BSAF-Based Binary Offloading for Edge AI: Energy-Efficient Deep Learning and Spatial Computing in Mixed Reality

Mahin Khan Mahadi*, Daniel Mawunyo Doe*, Xiangfang Li*, Zhu Han†, and Lijun Qian*
*Department of Electrical and Computer Engineering, Prairie View A&M University, Prairie View, TX 77446, USA
{mmahadi, dmdoe, xili}@pvamu.edu, lijunqian@ieee.org
†Department of Electrical and Computer Engineering, University of Houston, Houston, TX 77204, USA
hanzhu22@gmail.com

***Abstract*—This paper presents a binary offloading strategy within the binary spatial allocation framework (BSAF) to enhance energy efficiency and task accuracy in mixed reality (MR) applications. We utilize the Microsoft HoloLens 2 and an edge server configured with You Only Look Once (YOLO) models. Our proposed framework dynamically selects local or edge execution based on scene complexity and resource constraints. The decision is driven by a closed-form utility function that incorporates accuracy, latency, and energy consumption, with Lagrangian relaxation ensuring constraint feasibility. Experimental results demonstrate that binary offloading achieves competitive accuracy ($89-91\%$) compared to full offloading ($92-94\%$), while reducing average latency by up to 37% and energy consumption by 44%, respectively. Furthermore, binary offloading sustains a 48% battery level after 50 minutes while the battery is completely depleted under full offloading.**

***Index Terms*—Binary offloading, spatial computing, mixed reality (MR), edge computing, energy-efficient task allocation.**

I. Introduction

The incorporation of spatial computing into extended reality (XR) and mixed reality (MR) systems has revolutionized how humans interact with technology [1]. Devices like the Apple Vision Pro and HoloLens 2 exemplify this transformation by blending digital elements into physical environments to offer immersive and context-aware experiences [2]. XR and MR technologies are rapidly advancing in diverse domains such as medicine, education, urban planning, and entertainment [3]. However, realizing the full potential of XR and MR remains constrained by the high computational demands of tasks such as 3D mapping, object detection, and real-time user interaction, which often exceed the capabilities of portable devices [4].

To overcome these constraints, many XR and MR systems rely on multi-access edge computing (MEC) to offload computation-intensive tasks. Yet, designing adaptive, low-latency offloading strategies that balance accuracy, responsiveness, and energy consumption remains a significant challenge. Recent learning-based approaches have made progress [5]–[7], for example by addressing usability challenges, improving XR performance with fog-cloud architectures, and introducing DRL-based offloading schemes, which are deep reinforcement learning methods that learn optimal offloading policies through interaction with dynamic environments. However, they often struggle with the context-dependent nature of XR. Binary and partial offloading methods [8]–[10] aim to optimize resource distribution but suffer from frequent communication overhead and lack robust adaptation. Optimization techniques like ODM-BCSA [11], which employs a binary cuckoo search algorithm for offloading decisions, and fine-grained DNN partitioning [12], which distributes neural network layers across device and edge to reduce overhead, show promise but lack spatial adaptability. Other innovative solutions, including blockchain integration [13] and UAV-based MARL systems [14], which use unmanned aerial vehicles in combination with multi-agent reinforcement learning to coordinate offloading decisions across aerial edge nodes, often remain domain-specific or overlook immersive task constraints. A notable exception is the MPOAR (Mobility and Privacy-Aware Offloading for AR Applications) framework proposed by the authors in [15], which jointly optimizes latency, energy consumption, and mobility using a multi-objective meta-heuristic approach. These studies collectively highlight the need for XR-specific, context-aware, and resource-efficient offloading frameworks.

Moreover, while advanced object detection algorithms such as YOLO, SIFT, and ORB [16], [17] enhance spatial awareness in XR, their computational complexity limits their feasibility for real-time deployment on constrained devices. YOLO (You Only Look Once) is a real-time deep learning-based object detection model that processes images in a single pass, whereas SIFT (Scale-Invariant Feature Transform) and ORB (Oriented FAST and Rotated BRIEF) are classical computer vision algorithms that detect and describe local features invariant to scale, rotation, and noise. Benchmarks like XRBench [18] offer performance metrics but lack depth in evaluating latency–energy trade-offs. Industrial XR applications, especially within emerging metaverse platforms, continue to face scalability and multitasking constraints under strict energy and latency budgets [19], [20]. These limitations reveal a critical research gap: how to balance accuracy, latency, and energy consumption in dynamic XR and MR environments.

Motivated by these challenges, this study addresses the interrelated trade-offs among accuracy, latency, and energy efficiency in real-time MR applications. Current approaches typically address individual metrics or lack adaptability in real-time, multitasking MR contexts. This work makes the following contributions:

- We propose real-time binary spatial allocation framework

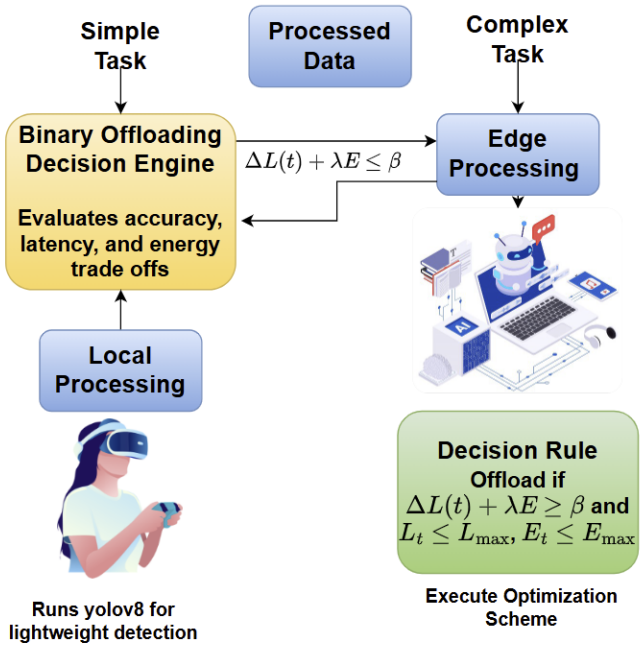


Fig. 1: System architecture with proposed latency optimization scheme

(BSAF) for MR task offloading, balancing accuracy, latency, and energy via utility-based local/edge selection.

- We design dynamic utility-based offloading that evaluates frame complexity and resource constraints for adaptive task allocation between HoloLens 2 and edge servers.
- We demonstrate experimental validation showing BSAF reduces latency, energy, and network overhead while maintaining competitive accuracy across 100 real-world scenes

II. System Model

A. System Overview

The proposed binary offloading system is developed to optimize object detection in mixed reality (MR) applications by dynamically leveraging both local processing on the HoloLens 2 and edge computing resources. The system employs a decision engine that determines whether a task should be executed locally or offloaded to an edge server based on critical factors such as accuracy, latency, and energy efficiency. The system architecture includes five key components: HoloLens 2 running YOLOv8 for low-latency detection; an edge server executing YOLOv11 [21] for high-accuracy, resource-intensive tasks; a network layer enabling fast communication via Wi-Fi or 5G; a decision engine for real-time binary offloading; and a monitoring module that tracks energy, latency, and accuracy to guide task allocation. This system ensures that computationally demanding tasks are offloaded to the edge while maintaining real-time responsiveness on the HoloLens 2, achieving a balanced trade-off between performance and energy efficiency. The system architecture with the proposed latency optimization scheme is shown in Fig. 1.

B. Network Model

We consider a system comprising a set of HoloLens 2 devices, denoted by $\mathcal{N} = \{1, 2, \ldots, N\}$, where N is the total number of devices, and each device is indexed by $n \in \mathcal{N}$. A set of edge servers, denoted by $\mathcal{S} = \{1, 2, \ldots, S\}$, where S is the total number of servers, and each server is indexed by $s \in \mathcal{S}$, is deployed across spatial subregions. The spatial environment is divided into Z subregions, indexed by $z = \{1, 2, \ldots, Z\}$, where Z is the total number of subregions. Each HoloLens device $n \in \mathcal{N}$ captures a frame workload F, representing data (e.g., video frames) collected via its on-device camera. The system dynamically decides whether to process the workload locally on the HoloLens or offload it to an edge server $s \in \mathcal{S}$, based on trade-offs involving latency, energy consumption, and computational requirements.

The total frame workload F is evenly partitioned across the Z subregions. For each subregion z, the workload is either processed locally or offloaded to an edge server, defined as:

$$F_z = \begin{cases} F_{n,z}^{\text{loc}} = \chi_{n,z}^{\text{loc}} \cdot \phi_{n,z}^{\text{loc}} \cdot \frac{F}{Z}, \\ F_{s,z}^{\text{off}} = \chi_{s,z}^{\text{off}} \cdot \phi_{s,z}^{\text{off}} \cdot \frac{F}{Z}, \end{cases} \tag{1}$$

where, $F_{s,z}^{\text{off}}$ denote the workload processed locally by device n or offloaded to server s in subregion z, respectively, with $\chi_{n,z}^{\text{loc}}$ and $\chi_{s,z}^{\text{off}}$ being binary decision indicators (equal to 1 if the workload is allocated locally or offloaded, and 0 otherwise), $\phi_{n,z}^{\text{loc}}$ and $\phi_{s,z}^{\text{off}}$ representing sensing availability indicators (equal to 1 if device n or server s can support the workload in z, and 0 otherwise), and $\frac{F}{z}$ being the total workload fraction assigned to subregion z.

The computation time for processing the workload in subregion z, denoted by t_z, depends on whether the task is executed locally or offloaded:

$$t_z = \begin{cases} t_{n,z}^{\text{loc}} = \frac{F_{n,z}^{\text{loc}} \cdot C_{n,z}}{C_n}, \\ t_{s,z}^{\text{off}} = \frac{F_{s,z}^{\text{off}} \cdot C_{s,z}}{C_s}, \end{cases} \tag{2}$$

where, $t_{n,z}^{\text{loc}}$ and $t_{s,z}^{\text{off}}$ denote the processing time (in seconds) for workload in subregion z on device n and server s, respectively, C_n and C_s represent the computational capacities (in cycles/second) of HoloLens device n and edge server s, $C_{n,z}$, $C_{s,z}$ are the computational complexities (in cycles/unit workload) for local and offloaded processing in subregion z.

The wireless uplink rate between HoloLens device n and edge server s, denoted by $R_{n,s}$, is given by:

$$R_{n,s} = b_{n,s} \cdot \log\left(1 + \frac{P_{n,s} \cdot |H_{n,s}|^2 \cdot (d_{n,s})^{-\theta}}{N_{n,s}}\right), \tag{3}$$

where, $b_{n,s}$ is the allocated bandwidth (in Hz) for the link between device n and server s, $P_{n,s}$ represents the transmit power (in watts) of device n when communicating with server s, $H_{n,s}$ denotes the channel gain between device n and server s, $d_{n,s}$ is the distance (in meters) between device n and server s, θ is the path-loss exponent modeling signal attenuation over distance, and $N_{n,s}$ is the noise power (in watts) at server s.

The total data size for offloading or local processing in subregion z, denoted by Π_{F_z}, accounts for the number of transmission hops:

$$\Pi_{F_z} = \begin{cases} \Pi^{\text{loc}}_{n,z} = F^{\text{loc}}_{n,z} \cdot \left(\left\lfloor \frac{D_n}{\delta} \right\rfloor + 1 \right), \\ \Pi^{\text{off}}_{s,z} = F^{\text{off}}_{s,z} \cdot \left(\left\lfloor \frac{D_s}{\delta} \right\rfloor + 1 \right), \end{cases} \tag{4}$$

where $\Pi^{\text{loc}}_{n,z}$ and $\Pi^{\text{off}}_{s,z}$ represent the data sizes for local processing by device n and offloaded processing to server s in subregion z, respectively, D_n and D_s denote the total transmission path lengths (in meters) for local and edge server cases, δ is the effective single-hop communication range (in meters), and $\lfloor \cdot \rfloor$ is the floor function which returns the greatest integer less than or equal to the input value.

The total round-trip time, t_{RTT}, represents the time required for one complete update cycle, including sensing, computation, and communication. It is computed as:

$$t_{\text{RTT}} = \begin{cases} \sum_{z=1}^{Z} \left(t_{F^{\text{loc}}_{n,z}} + t^{\text{loc}}_{n,z} + t_{n,s} \right), & \text{for local processing,} \\ \sum_{z=1}^{Z} \left(t_{F^{\text{off}}_{s,z}} + t^{\text{off}}_{s,z} + t_{n,s} \right), & \text{for edge processing,} \end{cases} \tag{5}$$

where, $t_{F^{\text{loc}}_{n,z}}$ and $t_{F^{\text{off}}_{s,z}}$ denote time to sense or capture frame data for local and offloaded cases in subregion z, respectively (in seconds), with $t_{n,s}$ denoting communication time between device n and server s (in seconds).

The offloading decision is determined by a utility-driven trade-off, balancing accuracy, latency, and energy consumption:

$$\alpha \cdot \Delta A > \beta \cdot \Delta L + \gamma \cdot \Delta E, \tag{6}$$

$$\text{subject to: } L_t \leq L_{\max}, \;\; E_t \leq E_{\max}, \tag{7}$$

where ΔA represents the gain in detection accuracy when offloading compared to local processing; ΔL denotes the additional latency cost incurred by offloading; ΔE represents the additional energy consumption due to offloading; α, β, γ are weighting coefficients for accuracy, latency, and energy, respectively; L_t denotes the total latency for the task (in seconds); E_t represents the total energy consumption for the task (in joules); and $L_{\max}$ and $E_{\max}$ represent the maximum allowable latency and energy consumption, respectively.

These constraints ensure that task offloading maintains real-time performance while optimizing system-wide efficiency. To improve realism and capture contention in multi-user edge computing, we extend the network model to include system-level constraints on bandwidth, computational capacity, and queuing delays.

a) Bandwidth Contention Constraint: Let $b_{n,s}$ represent the bandwidth (in Hz) allocated by HoloLens device n for communication with edge server s. The total bandwidth consumption must not exceed the server's bandwidth budget B_s:

$$\sum_{n=1}^{N} b_{n,s} \leq B_s, \;\; \forall s \in \mathcal{S}. \tag{8}$$

This constraint ensures fair allocation of wireless resources and prevents bandwidth saturation during simultaneous offloading.

b) Edge Server Computational Capacity Constraint: Let C^E_i denote the computation demand (in CPU cycles) of task i when offloaded to the edge. The cumulative workload offloaded to edge server s must not exceed its total processing budget $C^{\max}_s$:

$$\sum_{i=1}^{N} b_i \cdot C^E_i \leq C^{\max}_s, \;\; \forall s \in \mathcal{S}. \tag{9}$$

This accounts for shared edge resources and avoids overloading when multiple MR devices simultaneously offload tasks.

c) Queuing Delay Model: In practical deployments, tasks may experience waiting times at edge servers due to queuing. The average queuing delay D_q at server s is given by:

$$D_q = \frac{\rho_s}{\mu_s(1-\rho_s)}, \quad \rho_s = \frac{\lambda_s}{\mu_s}, \tag{10}$$

where λ_s is the task arrival rate and μ_s is the service rate at server s. The term ρ_s represents the utilization factor of server s, indicating the fraction of time it is busy. The total offloading latency is then updated as:

$$L_i(b_i) = (1 - b_i)L^L_i + b_i(L^E_i + D_q). \tag{11}$$

This latency model reflects congestion-induced delay, which is especially critical in bandwidth-limited environments with fluctuating load. These additional constraints and the queuing model strengthen the BSAF framework's ability to capture resource contention, scalability issues, and shared capacity limitations in realistic edge-assisted MR deployments. They are later enforced in the system-wide optimization constraints described in Section III.

C. Utility Model

To guide binary offloading decisions in the BSAF framework, a utility model quantifies the trade-offs among accuracy, latency, and energy consumption for real-time mixed reality applications. The utility function for each task J_i is:

$$U_i(b_i) = \alpha A_i(b_i) - \beta L_i(b_i) - \gamma E_i(b_i), \tag{12}$$

where $b_i \in \{0, 1\}$ indicates local execution ($b_i = 0$, using YOLOv8 on HoloLens 2) or offloading ($b_i = 1$, using YOLOv11 on the edge server), $A_i(b_i)$ represents detection accuracy, $L_i(b_i)$ denotes latency, and $E_i(b_i)$ measures energy consumption, with weights α, β, and γ prioritizing these metrics. The metrics are expressed as:

$$\begin{aligned} &A_i(b_i) = (1 - b_i)A^L_i + b_i A^E_i, L_i(b_i) = (1 - b_i)L^L_i + b_i L^E_i, \\ &E_i(b_i) = (1 - b_i)E^L_i + b_i E^E_i. \end{aligned} \tag{13}$$

where superscripts L and E denote local and edge execution, respectively. Leveraging system parameters (e.g., computation time and uplink rate from Section II-B) and monitored conditions (e.g., network bandwidth, battery status), the utility model maximizes $U_i(b_i)$ under constraints $L_i(b_i) \leq L_{\max}$ and $E_i(b_i) \leq E_{\max}$, forming the foundation for the optimization problem in Section III.

III. Problem Formulation and Binary Offloading Decision Model in the BSAF Framework

The BSAF introduces a binary offloading strategy for real-time MR applications. Each task is either offloaded to the edge or processed locally, allowing efficient, scalable decisions under strict constraints on latency, energy, and accuracy.

A. Problem Statement

The goal is to maximize a utility function that captures trade-offs among accuracy, latency, & energy for each task J_i:

$$U_i(b_i) = \alpha A_i(b_i) - \beta L_i(b_i) - \gamma E_i(b_i). \tag{14}$$

Here, $b_i \in \{0, 1\}$ indicates local ($b_i = 0$) or edge execution ($b_i = 1$). The optimization is subject to:

$$L_i(b_i) \leq L_{\max}, \;\; E_i(b_i) \leq E_{\max}. \tag{15}$$

Latency and energy are expressed as:

$$L_i(b_i) = (1 - b_i)L_i^L + b_i L_i^E, \;\; E_i(b_i) = (1 - b_i)E_i^L + b_i E_i^E. \tag{16}$$

To generalize for N tasks:

$$\max_{\{b_i\}} \sum_{i=1}^{N} [\alpha A_i(b_i) - \beta L_i(b_i) - \gamma E_i(b_i)] \tag{17}$$

$$\text{subject to: } L_i(b_i) \leq L_{\max}, \;\; E_i(b_i) \leq E_{\max}, \;\; \forall i. \tag{18}$$

This resembles a 0–1 knapsack problem and can be extended with bandwidth and capacity constraints.

B. Deriving the Optimal Offloading Decision

The utility difference between edge and local execution is:

$$U_i(1) - U_i(0) = \alpha \Delta A_i - \beta \Delta L_i - \gamma \Delta E_i, \tag{19}$$

$$\text{where: } \Delta A_i = A_i^E - A_i^L, \;\; \Delta L_i = L_i^E - L_i^L, \;\; \Delta E_i = E_i^E - E_i^L. \tag{20}$$

The binary decision rule becomes:

$$b_i = \begin{cases} 1 & \text{if } \alpha \Delta A_i > \beta \Delta L_i + \gamma \Delta E_i, \\ 0 & \text{otherwise} \end{cases}$$

C. Incorporating Latency and Energy Constraints

To ensure feasibility, the decision must also satisfy:

$$b_i L_i^E + (1 - b_i)L_i^L \leq L_{\max}, \;\; b_i E_i^E + (1 - b_i)E_i^L \leq E_{\max}. \tag{21}$$

The decision rule becomes:

$$b_i = \begin{cases} 1 & \text{if } \alpha \Delta A_i > \beta \Delta L_i + \gamma \Delta E_i, \\ & L_i^E \leq L_{\max}, \, E_i^E \leq E_{\max}, \\ 0 & \text{otherwise} \end{cases}$$

D. Final Decision Rule (Closed-Form)

The final decision rule can be written compactly:

$$b_i = \begin{cases} 1 & \text{if } \alpha(A_i^E - A_i^L) > \beta(L_i^E - L_i^L) + \gamma(E_i^E - E_i^L), \\ & L_i^E \leq L_{\max}, \, E_i^E \leq E_{\max}, \\ 0 & \text{otherwise} \end{cases}$$

This expression enables efficient, real-time decisions while respecting system constraints.

E. Lagrangian Relaxation

The binary offloading decision rule effectively balances utility with latency and energy constraints. However, in practical scenarios with dynamic network conditions or varying task complexities, strict enforcement of these constraints may make a feasible solution unattainable. To introduce flexibility while preserving constraint awareness, we apply Lagrangian relaxation. The Lagrangian function integrates the utility function and the constraint violations using multipliers:

$$\mathcal{L}(b_i, \lambda, \mu) = \sum_{i=1}^{N} [\alpha A_i(b_i) - \beta L_i(b_i) - \gamma E_i(b_i)] + \lambda \sum_{i=1}^{N} (L_{\max} - L_i(b_i)) + \mu \sum_{i=1}^{N} (E_{\max} - E_i(b_i)). \tag{22}$$

Here, λ and μ are Lagrange multipliers that penalize latency and energy constraint violations, respectively. To solve the relaxed problem, these multipliers are iteratively updated using subgradient descent:

$$\lambda^{(t+1)} = \lambda^{(t)} + \eta \cdot (L_{\max} - L_i(b_i)), \tag{23}$$

$$\mu^{(t+1)} = \mu^{(t)} + \eta \cdot (E_{\max} - E_i(b_i)), \tag{24}$$

where η is the learning rate. The algorithm continues until constraint violations fall below a predefined threshold. The decision rule can also be interpreted from the simplified utility difference form:

$$\mathcal{L}(b_i, \lambda, \mu) = \sum_{i=1}^{N} [\alpha \Delta A_i - (\beta + \lambda)\Delta L_i - (\gamma + \mu)\Delta E_i], \tag{25}$$

where $\Delta A_i = A_i^E - A_i^L$, $\Delta L_i = L_i^E - L_i^L$, and $\Delta E_i = E_i^E - E_i^L$. This formulation supports real-time adaptability by balancing optimality and constraint feasibility under variable conditions. The dual function is defined by minimizing the Lagrangian over the primal variable b_i:

$$g(\lambda, \mu) = \min_{b_i \in \{0,1\}} \mathcal{L}(b_i, \lambda, \mu). \tag{26}$$

This formulation allows the transformation of the constrained problem into a dual maximization problem:

$$\max_{\lambda \geq 0, \mu \geq 0} g(\lambda, \mu). \tag{27}$$

The Lagrange multipliers λ and μ are updated using subgradient methods as in Equations (25) and (26). A common stopping criterion is used:

$$\text{If } |\lambda^{(t+1)} - \lambda^{(t)}| < \epsilon \text{ and } |\mu^{(t+1)} - \mu^{(t)}| < \epsilon, \text{ then stop.} \tag{28}$$

Here, ϵ is a small positive threshold (e.g., 10^{-3}) indicating convergence. In practice, a maximum number of iterations (e.g., 1000) is also enforced to prevent infinite loops. This iterative process converges under a diminishing step size $\eta(t)$, assuming Slater's condition holds, which is typical in convex approximations of binary offloading. Although it does not guarantee a global optimum due to the non-convexity introduced by binary variables, it provides a flexible and

Algorithm 1: Binary Offloading and Edge Inference Process

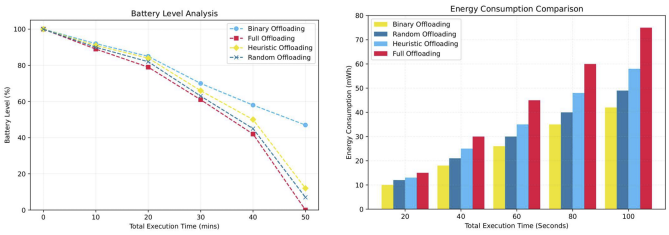
Step 1: Offloading Decision
Compute $\Delta A_i \leftarrow A_E - A_L$, $\Delta L_i \leftarrow L_E - L_L$, $\Delta E_i \leftarrow E_E - E_L$;
Compute utility: $U_i(1) - U_i(0) = \alpha \Delta A_i - \beta \Delta L_i - \gamma \Delta E_i$;
if $L_E \leq L_{max}$ ***and*** $E_E \leq E_{max}$ ***and*** $U_i(1) > U_i(0)$ **then**
    $b_i \leftarrow 1$; `// Offload`
else
    $b_i \leftarrow 0$; `// Process locally`
Step 2: Object Detection
Load YOLOv11 if $b_i = 1$, else YOLOv8;
Initialize input stream, set resolution, preprocess frames;
while *stream active* **do**
    Capture and preprocess frame;
    Run YOLO model, get outputs;
    foreach *object* **do**
        if *confidence* $>$ *threshold* **then**
            Annotate frame;
return b_i, *detection results*

constraint-aware solution. In single-task or decoupled settings, the decision rule behaves as a greedy optimal solution, while in multi-task scenarios, it functions as an efficient heuristic that balances utility and feasibility.

F. System Implementation

In the proposed framework, scene complexity is first assessed to decide whether a task should be processed locally or offloaded to the edge. This is determined using lightweight heuristics such as object count, edge density, and Shannon entropy. Low-complexity frames are processed on HoloLens 2, while complex scenes are offloaded to the edge server. After capturing a frame, the system computes its complexity and passes it to the binary offloading engine, which evaluates the utility function $U_i(b_i)$ based on accuracy, latency, and energy. If offloading offers higher utility within system constraints, the engine sets $b_i = 1$; otherwise, $b_i = 0$. YOLOv8 [22] runs locally, and YOLOv11 handles offloaded tasks where results are rendered in Unity. A monitoring module continuously tracks metrics such as battery status (via the HoloLens SDK), network latency (RTT), and bandwidth. These metrics serve as feedback for refining offloading decisions. Finally, a lightweight HTTP server enables asynchronous communication between Unity and the Python backend, exchanging telemetry data and offloading decisions via JSON and Unity XR Toolkit.

The binary offloading decision engine and edge-side object detection process are illustrated in Algorithm 1. The performance of the utility-based decision rule is sensitive to the weight parameters α, β, γ, which were empirically tuned through sensitivity analysis. By varying these weights and observing performance across test scenes, we identified a configuration that balanced utility under system constraints and proved consistently effective. While a formal ablation study is omitted, the results support the robustness of this tuning. Future work will explore adaptive methods such as Bayesian optimization or reinforcement learning to dynamically adjust the weights. The current model assumes independent tasks and does not account for queuing delays or bandwidth contention in multi-user scenarios. Additionally, hard latency deadlines are not explicitly modeled, which may limit applicability in congested environments. The algorithm has a runtime complexity of $\mathcal{O}(N)$, where N is the number of tasks, since each decision involves independent utility evaluation and constraint checks, making it scalable for real-time MR applications.



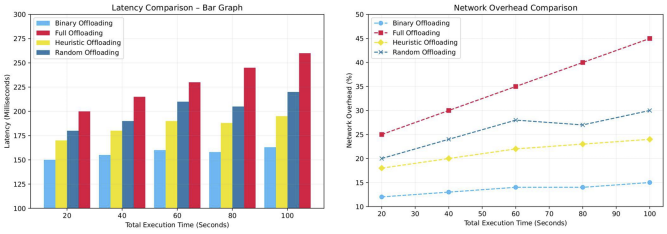
(a) (b)

Fig. 2: (a) Battery Level Depletion Trends and (b) Energy Consumption Comparison with Execution Time

IV. Results and Discussion

The evaluation was conducted on 100 mixed reality (MR) scenes captured using HoloLens 2 during indoor navigation and object interaction tasks. The dataset spans multiple sessions and reflects a range of real-world conditions, including object diversity (2 to 15 objects per scene, from simple tools to complex machinery), various lighting conditions (natural daylight, artificial lighting, and dim environments), motion blur and occlusion due to head and hand movements, and varying scene complexity measured through entropy and edge density analysis.

These scenarios represent common MR applications such as maintenance, mapping, and assisted navigation. This diversity ensures robust evaluation under realistic conditions. To assess the binary offloading framework, five metrics were considered: accuracy, latency, energy consumption, battery level, and network overhead. Each was compared across binary, full, heuristic, and random offloading strategies commonly referenced in offloading literature. This study used five measures to evaluate performance. These were prediction accuracy, the energy used for each task, the total delay or latency, the remaining battery level over time, and the network overhead, which compares data transfer time to computation time. Among the evaluated strategies, full offloading achieves the highest accuracy (92–94%) using the edge-deployed YOLOv11 model. Binary offloading remains competitive (89–91%) by adaptively using YOLOv8 locally for simple scenes and offloading complex ones, reducing network usage while maintaining reliability. Heuristic (83–85%) and random (75–80%) offloading perform worse due to less informed decisions. Figs. 2 and 3 illustrate the performance of different offloading strategies across four



(a) (b)

Fig. 3: (a) Latency Analysis and (b) Network Overhead Induced by Offloading Strategies

key metrics. In Fig. 2a, battery level depletion over time is shown. Here, binary offloading retains the highest battery life (48% after 50 minutes), while full offloading drains it most rapidly and heuristic and random strategies end below 15%. Fig. 2b presents energy consumption over time, where binary offloading is the most efficient ($42\ mWh$ at 100 seconds), compared to full offloading (up to $75\ mWh$) and other baselines. Fig. 3a compares end-to-end latency. Here, binary offloading achieves the lowest delay (150–$163\ ms$) through local-first execution, while full offloading suffers the highest (up to $260\ ms$), and heuristic and random strategies perform inconsistently. Finally, Fig. 3b shows network overhead, where binary offloading maintains minimal usage (12–15%) by avoiding unnecessary transmissions, unlike full offloading, which reaches up to 45%. Overall, the figures demonstrate that binary offloading strikes the best trade-off between performance and efficiency, offering a scalable, low-latency, and energy-conscious solution for real-time mixed reality tasks.

V. Conclusion

This paper proposed a binary offloading strategy within the BSAF framework to optimize task execution in MR object detection pipelines. The system selectively executed tasks on HoloLens 2 or offloaded these tasks to a remote edge server, guided by a utility-based decision engine that accounts for latency, accuracy, and energy. Quantitative evaluation showed that binary offloading maintains high detection accuracy ($89-91\%$), closely matching full offloading ($92-94\%$), while significantly reducing latency and energy consumption. Battery drain was also mitigated, preserving up to 48% charge after prolonged execution. Compared to heuristic and random offloading, binary offloading demonstrated superior adaptability and efficiency across all metrics.

Acknowledgements

This research was sponsored by the Army Research Office under Cooperative Agreement Numbers W911NF-23-1-0214 and W911NF-24-2-0133.

References

[1] K. M. Stanney, C. Hughes, H. Nye, E. Cross, C. Boger, and S. Deming, "Interaction design for augmented, virtual, and extended reality environments," in *Interaction techniques and technologies in human-computer interaction*, pp. 355–405, CRC Press, 2024.

[2] V. Yadav and S. Kumar, "Future of augmented reality: A comprehensive study on apple vision pro.," *International Journal of Applied Marketing & Management*, vol. 9, p. 26, Jan 2024.

[3] I. Yaqoob, K. Salah, R. Jayaraman, and M. Omar, "Metaverse applications in smart cities: Enabling technologies, opportunities, challenges, and future directions," *Internet of Things*, vol. 23, p. 100884, Oct 2023.

[4] A. Çöltekin, I. Lochhead, M. Madden, S. Christophe, A. Devaux, C. Pettit, O. Lock, S. Shukla, L. Herman, Z. Stachoň, *et al.*, "Extended reality in spatial sciences: A review of research challenges and future directions," *ISPRS International Journal of Geo-Information*, vol. 9, p. 439, Jul 2020.

[5] V. Bhaskaran and U. Mahbub, "Immersive user experiences: Trends and challenges of using xr technologies," in *Computer Vision: Challenges, Trends, and Opportunities*, pp. 260–278, CRC Press, Jul 2024.

[6] E.-S. Lee and B.-S. Shin, "Enhancing the performance of xr environments using fog and cloud computing," *Applied Sciences*, vol. 13, p. 12477, Nov 2023.

[7] B. Trinh and G.-M. Muntean, "A deep reinforcement learning-based offloading scheme for multi-access edge computing-supported extended reality systems," *IEEE Transactions on Vehicular Technology*, vol. 72, pp. 1254–1264, Sep 2022.

[8] X. Yu, S. Zhou, and B. Wei, "Dependent task offloading and resource allocation via deep reinforcement learning for extended reality in mobile edge networks," *Electronics*, vol. 13, p. 2528, Jun 2024.

[9] B. Trinh and G.-M. Muntean, "A deep reinforcement learning-based resource management scheme for sdn-mec-supported xr applications," in *IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, pp. 790–795, IEEE, Jan 2022.

[10] I. Aliyu, S. Oh, N. Ko, T.-W. Um, and J. Kim, "Dynamic partial computation offloading for the metaverse in in-network computing," *IEEE Access*, vol. 11, pp. 1–12, Dec 2023.

[11] M. Alqarni, A. Cherif, and E. Alkayyal, "Odm-bcsa: An offloading decision-making framework based on binary cuckoo search algorithm for mobile edge computing," *Computer Networks*, vol. 226, p. 109647, May 2023.

[12] E. Kilcioglu, H. Mirghasemi, I. Stupia, and L. Vandendorpe, "An energy-efficient fine-grained deep neural network partitioning scheme for wireless collaborative fog computing," *IEEE Access*, vol. 9, pp. 79611–79627, May 2021.

[13] L. J. R. Lopez, D. Millan Mayorga, L. H. Martinez Poveda, A. F. C. Amaya, and W. Rojas Reales, "Hybrid architectures used in the protection of large healthcare records based on cloud and blockchain integration: A review," *Computers*, vol. 13, p. 152, Jun 2024.

[14] W. Liu, B. Li, W. Xie, Y. Dai, and Z. Fei, "Energy efficient computation offloading in aerial edge networks with multi-agent cooperation," *IEEE Transactions on Wireless Communications*, vol. 22, pp. 5725–5739, Jan 2023.

[15] K. Peng, P. Liu, M. Bilal, X. Xu, and E. Prezioso, "Mobility and privacy-aware offloading of ar applications for healthcare cyber-physical systems in edge computing," *IEEE Transactions on Network Science and Engineering*, vol. 10, pp. 2662–2673, Jun 2022.

[16] Y. Goh, M. Choi, J. Jung, and J.-M. Chung, "Partial offloading mec optimization scheme using deep reinforcement learning for xr real-time m&s devices," in *IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1–3, IEEE, Jan 2022.

[17] W. Burger and M. J. Burge, "Scale-invariant feature transform (sift)," in *Digital Image Processing: An Algorithmic Introduction*, pp. 709–763, Springer, Jul 2022.

[18] W. Ma and F. Xu, "Study on computer vision target tracking algorithm based on sparse representation," *Journal of Real-Time Image Processing*, vol. 18, pp. 407–418, Apr 2021.

[19] F. Siddiqui, S. Zafar, S. Khan, and N. Iftekhar, "Computer vision analysis of brief and orb feature detection algorithms," in *International Conference on Computing in Engineering & Technology*, (Singapore), pp. 425–433, Springer, Feb 2022.

[20] H. Kwon, K. Nair, J. Seo, J. Yik, D. Mohapatra, D. Zhan, J. Song, P. Capak, P. Zhang, P. Vajda, *et al.*, "Xrbench: An extended reality (xr) machine learning benchmark suite for the metaverse," *Proceedings of Machine Learning and Systems*, vol. 5, pp. 1–20, Mar 2023.

[21] E. H. Alkhammash, "Multi-classification using yolov11 and hybrid yolo11n-mobilenet models: A fire classes case study," *Fire*, vol. 8, p. 17, Jan 2025.

[22] M. K. Mahadi, R. Rahad, M. S. Mobassir, A. Rahman, A. Shafiullah, and M. M. Nishat, "Implementation of personal safety equipment tracking & detection by deepsort & yolov8," in *International Conference on Inventive Computation Technologies (ICICT)*, pp. 1969–1974, IEEE, Apr 2024.