Monash University

Faculty of Information Technology

Master of INFORMATION TECHNOLOGY

A Context-Aware Computing Framework for Performance Improvement in Distributed Vision-Based Augmented Reality

---

Author:
Shuyang Yan

Student ID:
35009411

Course:
FIT5128

Supervisors:
Pari Delir Haghighi
Jiazhou 'Joe' Liu
Fucai Ke

Part 2 Word Count:
7782

May 29, 2026

MONASH University

Contents

Part 1

General Literature Review

Monash University

Faculty of Information Technology

Master of Information Technology

Literature Review

A Context-Aware Computing Framework for Performance Improvement in Distributed Vision-Based Augmented Reality

Author:
Shuyang Yan

Student ID:
35009411

Supervisors:
Jiazhou 'Joe' Liu
Pari Delir Haghighi
Fucai Ke

Semester 2, 2025

MONASH University

Contents

1 Introduction

Augmented Reality (AR) provides users with immersive experiences by overlaying the physical world with digital information [1]. It is increasingly becoming a part of everyday life and is enhancing experiences in fields such as education, healthcare, and construction. This widespread adoption has been driven by the growing availability of mobile platforms, including smartphones, tablets, and wearable head-mounted displays (HMDs), which allow users to access AR applications anytime and anywhere [2]. Meanwhile, to achieve better immersive experiences, modern AR applications often rely on Computer Vision (CV) models to identify, track, and understand real-world environments. Given this critical dependence on Computer Vision technologies, this paper refers to AR systems that integrate CV models as vision-based AR systems.

However, integrating AR with CV models requires substantial computational resources, which often exceed the capabilities of mobile devices. To overcome these limitations, a common approach is to adopt distributed computing, where mobile AR devices offload intensive tasks to backend servers, typically located in the cloud or at the edge. In this setup, AR devices transmit captured images to backend servers for processing (e.g., object detection) and await the returned results for rendering, a server–client workflow that inherently introduces end-to-end latency. This problem is further exacerbated by user mobility, as inference results must remain precisely aligned with dynamic real-world environments [3]. Such latency not only degrades rendering quality but also diminishes the user's Quality of Experience (QoE) [4], underscoring the urgent need to minimize transmission delays.

As a result, distributed vision-based AR systems confront a fundamental performance challenge: they must operate within the inherent constraints of mobile device resources while simultaneously managing frequent data transfers that are highly prone to latency. Therefore, achieving a seamless user experience demands a careful balance between computational efficiency, communication delay, and model accuracy. This difficult trade-off is now a primary focus of research in the field.

Context-aware computing offers a highly promising solution to this challenge. As defined by Abowd et al. [5], "Context is any information that can be used to characterize the situation of an entity." Context awareness, therefore, refers to a system's ability to utilize this contextual information to intelligently adjust its behavior accordingly, thereby providing better services. Prior to its integration with mobile AR systems, context-aware computing had already found widespread application across diverse devices and tasks. For example, smartphones leverage accelerometers and gyroscopes to recognize a user's motion state (e.g., walking, running). Integrating context-aware computing into AR systems primarily helps in optimizing resource

allocation by intelligently managing data and computational tasks. For instance, a system can leverage context information such as network type and power level to dynamically offload CV tasks to cloud servers. Indeed, AR systems, particularly AR HMD devices, are inherently well-suited for context-awareness integration [6]. This is due to their extensive array of sensors designed for immersive interactions, which allow them to capture real-world data and engage with diverse contextual scenarios.

Previous work shows that context-aware computing has great potential to improve latency and energy consumption [7]. However, in the field of vision-based AR, the use of context-aware computing remains under-explored. This thesis aims to address this gap by developing a context-aware approach to improve the performance of distributed vision-based AR systems. This project first identifies the key contextual information and then uses a fuzzy logic–based context reasoning method to reduce both latency and energy consumption.

Based on these considerations, this literature review aims to systematically review and analyze existing research related to vision-based AR systems, with a particular focus on their integration with context-aware computing. This review first clarifies the performance challenges inherent in these AR systems when performing CV tasks. Then it explores how researchers leverage context-aware computing as an optimization technique to address these challenges. This review will critically assess the cost-benefit trade-offs of this technological combination in existing research, with the aim of identifying current research gaps.

The structure of this paper is as follows: Section 2 provides a substantive literature review on vision-based AR systems and their integration with context-awareness. Section 3 summarizes the state of the art in this area and identifies the research gap. Section 4 proposes a research plan on this gap and the corresponding project timeline. Section 5 concludes this paper.

2 Substantive Literature Review

This chapter is composed of 2 sections. Section 2.1 will first review vision-based AR systems, and Section 2.2 will focus on context-aware adaptation approaches for AR systems.

2.1 Vision-based AR Systems

2.1.1 Vision-based Tasks

Augmented Reality (AR) has a history spanning more than five decades, but its recent integration with Computer Vision (CV) models has significantly revitalized the field [8]. CV tasks form the foundation of vision-based AR systems, enabling them to perceive, interpret, and augment the physical environment. A clear understanding of the characteristics of these tasks is therefore essential for analyzing the challenges such systems face. Drawing on established taxonomies [9], [10], [11], [12], [13], CV tasks can be categorized by their computational complexity and latency sensitivity, as summarized in Table 1. These CV tasks vary widely in both computational requirements and latency sensitivity. This diversity introduces fundamental tensions when deploying AR systems on resource-constrained mobile devices, shaping how applications balance local execution against remote offloading. These trade-offs will be further illustrated through domain-specific implementations in the following section.

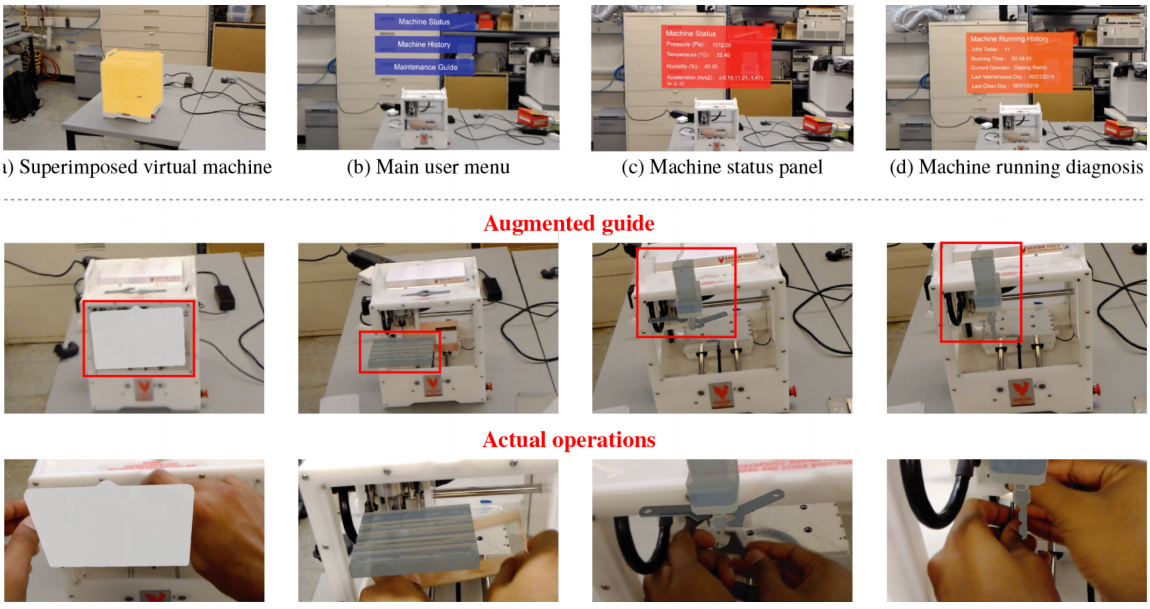
Table 1: Summary of Computer Vision Tasks in AR Systems Sorted by Task Complexity

CV Task	**Task Definition**	**AR Application Examples**	**Task Complexity**	**Computational Demand**	**Latency Sensitivity**
Image Classification	Assigning a label to an entire image to recognize the class or category of the object [13]	Indoor/Outdoor recognition, retail product category	Low	Low	Low
Image Retrieval	Finding and returning similar images from a database given a query image [9]	AR shopping (product lookup), landmark recognition	Low	Medium	Medium

Continued on next page

Table 1 – *Continued from previous page*

CV Task	**Task Definition**	**AR Application Examples**	**Task Complexity**	**Computational Demand**	**Latency Sensitivity**
Object Detection	Identifying and localizing multiple objects within an image, typically represented by bounding boxes [11]	Assembly guidance, pedestrian detection in AR-HUD	Medium	Medium	High
Object Tracking	Continuously following objects across video frames to maintain alignment [12]	Safety monitoring, traffic tracking	Medium	Medium	Very High
Image Segmentation	Pixel-level classification that assigns a class label to each pixel in an image for precise scene understanding [11]	Accurate occlusion handling	High	High	Depends
Pose Estimation	Estimating the position and orientation of objects or human bodies in 2D/3D space from images [10]	Gesture-based AR	High	Medium-High	Very High
Action and Activity Recognition	Identifying complex human actions and activities over temporal sequences [10]	Rehabilitation	High	High	Medium-High



ı) Superimposed virtual machine (b) Main user menu (c) Machine status panel (d) Machine running diagnosis

Figure 1: Top row: Machine information monitoring and visualization; Bottom two rows: Augmented maintenance guide [14].

2.1.2 Applications and Challenges

The deployment of vision-based AR across different domains reveals how application requirements, task complexity, and device constraints collectively force systems into different optimization trade-offs. By examining implementations across manufacturing, driving, and consumer applications, we can identify emerging challenges and recurring patterns in how systems handle them.

Manufacturing Applications Manufacturing is among the most computationally demanding AR domains, where precision and reliability are critical. To meet these requirements, CV tasks are often offloaded to the edge or cloud server. Sun et al. [14] illustrate this in their industrial IoT-AR system: conventional approaches struggled with textureless and complex-shaped objects commonly found in factory settings. To address these challenges, they proposed combining deep learning with AR by leveraging both color and depth images to improve accuracy and robustness. The heavy task of target machine localization and pose estimation was executed on GPU-enabled servers, ensuring accuracy regardless of the computational intensity.

More recently, Raj et al. [15] developed a Mixed Reality assembly guidance system integrating YOLOv8 for object detection and ByteTrack for object tracking.

Their system addressed scalability gaps in prior AR solutions for complex and dynamic assembly tasks. However, they reported that achieving real-time performance required high-end hardware (RTX 3090), making deployment on mobile devices infeasible. In their framework, the Microsoft HoloLens 2[1] communicated with a local server executing deep learning tasks. Although latency is low in this experimental setup, real-world deployment on cloud servers would likely introduce significantly higher delays, further constraining practical usability.

Interestingly, Židek et al. [16] took a different approach, executing CNN-based object detection entirely on smartphones and tablets. Their rationale was pragmatic: many assembly stations lack reliable network connectivity, making cloud dependency impractical. Moreover, object detection is relatively lightweight compared to tasks such as pose estimation, and its latency sensitivity often outweighs computational demands. Deploying directly on mobile devices therefore ensured timely responses despite limited computational resources. This example highlights how specific task requirements and deployment environments can lead to divergent optimization choices in manufacturing AR.

Driving Applications In the driving domain, where safety is paramount, CV tasks must be processed in real time, as even a 100 ms delay can translate into several meters of vehicle movement at highway speeds. Consequently, most AR-driving systems prioritize local execution to ensure responsiveness, often at the expense of model complexity and accuracy.

Abdi and Meddeb [17] proposed a projection-based AR system that overlays key information directly into the driver's field of view, implementing object detection, recognition, and localization entirely on the local AR head-up display (AR-HUD). This design reduces cognitive load and guides driver attention toward potential hazards. However, state-of-the-art detection models still operate at only 5 to 17 fps, limiting their practicality in real-world driving. Although higher-performance methods exist, they demand greater computational resources than mobile devices can provide.

Anderson et al. [18] pushed the speed–accuracy trade-off further by deliberately employing the lightweight Viola-Jones algorithm instead of more accurate alternatives such as YOLO. While Viola-Jones was state-of-the-art two decades earlier, it remains viable for driving AR due to its speed, achieving 32 fps on Microsoft HoloLens. However, its detection range is limited to approximately one car length, which is well within a distance drivers should already perceive visually.

[1]Microsoft HoloLens, Available at: `https://learn.microsoft.com/en-us/hololens/` (accessed Sep. 19, 2025).

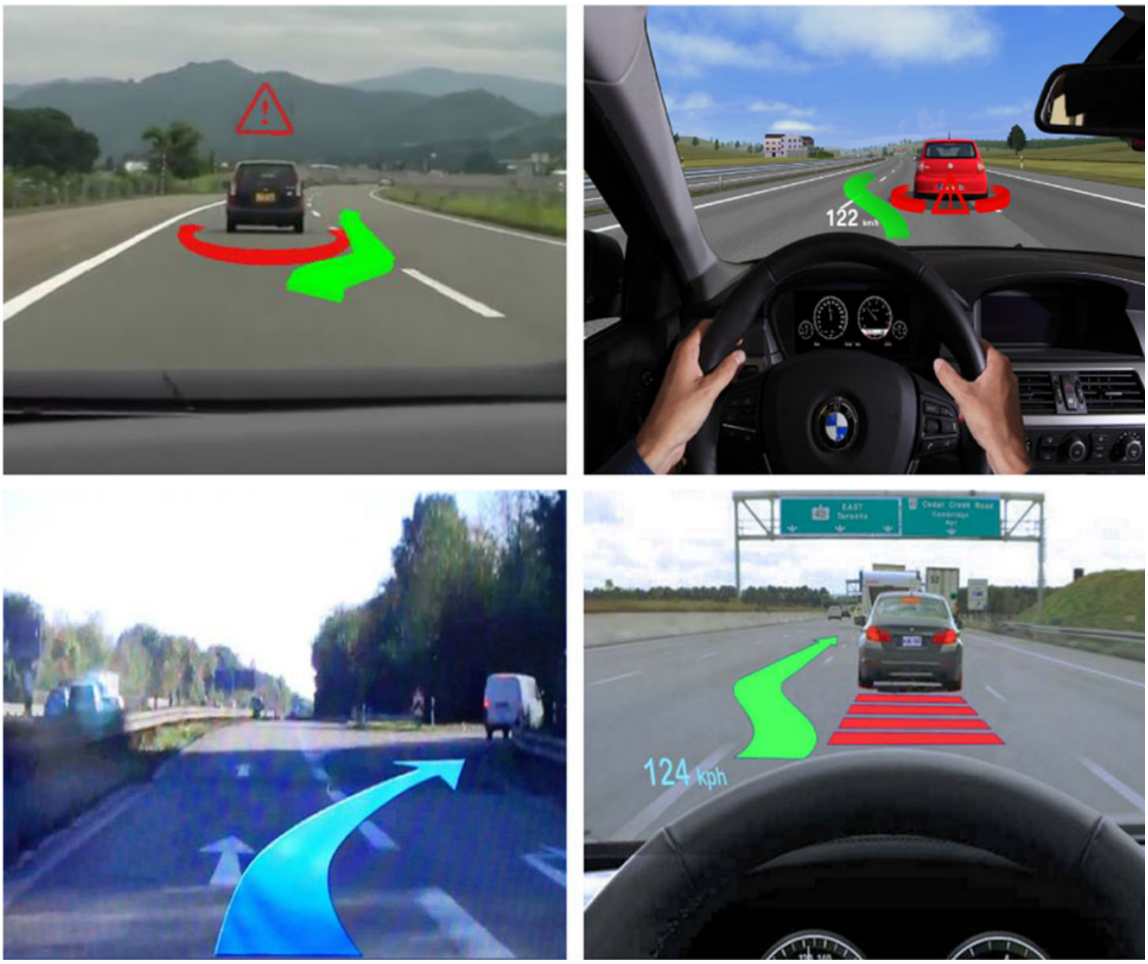


Figure 2: Improving Road Safety with Information Visualization[17].

Figure 3: Driver wearing Microsoft Hololens [18].

More recently, Roh and Lee [19] developed a more sophisticated system on the AR-HUD that integrates human detection (Mask R-CNN), personal mobility recognition (DMNAD), and movement direction estimation (MEBOW, based on HRNet and ResNet). This system addresses an important gap by identifying fast-moving personal mobility users and predicting their future actions. However, the computational intensity of these tasks limited the system to simulation-based validation, which was insufficient for real-world deployment.

Consumer Applications: Retail and Navigation Consumer-facing AR applications in retail and navigation face a different set of constraints, shaped largely by device heterogeneity and unpredictable usage contexts. These systems must support devices ranging from flagship smartphones to entry-level models, while operating in environments that vary from well-connected retail stores to cellular dead zones.

Rao et al. [20] introduced a mobile outdoor AR system for geovisualization that sought to overcome the network fragility of client–server approaches by implementing object detection locally using the lightweight Single Shot MultiBox Detector (SSD). However, even after optimization, the method achieved only 2 FPS on a mobile CPU. This reflects a persistent trade-off between speed and accuracy: further simplifying

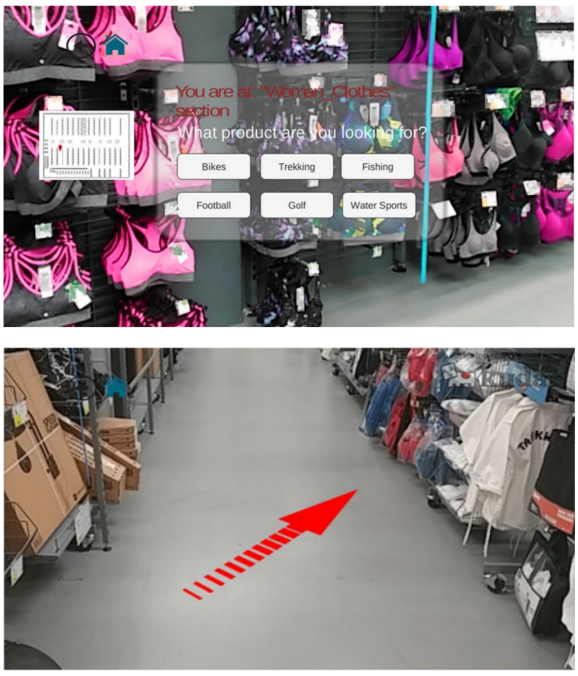


Figure 4: In-store Guidance Example[21].

SSD could improve responsiveness but would likely degrade detection precision, while better performance would require more powerful processors.

Cruz et al. [21] explored retail navigation, combining deep learning with AR to provide precise in-store localization and interactive product previews. The system relied on a client–server architecture with computationally intensive classification running on remote servers. This design avoided overloading mobile hardware but introduced two major limitations. First, it required constant Internet connectivity, making it unsuitable for offline scenarios. Second, it incurred significant latency: although server processing itself took only 0.014 seconds, end-to-end communication added an average of 1.5 seconds across different networks (Wi-Fi, 3G/4G). These delays undermined the seamless user experience required for real-time AR guidance.

Fuchs et al. [22] examined the feasibility of using CV models for product recognition in retail environments, testing different CNN architecture combinations, including Inception/Faster R-CNN, ResNet/Retinanet, and MobileNet/SSD for image classification and object detection tasks. Their experiments revealed significant performance trade-offs: while Retinanet achieved the highest accuracy at 93.8% mAP, it could only deliver 6 fps when executed remotely on Google Cloud. In contrast,

MobileNet running locally achieved 12 fps, demonstrating the tension between model accuracy and real-time performance. The study also revealed substantial variation in data requirements across CV tasks. Image classification achieved 90% accuracy with only six images per product and 95% with 26, whereas object detection required at least 42 instances for 90% mAP and over 100 to approach 95%. These results highlight the markedly higher computational demands of object detection compared to image classification, underscoring the need for AR system design to account for the specific characteristics and resource requirements of each CV task.

Table 2: Summary of Vision-Based AR Systems across Application Domains Sorted by Domain

Article	**CV models/algorithms**	**CV tasks**	**AR devices**	**Platform**	**Domain**
Židek et al. [16]	CNN	Object Detection	Smartphone: Samsung S7, Tablet: Lenovo Tab3	Local	Assembly
Sun et al. [14]	DNN	Pose Estimation	AR-HMD: Microsoft Hololens	Server	Manufacturing
Raj et al. [15]	Object Detection: YOLOv8, Object Tracking: ByteTrack	Object Detection, Object Tracking	AR-HMD: Microsoft HoloLens 2	Server	Assembly
Abdi and Meddeb [17]	CNN	Object Detection, Recognition and Localization	AR-HUD	Local	Driving
Anderson et al. [18]	Object Detection: Viola-Jones algorithm, Lane Detection: Custom algorithm based on Kaur et al.'s proposal [23]	Object Detection, Lane Detection	AR-HMD: Microsoft Hololens	Local	Driving
Roh and Lee [19]	Object Detection: Mask R-CNN, Pose Estimation: MEBOW, Action Recognition: DMNAD	Object Detection, Action Recognition, Pose Estimation	AR-HUD	Local	Driving
Rao et al. [20]	SSD	Object Detection	Smartphone	Local	Geovisualization for navigation

Continued on next page

Table 2 – *Continued from previous page*

Article	**CV models/algorithms**	**CV tasks**	**AR devices**	**Platform**	**Domain**
Cruz et al. [21]	CNN	Scene Classification, User Localization	Smartphone	Server	Shopping navigation
Fuchs et al. [22]	Image Classification: Inception Resnet V2, Object Detection: Focal Pyramid Networks (Retinanet)	Image Classification, Object Detection	AR-HMD: Microsoft Hololens, Smartphone: OnePlus 6t	Dual-mode	Retail

2.1.3 Fundamental Trade-offs

The diverse applications examined above converge on three fundamental challenges that define the design space of vision-based AR systems, each creating cascading constraints that force suboptimal compromises.

Computational complexity versus device capability. As demonstrated in above cases, advanced CV tasks usually demand resources far beyond the limits of mobile hardware. For instance, image segmentation, as a pixel-level classification task, involves processing millions of pixels in megapixel images, making dense prediction highly computationally expensive. While pixel-wise image segmentation has seen significant progress, the substantial computational complexity involved in dense prediction remains a major hurdle for its practical deployment [11]. This mismatch forces systems to either sacrifice accuracy with simplified local models or rely on remote processing at the cost of latency.

Latency sensitivity versus network limitations. Offloading provides access to powerful computational resources, but network transmission introduces delays that undermine the seamless experience of AR. The offloading pipeline, including data transmission, remote inference and result retrieval, is the primary source of this latency, and it is further exacerbated by network instability or congestion. AR applications are particularly latency-sensitive: even minor delays between user movement and system response can disrupt immersion or induce motion sickness. Guan et al. [24] suggest that end-to-end latency should remain around 33 ms to ensure optimal user experience. Moreover, many CV tasks themselves depend on real-time input, making them especially vulnerable to latency introduced during offloading.

High energy consumption versus mobile power supply. Energy remains one of the most critical yet often underestimated constraints in AR systems. Continuous execution of CV tasks rapidly drains device batteries, and current mobile power supplies are insufficient to sustain full-day usage. Ren et al. [25] report sobering figures: Google Glass[2] operates for less than one hour when recording video, Microsoft HoloLens lasts only two to three hours in typical use, and Recon Jet achieves four hours under conservative settings.

These challenges are further compounded by the static nature of current optimization approaches. Manufacturing systems typically offload CV tasks to remote

[2]Google Glass, Available at: `https://www.google.com/glass/photography/` (accessed Sep. 19, 2025).

servers, driving systems adopt lightweight models at the expense of accuracy, and consumer applications rely on separate frameworks tailored to specific scenarios. These rigid approaches prevent systems from dynamically navigating trade-offs in response to real-time conditions. Overcoming these limitations requires mechanisms that can sense the operational context and adapt execution strategies accordingly. This insight motivates the exploration of context-aware computing, discussed in the following section, which seeks to transform static trade-offs into dynamic optimizations.

2.2 Context-aware Strategies for AR Systems

2.2.1 Overview of Context-Aware Computing

Context-aware computing, as defined by Abowd et al. [5], refers to systems that utilize "any information that characterizes the situation of an entity." Such systems operate through three core processes: sensing contextual information, reasoning over it, and dynamically adapting system behavior. This paradigm is especially relevant for mobile AR systems, where HMDs or smartphones provide rich sensor data, yet face stringent latency and energy constraints in dynamic environments. Context-aware approaches enable them to move beyond static optimization and achieve adaptive trade-offs responsive to real-world conditions.

2.2.2 Key Contextual Information in AR Systems

The effectiveness of context-aware computing in AR systems depends critically on identifying which contextual information to capture and utilize. Mobile AR systems, with their rich sensor arrays, generate diverse contextual data that must be systematically classified to enable effective adaptation strategies.

Building upon Grubert et al.'s [26] comprehensive taxonomy, contextual information in AR systems can be organized into three fundamental domains: human factors, environmental factors, and system factors. Each domain encompasses distinct subcategories.

Human Factors encompass both personal and social dimensions of user context. Personal factors include physiological states (anatomy, age, impairments), perceptual and cognitive capabilities, affective states, preferences, and user activities or actions. Social factors extend beyond individual users to consider social networks and places. For instance, Xi et al. [27] demonstrate a sophisticated multimodal context integration on HoloLens 2, employing advanced AI models to interpret complex intentions

by fusing voice commands and pointing gestures. This system demonstrates how user contextm such as personal actions and interaction contexts, can be employed to create more intuitive AR experiences.

Environmental Factors describe the external physical and technical surroundings where AR interaction occurs. Physical factors include raw measurements (temperature, time, location) and derived factors (spatial configuration, object presence/absence, motion patterns). Digital factors encompass the characteristics of available digital information including type, quality, and quantity. Infrastructure factors address technical elements like network connectivity that support but are not inherently part of the AR system. Cao et al. [1] leverages environmental context factors, primarily air quality (PM2.5 prediction), along with wind speed, humidity, and temperature, to develop a context-aware AR navigation system (ConAR). This system uses 5G edge computing to process this data, providing user-personalized routing recommendations on HoloLens 2 to help users avoid areas with unhealthy air quality during navigation.

System Factors concern the technical characteristics and capabilities of the AR platform itself. This includes computational resources (CPU/GPU capacity, memory, battery), input/output modalities (display types, touch interfaces, speech recognition), and sensor configurations (cameras, IMUs, GPS). These factors fundamentally constrain what adaptations are feasible. Ran et al. [28] demonstrate comprehensive system-aware optimization by monitoring multiple device parameters (e.g., device battery levels, available computational resources) to make three-level adaptation decisions. This multidimensional adaptation illustrates how system context can drive sophisticated optimization approaches that maintain AR performance under varying resource constraints.

2.2.3 Context-aware Adaptation Approaches in AR Systems

Building on the discussion of contextual information, an important question is how AR systems leverage such information to enhance system performance. This section reviews representative context-aware systems and identifies six adaptation approaches, targeting four key optimization objectives.

Latency Reduction Approaches:

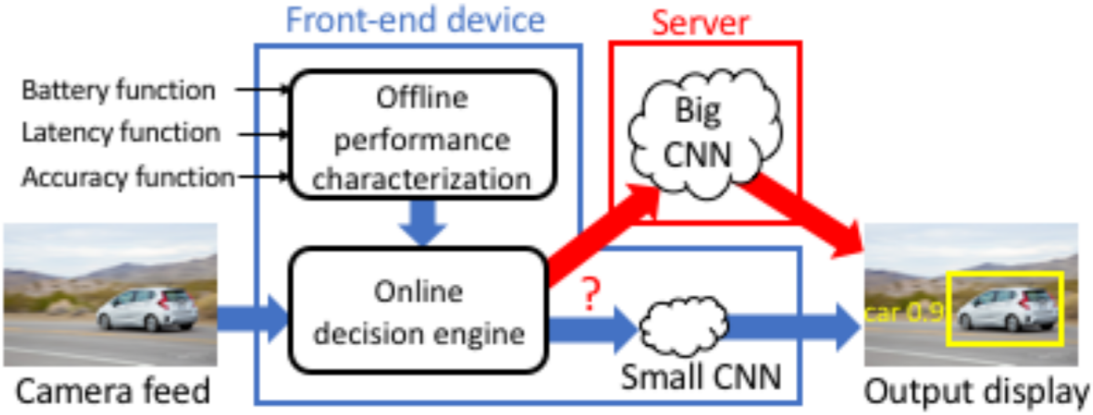


Figure 5: DeepDecision System Workflow[28].

Dynamic offloading has been the most prominent approach to reducing latency in AR systems. CloudAR [3] introduced a hybrid but static offloading scheme, permanently assigning image recognition tasks to cloud servers while maintaining Six Degrees of Freedom (6DoF) tracking tasks locally. Although effective under stable connections, this rigid architecture was vulnerable to network fluctuations. DeepDecision [28] addressed this limitation by introducing an adaptive mechanism that monitors real-time network conditions and battery state to flexibly switch between local lightweight models and remote complex CNN, achieving 15 FPS with an average latency of approximately 66 ms. Unlike static partitioning, dynamic offloading reduces latency by making real-time decisions on whether to process tasks locally or in the cloud, depending on network or device status. Shortly thereafter, MARVEL [29] combined the strengths of both approaches: like CloudAR, it decomposed CV tasks between local and cloud processing, with local devices performing inertial tracking and optical flow while the cloud selectively provided 6DoF image localization; and similar to DeepDecision, it dynamically offloaded only when local inertial and visual data were inconsistent, offloading a subset of images to the cloud for calibration. Collectively, these systems illustrate the evolution of AR workload management from rigid static partitioning to adaptive, context-aware, and task-specific offloading strategies.

Data filtering provides another pathway to latency reduction. By intelligently narrowing the data or task space before processing, filtering reduces the amount of computation and communication required, thereby shortening response times. Smart Lens [2] applied context-based filtering by predicting likely targets using user loca-

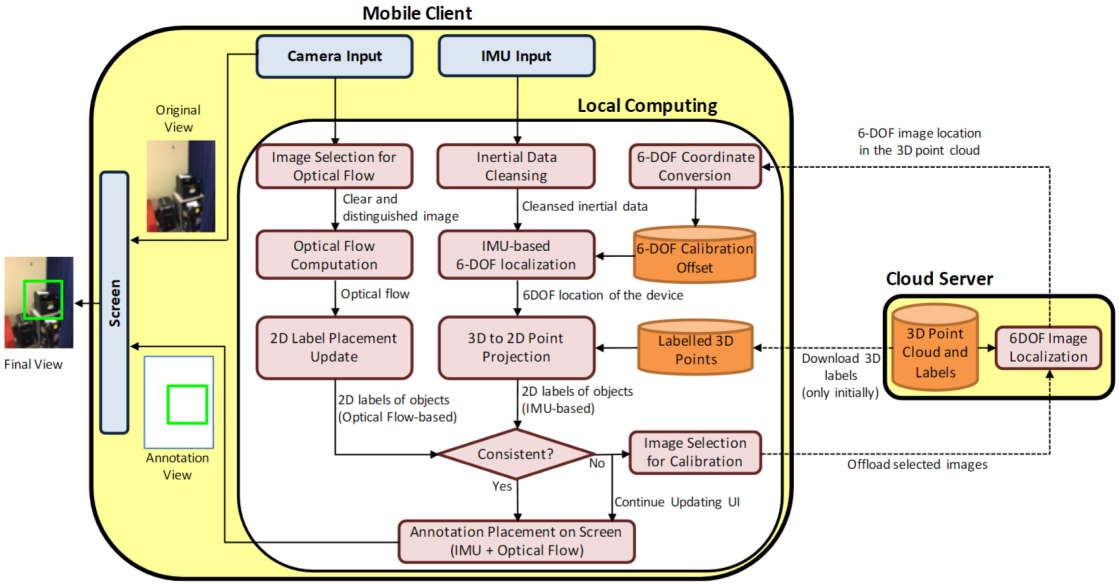


Figure 6: Marvel System Workflow [29].

tion and time, thereby downloading only a relevant subset of the dataset from the server. This reduced the number of candidate objects for recognition and improved response latency. In contrast, OverLay [30] adopted a location-independent filtering approach to avoid reliance on GPS, which are often unreliable indoors. Instead, it used gyroscope data to track user rotation and temporal patterns of object observations, constructing a spatiotemporal graph to predict likely upcoming objects. This reduced the candidate image set from 100 annotations to 10, bringing median end-to-end latency down to 180 ms. Later, Jain et al. [31] proposed another system (VisualPrint) which took a different approach than the previous two cases by filtering visual data at the source rather than reducing the server's search space. It identified high-entropy keypoints that were most likely to produce unique matches, reducing the keypoints of the scene from thousands to just 200–500. By sending only these most distinctive visual fingerprints to the cloud, it reduced network bandwidth consumption by an order of magnitude, requiring only 51.2 KB compared to 523 KB for whole frame uploads.

Adaptive data quality reduces latency by dynamically adjusting data fidelity to match real-time constraints. DeepDecision [28] exemplifies this approach by incorporating multi-level data quality adjustments (e.g., resolution, compression, and frame rate) based on bandwidth and device status. During offloading, it adapts frame resolution and video compression to current network conditions, choosing lower target

bitrates to reduce transmission delay when bandwidth is limited. Through this adaptive approach, DeepDecision consistently outperforms fixed-quality baselines that cannot respond to changing context. Liu et al. [32] extended this idea with Dynamic RoI Encoding (DRE), which applies different compression levels across different regions of the same frame. The system leverages the CNN's region proposal network to identify candidate RoIs. These regions are encoded with near-lossless quality to preserve recognition accuracy, while background areas are compressed more aggressively to save bandwidth. In contrast to DeepDecision's uniformed image-level adjustments, DRE enables spatial selectivity within a single image that determines which image regions deserve higher quality encoding. This enables more intelligent allocation of bandwidth that reduces latency while maintaining high detection accuracy.

Energy Conservation Approaches:

Dynamic model switching alternates between models of different complexity to balance accuracy and resource consumption. MARLIN [33] adopted an event-driven pipeline for object tracking tasks, replacing energy consuming continuous DNN inference with selective DNN activation triggered by significant scene changes. This system switches between costly DNN detection and lightweight incremental tracking, reducing DNN invocations by 73% and lowering power consumption from 1,724 mW to as little as 319 mW. However, its binary mechanism often overlooked gradual scene change.

Adaptive Quality Scaling dynamically adjusts AR rendering to balance visual fidelity and energy consumption. Chou et al. [34] proposed an approach to first detect six user statuses and then dynamically adjust both physical view parameters (brightness/frame rate) and virtual object rendering rates based on these six user attention levels. This framework achieves up to 10% power savings on OLED displays. In contrast, Sahu et al. [35] developed a more sophisticated AI-driven framework that adjusts the quality of AR rendering based on battery status and network conditions. Their system employs Deep Q-Networks (DQN) for reinforcement learning and information theory-based adaptive mechanisms to compute optimal quality settings, reducing energy consumption by 30% while maintaining task success rates above 90%. The key distinction is that Chou et al.'s [34] method relies on predefined rules based only on user context, whereas Sahu et al.'s [35] approach enables AI-driven, learning-based adaptation to broader system and environmental contexts.

Accuracy Enhancement Approaches:

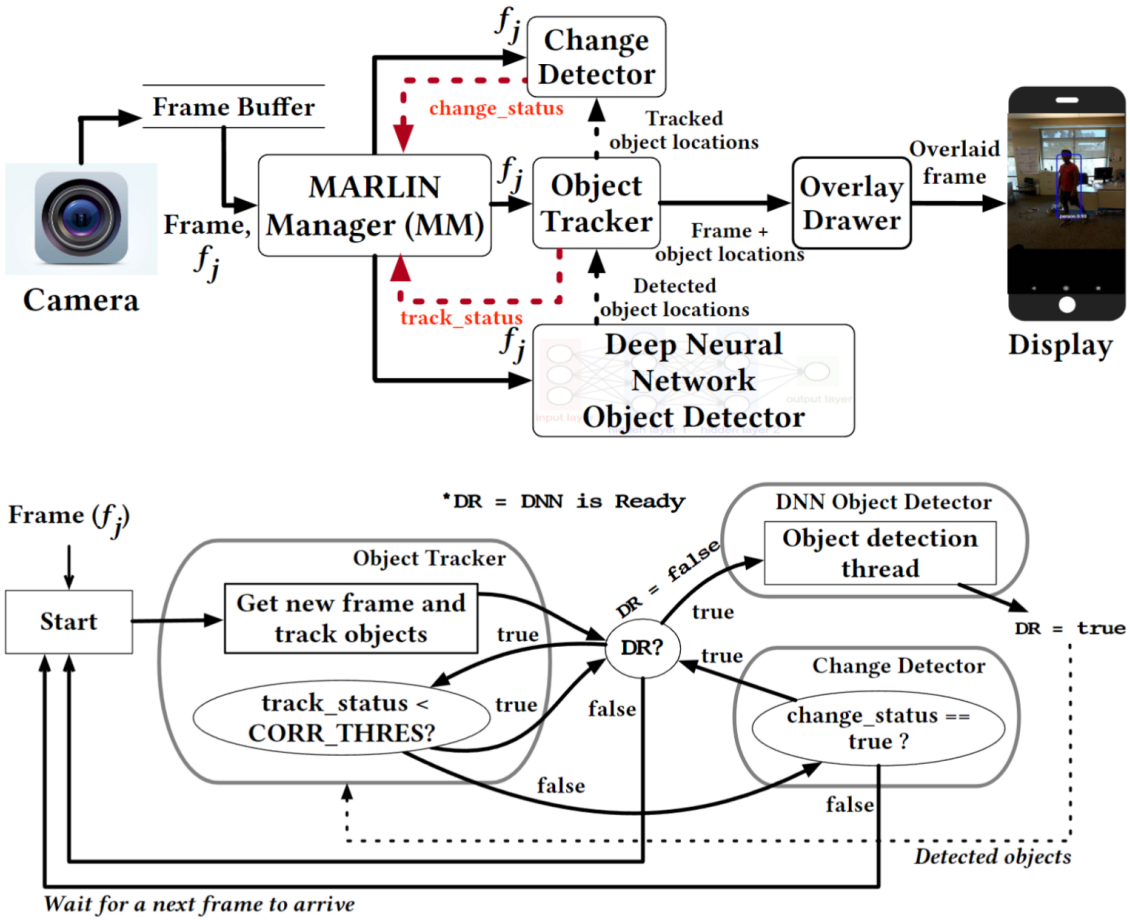


Figure 7: Above: MARLIN System Workflow; Below: MM Decision Flow[33].

Data caching enhances model accuracy in mobile AR by ensuring continuity of recognition results. Glimpse [36] exemplifies temporal caching at the device level: when a frame is offloaded for server-side recognition, the device stores all subsequent intermediate frames in an active cache. Once the (possibly stale) server result returns, the system tracks the recognized object forward through cached frames via optical flow, catching up to the latest view and preserving accuracy despite multi-frame delays. In contrast, Ren et al. [25] introduced spatial caching in a hierarchical edge–cloud architecture. A lightweight edge cache deployed at base stations stores frequently accessed 3D models, annotations, and recent video streams, while a cloud database maintains less popular content. Incoming data is first processed at the edge, with unmatched queries escalated to the cloud, and a popularity-based policy ensures efficient distribution across layers. Thus, Glimpse demonstrates local caching for temporal coherence, whereas Ren's design emphasizes distributed caching across edge and cloud for spatial efficiency.

In summary, the six adaptation approaches reviewed above directly mitigate the key challenges identified in Section 2.1.3. Latency is alleviated through dynamic offloading, data filtering, and adaptive data quality, which reduce transmission delays and computation time. Energy consumption is addressed by dynamic model switching and Adaptive Quality Scaling, enabling efficient use of device resources without compromising usability. Accuracy is enhanced through data caching, which maintains recognition continuity despite network delays. Collectively, these strategies illustrate how context-aware computing can balance computational complexity, energy efficiency, latency sensitivity, and inference accuracy to improve the overall performance of distributed vision-based AR systems.

3 Summary of the State of the Art

The development of context-aware AR systems reveals clear progress but also persistent limitations that define current research frontiers. Three trends illustrate recent advances. First, multimodal fusion has become a critical capability, with systems increasingly combining diverse contextual information for richer situational reasoning. For example, CARING-AI [37] leverages generative AI to integrate contextual information from multimodal inputs—users can provide context through voice, text, or direct environmental scanning. This approach achieves a code-less and MoCap-free workflow for authoring AR instructions. Second, the field has progressed from isolated optimizations to multi-strategy frameworks, where several adaptation approaches are orchestrated simultaneously. Sahu et al. [35] exemplify this trend by integrating reinforcement learning–based offloading with Adaptive Quality Scaling in

their system. Third, recent work shows a paradigm reversal: deep learning, once the primary computational burden requiring context-aware management, now serves as the optimization mechanism itself. Mustafa et al. [38] comprehensively demonstrate this shift in their survey of DNN-based computation offloading, showing how neural networks are increasingly employed not just as the workload to be managed, but as intelligent decision-makers that predict optimal offloading strategies, estimate channel conditions, and dynamically allocate resources based on learned patterns from complex MEC environments. Together, these trends mark a clear shift from single-modal, isolated techniques toward more integrated, intelligent, and AI-driven context-aware AR systems.

Despite these advances, critical limitations remain. Most systems still optimize only a single performance dimension, none of them cover the full range of adaptation approaches and optimization objectives identified in Section 2.2.3. A further challenge lies in decision conflicts: while DeepDecision [28] demonstrates simultaneous optimization of latency, energy, and accuracy, it lacks mechanisms to resolve trade-offs when objectives compete (e.g., when reducing latency lowers accuracy). This is partly due to the inconsistent evaluation metrics: some studies report latency, others measure energy savings or recognition accuracy. No work provides a comprehensive assessment across latency, energy, accuracy simultaneously, limiting comparability and systematic progress. Moreover, user agency is limited: current systems impose developer-defined strategies, leaving users little control over trade-offs. For example, Mascot [39] allows only developers to configure domain-specific context, QoE requirements, and optimization objectives at the design stage, but it still lacks mechanisms for user-defined customization.

In summary, despite advances in multimodal fusion, multi-strategy adaptation, and AI-driven optimization, context-aware AR research remains fragmented. Most systems optimize only one performance dimension, lack mechanisms to resolve conflicts between competing objectives, and use inconsistent evaluation metrics, limiting comparability. Moreover, user agency is minimal, with strategies largely imposed by developers. These gaps point to the need for holistic frameworks that integrate multiple strategies, enable conflict resolution, standardize evaluation, and incorporate user-driven customization to achieve robust and adaptable AR experiences.

4 Research Project Plan

This project aims to address the limitations identified in the current state of the art by designing and evaluating a context-aware computing framework for distributed vision-based AR systems. The proposed framework will explicitly model multiple

forms of contextual information, develop mechanisms for multi-context reasoning, and implement adaptive optimization approaches to improve system performance across latency, resource consumption, and accuracy.

4.1 Research Question and Aims

The overarching research question (RQ) guiding this project is:

RQ: In distributed vision-based AR systems, how can context-aware computing be used to improve system performance (i.e., latency, resource consumption, and accuracy)?

This research question can be decomposed into four sub-questions:

RQ1. What contextual attributes should be considered to effectively represent device, network, user, and environmental states?

RQ2. How can multiple contextual attributes be aggregated and reasoned about the current context related to system performance?

RQ3. How can the output from RQ1 and RQ2 be used to develop a context-aware approach for vision-based AR systems to improve system performance?

RQ4. How can we evaluate the proposed context-aware approach?

4.2 Research Design and Method

4.2.1 Research Design

The project will proceed in four stages:

1. **Literature Review**
   A structured review of vision-based AR and context-aware computing literature summarizes the selection of contextual attributes and identifies existing adaptive optimization approaches, as well as highlights unresolved research gaps.

2. **Design of the context-aware approach for the vision-based AR systems**
   The proposed architecture consists of three main components. The Apple Vision Pro[3] serves as the AR device, capturing images and displaying detection results with bounding boxes. The Context-aware Approach module forms the

[3]Apple Vision Pro, Available at: `https://www.apple.com/apple-vision-pro/` (accessed Sep. 19, 2025).

system's core, comprising context collection, context inference, and context-aware adaptation. The Cloud Server handles computationally intensive object detection tasks when offloading is deemed beneficial based on contextual conditions.

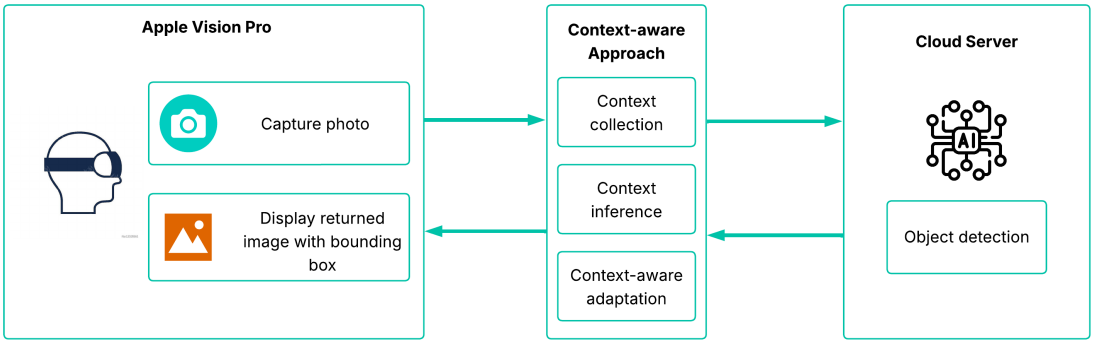


Figure 8: Architecture of the Distributed Context-Aware AR Prototype

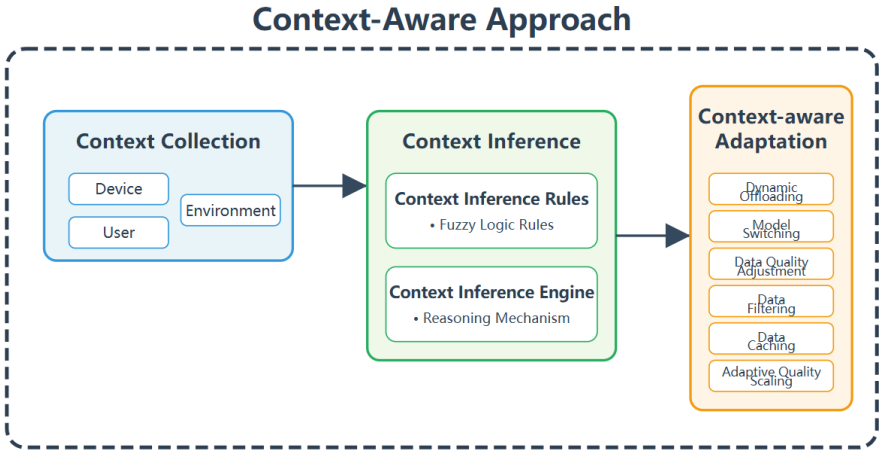


Figure 9: Proposed Context-Aware Approach for Vision-Based AR Systems

As shown in Figure 9, the proposed context-aware approach for vision-based AR systems consists of three main components working in sequence to optimize system performance:

- **Context Collection:** This component is responsible for gathering real-time contextual information from a multi-layered model (device, user, environment) within the AR ecosystem.
- **Context Inference:** This component processes the collected contextual data to understand the current operational context and predict effective adaptation approaches. It comprises two sub-components:
  - **Context Inference Rules**: Implements a fuzzy logic-based rule system that defines relationships between contextual attributes and performance outcomes. These rules encode expert knowledge about how different combinations of context factors impact latency, energy consumption, and accuracy. For example, rules might specify that "IF network bandwidth is HIGH AND battery level is LOW, THEN prioritize cloud offloading with compressed data."
  - **Context Inference Engine**: The reasoning mechanism that processes contextual inputs through the defined rules to generate adaptation decisions. It performs multi-objective reasoning to balance competing goals (latency vs. accuracy vs. energy) and resolves conflicts when objectives compete.
- **Context-aware Adaptation:** This component executes the adaptation strategies determined by the inference engine, implementing specific adaptation approaches to improve system performance.

These components work together in a continuous feedback loop, where the Context Collection component monitors the effects of adaptations on performance metrics (latency, energy consumption, accuracy), enabling the system to refine its strategies over time and maintain performance across varying operational conditions.

3. **Prototype Development**
   A distributed AR prototype will be implemented on Apple Vision Pro using Swift[4], SwiftUI[5], RealityKit[6], and ARKit, supported by a cloud backend for intensive computation. Object detection will be realized through Ground-

[4]Apple Swift, Available at: `https://developer.apple.com/swift/` (accessed Sep. 21, 2025).

[5]Apple SwiftUI, Available at: `https://developer.apple.com/swiftui/` (accessed Sep. 21, 2025).

[6]Apple RealityKit, Available at: `https://developer.apple.com/documentation/realitykit` (accessed Sep. 21, 2025).

ingDINO[7], an open-set detector capable of aligning visual features with natural language prompts, enabling flexible recognition beyond fixed categories. The model will be accessed via Hugging Face[8], which hosts pretrained checkpoints and inference APIs. GroundingDINO will run in the cloud, with AR devices sending frames and receiving detection results in real time.

4. **Evaluation**
   Quantitative experiments will assess the performance of this prototype under different contexts.

4.2.2 Research Method

The research will adopt a quantitative evaluation methodology. Evaluation will be based on three main categories of metrics:

- **Accuracy metrics**: mean Average Precision (mAP), recall, intersection-over-union (IoU), and F1-score will be used to measure the accuracy of vision-based tasks.
- **Time metrics**: end-to-end latency, inference time, data transfer time, and HMD rendering time will capture the responsiveness of the system across local and cloud execution.
- **Computing resource metrics**: battery level and CPU/GPU/memory usage will be tracked to evaluate system resource consumption.

These metrics will be compared across different adaptation approaches (e.g., dynamic offloading, model switching and data quality adjustment) to assess their impact under varying contexts compared with static baselines.

4.3 Data Collection

Three main types of data will be collected:

1. **Real-world image dataset**
   To evaluate object detection performance, a suitable real-world dataset will be selected for testing with GroundingDINO. This dataset will be pre-processed

[7]IDEA-Research/GroundingDINO, Available at: `https://github.com/IDEA-Research/GroundingDINO` (accessed Sep. 21, 2025).

[8]Hugging Face, Available at: `https://huggingface.co/` (accessed Sep. 21, 2025).

into at least three different image resolutions, which will serve as contextual variables to analyze how model performance and system behavior vary with input quality.

2. **Context information**
   Contextual data will be collected from three dimensions:

   - *Device context:* battery level, CPU usage, GPU usage, and memory consumption.
   - *Environment context:* time, lighting conditions, network type, network latency, and network throughput.
   - *User context:* user-defined preferences such as acceptable latency or accuracy thresholds.

3. **System performance metrics**
   System-level logs (e.g., end-to-end latency, inference time, data transfer time, HMD rendering time, resource utilization) and model accuracy metrics (e.g., mAP, recall, IoU, F1-score) will be collected for quantitative analysis, as described in Section 4.2.2.

4.4 Ethics and Data Privacy

This research does not involve human participants, personal data, or sensitive information. All contextual and performance data will be generated by devices in controlled environments. Thus, no ethical approval is required, and privacy concerns are minimal. Nevertheless, best practices for secure data storage and reproducibility will be observed. For real-world testing, care will be taken to avoid recording identifiable pedestrians or private content.

4.5 Research Timeline

Table 3: Research Timeline

Period	Tasks and Milestones
Semester 2, 2025	
Weeks 1–4	• Conduct literature review on AR, CV, and context-aware computing • Determine the research question • Identify CV models used and CV tasks executed • Identify contextual attributes and performance metrics
Week 5	Determine the structure of this literature review
Week 6	Draft Section 1 and Section 2.1
Week 7	Draft Section 2.2, Section 3 and Section 4
Week 8	Complete Section 5 and submit the final version
Weeks 9–13	• Learn Apple Vision Pro development • Learn CV model deployment • Design context inference rules
Week 14	Interim presentation
Summer Break	
	Prototype development
Semester 1, 2026	
Weeks 1–3	Refine and evaluate the prototype
Weeks 4–8	• Analyse collected data • Complete the draft thesis
Weeks 9–12	• Revise thesis • Submit thesis • Prepare for the final presentation

5 Conclusion

This literature review has systematically examined the integration of context-aware computing with vision-based AR systems, revealing both significant progress and

persistent challenges. The analysis demonstrates that while vision-based AR systems face fundamental trade-offs between computational complexity, latency sensitivity, and energy consumption, context-aware computing offers promising solutions through adaptive optimization strategies. Six key adaptation approaches have been identified: dynamic offloading, data filtering, adaptive data quality, dynamic model switching, Adaptive Quality Scaling, and data caching. However, current implementations remain fragmented, with most systems optimizing single performance dimensions and lacking comprehensive evaluation frameworks.

To overcome this gap, the project will comprehensively model diverse contextual attributes, develop reasoning mechanisms to manage multi-objective trade-offs, and implement adaptive approaches for distributed AR. By moving from isolated optimizations to an integrated framework, this project contributes a practical framework for robust, adaptive AR systems. Ultimately, it advances the vision of AR systems that are both context-sensitive and user-centric, providing a foundation for future applications in real-world, resource-constrained environments.

References

[1] J. Cao, X. Liu, X. Su, S. Tarkoma, and P. Hui, "Context-aware augmented reality with 5g edge," in *2021 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2021, pp. 1–6.

[2] N. Z. Naqvi, K. Moens, A. Ramakrishnan, D. Preuveneers, D. Hughes, and Y. Berbers, "To cloud or not to cloud: A context-aware deployment perspective of augmented reality mobile applications," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, pp. 555–562.

[3] W. Zhang, S. Lin, F. H. Bijarbooneh, H. F. Cheng, and P. Hui, "Cloudar: A cloud-based framework for mobile augmented reality," in *Proceedings of the on Thematic Workshops of ACM Multimedia 2017*, 2017, pp. 194–200.

[4] A. Younis, B. Qiu, and D. Pompili, "Latency-aware hybrid edge cloud framework for mobile augmented reality applications," in *2020 17th Annual IEEE international conference on sensing, communication, and networking (SECON)*, IEEE, 2020, pp. 1–9.

[5] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *International symposium on handheld and ubiquitous computing*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 304–307.

[6] A. Seeliger, R. P. Weibel, and S. Feuerriegel, "Context-adaptive visual cues for safe navigation in augmented reality using machine learning," *International Journal of Human–Computer Interaction*, vol. 40, no. 3, pp. 761–781, 2024.

[7] F. Farahbakhsh, A. Shahidinejad, and M. Ghobaei-Arani, "Context-aware computation offloading for mobile edge computing," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 5, pp. 5123–5135, 2023.

[8] S. Minaee, X. Liang, and S. Yan, "Modern augmented reality: Applications, trends, and future directions," *arXiv preprint arXiv:2202.09450*, 2022.

[9] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing*, vol. 187, pp. 27–48, 2016.

[10] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, no. 1, p. 7 068 349, 2018.

[11] X. Feng, Y. Jiang, X. Yang, M. Du, and X. Li, "Computer vision algorithms and hardware implementations: A survey," *Integration*, vol. 69, pp. 309–320, 2019.

[12] N. Le, V. S. Rathour, K. Yamazaki, K. Luu, and M. Savvides, "Deep reinforcement learning in computer vision: A comprehensive survey," *Artificial Intelligence Review*, vol. 55, no. 4, pp. 2733–2819, 2022.

[13] R. Archana and P. E. Jeevaraj, "Deep learning models for digital image processing: A review," *Artificial Intelligence Review*, vol. 57, no. 1, p. 11, 2024.

[14] Y. Sun et al., "Towards industrial iot-ar systems using deep learning-based object pose estimation," in *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, IEEE, 2019, pp. 1–8.

[15] S. Raj, B. Karmakar, G. Kumar, A. Mukhopadhyay, R. Chandrahas, and P. Biswas, "Comparing computer vision models for low resource dataset to develop a mixed reality based manual assembly assistant," *Discover Robotics*, vol. 1, no. 1, p. 5, 2025.

[16] K. Židek, A. Hosovsky, J. Piteľ, and S. Bednár, "Recognition of assembly parts by convolutional neural networks," in *Advances in Manufacturing Engineering and Materials: Proceedings of the International Conference on Manufacturing Engineering and Materials (ICMEM 2018), 18–22 June, 2018, Nový Smokovec, Slovakia*, Cham: Springer International Publishing, 2018, pp. 281–289.

[17] L. Abdi and A. Meddeb, "Driver information system: A combination of augmented reality, deep learning and vehicular ad-hoc networks," *Multimedia Tools and Applications*, vol. 77, no. 12, pp. 14 673–14 703, 2018.

[18] R. Anderson, J. Toledo, and H. ElAarag, "Feasibility study on the utilization of microsoft hololens to increase driving conditions awareness," in *2019 SoutheastCon*, IEEE, 2019, pp. 1–8.

[19] D. H. Roh and J. Y. Lee, "Augmented reality-based navigation using deep learning-based pedestrian and personal mobility user recognition—a comparative evaluation for driving assistance," *IEEE Access*, vol. 11, pp. 62 200–62 211, 2023.

[20] J. Rao, Y. Qiao, F. Ren, J. Wang, and Q. Du, "A mobile outdoor augmented reality method combining deep learning object detection and spatial relationships for geovisualization," *Sensors*, vol. 17, no. 9, p. 1951, 2017.

[21] E. Cruz et al., "An augmented reality application for improving shopping experience in large retail stores," *Virtual Reality*, vol. 23, no. 3, pp. 281–291, 2019.

[22] K. Fuchs, T. Grundmann, and E. Fleisch, "Towards identification of packaged products via computer vision: Convolutional neural networks for object detection and image classification in retail environments," in *Proceedings of the 9th International Conference on the Internet of Things*, 2019, pp. 1–8.

[23] G. Kaur and D. Kumar, "Lane detection techniques: A review," *International Journal of Computer Applications*, vol. 112, no. 10, 2015.

[24] Y. Guan, X. Hou, N. Wu, B. Han, and T. Han, "Deepmix: Mobility-aware, lightweight, and hybrid 3d object detection for headsets," in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, 2022, pp. 28–41.

[25] J. Ren, Y. He, G. Huang, G. Yu, Y. Cai, and Z. Zhang, "An edge-computing based architecture for mobile augmented reality," *IEEE Network*, vol. 33, no. 4, pp. 162–169, 2019.

[26] J. Grubert, T. Langlotz, S. Zollmann, and H. Regenbrecht, "Towards pervasive augmented reality: Context-awareness in augmented reality," *IEEE transactions on visualization and computer graphics*, vol. 23, no. 6, pp. 1706–1724, 2016.

[27] M. Xi, M. Perera, S. Anderson, and M. Adcock, "Towards situated imaging," in *2024 IEEE International Conference on Artificial Intelligence and eXtended and Virtual Reality (AIxVR)*, IEEE, 2024, pp. 85–89.

[28] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018-IEEE conference on computer communications*, IEEE, 2018, pp. 1421–1429.

[29] 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 the 16th ACM Conference on Embedded Networked Sensor Systems*, 2018, pp. 292–304.

[30] P. Jain, J. Manweiler, and R. Roy Choudhury, "Overlay: Practical mobile augmented reality," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, 2015, pp. 331–344.

[31] P. Jain, J. Manweiler, and R. Roy Choudhury, "Low bandwidth offload for mobile ar," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016, pp. 237–251.

[32] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *The 25th annual international conference on mobile computing and networking*, 2019, pp. 1–16.

[33] K. Apicharttrisorn, X. Ran, J. Chen, S. V. Krishnamurthy, and A. K. Roy-Chowdhury, "Frugal following: Power thrifty object detection and tracking for mobile augmented reality," in *Proceedings of the 17th conference on embedded networked sensor systems*, 2019, pp. 96–109.

[34] P. H. Chou, S. E. Wei, and C. H. Lin, "Content-based power-saving design for augmented reality applications on mobile devices," in *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*, 2024, pp. 1–6.

[35] D. Sahu et al., "Edge assisted energy optimization for mobile ar applications for enhanced battery life and performance," *Scientific Reports*, vol. 15, no. 1, p. 10 034, 2025.

[36] T. Y. H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of the 13th ACM conference on embedded networked sensor systems*, 2015, pp. 155–168.

[37] J. Shi et al., "Caring-ai: Towards authoring context-aware augmented reality instruction through generative artificial intelligence," in *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 2025, pp. 1–23.

[38] E. Mustafa et al., "Deep neural networks meet computation offloading in mobile edge networks: Applications, taxonomy, and open issues," *Journal of Network and Computer Applications*, vol. 226, p. 103 886, 2024.

[39] N. Z. Naqvi, J. Devlieghere, D. Preuveneers, and Y. Berbers, "Mascot: Self-adaptive opportunistic offloading for cloud-enabled smart mobile applications with probabilistic graphical models at runtime," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, IEEE, 2016, pp. 5701–5710.

Part 2

The Research Paper

A Context-Aware Computing Framework for Performance Improvement in Distributed Vision-Based Augmented Reality

Shuyang Yan

Abstract

Distributed vision-based augmented reality (AR) systems often offload computationally intensive computer vision tasks to remote servers, but this introduces network-dependent latency that can degrade user experience. This paper presents a context-aware adaptation mechanism for a distributed AR prototype using Apple Vision Pro as the client. The proposed mechanism collects runtime network and device contexts, uses a fuzzy situation inference engine to infer current network situations, and adapts image resolution and network probing interval accordingly. Experiments under nine controlled network scenarios show that the proposed mechanism consistently reduces end-to-end latency compared with a non-context-aware baseline. The improvement mainly comes from reduced client-to-server transfer time, while the additional client-side processing overhead remains relatively small. Runtime results further show that adaptive probe interval reduces unnecessary network probing and improves runtime compared with fixed-frequency probing. These results demonstrate that context-aware adaptation can improve the responsiveness of distributed vision-based AR systems while limiting the additional resource cost.

CCS Concepts

• **Human-centered computing** → **Mixed / augmented reality**; *Ubiquitous and mobile computing systems and tools*; • **Networks** → *Network performance evaluation*; • **Computer systems organization** → *Client-server architectures*; • **Computing methodologies** → Object detection.

Keywords

Context-aware computing, augmented reality, Apple Vision Pro, distributed vision-based AR, fuzzy situation inference, network probing, adaptive resolution, end-to-end latency

1 Introduction

Augmented Reality (AR) overlays digital information onto the physical world and has been increasingly applied in areas such as manufacturing [28], navigation [25], and retail [8]. As AR applications move from simple visual overlays toward more interactive and intelligent experiences, they need to understand not only where to place virtual content, but also what exists in the surrounding environment. This shift has made Artificial Intelligence (AI), especially Computer Vision (CV), an important component of modern AR systems. By analyzing camera input, CV models can support tasks such as object detection [32], object tracking [23], pose estimation [28], and scene understanding [7]. For example, in retail navigation, Cruz et al. [7] combined deep learning-based scene classification and user localization with AR to support in-store navigation and product-related interaction. In this paper, AR systems that integrate such CV capabilities are referred to as vision-based AR systems.

Despite their potential, vision-based AR systems face major performance challenges. Existing AR systems have shown that the computational burden of CV tasks can challenge the real-time processing capability of mobile AR devices [1, 3, 25, 32]. A common solution is to adopt a distributed architecture, where the AR device captures images or video frames and offloads heavy inference tasks to an edge or cloud server [7, 23, 28]. This approach provides access to stronger computational resources, but it also introduces communication overhead. Latency is particularly critical in AR because virtual content must remain aligned with a changing physical environment [30]. Delayed results may reduce overlay accuracy, interrupt interaction, and degrade the user's Quality of Experience (QoE). In addition, mobile AR devices remain constrained by limited battery life, CPU/GPU resources, and memory capacity. Ren et al. [26] highlight the severity of this limitation, reporting short operating times for representative AR devices under demanding workloads. Therefore, distributed vision-based AR systems must balance three competing requirements: low latency, low resource consumption, and acceptable task accuracy.

To address these challenges, this paper proposes a context-aware adaptation mechanism for distributed vision-based AR systems. The proposed system uses an Apple Vision Pro (M2 model)[1] as the AR client to send images, while a remote server performs object detection inference. Instead of relying on fixed system parameters, the client collects runtime contextual information, including network bandwidth, network delay, and battery level. These contexts are interpreted by a fuzzy situation inference engine and used to adapt two key system parameters: image resolution and network probing interval. Image resolution is adapted to reduce transmission overhead and end-to-end latency, while the probing interval is adapted to balance context freshness with monitoring cost. Through this design, the system aims to improve the trade-off among latency, resource consumption, and detection accuracy compared with fixed-parameter baselines.

This work makes three main contributions. First, it designs and implements a distributed vision-based AR prototype on Apple Vision Pro, where the object detection task is offloaded to a remote server. Second, it proposes a context-aware adaptation mechanism that models runtime network and device contexts, infers high-level network situations using fuzzy situation inference, and dynamically adapts image resolution and network probing interval. Third, it evaluates the prototype under nine controlled bandwidth-delay scenarios, showing that the proposed mechanism consistently reduces end-to-end latency mainly by reducing client-to-server transfer time, while adaptive probing helps limit the runtime overhead of context monitoring.

[1] Apple Vision Pro technical specifications. Available at: https://support.apple.com/en-au/117810 (accessed May 31, 2026).

2 Related Work

Context-aware computing has been widely studied as a way to enable systems to adapt their behavior according to changing operational conditions. Abowd et al. [2] define context as "any information that can be used to characterize the situation of an entity." Based on this definition, a context-aware system does not simply collect raw data; rather, it senses relevant contextual attributes, reasons about their meaning, and adapts system behavior accordingly.

AR systems are well suited to context-aware computing because they continuously sense both the physical environment and their own operating state [27]. Through cameras, inertial sensors, and eye-tracking modules, AR devices can capture diverse user, device, and environmental contexts. Meanwhile, distributed AR applications are highly sensitive to runtime changes, such as network quality, battery level, and computational load. Therefore, context-aware computing provides a suitable mechanism for adapting system behavior according to current conditions, rather than relying on fixed configurations.

Despite these advantages, existing context-aware AR systems still show several limitations. First, some systems rely mainly on a single type of low-level context. For example, Liu et al.'s [18] edge-assisted AR object detection system adapts encoding and compression decisions primarily using derived visual information, such as candidate regions of interest (RoIs) from previous CNN inference results, motion vectors, and macroblock types in encoded video frames. These contexts mainly describe frame-level visual changes, rather than broader runtime conditions such as battery level, network conditions, or time and space information.

Second, some systems depend on contexts that are collected and processed before runtime. For instance, VisualPrint [13] collects visual data through offline wardriving and uses it to derive global knowledge about the uniqueness of visual features. This knowledge is pre-computed on the server, stored in Bloom-filter-based lookup tables, and downloaded to the client in advance. During runtime, the client only uses this pre-built knowledge to filter uploaded keypoints. Therefore, its adaptation relies on pre-collected visual context rather than real-time context modeling and reasoning.

More broadly, although these systems make effective use of specific contextual signals, most existing studies do not explicitly model contextual attributes or abstract them into higher-level situations. Instead, contexts are often used directly as isolated indicators for adaptation, with limited aggregation across different contextual dimensions. Without such modeling and abstraction, adaptation rules tend to remain tied to specific systems or scenarios, making it difficult to develop a general context-aware framework that can be applied across different runtime conditions.

Another limitation lies in the granularity of adaptation. Many existing systems still make relatively coarse-grained decisions. DeepDecision [24] dynamically switches between local lightweight models and remote complex models based on runtime conditions, while MARLIN [4] runs the expensive DNN detector only when the scene changes significantly and otherwise relies on lightweight object tracking to update object positions. Although these systems demonstrate the benefit of context-aware adaptation, their decisions are mainly made at the system or pipeline level, such as which execution mode to use or whether to activate a DNN detector. By contrast, this study enables more fine-grained adaptation by using runtime contexts to dynamically adjust specific system parameters, including image resolution and network probing interval.

Finally, many existing approaches focus on a single optimization objective. For example, Chou et al. [6] mainly aim to reduce energy consumption by adjusting display brightness, frame rate, and virtual object rendering rate according to user attention levels. Smart Lens [22] mainly targets latency reduction by using location and time to narrow the candidate search space before recognition. Although these approaches are effective for their specific goals, distributed vision-based AR systems usually need to balance latency, resource consumption, and detection accuracy. Therefore, this paper aims to address the above limitations by modeling multiple runtime contexts, inferring high-level system situations through fuzzy logic, and enabling finer-grained context-aware adaptation.

3 System Design and Implementation

This section presents the design and implementation of the proposed distributed context-aware vision-based AR system. The system is designed to support object detection on Apple Vision Pro while adapting runtime parameters according to changing network and device conditions. This section focuses on the overall prototype architecture, the client-side AR application, and the server-side inference and probing services. The detailed design of the context-aware module is presented in the next section.

3.1 System Overview

Figure 1 shows the overall architecture of the proposed system. The system follows a distributed client-server design, as vision-based AR applications often rely on computationally intensive CV models that are difficult to execute efficiently on mobile devices. It consists of three main components: an Apple Vision Pro client, a context-aware module, and a remote server hosting the Grounding DINO model.

The end-to-end workflow begins with the client obtaining an image and preparing it for transmission. Before the image is sent to the server, it is preprocessed according to the current adaptation decision. For example, the image may be downsampled to a selected resolution to reduce transmission overhead under constrained network conditions. The processed image and detection parameters are then sent to the server through an HTTP inference request. After receiving the request, the server performs object detection and returns structured detection results, including object labels, confidence scores, and bounding-box coordinates. The client then maps the returned bounding boxes to the displayed image region and visualizes the detection results in the AR interface.

3.2 Client-Side AR Application

The client-side application is developed for the Apple visionOS platform[2]. It is mainly implemented in Swift, with SwiftUI used for the user interface, PhotosUI for local image selection, ImageIO for image downsampling, and URLSession for network communication. In the current prototype, the client supports image-based object detection for AR scenarios: the user selects an image from

[2]Apple visionOS developer documentation. Available at: https://developer.apple.com/visionos/ (accessed May 31, 2026).

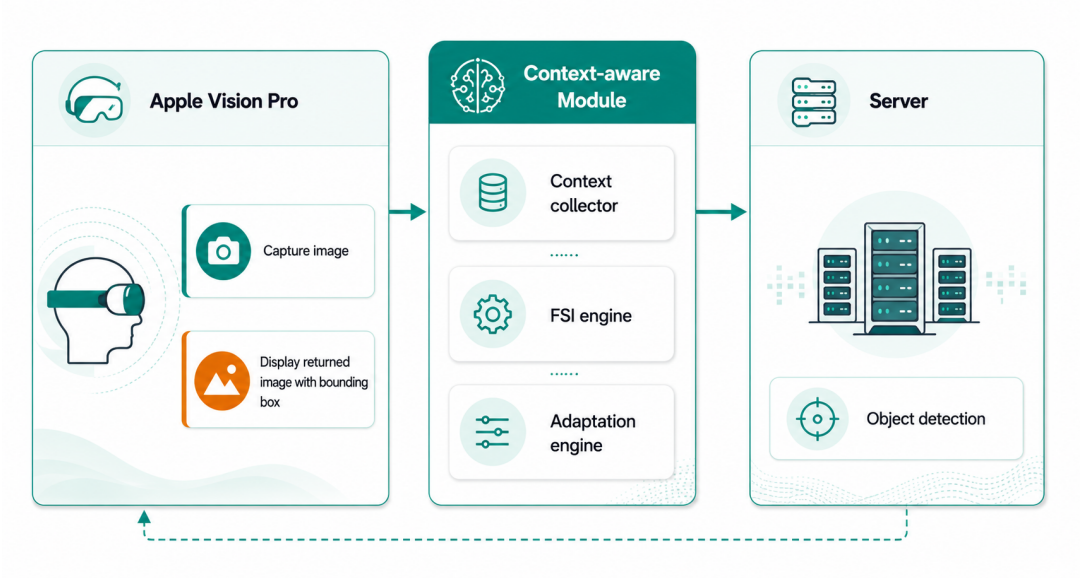


Figure 1: Overall architecture of the proposed AR system.

the local photo library and provides an open-vocabulary detection prompt. This prompt is prepared to match the capability of the Grounding DINO model, which can detect objects specified by natural-language prompts rather than relying only on predefined object categories. The client then preprocesses the image according to the current adaptation decision, packages the image and detection parameters into a `multipart/form-data` request, and sends it to the backend FastAPI service through HTTP POST. After server-side inference, the detection results are returned as object labels, confidence scores, and normalized bounding-box coordinates. The client maps these normalized coordinates to the displayed image area and overlays bounding boxes and labels on the original image. The client also records request-level information, such as request identifier, selected image resolution, timestamps, and adaptation decisions, for later experimental analysis.

3.3 Server-Side Inference and Probing Services

The server-side service is implemented in Python using FastAPI. It provides a `/v1/detect` HTTP inference endpoint that receives images and detection parameters from the Apple Vision Pro client. In addition to the FastAPI-based inference API, the server also runs an independent UDP network probing service. This service listens on `0.0.0.0:9999` by default and supports client-side measurements of network delay, clock synchronization, and packet-train rate. Separating the HTTP inference service from the UDP probing service allows the system to measure network conditions without relying on the heavier image upload pipeline.

The object detection backend is based on Grounding DINO [19]. This model is selected because it supports open-vocabulary object detection, allowing the system to detect objects specified by natural-language prompts rather than being limited to a fixed set of predefined classes. This capability is particularly suitable for AR scenarios, where target objects may vary across environments and

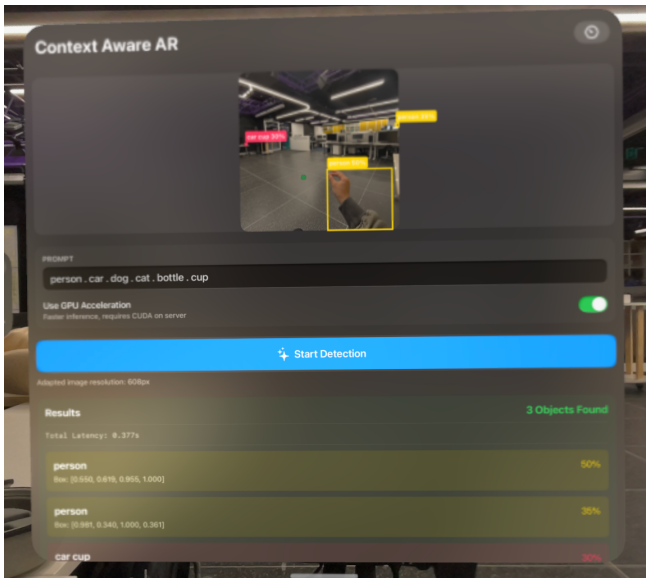


Figure 2: Client-side interface of the Apple Vision Pro prototype.

use cases. In the proposed system, the client sends a user-defined prompt together with the image, and the server uses Grounding DINO to localize the corresponding objects in the scene. Depending on the request configuration, inference is executed on either CUDA or CPU. The raw model outputs are then post-processed to extract object labels, confidence scores, and bounding-box coordinates, which are returned to the client as structured detection results.

The server also records request-level timing information for experimental analysis. These logs include key stages such as image

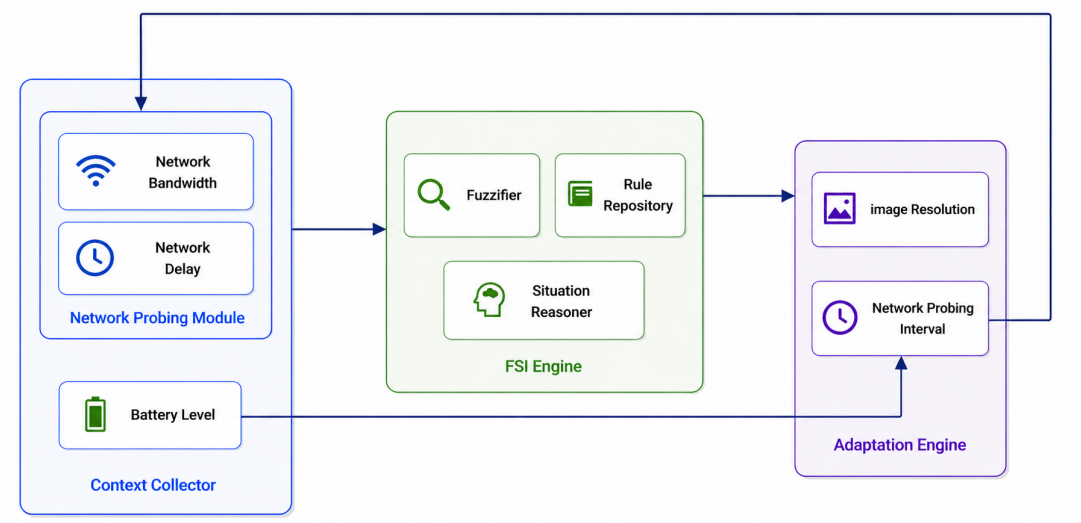


Figure 3: Internal components of the context-aware module.

decoding, preprocessing, model inference, post-processing, and response generation. Together with client-side logs, these records allow the system to analyze the latency composition of the distributed AR pipeline under different network conditions and adaptation settings.

4 Context-Aware Module

Figure 3 shows the internal structure of the context-aware adaptation module. The module consists of three parts: the context collector, the Fuzzy Situation Inference (FSI) engine, and the adaptation engine.

4.1 Context Collector

4.1.1 Context Selection. The context collector is responsible for gathering runtime contextual attributes from the system. Since the proposed system offloads object detection from the client to a backend server, network condition is treated as the primary context in this work. DeepDecision [24] shows that, in edge-based video analytics, network transmission time can dominate the total latency when visual analysis is executed remotely. Following this observation, this work characterizes network conditions using two contextual attributes: network bandwidth and network delay, hereafter referred to as bandwidth and delay.

In addition to network-related contexts, device resource constraints are also considered. Typical device contexts in mobile and AR systems may include battery level, memory usage, device temperature, CPU usage, and GPU usage. In this work, battery level is selected as a representative device-side context, rather than as the only relevant device factor.

This design keeps the current prototype lightweight while preserving the extensibility of the proposed framework. Additional device contexts can be incorporated in the future by adding corresponding context collectors and fuzzy inference rules. Therefore, the selected contexts provide a lightweight but representative basis for demonstrating how runtime network and device conditions can support adaptive behavior in distributed vision-based AR systems.

4.1.2 Network Probing Module. Among the three contexts selected in this work, battery level can be collected directly from the client device through Apple's `UIDevice` battery API[3]. In contrast, quantitative network contexts such as bandwidth and delay cannot be directly obtained as stable numerical values from standard APIs. Therefore, this work implements an active network probing module to estimate these two contexts at runtime.

The network probing module performs two types of measurements: delay probing and bandwidth probing. Both measurements are implemented using lightweight UDP probes between the client and the backend server. In both cases, the client sends UDP packets to the server, and the system estimates the current network condition from the timing behavior of these packets. The main difference lies in how the probe packets are structured and how the corresponding measurement results are derived.

Delay probing is implemented by measuring the round-trip time (RTT) of small UDP echo packets between the client and the server. For bandwidth measurement, however, this work does not adopt a simple RTT-based throughput estimation method. Although such a method is straightforward, it would make the bandwidth estimate strongly dependent on RTT, which is already used to represent network delay. This coupling would blur the distinction between the two network contexts and make it more difficult to analyze their independent effects on system adaptation.

To obtain a more independent bandwidth estimate, this work adopts a PTR-style packet-train method for available bandwidth estimation [12]. Other available-bandwidth probing techniques, such as PathMon [15] and WBest [16], were less suitable for the current prototype because they require accurate receiver-side inter-packet timing, while the receiver in this system runs on Windows

[3]Apple Developer Documentation, `UIDevice.batteryLevel`. Available at: https://developer.apple.com/documentation/uikit/uidevice/batterylevel (accessed May 31, 2026).

and records packet arrival timestamps only at the application level. In PTR, the client sends UDP packet trains with different sending rates, and the receiver estimates the available bandwidth from the aggregate receiving rate around the turning point where the output rate no longer increases proportionally with the input rate. This makes PTR more suitable for this prototype because its estimate is based on packet-train level rate changes rather than precise individual packet-gap measurements.

After obtaining the raw delay and bandwidth estimates, the network probing module further smooths these measurements using a Flip-Flop Filter (FF). FF was proposed in SANE [14] to balance stability and agility in network estimation, addressing the limitation that a fixed-gain EWMA filter is usually biased either toward stability or toward fast reaction. Instead of using a single smoothing strength, FF switches between a more agile update and a more stable update according to whether the current observation falls within normal variation. In this system, the filter is applied before the situation inference stage so that transient probing errors are dampened, while persistent changes in bandwidth or delay can still be reflected in the context values.

The filtered bandwidth and delay estimates are passed to the FSI engine as low-level network contexts. These values are not used directly as adaptation actions. Instead, they are further processed to infer the current network situation and guide subsequent image resolution and probing interval adaptation.

4.2 Fuzzy Situation Inference Engine

Figure 4: Interface of the Fuzzy Situation Inference tab.

The Fuzzy Situation Inference (FSI) engine is the reasoning component of the context-aware module. Its role is to transform low-level runtime contexts into higher-level situation information that can be used by the adaptation engine.

The design follows the FSI approach proposed in previous context-aware computing research. FSI integrates fuzzy logic with context modeling to reason about imprecise and ambiguous real-world situations from multiple contextual attributes [10]. In particular, this work adopts the general FSI structure, including the fuzzifier, rule repository, and situation reasoner, and applies it to network situation inference in distributed vision-based AR systems. This is suitable because network conditions in such systems are rarely binary or perfectly stable. Instead of being treated simply as either "good" or "bad", a network condition may gradually shift from one situation to another. By computing confidence values for multiple possible situations, FSI can capture such intermediate states and support smoother adaptation decisions than hard threshold rules.

4.2.1 Fuzzifier. The Fuzzifier maps the measured context values to their corresponding linguistic terms and computes the membership degree of each value in those terms. In this work, the two network contexts, bandwidth and delay, are first measured as crisp numerical values and then converted into fuzzy representations. For example, if the measured bandwidth is 5 Mbps and bandwidth is described using three terms, *low*, *medium*, and *high*, the fuzzifier computes the membership degree of 5 Mbps in each of these terms. Therefore, each linguistic term can be regarded as a fuzzy set consisting of value–membership pairs.

In this implementation, trapezoidal functions are used to calculate membership degrees. Each trapezoidal function is specified by four parameters (a, b, c, d), which determine the numerical range and transition boundaries of a linguistic term. This work defines the parameter configurations for the bandwidth and delay terms, with reference to relevant broadband and network performance standards [5, 20]. The resulting configurations are shown in Table 1.

Table 1: Fuzzy terms and trapezoidal membership parameters for network contexts.

Context	Term	Parameters (a, b, c, d)
Bandwidth (Mbps)	Low	$(1, 1, 5, 20)$
	Medium	$(5, 20, 40, 60)$
	High	$(40, 60, 200, 200)$
Delay (ms)	Low	$(1, 1, 16, 74)$
	Medium	$(16, 74, 344, 460)$
	High	$(344, 460, 1200, 1200)$

4.2.2 Rule Repository. The rule repository stores the predefined definitions of different situations. In this work, it defines how different combinations of bandwidth and delay correspond to different network situations. Since both bandwidth and delay are divided into three linguistic terms, *low*, *medium*, and *high*, this work defines nine network situation rules.

Each rule follows the form: if bandwidth is X and delay is Y, then the current situation is denoted as XB-YD. Here, B and D refer to bandwidth and delay, respectively, and $X, Y \in \{L, M, H\}$ represent low, medium, and high levels. Table 2 lists the nine rules used in this work.

Together, these predefined rules provide the basis for the situation inference process described in the next section.

4.2.3 Situation Reasoner. The situation reasoner uses the rules stored in the rule repository to infer network situations from the membership degrees produced by the fuzzifier. Specifically, the fuzzifier first maps numerical context values, such as bandwidth and delay, to linguistic terms and membership degrees, and the

Table 2: Predefined rules for network situations.

Rule	Conditions	Situation
R1	Bandwidth is Low and Delay is Low	LB-LD
R2	Bandwidth is Low and Delay is Medium	LB-MD
R3	Bandwidth is Low and Delay is High	LB-HD
R4	Bandwidth is Medium and Delay is Low	MB-LD
R5	Bandwidth is Medium and Delay is Medium	MB-MD
R6	Bandwidth is Medium and Delay is High	MB-HD
R7	Bandwidth is High and Delay is Low	HB-LD
R8	Bandwidth is High and Delay is Medium	HB-MD
R9	Bandwidth is High and Delay is High	HB-HD

situation reasoner then aggregates these membership degrees to compute the confidence value of each possible situation.

The confidence of a situation is calculated by aggregating the weighted membership degrees of the context attributes involved in the corresponding rule:

$$\text{Confidence}(S_j) = \sum_{i=1}^{n} w_i \mu(x_i), \tag{1}$$

Here, S_j denotes the j-th situation rule and x_i denotes the i-th condition in that rule. The term $\mu(x_i)$ is the membership degree of the linguistic term in the corresponding condition, as computed by the fuzzifier, while w_i is the weight of that condition in defining the situation. In this work, bandwidth and delay are treated as two equally important conditions for defining a rule, and therefore both are assigned a weight of 0.5.

For example, for the situation *MB-HD*, the reasoner retrieves the membership degree of the measured bandwidth in the *medium* bandwidth term and the membership degree of the measured delay in the *high* delay term. These two memberships are then combined using their weights to produce the confidence value of the *MB-HD* situation.

The reasoner repeats this process for all predefined rules and produces a situation confidence vector. Each element in this vector corresponds to one possible network situation, such as *LB-LD*, *MB-HD*, or *HB-MD*. Instead of abstracting the current network condition into a single hard situation, the confidence vector preserves the gradual relationship between situations. The adaptation engine can then use these confidence values to make smoother and more fine-grained adaptation decisions.

4.3 Adaptation Engine

The adaptation engine uses the outputs of the previous two components to compute and adapt concrete system parameters. In the current prototype, it controls two targets: image resolution and network probing interval.

These two targets are adapted through different mechanisms. Resolution adaptation follows a situation-aware mechanism and uses the inferred network situations produced by the FSI engine. In contrast, probe interval adaptation follows a resource-aware mechanism and directly uses runtime contexts collected by the context collector, such as battery level. Therefore, the adaptation engine combines situation-aware and resource-aware mechanisms to adjust system behavior at runtime.

4.3.1 Image Resolution Adaptation. Image resolution is adapted according to the inferred network situations. In general, a more constrained network condition, such as low bandwidth or high delay, leads the client to use a lower resolution to reduce transmission overhead, and vice versa.

Following the situation-aware adaptation strategy used in previous work [10], an initial resolution is first calculated for each possible situation. These initial resolution values are then aggregated according to the confidence values of the corresponding situations. Specifically, the initial resolution r_i for situation s_i is calculated as:

$$r_i = LB_r + ((UB_r - LB_r) \times (1 - C(s_i))), \tag{2}$$

where LB_r and UB_r denote the lower and upper bounds of image resolution, and $C(s_i)$ denotes the level of criticality of situation s_i. The criticality level is an application-specific value defined by the system designer when specifying the situation rules. A higher criticality level results in a lower initialized resolution.

Since the FSI engine produces confidence values for multiple possible situations rather than a single hard situation, the final adapted resolution is computed as an aggregated value weighted by confidence:

$$\hat{r} = \frac{\sum_{i=1}^{n} \mu(s_i) r_i}{\sum_{i=1}^{n} \mu(s_i)}, \tag{3}$$

where $\mu(s_i)$ is the confidence value of situation s_i, r_i is its initialized resolution value, and n is the number of predefined situations. This allows the resolution to change smoothly with gradual changes in different situations.

4.3.2 Probe Interval Adaptation. Probe interval adaptation determines how frequently the network probing module should measure bandwidth and delay. Unlike image resolution adaptation, which uses the inferred situations from the FSI engine, probe interval adaptation is directly controlled by two criticality values: network criticality and battery criticality.

Network criticality reflects the degree of network fluctuation. This work measures network fluctuation using the coefficient of variation (CoV), which has been used in previous network studies to characterize throughput variability and stability [9, 29]. For a recent window of samples, CoV is computed as:

$$\text{CoV} = \frac{\sigma}{\mu}, \tag{4}$$

where μ and σ denote the mean and standard deviation of the samples in the window. In this prototype, CoV is calculated separately for bandwidth and RTT samples. The two CoV values are then normalized into bounded scores, and the larger score is used as the final network criticality:

$$C_N = \max(B_{\text{score}}, D_{\text{score}}), \tag{5}$$

where B_{score} and D_{score} denote the normalized fluctuation scores of bandwidth and delay, respectively. This conservative design treats the network as unstable if either bandwidth or delay fluctuates significantly, leading to more frequent probing.

Battery criticality reflects the device-side resource pressure caused by battery drain. While the original resource-aware adaptation model uses a linear mapping from resource availability to criticality, this work adopts an exponential mapping to better emphasize the increasing resource pressure at lower battery levels. This design is motivated by prior battery-aware research, where battery-related dynamics have been modelled using exponential fitting [11]. Battery criticality is calculated as:

$$C_B = e^{-5.0 \times B_{\text{level}}}, \tag{6}$$

where B_{level} denotes the normalized battery level in the range $[0, 1]$. As the battery level decreases, C_B increases, indicating that the system should reduce probing frequency to save device resources.

Based on these two criticality values, the system first computes two initial probe intervals. Let LB_I and UB_I denote the lower and upper bounds of the probing interval. For network criticality, a higher fluctuation level leads to a shorter interval:

$$I_N = LB_I + (UB_I - LB_I)(1 - C_N). \tag{7}$$

For battery criticality, a higher resource pressure leads to a longer interval:

$$I_B = LB_I + (UB_I - LB_I)C_B. \tag{8}$$

Finally, the two initial intervals are aggregated using their corresponding criticality values:

$$I_{\text{final}} = \frac{C_N I_N + C_B I_B}{C_N + C_B}. \tag{9}$$

where I_N is the interval suggested by network fluctuation, I_B is the interval suggested by battery level, and I_{final} is the final probing interval used by the network probing module. This design balances context freshness and resource saving.

5 Evaluation Methodology

This section describes the evaluation objectives, experimental setup, performance metrics, and experimental procedures used to assess the effectiveness of the proposed context-aware adaptation module.

5.1 Evaluation Objectives

The evaluation focuses on two main objectives:

- **Latency improvement:** evaluating whether the proposed context-aware adaptation mechanism reduces end-to-end latency compared with a static baseline.
- **Energy efficiency:** evaluating whether adaptive probe interval adjustment reduces energy consumption and extends device operating time compared with a fixed probing interval.

To answer these research questions, two groups of experiments were conducted. The first experiment compares the end-to-end latency of the system under **withoutCA** and **withCA** configurations. In addition to evaluating latency improvement, this experiment also verifies whether the adaptation mechanism is correctly triggered by examining the changes in selected image resolutions.

The second experiment evaluates the impact of probe interval adaptation on device operating time and energy consumption. Specifically, it compares context-aware configurations with and without probe interval adaptation, together with a fully static **WithoutCA** baseline.

5.2 Experimental Setup

5.2.1 Hardware Configuration. The experiments were conducted using an Apple Vision Pro as the AR client and a desktop workstation as the backend inference server. Table 3 summarizes the hardware configuration used throughout the evaluation.

Table 3: Hardware configuration used in the experiments.

Component	Item	Specification
Client	Device	Apple Vision Pro (M2 model)
Client	Processor	Apple M2
Client	Memory	16 GB unified memory
Client	Storage	512 GB
Client	Operating System	visionOS 26.4
Server	CPU	AMD Ryzen 9 9900X
Server	GPU	NVIDIA GeForce RTX 4090
Server	GPU Memory	24 GB dedicated GPU memory
Server	Memory	128 GB RAM
Server	Operating System	Windows 11

5.2.2 Network Configuration. The Apple Vision Pro and the backend server were connected through a local area network. The Apple Vision Pro accessed the network over IEEE 802.11ac WiFi, while the backend server was connected to the same network infrastructure. Prior to network emulation, the average round-trip time (RTT) between the client and server was measured using 500 ICMP echo packets, resulting in an average RTT of 6.28 ms. In addition, a 10-second `iperf3` test measured an average uplink throughput of approximately 72 Mbps and a downlink throughput of approximately 176 Mbps.

Network conditions used in the evaluation were emulated using Apple's Network Link Conditioner (NLC)[4], which applies bandwidth limits, additional delay, and packet loss at the operating-system level. In this study, only uplink bandwidth and delay were varied, while packet loss was fixed at 0% to isolate the effects of network throughput and transmission delay. Since NLC only constrains the existing network rather than increasing its capacity, the effective throughput under each profile is bounded by both the physical network capacity and the configured bandwidth limit.

Based on this mechanism, nine network scenarios were configured by combining three uplink bandwidth limits with three delay levels, as summarized in Table 4. In the scenario labels, LB, MB, and HB denote low-, medium-, and high-bandwidth configurations, while LD, MD, and HD denote low-, medium-, and high-delay configurations. Although these labels follow the same naming convention as the inferred situations used by the FSI engine, they have

[4]Apple Developer Documentation, "Testing your app under different network conditions." Available at: https://developer.apple.com/library/archive/documentation/FileManagement/Conceptual/On_Demand_Resources_Guide/TestingPerformance.html (accessed May 31, 2026).

different meanings: a network scenario refers to an externally configured NLC profile, whereas a situation refers to a runtime context state inferred from measured context values.

The downlink bandwidth limit was fixed at 250 Mbps across all scenarios because the response payloads were much smaller than the uploaded images. The delay values were applied symmetrically to uplink and downlink traffic. Therefore, delay settings of 75 ms and 300 ms introduce approximately 150 ms and 600 ms of additional RTT, respectively, excluding the baseline network RTT. Packet loss was fixed at 0% in all scenarios.

Table 4: Network scenarios configured using NLC.

Scenario	Uplink BW (Mbps)	Delay (ms)
LB-LD	1	0
LB-MD	1	75
LB-HD	1	300
MB-LD	25	0
MB-MD	25	75
MB-HD	25	300
HB-LD	100	0
HB-MD	100	75
HB-HD	100	300

The selected bandwidth and delay values were chosen to cover the fuzzy terms defined in Table 1. Specifically, each fuzzy term is represented by a trapezoidal membership function, whose two middle parameters define the range where the membership degree is 1. The configured values were selected from these full-membership ranges so that each scenario clearly corresponds to the intended low, medium, or high term instead of lying near the boundary between adjacent terms.

5.2.3 Adaptation Configuration. The proposed system adapts two runtime parameters: image resolution and network probe interval. The image resolution adaptation range was configured between 416 px and 800 px, while the probe interval adaptation range was configured between 1 s and 10 s. For comparison, the fixed-probing baseline used a constant probe interval of 1 s.

The image resolution range was determined through an offline benchmarking study using Grounding DINO. Since image resolution is the only variable that directly affects the object detection process in this system, the evaluation was conducted on the server side without involving the AR client. A total of 500 images from the COCO dataset [17] were processed at five different input resolutions: 256 px, 416 px, 512 px, 640 px, and 800 px. All other inference settings remained unchanged.

The upper bound of 800 px was chosen based on the preprocessing behavior of Grounding DINO. In the default inference pipeline, Grounding DINO applies a resize transform that scales the image so that the shorter side is resized to 800 pixels, while the longer side is constrained by a maximum size of 1333 pixels. Therefore, although images captured by the Apple Vision Pro have a native resolution of 2560×2560 pixels, they are resized to approximately 800×800 pixels before being processed by the model. As a result, sending images at resolutions higher than 800 px does not provide additional effective input detail for inference, but would increase transmission overhead.

Table 5 summarizes the benchmarking results. Overall, reducing the input resolution from 800 px to 416 px substantially reduces the number of transmitted pixels while maintaining relatively stable detection performance. However, further reducing the resolution to 256 px leads to a much larger accuracy drop. Based on these observations, the image resolution adaptation range was configured between 416 px and 800 px for all subsequent experiments.

Table 5: Grounding DINO detection performance under different input resolutions.

Resolution (px)	AP	Pixel Reduction	AP Drop
800	0.471	–	–
640	0.454	36.0%	3.6%
512	0.428	59.0%	9.1%
416	0.403	73.0%	14.4%
256	0.213	89.8%	54.8%

The probe interval range of 1–10 s was selected empirically to provide a balance between context freshness and probing overhead. A shorter interval enables faster detection of network changes but increases energy consumption and network traffic, whereas a longer interval reduces monitoring overhead at the expense of slower probing. The fixed-probing baseline was configured with a constant interval of 1 s to represent the most aggressive monitoring strategy.

5.3 Dataset

The evaluation experiments used images from the ESIQA dataset [31], which contains native Apple Vision Pro images captured in real-world environments and stored in the HEIC format. To provide a consistent experimental workload, approximately 150 images were selected after filtering, with file sizes ranging from 700 KB to 1 MB and a native resolution of 2560×2560 pixels. The same image set was used throughout all experiments to ensure that observed differences in latency, energy consumption, and adaptation behavior were attributable to system configurations rather than variations in image content.

5.4 Performance Metrics

The evaluation mainly reports latency and runtime, with probe count included as a supplementary metric. Latency is used to evaluate the responsiveness of the distributed object detection pipeline, while runtime reflects the practical operating time of the Apple Vision Pro under continuous use. Probe count is reported to provide additional insight into the overhead of network context monitoring.

5.4.1 Latency Metrics. Latency is the primary performance metric in this study. As illustrated in Figure 5, the total latency of one detection request is decomposed into client processing time, network transfer time, and server processing time:

$$T_{\text{total}} = t_1 + t_2 + t_3 \tag{10}$$

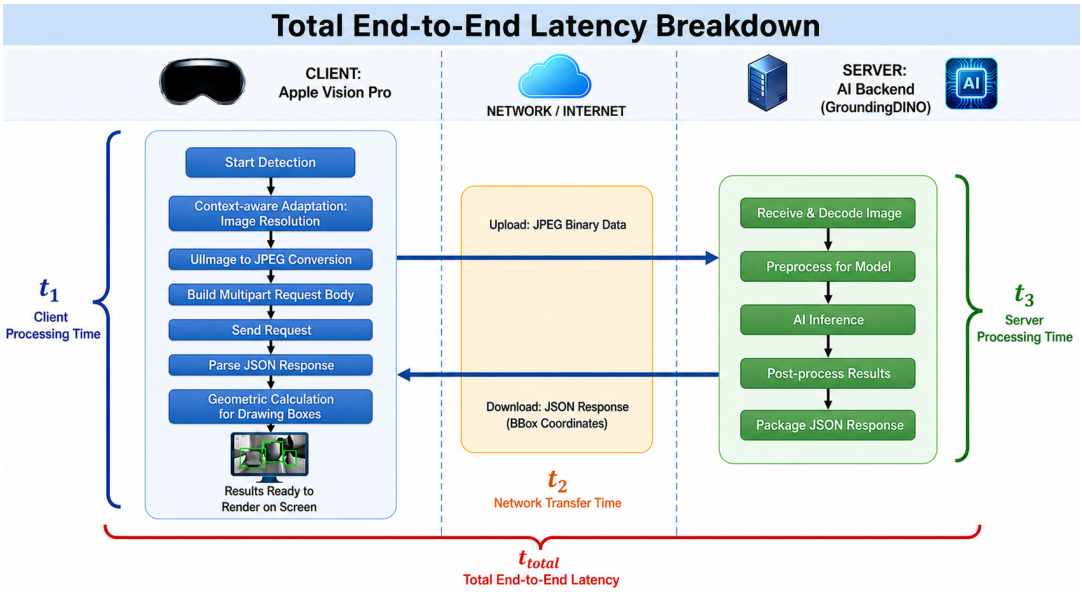


Figure 5: End-to-end latency breakdown.

where T_{total} denotes the total end-to-end latency, measured from the moment the client starts a detection request until the detection results have been parsed and the bounding-box coordinates have been prepared for rendering. The three components represent different stages of the request pipeline. Client processing time t_1 captures client-side overhead, including image adaptation, image encoding, request construction, response parsing, and rendering preparation. Network transfer time t_2 represents the HTTP communication time between the Apple Vision Pro client and the backend server, excluding server-side processing time. Server processing time t_3 represents backend processing after the complete request body has been received and before the response is ready to be sent, including image decoding, model preprocessing, Grounding DINO inference, result post-processing, and JSON response generation.

Since image resolution adaptation mainly affects the amount of uploaded image data, this study further decomposes network transfer time t_2 into client-to-server and server-to-client transfer times:

$$t_2 = t_{\text{c2s}} + t_{\text{s2c}} \tag{11}$$

The client-to-server transfer time t_{c2s} estimates the upload time of the encoded image request, while the server-to-client transfer time t_{s2c} estimates the download time of the detection response. Since the client and server use different clocks, a clock offset Δ is estimated before each run and applied when calculating these one-way transfer components.

This finer-grained decomposition helps identify whether changes in total latency are mainly caused by reduced image upload time, reduced server-side processing time, or other client-side effects. In the latency analysis, this study mainly reports the mean values of T_{total}, t_1, t_2, t_{c2s}, and t_{s2c} under each experimental condition.

5.4.2 Runtime Metric. Runtime is used to evaluate the impact of the proposed adaptation mechanism on Apple Vision Pro battery endurance. It is defined as the continuous operating time of the device from 100% battery level to 5% battery level during the experiment:

$$T_{\text{runtime}} = T_{5\%} - T_{100\%} \tag{12}$$

where $T_{100\%}$ and $T_{5\%}$ denote the timestamps when the battery level is recorded as 100% and 5%, respectively.

The 5% endpoint is used instead of complete battery depletion because the `UIDevice` battery API reports battery level changes at approximately 5% granularity. Using 5% as the endpoint therefore provides a more reliable and convenient way to record runtime while avoiding uncertainty near complete shutdown.

This metric is used to assess whether context-aware adaptation can extend the practical operating time of the AR device.

5.4.3 Monitoring Overhead Metric. Probe count is reported as a supplementary metric to reflect the overhead of network context monitoring. It records the number of network probing operations performed during a fixed experimental period. This metric is mainly used to examine whether probe interval adaptation can reduce unnecessary probing under comparable conditions.

5.5 Experimental Procedure

The evaluation consists of two main experiments: a latency experiment and a runtime experiment.

5.5.1 Clock Offset Calibration for Latency Measurement. Before each latency test run, an initial clock offset calibration was performed between the Apple Vision Pro client and the Windows server. This step was required only for estimating one-way transfer times in the latency breakdown. It does not modify the system clock of either device; instead, it establishes a mapping from the client monotonic timeline to the server monotonic timeline.

The calibration uses a lightweight UDP-based four-timestamp procedure, following the timestamp exchange principle used in network time synchronization protocols [21]. For each synchronization probe, the client records the send and receive timestamps, while the server records the receive and send timestamps. Let t_1 denote the client send time, t_2 the server receive time, t_3 the server send time, and t_4 the client receive time. The clock offset is estimated as:

$$\Delta = \frac{(t_2 - t_1) + (t_3 - t_4)}{2} \tag{13}$$

A client-side timestamp T_{client} can then be mapped to the server timeline as:

$$T_{\text{server}} = T_{\text{client}} + \Delta \tag{14}$$

The estimated offset was fixed during each latency test run. Therefore, the one-way client-to-server and server-to-client transfer times are treated as estimated values, as they may still be affected by clock drift and asymmetric network delay. In contrast, total end-to-end latency and local processing durations are measured directly using monotonic clocks.

5.5.2 Latency Experiment. The latency experiment compares two system configurations:

- **withoutCA**: the baseline configuration without context-aware adaptation;
- **withCA**: the proposed configuration with context-aware adaptation enabled.

The experiment was conducted under the nine network scenarios listed in Table 4. For each scenario, NLC was used to configure the target bandwidth and delay settings. Both configurations used the same set of 150 HEIC images to ensure that latency differences were mainly attributable to system configuration and network scenario rather than image content.

Before each test run, the clock offset calibration described above was performed and recorded. The client then processed the 150 images continuously and sent them to the backend detection server. During each request, the system automatically recorded all latency metrics defined in Section 5.4.1, together with relevant contextual and adaptation information, including measured bandwidth, measured delay, inferred situation, situation confidence, and selected image resolution.

Each network scenario was repeated for three test runs, resulting in 3×150 detection requests per scenario for each system configuration.

5.5.3 Runtime Experiment. The runtime experiment evaluates whether probe interval adaptation can extend Apple Vision Pro battery endurance and reduce network monitoring overhead.

Three system configurations were evaluated:

- **withoutCA**: the baseline configuration without context-aware adaptation;
- **CA with adaptive probe interval**: the context-aware configuration with probe interval adaptation enabled;
- **CA with fixed probe interval**: the context-aware configuration with resolution adaptation enabled but probe interval adaptation disabled.

For each runtime test, the Apple Vision Pro was charged to 100% before the experiment started. The client then continuously processed the image dataset in a loop and repeatedly sent detection requests to the backend server. The test stopped when the battery level reached 5%. This endpoint was selected because the `UIDevice` battery API reports battery level changes at approximately 5% granularity and it is a more reliable stopping point than complete battery depletion.

Each configuration was repeated three times. During each run, the system recorded the total runtime, battery-level changes, and probe count.

5.5.4 Data Processing. After data collection, latency logs were processed using an observation-level IQR filtering and image-level aggregation pipeline. For each network condition and system configuration, upper outliers were removed using the 1.5×IQR rule based on total latency. The remaining repeated measurements were then averaged at the image level before computing mean latency.

6 Results and Discussion

This section presents and discusses the experimental results based on the evaluation methodology described above.

6.1 End-to-End Latency Reduction

Figure 6 compares the mean end-to-end latency of *withoutCA* and *withCA* under nine controlled bandwidth-delay scenarios. The results show that the proposed context-aware mechanism consistently reduces latency in all tested network conditions. When all scenarios are weighted equally, the average latency decreases from 5309 ms to 850 ms, corresponding to an overall reduction of approximately 84.0%.

The improvement is particularly clear under constrained network conditions. For example, in the low-bandwidth scenarios, the latency is reduced from 7810 ms to 858 ms under LB-LD and from 9651 ms to 1642 ms under LB-HD. This indicates that the proposed mechanism is most beneficial when network transmission is the main performance bottleneck.

The lower panel further confirms this trend: all bootstrap confidence intervals of the mean latency difference are above zero, and all scenarios are marked with an asterisk. This suggests that the latency reduction is consistent across different bandwidth-delay combinations, rather than being caused by a small number of favorable cases.

6.2 Latency Component Breakdown

The proposed context-aware mechanism directly affects two latency components: client-side processing time t_1 and client-to-server transfer time t_{c2s}. Specifically, image resolution adaptation is performed on the client side, which would increase t_1. At the same time, resizing reduces the uploaded image size, which is expected to reduce t_{c2s}. Therefore, Figure 7 compares these two components under *withoutCA* and *withCA*.

The results show that *withCA* consistently increases t_1 across the nine network scenarios. This confirms that the context-aware mechanism introduces additional client-side overhead due to adaptation decision-making and image resizing. However, this increase

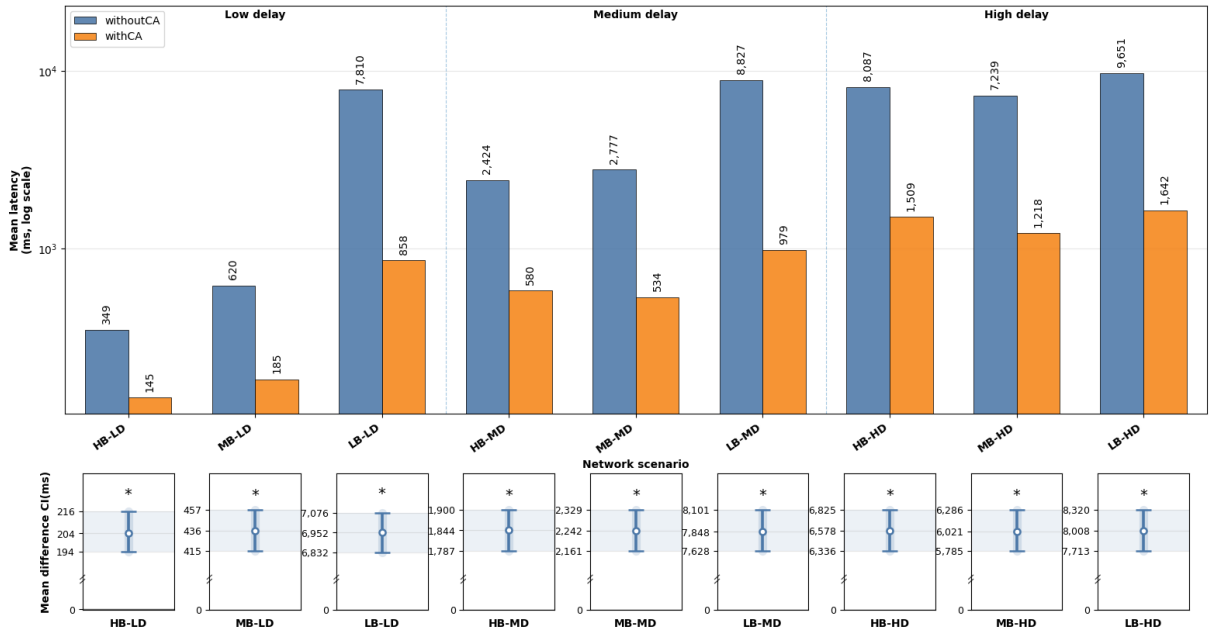


Figure 6: Mean end-to-end latency comparison between *withoutCA* and *withCA* across nine controlled network scenarios. The upper panel shows mean latency on a logarithmic scale, while the lower panel shows the bootstrap confidence intervals of the mean latency reduction. The asterisk (*) indicates that the observed mean difference is greater than zero.

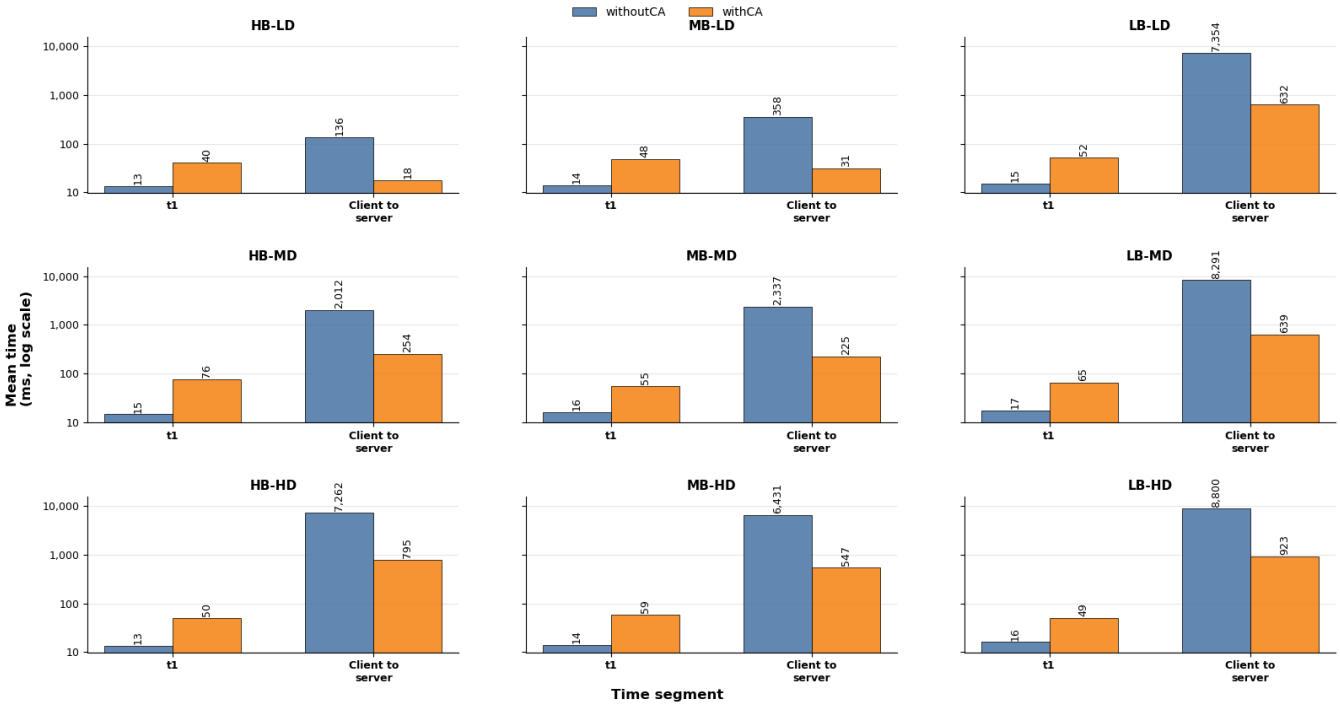


Figure 7: Comparison of client-side processing time t_1 and client-to-server transfer time under *withoutCA* and *withCA* across nine controlled network scenarios.

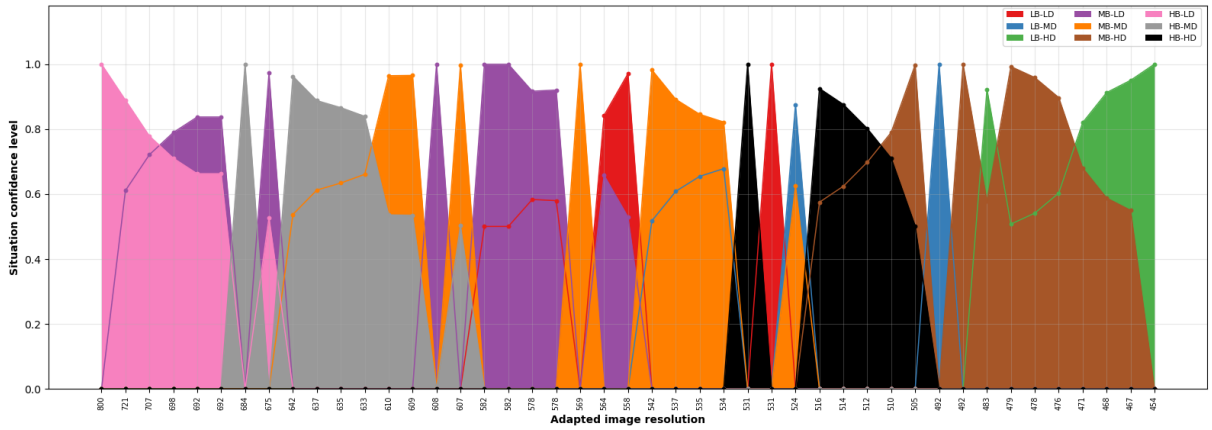


Figure 8: Visualization of the internal adaptation behavior of the context-aware module. Each colored area represents the confidence level of one inferred network situation, and each x-axis tick label shows the adapted image resolution of a selected record.

is relatively small. Across all scenarios, t_1 increases by about 40 ms on average.

In contrast, t_{c2s} is substantially reduced under *withCA*. This reduction is especially clear under LB- or -HD scenarios, where upload time is the main communication bottleneck. On average, t_{c2s} decreases by approximately 4324 ms across 9 scenarios. Therefore, the additional client-side overhead does not offset the much larger reduction in upload time.

To further relate these component-level changes to the overall latency improvement, the net change of these two components was compared with the total end-to-end latency reduction reported in Figure 6. The average total latency reduction is approximately 4459 ms, while the reduction in t_{c2s} minus the additional t_1 overhead is approximately 4284 ms. This accounts for about 96% of the total latency reduction. These results indicate that the main source of the end-to-end latency improvement is the reduction in t_{c2s}.

6.3 Internal Adaptation Behavior

Figure 8 visualizes the internal adaptation process of the context-aware module. Each colored area represents the confidence level of one inferred network situation. Each x-axis position corresponds to one selected adaptation record from the experimental logs, and the tick label indicates the adapted image resolution used for that record.

To make the visualization interpretable, 45 records were selected from the logs in total. Specifically, for each of the nine inferred network situations, five records with the highest confidence values were selected. Therefore, each point in Figure 8 corresponds to one real adaptation record. The connecting lines between points are only visual guides and should not be interpreted as a temporal trace, because the selected records are discrete samples rather than a continuous time series.

The figure shows that different inferred network situations are associated with different adapted resolutions. Favorable situations, such as HB-LD, generally correspond to higher adapted resolutions, while more constrained situations tend to result in lower resolutions. Intermediate situations occupy the middle range and may overlap with neighbouring situations. This overlap is expected because the fuzzy situation inference engine does not assign each record to a single hard category. Instead, it outputs confidence values for multiple possible situations.

This result demonstrates that the proposed context-aware module performs fine-grained adaptation rather than applying a fixed resolution for each inferred network situation. By using situation confidence values, the system can produce smooth adaptation decisions that reflect uncertainty and gradual changes in network conditions.

6.4 Runtime and Probe Interval Adaptation

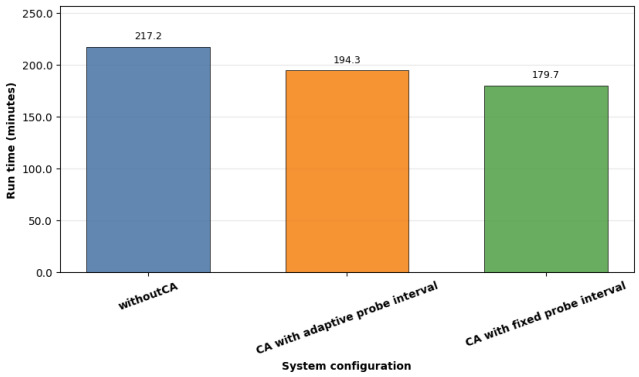


Figure 9: Runtime comparison among three system configurations: *withoutCA*, *CA with adaptive probe interval*, and *CA with fixed probe interval*.

Figure 9 compares the runtime of three system configurations. The *withoutCA* baseline achieves the longest runtime of 217.2 minutes because it disables the context-aware adaptation. Among the two context-aware configurations, *CA with adaptive probe interval*

runs for 194.3 minutes, while *CA with fixed probe interval* runs for 179.7 minutes. Therefore, probe interval adaptation extends runtime by 14.6 minutes, corresponding to an 8.1% improvement over the fixed-probe configuration.

This runtime improvement is closely related to the reduction in probing activity. The adaptive-probe configuration performs 1483 probes, whereas the fixed-probe configuration performs 2848 probes. This means that adaptive probing reduces the total number of probes by 1365, or 47.9%.

These results show that frequent network probing introduces a measurable runtime cost. By increasing the probe interval when intensive monitoring is unnecessary, the adaptive strategy reduces probing overhead and improves runtime while still retaining the context-aware adaptation mechanism. This suggests that probe interval adaptation is useful for controlling the resource cost of context awareness, rather than using a fixed monitoring frequency throughout execution.

6.5 Overall Discussion

Overall, the results show that the proposed context-aware mechanism improves system responsiveness mainly by reducing client-to-server transfer time. The additional client-side processing time is relatively small compared with the reduction in upload time, especially under constrained network conditions. The results also reveal the cost of context awareness. Network probing and local adaptation reduce runtime compared with the baseline, but the adaptive probe interval mitigates this cost by reducing unnecessary probes. Therefore, the proposed system achieves a practical trade-off: it substantially reduces latency while using adaptive monitoring to limit the runtime overhead introduced by context-aware adaptation.

7 Conclusion

This paper presented a context-aware adaptation mechanism for distributed vision-based AR systems. The proposed system uses Apple Vision Pro as the client device and offloads the object detection task to a remote server. To improve system responsiveness, the client collects network and device contexts, infers the current network situation using a fuzzy situation inference engine, and adapts image resolution and network probing interval accordingly.

The experimental results show that the proposed mechanism consistently reduces end-to-end latency across nine controlled bandwidth-delay scenarios. The main source of improvement is the reduction in client-to-server transfer time achieved through resolution adaptation. Although the mechanism introduces additional client-side processing overhead, this overhead is much smaller than the upload-time reduction. The runtime experiment further shows that probe interval adaptation can reduce unnecessary network probing and improve runtime compared with fixed-frequency probing.

This work also has several limitations. First, network context is inherently difficult to measure accurately. Network bandwidth and delay can fluctuate rapidly, and the current context collector is not fully stable. As a result, the collected network context may contain noise. Second, the number of contextual attributes used in the current prototype is still limited. The system does not fully exploit the sensing capabilities of Apple Vision Pro, such as eye gaze, user posture, or other interaction-related contexts. Third, although Grounding DINO supports open-vocabulary prompts, this feature is not fully integrated into the context-aware mechanism. In the current prototype, the prompt is mainly treated as an input parameter rather than a contextual factor that can influence adaptation decisions.

Future work should improve the robustness of context collection, incorporate richer device, user, and interaction contexts, and explore how open-vocabulary prompts can be integrated into the adaptation process. Further evaluation should also examine the relationship between adapted resolution and detection accuracy, so that latency, resource, and recognition quality can be jointly optimized.

References

[1] Lotfi Abdi and Aref Meddeb. 2018. Driver Information System: A Combination of Augmented Reality, Deep Learning and Vehicular Ad-hoc Networks. *Multimedia Tools and Applications* 77, 12 (2018), 14673–14703. doi:10.1007/s11042-017-5054-6

[2] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. 1999. Towards a Better Understanding of Context and Context-Awareness. In *Handheld and Ubiquitous Computing*. Springer Berlin Heidelberg, Berlin, Heidelberg, 304–307. doi:10.1007/3-540-48157-5_29

[3] Ryan Anderson, Juan Toledo, and Hala ElAarag. 2019. Feasibility Study on the Utilization of Microsoft HoloLens to Increase Driving Conditions Awareness. In *2019 SoutheastCon*. IEEE, Piscataway, NJ, USA, 1–8. doi:10.1109/SoutheastCon42311.2019.9020354

[4] Kittipat Apicharttrisorn, Xukan Ran, Jiasi Chen, Srikanth V. Krishnamurthy, and Amit K. Roy-Chowdhury. 2019. Frugal Following: Power Thrifty Object Detection and Tracking for Mobile Augmented Reality. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems (SenSys '19)*. ACM, New York, NY, USA, 96–109. doi:10.1145/3356250.3360044

[5] Australian Competition and Consumer Commission. 2025. *Broadband Speed Claims: Industry Guidance.* Industry Guidance Version 5. Australian Competition and Consumer Commission, Canberra, Australia. https://www.accc.gov.au/about-us/publications/guidance-on-broadband-speed-claims-for-retail-service-providers

[6] Ping-Han Chou, Shih-En Wei, and Chun-Han Lin. 2024. Content-Based Power-Saving Design for Augmented Reality Applications on Mobile Devices. In *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '24)*. ACM, New York, NY, USA, 1–6. doi:10.1145/3665314.3670802

[7] Edmanuel Cruz, Sergio Orts-Escolano, Francisco Gomez-Donoso, Carlos Rizo, Jose Carlos Rangel, Higinio Mora, and Miguel Cazorla. 2019. An Augmented Reality Application for Improving Shopping Experience in Large Retail Stores. *Virtual Reality* 23, 3 (2019), 281–291. doi:10.1007/s10055-018-0338-3

[8] Klaus Fuchs, Tobias Grundmann, and Elgar Fleisch. 2019. Towards Identification of Packaged Products via Computer Vision: Convolutional Neural Networks for Object Detection and Image Classification in Retail Environments. In *Proceedings of the 9th International Conference on the Internet of Things (IoT 2019)*. ACM, New York, NY, USA, 1–8. doi:10.1145/3365871.3365899

[9] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating Systems Review* 42, 5 (July 2008), 64–74. doi:10.1145/1400097.1400105

[10] Pari Delir Haghighi, Averi Perera, Maria Indrawan-Santiago, and Tuan Minh Huynh. 2014. Situation-Aware Mobile Health Monitoring. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MOBIQUITOUS 2014)*. ICST, 248–256. doi:10.4108/icst.mobiquitous.2014.257974

[11] Liang He, Guozhu Meng, Yu Gu, Cong Liu, Jun Sun, Ting Zhu, Yang Liu, and Kang G. Shin. 2017. Battery-Aware Mobile Data Service. *IEEE Transactions on Mobile Computing* 16, 6 (June 2017), 1544–1558. doi:10.1109/TMC.2016.2597842

[12] Ningning Hu and Peter Steenkiste. 2003. Evaluation and Characterization of Available Bandwidth Probing Techniques. *IEEE Journal on Selected Areas in Communications* 21, 6 (Aug. 2003), 879–894. doi:10.1109/JSAC.2003.814505

[13] Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. 2016. Low Bandwidth Offload for Mobile AR. In *Proceedings of the 12th International on Conference on Emerging Networking Experiments and Technologies (CoNEXT '16)*. ACM, New York, NY, USA, 237–251. doi:10.1145/2999572.2999587

[14] Minkyong Kim and Brian D. Noble. 2000. *SANE: Stable Agile Network Estimation.* Technical Report CSE-TR-432-00. Department of Electrical Engineering

and Computer Science, University of Michigan. https://www.cse.umich.edu/techreports/cse/00/CSE-TR-432-00.pdf

[15] Diane Kiwior, J. Kingston, and A. Spratt. 2004. PATHMON: A Methodology for Determining Available Bandwidth over an Unknown Network. In *2004 IEEE/Sarnoff Symposium on Advances in Wired and Wireless Communication*. IEEE, New York, NY, USA, 27–30. doi:10.1109/SARNOF.2004.1302833

[16] Mingzhe Li, Mark Claypool, and Robert E. Kinicki. 2008. WBest: A Bandwidth Estimation Tool for IEEE 802.11 Wireless Networks. In *2008 33rd IEEE Conference on Local Computer Networks (LCN)*. IEEE, 374–381. doi:10.1109/LCN.2008.4664193

[17] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *Computer Vision – ECCV 2014 (Lecture Notes in Computer Science, Vol. 8693)*. Springer, Cham, 740–755. doi:10.1007/978-3-319-10602-1_48

[18] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge Assisted Real-Time Object Detection for Mobile Augmented Reality. In *The 25th Annual International Conference on Mobile Computing and Networking (MobiCom '19)*. ACM, New York, NY, USA, 1–16. doi:10.1145/3300061.3300116

[19] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and Lei Zhang. 2024. Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection. In *Computer Vision – ECCV 2024*. Springer Nature Switzerland, 38–55. doi:10.1007/978-3-031-72970-6_3

[20] MEF Forum. 2021. *MEF 23.2.2: Amendment to MEF 23.2: Satellite Performance Tier*. MEF Standard MEF 23.2.2. MEF Forum. https://www.mplify.net/resources/mef-23-2-2-satellite-performance-tier/

[21] David L. Mills. 1991. Internet Time Synchronization: The Network Time Protocol. *IEEE Transactions on Communications* 39, 10 (Oct. 1991), 1482–1493. doi:10.1109/26.103043

[22] Nayyab Zia Naqvi, Karel Moens, Arun Ramakrishnan, Davy Preuveneers, Danny Hughes, and Yolande Berbers. 2015. To Cloud or Not to Cloud: A Context-Aware Deployment Perspective of Augmented Reality Mobile Applications. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15)*. ACM, New York, NY, USA, 555–562. doi:10.1145/2695664.2695880

[23] Subin Raj, Bikram Karmakar, Gyanig Kumar, Abhishek Mukhopadhyay, Rohit Chandrahas, and Pradipta Biswas. 2025. Comparing Computer Vision Models for Low Resource Dataset to Develop a Mixed Reality Based Manual Assembly Assistant. *Discover Robotics* 1, 1 (2025), 5. doi:10.1007/s44430-025-00005-1

[24] Xukan Ran, Haoliang Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. 2018. DeepDecision: A Mobile Deep Learning Framework for Edge Video Analytics. In *IEEE INFOCOM 2018 – IEEE Conference on Computer Communications*. IEEE, Piscataway, NJ, USA, 1421–1429. doi:10.1109/INFOCOM.2018.8485905

[25] Jinmeng Rao, Yuan Qiao, Fang Ren, Jun Wang, and Qingyun Du. 2017. A Mobile Outdoor Augmented Reality Method Combining Deep Learning Object Detection and Spatial Relationships for Geovisualization. *Sensors* 17, 9 (2017), 1951. doi:10.3390/s17091951

[26] Jinke Ren, Yinghui He, Guan Huang, Guanding Yu, Yunlong Cai, and Zhaoyang Zhang. 2019. An Edge-Computing Based Architecture for Mobile Augmented Reality. *IEEE Network* 33, 4 (2019), 162–169. doi:10.1109/MNET.2018.1800132

[27] Arne Seeliger, Robert P. Weibel, and Stefan Feuerriegel. 2024. Context-Adaptive Visual Cues for Safe Navigation in Augmented Reality Using Machine Learning. *International Journal of Human–Computer Interaction* 40, 3 (2024), 761–781. doi:10.1080/10447318.2022.2122114

[28] Yongbin Sun, Sai Nithin Reddy Kantareddy, Joshua Siegel, Alexandre Armengol-Urpi, Xiaoyu Wu, Hongyu Wang, and Sanjay E. Sarma. 2019. Towards Industrial IoT-AR Systems Using Deep Learning-Based Object Pose Estimation. In *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*. IEEE, Piscataway, NJ, USA, 1–8. doi:10.1109/IPCCC47392.2019.8958753

[29] Yi Sun, Xiaoqi Yin, Nanshu Wang, Junchen Jiang, Vyas Sekar, Yun Jin, and Bruno Sinopoli. 2015. Analyzing TCP Throughput Stability and Predictability with Implications for Adaptive Video Streaming. arXiv:1506.05541 [cs.NI] doi:10.48550/arXiv.1506.05541

[30] Wenxiao Zhang, Sikun Lin, Farshid Hassani Bijarbooneh, Hao Fei Cheng, and Pan Hui. 2017. CloudAR: A Cloud-Based Framework for Mobile Augmented Reality. In *Proceedings of the Thematic Workshops of ACM Multimedia 2017*. ACM, New York, NY, USA, 194–200. doi:10.1145/3126686.3126739

[31] Xilei Zhu, Liu Yang, Huiyu Duan, Xiongkuo Min, Guangtao Zhai, and Patrick Le Callet. 2025. ESIQA: Perceptual Quality Assessment of Vision-Pro-based Egocentric Spatial Images. *IEEE Transactions on Visualization and Computer Graphics* 31, 5 (2025), 2277–2287. doi:10.1109/TVCG.2025.3549174

[32] Kamil Židek, Alexander Hošovský, Jozef Piteľ, and Slavomír Bednár. 2018. Recognition of Assembly Parts by Convolutional Neural Networks. In *Advances in Manufacturing Engineering and Materials*. Springer International Publishing, Cham, 281–289. doi:10.1007/978-3-319-99353-9_30

Part 3

Appendices

Appendix A: Project Artefacts and Supporting Materials

Table 1 lists the main project artefacts provided as supplementary materials. These materials include the HEIC image dataset used in the experiments, the Apple Vision Pro client implementation, the backend server implementation, and the data cleaning and visualisation code used for analysis.

Table 1: Supplementary project artefacts.

Material	Description and Link
HEIC image dataset	A dataset of Apple Vision Pro HEIC images used as the image workload for the experimental evaluation. The images were used to test the image upload, adaptation, and server-side object detection pipeline. https://drive.google.com/file/d/18PRVSaD7dbElecJAs55_2mTgMISlidat/view?usp=drive_link
Client code	The Apple Vision Pro client implementation. The client supports image selection, context-aware image resolution adaptation, request construction, result visualisation, battery monitoring, network probing control, and experimental logging. https://drive.google.com/file/d/1FcF7PF81ICrK08xi7yoBQ8QDW5mG5y83/view?usp=drive_link
Server code	The backend server implementation. The server receives image detection requests, runs the object detection model, processes detection results, and returns object labels, confidence scores, and bounding-box coordinates to the client. https://drive.google.com/file/d/1Zt552oC5uL8UL1VDI5cdt-7ifPWALcqa/view?usp=drive_link
Data cleaning and visualisation code	The analysis code used to clean experimental logs, process latency and context records, remove outliers where applicable, compute summary statistics, and generate visualisations for the results section. https://drive.google.com/file/d/10Kl51zqhyZ-aEu-l3S8gUDR6iqoYPn92/view?usp=drive_link

Appendix B: Example CSV Logs

This appendix provides examples of the CSV logs generated during the experiments. These files demonstrate the type of runtime data collected by the prototype, including context values, latency measurements, battery-drain summaries, and probe-interval records.

B.1 Context Log Example

The context log records the contextual inputs and adaptation outputs for each image request. The example file is:

`withCA_hbmd_contexts_r1.csv`

The main columns include image identifier, request status, measured bandwidth, measured delay, network fluctuation score, battery level, selected image resolution, selected probe interval, inferred fuzzy situations, and situation confidence values.

Table 2: Columns in the context log example.

Column	**Meaning**
Image Number / Image Name	Identifies the tested image.
Status / Error	Records whether the request was successful and any error message.
Bandwidth(Mbps)	Measured network bandwidth context.
Delay(ms)	Measured network delay context.
NetworkFluctuationScore	Estimated network fluctuation score.
Battery Level	Current battery level of the device.
Selected Resolution	Image resolution selected by the adaptation mechanism.
Probe Interval(s)	Network probing interval selected by the adaptation mechanism.
Situations	List of fuzzy situations considered by the inference engine.
Situation Confidences	Confidence values associated with the fuzzy situations.

An excerpt of the file is shown below.

```
Image Number,Image Name,Status,Error,Bandwidth(Mbps),Delay(ms),NetworkFluctuationScore,Battery Level,Selected Resolution,Probe Interval(s),
      Situations,Situation Confidences
1,1.HEIC,success,,29,157,0.942906,1.000000,569,1.511,b_low_d_low;b_low_d_med;b_low_d_high;b_med_d_low;b_med_d_med;b_med_d_high;b_high_d_low;
      b_high_d_med;b_high_d_high,0.000000;0.000000;0.000000;0.000000;1.000000;0.000000;0.000000;0.000000;0.000000
2,6.HEIC,success,,29,157,0.942906,1.000000,569,1.511,b_low_d_low;b_low_d_med;b_low_d_high;b_med_d_low;b_med_d_med;b_med_d_high;b_high_d_low;
      b_high_d_med;b_high_d_high,0.000000;0.000000;0.000000;0.000000;1.000000;0.000000;0.000000;0.000000;0.000000
3,7.HEIC,success,,29,157,0.942906,1.000000,569,1.511,b_low_d_low;b_low_d_med;b_low_d_high;b_med_d_low;b_med_d_med;b_med_d_high;b_high_d_low;
      b_high_d_med;b_high_d_high,0.000000;0.000000;0.000000;0.000000;1.000000;0.000000;0.000000;0.000000;0.000000
```

B.2 Latency Log Example

The latency log records the end-to-end latency and its major components for each image request. The example file is:

withCA_hbmd_latency_r1.csv

This log supports the analysis of total latency and its decomposition into client-side processing, network transfer, and server-side processing.

Table 3: Columns in the latency log example.

Column	Meaning
Image Number / Image Name	Identifies the tested image.
Status / Error	Records whether the request was successful and any error message.
T Total Time(ms)	End-to-end latency for the complete request.
(t_1) Client Processing Time(ms)	Client-side image preparation and request construction time.
(t_2) Network Transfer Time(ms)	Total network transfer time.
(t_3) Server Processing Time(ms)	Server-side processing and inference time.
Client to Server Transfer Time(ms)	Upload-side network transfer time.
Server to Client Transfer Time(ms)	Download-side network transfer time.

An excerpt of the file is shown below.

```
Image Number,Image Name,Status,Error,T Total Time(ms),(t_1) Client Processing Time(ms),(t_2) Network Transfer Time(ms),(t_3) Server Processing
      Time(ms),Client to Server Transfer Time(ms),Server to Client Transfer Time(ms)
1,1.HEIC,success,,1306.18,66.06,873.76,366.37,561.15,312.06
2,6.HEIC,success,,594.01,40.72,472.83,80.46,237.33,234.94
3,7.HEIC,success,,607.57,54.38,477.17,76.02,240.85,235.76
4,9.HEIC,success,,1350.13,27.75,1226.83,95.54,1086.37,139.85
5,10.HEIC,success,,957.12,45.92,799.65,111.56,562.17,236.97
```

B.3 Battery Drain Summary Log Example

The battery-drain summary log records the overall result of a battery-drain experiment. The example file is:

battery_drain_summary_20260519_155727_52B59055.csv

This file summarises the start and end time, total duration, stop threshold, adaptation settings, number of upload attempts, successful uploads, completed image loops, and the associated probe-context log.

```
Start Time,End Time,Start Monotonic(ns),End Monotonic(ns),Total Duration(s),Stop Threshold,Context-Aware Image Adaptation At Start,Probe
      Interval Adaptation At Start,Stop Reason,Battery Level At Stop,Total Upload Attempts,Successful Uploads,Failed Uploads,Completed Image
      Directory Loops,Probe Rows,Probe CSV
2026-05-19T05:57:27Z,2026-05-19T06:59:05Z,20971875890500,24670199434667,3698.324,0.050000,true,true,battery_threshold_reached
      ,0.050000,8043,8043,0,53,277,/var/mobile/Containers/Data/Application/5FC57BD4-33CB-401D-AAFB-4E6000F7CFB0/Documents/test_results/
      battery_drain_probe_context_20260519_155727_52B59055.csv
```

B.4 Battery Drain Probe Context Log Example

The battery-drain probe context log records the network context and probe interval during the battery-drain experiment. The example file is:

battery_drain_probe_context_20260519_155727_52B59055.csv

This file supports analysis of how the probe interval changes over time in response to battery level and network fluctuation.

Table 4: Columns in the battery-drain probe context log.

Column	**Meaning**
Probe Number	Sequential identifier of the network probe.
Probe Timestamp	Timestamp when the probe record was generated.
Elapsed(s)	Elapsed time since the start of the battery-drain experiment.
Bandwidth(Mbps)	Measured network bandwidth.
Delay(ms)	Measured network delay.
NetworkFluctuationScore	Estimated network fluctuation score.
Battery Level	Current battery level during the experiment.
Probe Interval(s)	Current adaptive network probing interval.

An excerpt of the file is shown below.

```
Probe Number,Probe Timestamp,Elapsed(s),Bandwidth(Mbps),Delay(ms),NetworkFluctuationScore,Battery Level,Probe Interval(s)
1,2026-05-19T05:57:31Z,4.187,32.921886,6,0.364634,0.400000,5.500
2,2026-05-19T05:57:36Z,9.914,52.319426,6,0.365159,0.400000,5.498
3,2026-05-19T05:57:42Z,15.748,57.857365,6,0.340457,0.400000,5.594
4,2026-05-19T05:57:49Z,22.340,23.136349,7,0.461937,0.400000,5.021
5,2026-05-19T05:57:54Z,27.982,16.737453,6,0.540484,0.400000,4.551
6,2026-05-19T05:57:59Z,32.893,49.733481,7,0.537373,0.400000,4.571
7,2026-05-19T05:58:04Z,38.036,56.991182,6,0.530793,0.400000,4.612
```

Appendix C: Prototype User Interface Examples

This appendix provides screenshots of the Apple Vision Pro prototype interface. These screenshots illustrate the main functions implemented in the client, including benchmark testing, battery monitoring, image selection, context-aware adaptation, and object detection result display.

Figure 1: Performance diagnostics and benchmark test panel. This interface supports time-offset calibration, latency stress testing, image upload resolution control, and battery-drain test execution.

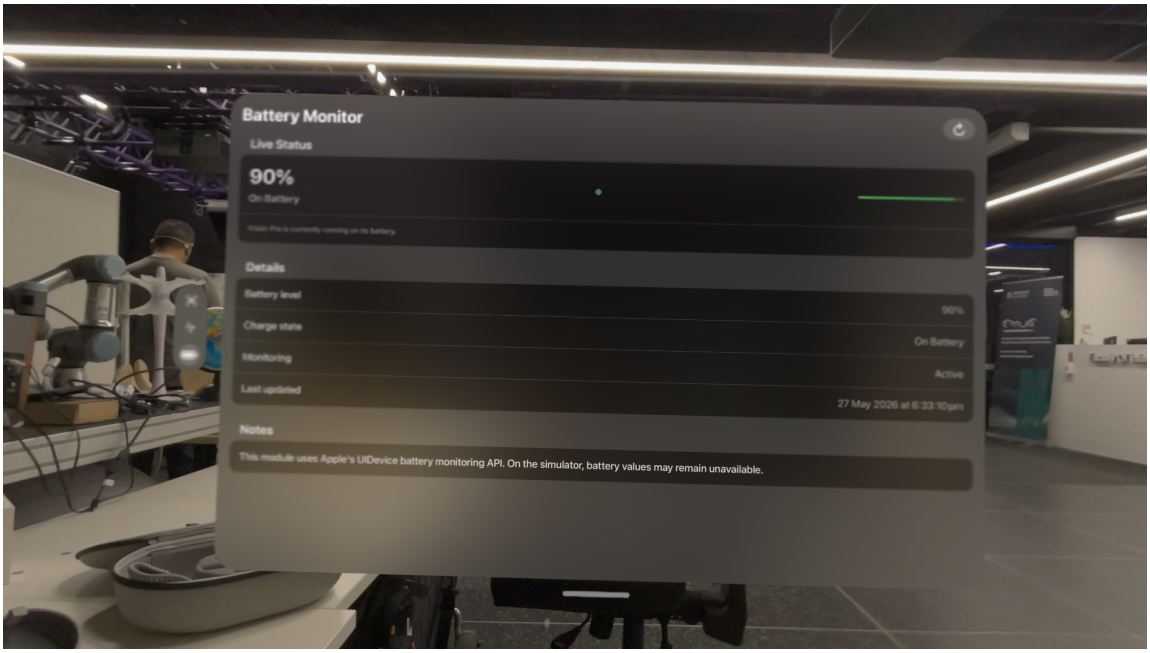


Figure 2: Battery monitor interface. This view displays the live battery level, charge state, monitoring status, and update time using the device battery monitoring API.

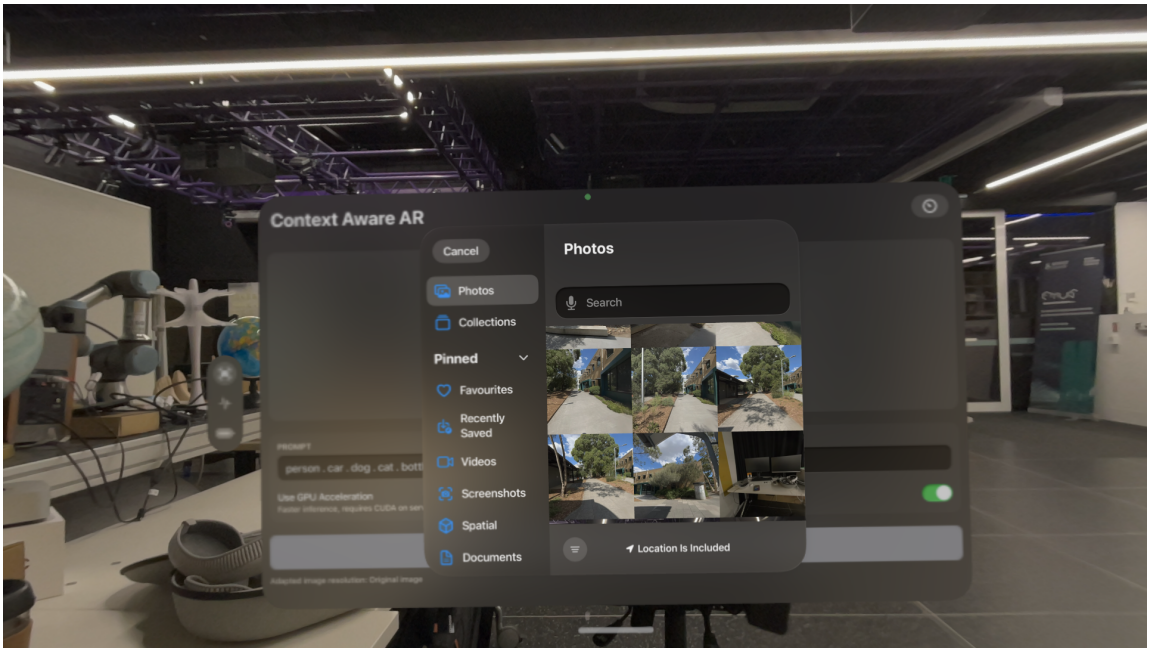


Figure 3: Image selection interface. The client allows the user to select local images from the photo library before sending them to the backend object detection service.

Figure 4: Context-aware module interface. This view displays the current network context values, fuzzy situation confidence values, selected image resolution, adaptive probe interval, and module switches.

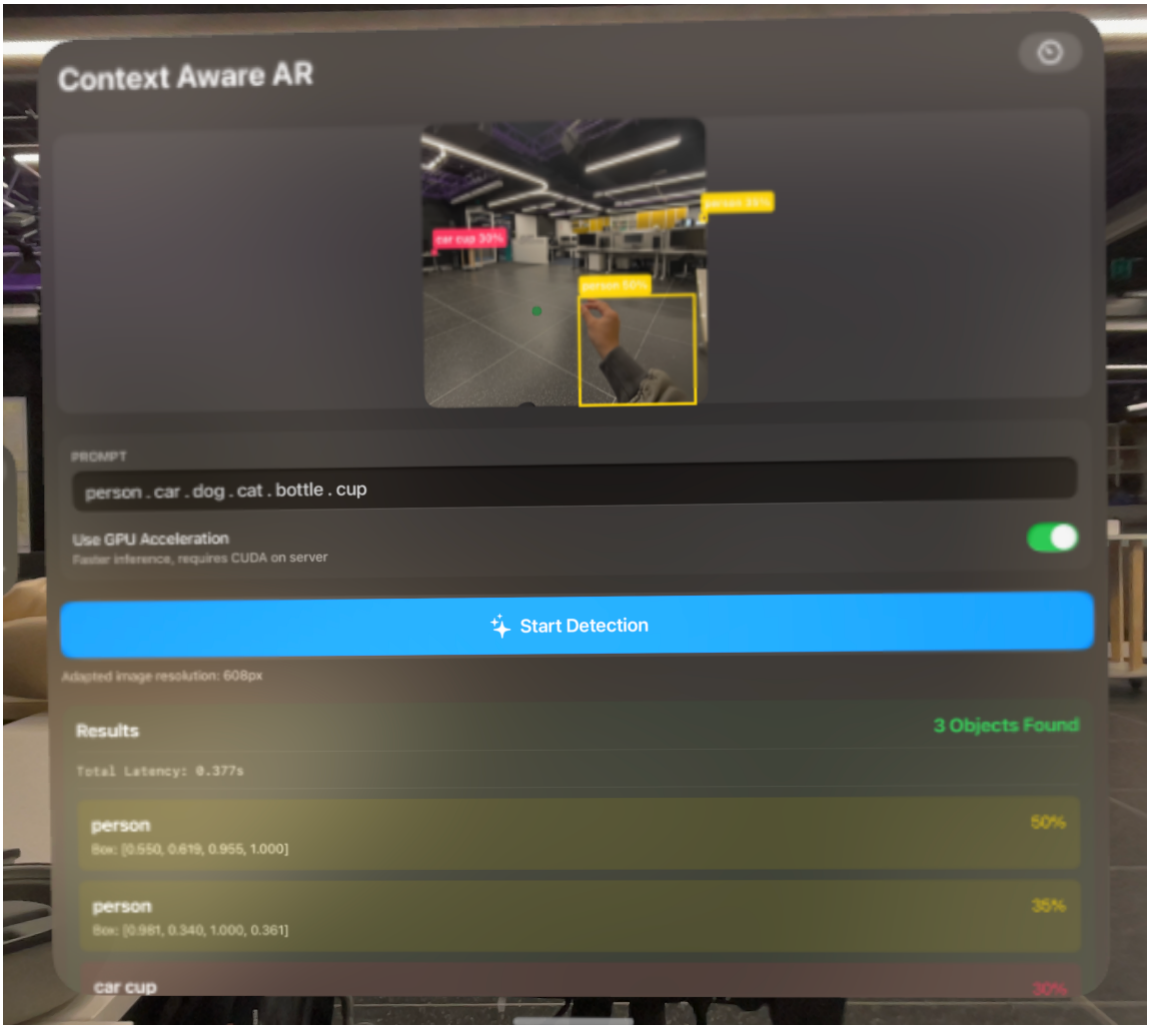


Figure 5: Object detection interface. This view shows the selected image, the open-vocabulary prompt, the adapted image resolution, bounding-box overlays, confidence scores, and the returned detection results.