on Wireless Communications*, pp. 1–1, 2023.
[25] G. Lan, J. M. Tomczak, D. M. Roijers, and A. Eiben, "Time efficiency in optimization with a bayesian-evolutionary algorithm," *Swarm and Evolutionary Computation*, vol. 69, p. 100970, 2022.
[26] LearnOpenGL. Face culling. https://learnopengl.com/Advanced-OpenGL/Face-culling.
[27] 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 *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, 2022, pp. 235–247.
[28] LightUp, "Animal safari ar - 3d learning," https://apps.apple.com/us/app/animal-safari-ar-3d-learning/id1468220377, Accessed: October, 2023.
[29] JigSpace Inc., "Augmented reality presentations," https://www.jig.space, Accessed: October, 2023.
[30] E. F. Gebran, "Ar creator and camera," https://apps.apple.com/nz/app/arvid-augmented-reality/id1276546297, Accessed: October, 2023.
[31] Adobe Inc., "Interactive augmented reality," https://apps.apple.com/us/app/adobe-aero/id1401748913, Accessed: October, 2023.