Breaking edge shackles: Infrastructure-free collaborative mobile augmented reality

Kittipat Apicharttrisorn*
kapic001@ucr.edu
University of California, Riverside
Riverside, CA, USA

Jiasi Chen
jiasi@cs.ucr.edu
University of California, Riverside
Riverside, CA, USA

Vyas Sekar
vsekar@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Anthony Rowe
agr@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Srikanth V. Krishnamurthy
krish@cs.ucr.edu
University of California, Riverside
Riverside, CA, USA

Abstract

Collaborative AR applications are gaining popularity, but have heavy computing requirements for identifying and tracking AR devices and objects in the ecosystem. Prior AR frameworks typically rely on edge infrastructure to offload AR's compute-heavy tasks. However, such infrastructure may not always be available, and continuously running AR computations on user devices can rapidly drain battery and impact application longevity. In this work, we enable infrastructure-free mobile AR with a low energy footprint, by using *collaborative time slicing* to distribute compute-heavy AR tasks across user devices. Realizing this idea is challenging because distributed execution can result in inconsistent synchronization of the AR virtual overlays. Our framework, FreeAR, tackles this with novel lightweight techniques for tightly synchronized virtual overlay placements across user views, and low latency recovery upon disruptions. We prototype FreeAR on Android and show that it can improve the virtual overlay positioning accuracy (with respect to the IOU metric) by up to 78%, relative to state-of-the-art collaborative AR systems, while also reducing power by up to 60% relative to a direct application of those prior solutions.

CCS Concepts

• **Human-centered computing → Ubiquitous and mobile computing systems and tools**.

Keywords

Mobile Augmented Reality, Energy Efficiency, Object Detection and Tracking, Simultaneous Localization and Mapping

ACM Reference Format:
Kittipat Apicharttrisorn, Jiasi Chen, Vyas Sekar, Anthony Rowe, and Srikanth V. Krishnamurthy. 2022. Breaking edge shackles: Infrastructure-free collaborative mobile augmented reality. In *ACM Conference on Embedded Networked Sensor Systems (SenSys '22), November 6–9, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3560905.3568546

*The corresponding author is currently a postdoctoral researcher at CyLab, Carnegie Mellon University, PA, USA, and can also be reached at kapichar@andrew.cmu.edu.

SenSys '22, November 6–9, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9886-2/22/11.
https://doi.org/10.1145/3560905.3568546

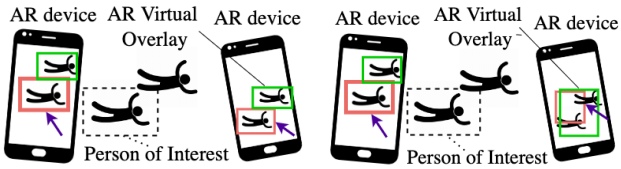
1 Introduction

Collaborative or multi-user AR experiences are on the rise, with examples including Pokemon Go's Buddy Adventures mode [51], Google's Just a Line virtual graffiti drawing app [22], and Meta-AR-App for education [64]. While many multi-user AR apps rely on cloud/edge infrastructure for heavy computations and sharing of information across devices, such infrastructure may be unavailable in many cases (*e.g.,* a search-and-rescue in a disaster zone or an ad-hoc AR game at a beach). In the first example, AR users may need to see virtual overlays around people needing rescue. In the second, users may interact with virtual coins hidden behind real-world objects (*e.g.,* palm tree) in a hide-and-seek game. In both cases, the virtual overlays (highlight around person, virtual coins) should be viewed in the correct locations with respect to the real-world objects by all the users; otherwise, a person might not be correctly identified, or the virtual coin might not be hidden.

Realizing these types of collaborative AR apps requires several steps. ***Step 1:*** An AR device must determine where to place a virtual overlay, based on an analysis of the real world scene; ***Step 2:*** While moving, the AR device must track its own pose (*i.e.,* position and orientation) and the pose of the virtual overlay, so that the overlay is at the correct location on the display; ***Step 3:*** The AR devices must communicate about the virtual overlays with each other, so that the overlays appear at consistent locations on all their displays.

For Steps 1 and 2, today's AR entails two sources of high-power computation that can drain a device's battery. For Step 1, deep neural networks (DNNs) are used by recent AR work [7, 37, 38] to correctly detect and classify objects with high accuracy (*e.g.,* to avoid cases like Fig. 1b where the virtual overlays are drawn over the wrong real world objects due to incorrect detection).

For Step 2, simultaneous localization and mapping (SLAM) is commonly used in AR [21, 33, 50] to allow a device to determine its pose in the real world. While these computations work well when executed on edge infrastructure, as shown in prior work [9, 13, 36] and seen in our measurements, their high power consumption makes them unsuitable for a direct application in infrastructure-free settings. For example, SLAM [33, 42] consumes roughly 1.01 - 1.85 watts, while a DNN execution consumes 0.98 watts on a Google Pixel 4 (comparable to or even higher than what is incurred



(a) Desired case (b) Undesired case

Figure 1: Virtual overlays are (a) correctly placed, (b) misplaced due to poor synchronization or inaccurate object detection. The purple arrows point at the person of interest.

with video streaming [62, 71]). Although hardware offloading may reduce power consumption of these computations, we focus on software solutions that work on heterogeneous devices without requiring hardware accelerators (*e.g.,* GPU) [40, 55, 65].

A key observation we make is that many of the critical/high energy computations are *redundant across devices* in collaborative AR settings, unlike when each device operates independently. This is because 1) AR devices detect and track a common set of physical objects, and so only some (not all) of the devices need to re-detect the objects using DNNs, and 2) AR devices move around a common space, and so in principle, only some of them need to perform localization independently using SLAM. Thus, having all battery-operated AR devices perform these computations all the time is not only expensive but also wasteful in terms of energy consumption.

To enable long-lived (power efficient) AR experiences when there is no infrastructure support, we envision employing *collaborative time slicing*, wherein not every device continuously runs all heavy computations (DNNs and SLAM). Rather, such computations performed by a primary device are re-purposed by others (secondary devices); the role of the primary can be rotated as needed to distribute the energy drain. While seemingly a simple idea, it is very hard to realize collaborative time slicing in practice, such that the AR experience (in terms of virtual overlay placement accuracy) is similar to when all devices perform their own computations, expending high power. Specifically, we encounter the following challenges.

Synchronizing moving AR devices in new areas. To place virtual overlays with consistent positions and orientations in their views, AR devices need to synchronize their 3D coordinate systems. This is very difficult for two reasons. First, users can launch AR apps from different locations, and thus the devices do not share a common initial reference point for synchronization. Second, AR devices move independently and see the same scene from different viewpoints at different times, so it is difficult to determine a common coordinate system that all devices agree on. To address these challenges, we rely on spatiotemporal, repeat observations of the scene from different viewpoints to try to estimate a shared coordinate system [15–17, 39]. Our key observation is that *when* the users view a scene, matters as much as *what* they view. In other words, to construct the common coordinate system, two viewpoints that are recent but less similar in appearance might be preferable to two viewpoints that are older but more similar, since the scene may have changed over time (see Fig. 3). Prior approaches [13, 15, 17, 39] neglect this time factor, *i.e.,* they assume a static real world.

Recovering from failures due to abrupt motion. After synchronization, a tenet of collaborative time slicing is that the secondary AR devices keep track of their own poses in the agreed-upon coordinate system in a lightweight way. However, challenges in tracking arise if there are changes in the appearance of an object in the FoV or in an AR user's pose. To cope with such disruptions, we design triple-layered repair mechanisms, viz. view-based and location-based local repairs, and primary-assisted collaborative repair. The main idea is for a secondary to search for the object in view based on its previously saved appearances; or failing that, to display the virtual overlay at the object's previous locations; or if all else fails, to obtain updated object locations from the primary (tracking them using the heavy computations) and map them to its own view.

Representing virtual overlays in 3D coordinates. As an AR device moves around a 3D world, because the viewpoint (*e.g.,* relative distance and angle) from the device to the virtual overlay may have changed, the pose of a virtual overlay needs to be updated in 3D. This is challenging since the virtual overlay is not a real 3D object. We solve this problem, in a nutshell, as follows. The 2D object coordinates of the virtual overlay, provided by the DNN, are mapped onto the 3D coordinates from SLAM on the primary. These 3D coordinates are then shared with the secondary devices so that they can consistently project the coordinates to the devices according to their viewpoints. We believe that we are the first to harmonize the usage of DNNs and SLAM to correctly maintain virtual overlay poses as multiple devices move.

Contributions and Roadmap: In summary, our work makes the following contributions:

- We identify fundamental challenges in existing systems to support infrastructure-free AR (§ 2).
- We design, arguably, the first infrastructure-free AR system FreeAR (§ 3), which incorporates novel components to realize robust coordinate system synchronization and virtual overlay consistency. FreeAR's lightweight mechanisms save power yet ensure overlay placement accuracy across AR devices.
- We implement an end-to-end prototype on Android (§ 4), working on multiple smartphones without needing root access. Our implementation adds more than 10,000 lines of code to the code base [43]. Our code is available at the FreeAR website [6].
- We perform extensive experiments (§ 5) to evaluate and compare FreeAR's performance with two state of the art approaches, MARVEL [13] which uses edge infrastructure, and MARLIN [7] which performs power efficient on-device computations (no edge is involved). Our evaluations in various representative scenarios show that on average (i) FreeAR reduces power by 46% *and* improves the object detection accuracy by 43% in terms of IOU, compared to MARLIN, and (ii) FreeAR improves the object detection accuracy by 78% in terms of IOU with an 18% increase in power, compared to MARVEL (which benefits from edge infrastructure).

Ethics. This paper does not raise ethical concerns; human volunteer experiments were performed with IRB approval.

2 Motivation and AR landscape

In this section, we provide a detailed example use case, current AR methods, and energy measurements to motivate FreeAR's approach.

Example: Consider a scenario (Fig. 1a) where AR-equipped firefighters navigate a building to search and rescue trapped people. When the lead firefighter (left AR device) finds a person, her AR device automatically detects and highlights the person on its display

Features \ System	AR-Core [21]	AVR [48]	Liu et al. [37]	Edge-SLAM [9]	SPAR [50]	MAR-LIN [7]	MAR-VEL [13]	FreeAR
Energy efficient						✓	✓	✓
No edge infrastructure		✓				✓		✓
Multiple users	✓	✓			✓			✓
Pose tracking	✓	✓		✓	✓		✓	✓
Object detection			✓			✓		✓

Table 1: Comparison of FreeAR and related work

Operation	Power (W)	Operation	Power (W)
OS+Camera+Screen	3.016 ± 0.239	Optical Flow (OF) object tracking [10]	0.319 ± 0.072
IMU-based Tracking (§ 3.3)	0.361 ± 0.151	Image-based Localization (§ 3.3)	0.994 ± 0.438
WiFi P2P Send	0.166 ± 0.033	SLAM [33]	1.208 ± 0.164
WiFi P2P Receive	0.085 ± 0.027	DNN [58]	1.225 ± 0.308
Local Repair (§ 3.4.1)	0.650 ± 0.105	SLAM+DNN+OF	2.424 ± 0.402

Table 2: Energy expenditures for key operations in FreeAR. Averaged measurements with Google Pixel 4, Google Pixel 4a 5G, Google Pixel 5, and Samsung S21.

with a red rectangular overlay and a purple virtual arrow. When a supporting firefighter (right AR device) arrives, the person is also highlighted on his display. If the person or a firefighter moves, these overlays must be updated on the appropriate AR displays. Realizing this requires the three computation steps listed in § 1; however, these steps are done without communication infrastructure that may be damaged, so the firefighters need to form an infrastructure-free network among themselves to coordinate their activities.

Current AR landscape: Current AR systems fall short in the above infrastructure-free scenario and cannot run solely on a lightweight mobile device, where energy is of paramount importance. This is because (a) they require infrastructure support (cloud/edge) to provide consistent overlays [13, 21, 38], (b) they are unconcerned with device energy because they can offload heavyweight compute to the cloud/edge [9, 37, 48, 49], and/or (c) they do not allow for real time coordination between multiple AR devices [9, 13, 37, 38, 49].

Canonical AR solutions: Google ARCore [21] allows for coordination across devices, but requires access to the Google Cloud Platform, which synchronizes different user views. Similarly, Liu et al. [37] require edge support, offloading camera frames in order to perform heavyweight computations (DNNs) to detect the person in view. MARVEL [13] utilizes edge infrastructure, and unlike FreeAR, requires specialized hardware (depth camera or LiDAR) to generate an offline map for localization. MARLIN [7] focuses on power efficient object detection with DNNs, and SPAR [50] enables multi-user AR through SLAM without energy concerns. Related work is summarized in Table 1 and § 6. In this work, we investigate whether such computations and coordination can be done without infrastructure on the devices with low power.

Energy costs: A seemingly natural way of enabling infrastructure free AR would be to have AR devices operate independently and run DNNs (for step 1 in § 1, to detect the person) and SLAM (step 2, to keep track of the person and devices' poses). Prior work, such as [7, 37], run DNNs while the others [21, 33] run SLAM on nearly every frame. We empirically measure the energy consumption of such a strategy. We perform measurements on several smartphones (Google Pixel 4/4a/5 and Samsung S21), using VINS-AR [33] as the SLAM implementation, and EfficientDet [58] on Tensorflow Lite as the DNN. As shown in Table 2, the energy expenditure of running SLAM alone is 1.2 W, DNNs alone is 1.2 W, and SLAM

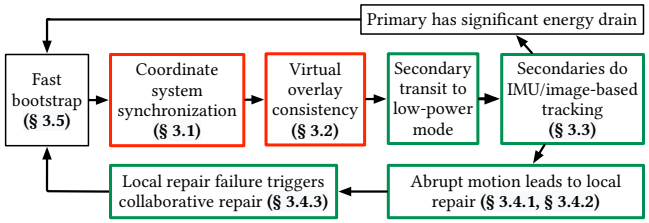


Figure 2: FreeAR's workflow: (red) synchronization phase, (green) steady-state (low-power) phase.

with DNNs and object tracking simultaneously is 2.4 W (averaging across the four different phone models mentioned above). The latter case has both SLAM and DNNs running simultaneously to keep track of existing virtual overlays and device poses, and to provide new virtual overlays, respectively. Note that this is the average energy consumed by a single device; with N users would consume approximately $\geq 2.4 \times N$ W of power.

A case for sharing: We thus observe a natural opportunity for energy savings — sharing common information about the virtual overlays' poses, and avoiding redundant computations as mentioned in § 1. Returning to the example (Fig. 1a), the lead firefighter's AR device (left) could initially detect the trapped persons and highlight the one needing immediate attention (with a virtual purple arrow). It can then share this information to supporting firefighters, having recently arrived, so that they do not have to repeat the computations already done by the lead's device. As the lead and supporting devices move around in a common area with overlapping viewpoints, they can share this information to each other and re-purpose their computations to save overall energy. If one device takes care of heavy computations for too long and drains significant energy, it can hand over these tasks to another device.

Our goal is to design a system that is able to overcome the practical challenges stated in § 1, but we cannot trivially apply prior methods because of the absence of supporting infrastructure, devices' continuous mobility, and low-power requirements for the longevity of the AR experience.

3 Design of FreeAR

As discussed, FreeAR uses collaborative time slicing to divide heavyweight computations across collaborating AR devices. In Fig. 2, we depict the high level workflow of FreeAR's functions and operations. At the beginning of a slice, FreeAR incorporates a novel coordinate system synchronization phase (or sync phase), where all devices converge to a common coordinate system (§ 3.1), and make the virtual overlays' 3D poses consistent across all the devices (§ 3.2). Thereafter, a chosen primary device runs SLAM and DNNs and is able to update the device pose, physical object locations, and the 3D virtual overlays as in traditional AR systems. On the other hand, the secondaries transition to a low-power mode, and will now track their locations in the converged coordinate system with lightweight methods (IMU/image-based tracking), and render the virtual overlays appropriately based on their own motion dynamics (§ 3.3). If a secondary experiences abrupt scene changes, the virtual overlays may be lost; then, FreeAR's local repair kicks in for rapid recovery (§ 3.4.1, 3.4.2). If local repair fails, FreeAR's collaborative repair is attempted (§ 3.4.3). FreeAR transitions to the next time slice either when the repair repeatedly fails, or when the primary's

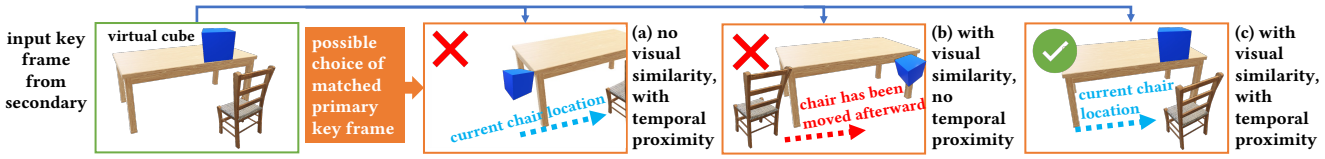


Figure 3: Performing coordinate system synchronization, the primary, choosing frames with (a) partial view or (b) misplaced chair, places the virtual cube at a wrong location. (c) considering the full view of table and chair *and* the current chair location leads to successful synchronization; hence, the virtual cube is placed correctly (on top right of the table).

energy drops, and it now chooses a new primary (§ 3.5). Next, we provide a more in-depth view of FreeAR's key components.

3.1 Coordinate system synchronization

The primary must be able to describe the 3D location and 3D orientation (*i.e.,* pose) of a physical object (and its associated virtual overlay) to a secondary, for the latter to locate the same object in space, and draw the virtual overlay at proper positions. Towards this, the primary and secondary need a common coordinate system to represent the poses of the virtual overlays, objects, and devices.

Prior methods: Centralized collaborative SLAM systems [28, 52, 70] allow multiple agents to coordinate in a common space by establishing a common coordinate system at a central server, which performs most of the heavy computations. On the other hand, decentralized systems [39] assume a pre-built map (database) of a location that allows distributed agents to work together. Finally, recent SLAM systems [16, 17] do not assume a central server or offline maps, but assume that the agents observe common landmarks at the same time to synchronize the coordinate systems.

Challenges: All the above systems fail to meet the requirements we have. (1) Infrastructure-free settings must not assume a central server, (2) impromptu operations (*e.g.,* emergency response) cannot assume pre-built offline maps, and (3) even if such an offline map is available, the actual scene may already have changed (*e.g.,* an object has been moved), making the visual features in an AR device's view differ from the features stored in the map, thus causing the coordinate system synchronization to fail. (4) Finally, AR users may not observe the same scene at the same time (*e.g.,* two firefighters are looking for trapped people at different corners of the room).

Key ideas: We observe that to converge to a common coordinate system, what we need is a common point with spatial and temporal proximity in terms of the views of the primary and secondary. That is, if we can find recent time instances where the primary and the secondary had similar views, those views can be used to synchronize them. Visual similarity between the primary and secondary is important because the more the visual features that are used to estimate the mapping (or homographic transformation [45]) between coordinate systems, the more accurate the estimation becomes [32, 66]. Fig. 3 shows an example where the primary and the secondary have different views; the virtual cube originally in the secondary's coordinate system is transformed to that in the primary's coordinate system using the synchronization output. The left frame is from a secondary. In Fig. 3a where the primary has a partial view of the table and chair, only some of the objects' visual features can be used to map to the secondary's frame, which has a full view of both objects. This mapping mismatch leads to a large synchronization error; hence, the virtual cube is rendered incorrectly in the primary's view (*e.g.,* to the left of the table). Thus, this well-known technique for synchronizing multiple SLAM agents [16, 17] does not work well when both (primary, secondary) are in motion unless pre-coordination is enabled to ensure similar views.

In addition to visual similarity, temporal freshness is also important since it increases the likelihood that the physical objects observed by the primary and secondary will be at nearly the same positions (not moved or moved very little). Recent systems [15–17, 39] ignore this temporal aspect by unrealistically assuming that (1) no objects in the scene have moved [15, 39] or (2) agents fully observe the same objects (*e.g.,* table and chair) at the same time [16, 17]. In Fig. 3b, the primary has a full view of the table and chair; however, a comparison between the frames from the primary and secondary should consider not only *what* features are in the frames (*e.g.,* those of table and chair), but also *where* they are (*e.g.,* chair features are on the right side of the frames). Without this, the frame in Fig. 3b where the chair is on the left side of the frame results in poor synchronization, and thus causes a virtual cube to be rendered incorrectly in the primary's view.

In sum, only when both visual similarity and temporal freshness are considered together is the synchronization very likely to be successful, leading to the correct rendering of the virtual cubes in both views. As an example in Fig. 3c, the primary is able to identify a key frame that (1) has a full view of table and chair (visual similarity) and (2) was captured not too long ago (temporal freshness), that matches with a secondary's frame. Using this match, it correctly renders the virtual cube on top right of the table due to a tight synchronization of the two coordinate systems.

Practical realization: Given this intuition, the secondary can send its recent (to be discussed) camera view to the primary which then searches through the history of its trajectory to find a set of reference frames for synchronization. The primary then checks the spatial correlation (visual) between the two sets of frames. It then chooses the pair (one from each set) that provides the best match to synchronize the coordinate systems of the two entities (*i.e.,* knowing their poses at those times, it creates a mapping).

To elaborate on the details of the above in practice, we leverage a well known technique for homographic transformation called Perspective-n-Point (PnP) method [32] (also used in [33, 39, 50]) as the underpinning of our synchronization method due to its low latency and acceptable power consumption. As input to a basic PnP operation (details to follow), **(1)** FreeAR chooses frames from the primary that are (a) similar in appearance and are (b) close in time to an input frame sent by the secondary. We call this selection of suitable input frames *improper frame avoidance.* **(2)** If there are multiple primary frames that are similar in space and time, FreeAR checks, for each of these frames, how many feature correspondences fit with the secondary's frame; the more features that fit, the better

the synchronization. We call this *variance suppression*. Below, we describe these two components in more detail.

Step 1: Improper Frame Avoidance. Coordinate system synchronization is fairly heavyweight, so it takes place on the primary. FreeAR's primary has access to its own entire frame history, but only to the most recent frame from a secondary; hence, it needs to search for the best matches from its history to establish a mapping with the secondary's frame, so it computes two scores:

- A visual similarity score, $v[i]$, where i is the index associated with the primary's candidate frame. The similarity is based on work in [19] where dictionaries of BRIEF visual features are matched between the primary's frame i and the secondary's frame. Note that FreeAR uses selected key frames (*e.g.,* those in which many features are detected) for coordinate system synchronization because they are less prone for visual distortion (*e.g.,* less blurriness), and the synchronization more likely succeeds. However, even in the case of failure, FreeAR will attempt another iteration of synchronization in the next time period (more details to follow).
- A time score that downweighs old frames, $0.99^{(t-t_i)/s}$, where t is the current time, t_i is the timestamp associated with frame i, and s is a normalization factor.

We then combine the visual and time scores to define the *frame proximity score (ps)* for primary's frame i as

$$ps[i] = v[i] * 0.99^{(t-t_i)/s} \quad (1)$$

and rank the frame indices i in a descending order of this score. Finally, we select the primary's *top-k* frames that maximize $ps[i]$. We choose the top k frames to increase the pool of candidates for high-quality matches, with further filtering below in Step 3.

Step 2: PnP synchronization method. The previous step returns k candidate frames from the primary's frame history for aiding synchronization, given a secondary's input frame. However, we still need to estimate the homographic transformations [45] (estimated spatial relations) between the coordinate systems of the primary and the secondary. PnP is a well-known technique [57] to do this, and so we only briefly summarize its usage here. First, it takes as inputs the primary frame (f_p, determined by Step 1), the secondary frame (f_s), and the pose at f_s ($T_{f_s \to s}$) with respect to the secondary's coordinate system (s). Second, it uses the intersection of the visual features in f_p and f_s to compute the pose of the secondary's frame with respect to the primary's coordinate system (p), using PnP, as $T_{f_s \to p}$. Finally, it estimates a 4x4 homogeneous transformation matrix $H_{p \to s} = (T_{f_s \to p})^{-1} \cdot (T_{f_s \to s})$, which is used to transform the 3D coordinates of an object from p to s as $o_s = H_{p \to s} \cdot o_p$, where o_p and o_s are the 3D vector coordinates of the object in the primary and secondary's coordinate systems, respectively. Details about how we obtain o_p are discussed in § 3.2.

Estimation of synchronization quality: The PnP solver fits a linear model ($H_{p \to s}$) from the 3D world points to visual features from the 2D camera frame inputs. More of the visual features fitting the linear model means that the linear model can consistently explain the observed data; thus, the synchronization is more likely to be accurate. In other words, $H_{p \to s}$ more likely represents the unknown ground truth transformation between the two coordinate systems. To use this information, FreeAR counts the number of visual feature correspondences that fits the linear model (within a tolerance threshold); this is called `numInliers` [57]. Recent work on PnP solvers [30, 32, 69] reports that the more inliers there are, the smaller the translation and rotation estimation errors become.

The primary runs the PnP solver for all the k frames from Step 1 and returns the largest found `numInliers` value, along with the corresponding $H_{p \to s}$, to the secondary. The value of k drives a tradeoff between higher-quality synchronization and the compute latency ($\approx k \times 70\ ms$). Guided experimentally, we choose $k = 3$.

Step 3: Variance Suppression. Upon receiving $H_{p \to s}$ and `numInliers`, the secondary uses `numInliers` to determine whether to accept the proposed synchronization. It only does so only if `numInliers` is greater than in the previous synchronization attempts. If this is true, it records both (a) $H_{p \to s}$ for transforming 3D points in the primary's coordinates (p) to those in its own coordinates (s), and (b) `numInliers` for comparison in the next synchronization iteration.

Finally, FreeAR checks `numInliers` > threshold for all the secondaries; if so, their coordinate systems are synchronized with that of the primary. We choose threshold = 5 because it is the minimum number for PnP to be successful [32].

3.2 Consistent virtual overlay placement

A key requirement of AR is that all users have a consistent view of a virtual overlay (in terms of its location, size, and orientation in the 3D world). For example, an overlay should appear larger on the display of a user closer to it, than of another who is further away. Thus, after coordinate system synchronization, the primary needs to share information about the physical objects in its view (whose locations are fixed in the 3D world), along with the relative positions of the virtual overlays with respect to these objects, with all the secondaries, so that all users will have consistent views.

Prior methods: Virtual overlays given by DNNs are in 2D, and AR users may observe the objects from different angles or distances, and so the same objects appear differently in terms of sizes, shapes, and orientations in different FoVs. Therefore, 2D virtual overlays directly shared by the primary can easily be mis-represented at a secondary's view (*e.g.,* as in Fig. 1b). SLAM generates key points in 3D, but cannot determine the locations for the virtual overlays from their 2D representation. Current AR systems focus on either DNNs [7, 37] or SLAM [9, 13, 50], but not both; others [23, 36] require edge infrastructure to compute the location of the virtual overlay on each user's view. Objectron [2, 20], detecting objects in 3D on mobile devices without edge infrastructure, in theory allows the user devices to utilize the 3D object coordinates to determine the correct placement of the virtual overlays, but it only works for a single user, without any virtual overlay sharing mechanism.

Challenges: Unfortunately, prior systems do not meet FreeAR's requirements. First, there is no edge infrastructure to coordinate virtual overlay placement. Second, we need a mechanism to represent these virtual overlays in 3D, and then perform a transformation to the 2D display of a different (secondary) user.

Key ideas and realization: Our vision is to harmonize the outputs of the DNN with that of SLAM in order to ensure the consistency of virtual overlay placement across primary and secondary devices. During the synchronization phase, AR devices run both DNNs and SLAM. DNNs provide information regarding physical objects in the environment by extracting features in an image view

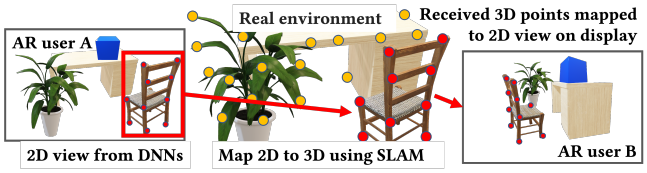


Figure 4: SLAM and DNNs cooperation maps 2D features of interest into 3D space allowing another AR user to view a consistent virtual overlay (virtual cube).

and returning 2D locations (u, v) where objects of interest (*e.g.,* person) likely appear. SLAM also extracts features of the environment, but instead of looking at a single image, it tracks those features in 3D continuously over time. By bundling information of feature movement (from frame to frame) and estimates of device pose changes using IMU, SLAM can estimate these features in the 3D world (x, y, z). For each (x, y, z) in the FoV observing the 3D world, we can project it to the 2D display as follows.

$$\left[u, v, 1\right]^T = \left[K\right] \left[R|t\right] \left[x, y, z, 1\right]^T \quad (2)$$

where K is an intrinsic camera parameter matrix (from a camera calibration), and R and t represent an estimated camera pose (rotation and translation, respectively) [29]. To harmonize the DNN and SLAM outputs, we consume a set of 3D points (x, y, z) output by SLAM, and map them to their corresponding projected 2D points (u, v) in the current view. Then, we filter the (u, v) by whether they lie within the 2D coordinates of a virtual overlay output by DNNs (*e.g.,* a bounding box). The 3D coordinates (x, y, z) corresponding to the filtered (u, v) are considered the 3D coordinates of the object.

FreeAR exploits this association as illustrated in Fig. 4. The primary device (user A) has the 2D coordinates of the chair from DNNs, which allows it to determine the 2D coordinates of the "blue cube" virtual overlay relative to the chair. Then, FreeAR converts the coordinates of the chair into 3D coordinates using SLAM and Eq. 2; subsequently from this, we are able to estimate the 3D coordinates of the virtual overlay, which are then conveyed to the secondaries. Each secondary then maps the virtual overlay's 3D coordinates onto its own 2D view again using Eq. 2 with its own R and t, and thus is able to place the virtual overlays properly on its display (on the table). Specifically, with changes in FoVs (or pose changes), (x, y, z) in Eq. 2 are fixed while R and t are updated; thus, u, v is projected into the view with proper sizing and orientation.

3.3 Lightweight device localization

Once the AR devices synchronize their coordinate systems and display virtual overlays consistently across devices, a steady state has been reached within the collaborative time slice. At this point, the secondary devices turn off SLAM and DNNs to save power. However, every secondary needs to continuously update its pose relative to its own coordinate system because when the primary shares new object information, the secondary will need to map it on to its view (which has changed) correctly. Note that it is relatively easy for the primary, which runs SLAM, to track its pose changes.

Prior methods: Previous systems [13, 53, 54] use an IMU to estimate pose changes in the 3D world (referred to as IMU-based tracking). However, this can accumulate drift and become inaccurate over time (10-20 s) [31] and result in displacements of virtual overlays from their correct positions. Recent systems such as [36] use image-based methods without considering IMUs (referred to as image-based localization) wherein a device uses a recent image frame (i) to search for a correspondence (a similar frame j) in its own trajectory history. Using these two frames (i and j), the device can compute a homographic transformation from the known pose at j to the current unknown pose at i. This transformation is used to update the device pose, and is re-computed as the device moves around. However, our measurements on different device models (Table 2) indicate that image-based localization consumes 0.99 W, which is 2.75 × higher power than IMU-based tracking (0.36 W); utilizing the former continuously can cause undesired power drain. Prior work [36] does not consider power efficiency as we do in FreeAR. However, a recent AR system [13] reduces power consumption on AR devices by offloading image-based localization to an edge infrastructure, not existing in our infrastructure-free settings.

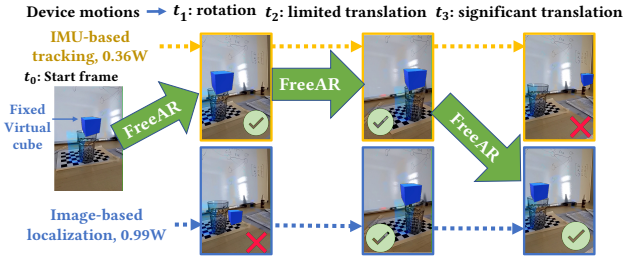


Figure 5: IMU-based and image-based localization results in high accuracy and low power updates of AR device poses.

Challenges: Unlike prior systems' assumptions, we do not have edge infrastructure and our solution has to be power efficient. Yet, we need to have high accuracy, *i.e.,* if a large drift occurs, FreeAR needs to recover from that drift and correctly place the virtual overlay in the view again. This process should incur low latency to update the pose of an AR device, in motion, almost in real time.

Key ideas: From the above, we make the observations that since image-based localization is relatively power heavy, we should use it sparingly if at all. Thus, we seek to use the IMU-based tracking to the extent possible. To increase its usability, we incorporate the impact of gravity to help improve the accuracy of basic IMU-based tracking. This is inspired by a prior work [53] that uses gravity estimation to improve tracking of a wrist watch with an arm model. We significantly build upon this to improve tracking of an AR device in free space where the user may move an arm or walk around in the space. We only trigger image-based localization upon need. Specifically, we find that while our modified IMU tracking is very robust to rotation and minor translation (also shown in [53, 54]), its error accumulation increases over larger translations. Thus, if the IMUs indicate significant translation (> 20cm), we trigger image-based localization to ameliorate the error.

A simple example in Fig. 5 showcases the benefits of integrating IMU and visual tracking with FreeAR. The upper row shows IMU-based tracking alone, and the lower row shows image-based tracking alone. An AR device at t_0 observes a physical cup, on top of which a virtual cube is rendered. As the device moves (without SLAM), the cube should remain fixed if the device pose is being tracked correctly. At t_1, the device rotates to the right (*e.g.,* right edge of the device moves closer to the user); here, FreeAR uses IMU to track its pose correctly (using the image-based method results in pose estimation error, *i.e.,* the cube shrinks and drops down a

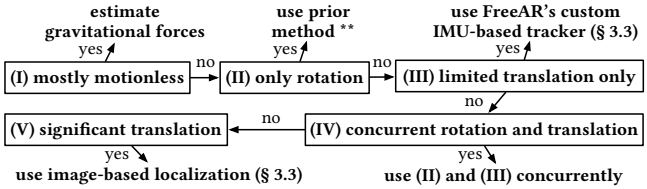


Figure 6: FreeAR's lightweight localization workflow; **[47]

bit). At t_2, the device experiences limited translation to the left and both IMU and image-based methods result in consistent device tracking; however, because IMU consumes less power, FreeAR uses it to update the device pose at this time. Finally at t_3, when the device is moved to the right with significant translation, IMU-based tracking results in a very large error and causes the cube to go further away from the cup to the far right of the frame. FreeAR's choice of image-based localization, however, helps regain the pose and thus, the cube placement on top of the cup. Next, we provide some details of FreeAR's combined IMU and image-based tracking.

Augmenting IMU Tracking with Gravity Estimates. We first integrated several publicly available IMU-based tracking methods [4, 27] into FreeAR, but found they did not perform well (*e.g.,* fixed virtual cube is displaced as the device moves). Hence, as mentioned, we develop our custom IMU-based tracker, inspired by two prior efforts, viz., Shen et al. [53] removing gravity from accelerometers under arm motion, and Solin et al. [54] tracking devices with legged or wheeled motion. Neither fulfils our need to track the device in 3D, with the user both moving her arm while holding the AR device and walking. Thus, we combine and build on these ideas to significantly improve tracking accuracy as follows: **(1)** When the device is mostly motionless (acceleration $(a_x, a_y, a_z) < 0.2m/s^2$), we estimate the gravitational forces in the three dimensions viz., (g_x, g_y, g_z). **(2)** When the user starts moving the device, we estimate the linear accelerations as $la_x = a_x - g_x, la_y = a_y - g_y$, and $la_z = a_z - g_z$. **(3)** Using standard physics kinematic equations [31], we estimate the translation every $\Delta t = 10$ ms.

We find that this simple method works well under the assumption that the user moves and stops occasionally (which provides a chance to re-estimate gravity). Since prior methods for rotation tracking work reasonably well, we incorporate the one from [47] into FreeAR.

Augmenting IMU Tracking with Visual Information. When IMU tracking indicates significant translation, the secondary device captures the recent camera frame and uses it to recover the pose within its own coordinate system. Setting this translation threshold too high can cause pose estimation to be off and virtual overlays to drift away; setting it too low will cause frequent invocations of the image-based approach, and thus induce high energy. Based on experiments with our smartphones, we set the threshold to 20 cm.

The image-based localization within FreeAR is similar to its coordinate system synchronization method. The key difference is that instead of comparing the secondary's most recent frame with the *primary*'s history of frames (Eq. 1), the comparison is made between the secondary's most recent frame and its own historical frames. Fig. 6 summarizes FreeAR's workflow for device pose tracking.

3.4 Recovery upon abrupt motion

While the above modules (§ 3.1-3.3) enable FreeAR to cope with gradual motion, they cannot fully handle a user's abrupt motion (e.g., a quick turn). Here, a secondary can lose track of an object

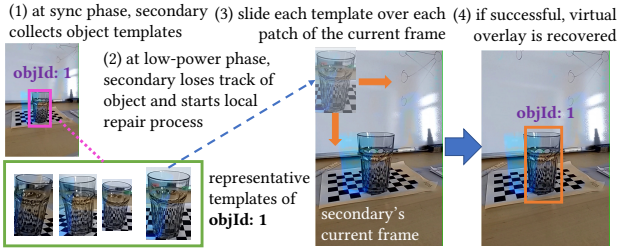


Figure 7: View-based Local Repair (VLR): a secondary uses collected templates to recover the lost virtual overlay.

and its virtual overlay can disappear. Abrupt pose changes also cause a loss in synchronization with the primary. A naive way to recover from such a loss is to trigger SLAM and DNNs for a reset; however, this consumes high power and importantly, induces long delays for the system to return to a steady state. Thus, we desire power thrifty, low latency repair mechanisms, to allow the secondary device (not running SLAM or DNNs) to (a) recover object locations in the device's 2D view correctly, and (b) place virtual overlays that are consistent with those in other AR users' views.

3.4.1 View-based local repair (VLR): Intuitively, if we know what object was lost, and can remember what it looked like, we can try to find it on our display in a lightweight way. Specifically, we can look for the lost object in historical frames, extract its features, and try to find the part of the display which has the same features.

Prior methods: There are many possible candidates from the literature for performing view-based local repair on the above basis. (1) *Template matching* [46] finds the location of a template image (of the physical object) in the current view. It slides the template over a template sized window (or patch) of the input image and compares the template and the patch (see (3) in Fig. 7). Then, if the patch with minimum difference to the template differs by lower than a threshold, it is considered to be the recovered 2D location of that object (see (4) in Fig. 7). (2) *DNN* [61]: detects and classifies physical objects in the current view. (3) A *cascade classifier* [44] uses Haar features to train a classifier to determine the location of a physical object (if present). Our experiments (on a Pixel 4 phone) indicate that template matching consumes the lowest power (0.56 W) with the smallest latency (20 ms), while the DNN (cascade classifier) consumes 0.98 (1.91) W with 250 (120) ms latency, as one might expect since they are considered heavyweight [7, 25].

Challenges: We cannot simply plug in template matching into FreeAR because there are too many templates (*e.g.,* 3600 templates in 2 mins), so blindly matching the current frame with all of these can induce large delays in recovery.

Key ideas: Our approach is to collect templates of the objects of interest (with which the AR virtual overlays are associated) during the sync phase. A secondary then uses those templates to re-locate lost objects and re-draw the virtual overlays. We filter out redundant templates and use fast template matching to recover multiple physical objects simultaneously (details to follow). Since both the template and candidate match were captured under similar conditions by the same device, VLR is very likely to succeed. We next describe VLR's two main components that accomplish this.

Intelligent template collection: When the DNN (during the sync phase) and the accelerometer indicate that a device is mostly motionless, candidate object templates from the camera frames are

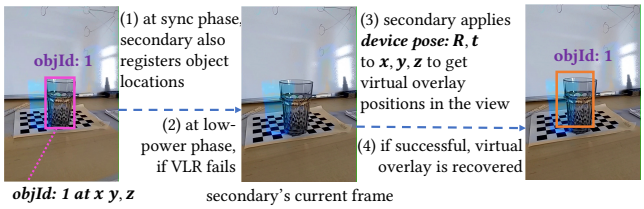


Figure 8: Location-based Local Repair (LLR): a secondary uses object locations and current pose to recover a virtual overlay.

obtained, as illustrated in step (1) in Fig. 7. To enable future local repairs, FreeAR needs to collect a diverse, yet compact set of templates for each object. It uses a color moment hash [59], a compressed representation of the image that is quick to compute (10ms) and compare (< 1ms), to determine if it should store a new template. It stores a new template based on two criteria: (1) the minimum hash distance compared to all previous templates must be greater than a threshold *or* (2) the minimum difference between a new template's width and height and all previous templates' widths and heights, is greater than a threshold. In a representative experiment in Fig. 7, we find that FreeAR stores four representative object templates (green boxed) for an object out of ≈ 3,000 frames.

Fast template matching: When a secondary loses an object due to abrupt motion, FreeAR first retrieves the templates associated with that object. It then waits for the device to be relatively motionless to ensure that a non-blurry camera frame is captured, appropriate for template matching, which is then performed (as described earlier in this subsection) and repeated for each template. The patch with the lowest sum of square differences (also lower than a threshold) is chosen as the recovered object location. Fig. 7 shows that the right most template in the green box matches the current frame and successfully places the bounding box in step (4). Our experiments show that template matching takes ≈ 60ms with high-precision object recovery (*e.g.,* IOU ≈ 0.7 − 0.8).

3.4.2 Location-based local repair (LLR): An object's appearance may change from how the secondary remembers it; in other words, visual features of the templates can deviate from the object features in the current view (*e.g.,* when viewing an object from a different angle). In such cases, VLR may fail, and to handle such cases, we imbibe a second layer of local repair. Here, we remember *where* the objects were before they were lost. With this location information, we can then recover the virtual overlays.

Straw-man methods: An AR device can use DNNs to recover the object locations. However, DNN executions drain significant energy from AR devices (see Table 2); hence, this method fails to serve as a good candidate for location based local repair.

Key ideas: During the sync phase, a secondary device not only collects templates for VLR, but also registers and updates an object's location in its own coordinate system. Leveraging FreeAR's SLAM and DNN cooperation method (§ 3.2), these registered object locations are in the 3D world, which is stationary. If VLR fails, FreeAR triggers LLR by projecting the last known 3D object location (x, y, z) to the secondary's view, making corrections based on the device pose (R, t) (discussed in § 3.3) and Eq. 2. Fig. 8 illustrates this simple, yet effective process in recovering a virtual overlay (orange bounding box). Note that FreeAR will not draw the virtual overlay if the object is not within the secondary's current view (Eq. 2).

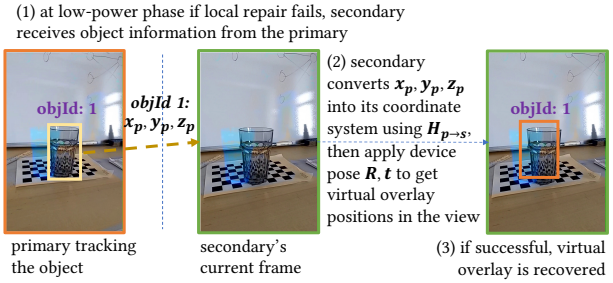


Figure 9: Primary-assisted Collaborative Repair (PCR): the primary shares object information with a secondary which can recover the position of the virtual overlay in its view.

3.4.3 Primary-assisted collaborative repair (PCR): In a few cases, LLR may also fail (*e.g.,* if the object has been moved) and we provide a third repair layer to try to prevent SLAM and DNNs executions on the secondaries. We observe that the primary, which still runs SLAM and DNNs, can share object (and virtual overlay) information with a secondary experiencing object loss; the latter can then use the information to recover the virtual overlays in the local view.

Prior methods: MARVEL [13] determines object locations offline and registers them on an edge infrastructure which shares these locations with AR devices to aid recovery of virtual overlays. EdgeSharing [36] uses DNNs running on an edge to determine object locations, and shares them with the user devices.

Challenges: We cannot directly apply these methods because of the lack of offline surveys and edge infrastructure. The primary is in motion (along with the secondaries), and thus cannot directly play the role of the fixed edge assumed in the prior systems.

Key ideas: The primary device, using SLAM and DNNs cooperation, estimates an object's 3D locations (x_p, y_p, z_p) and shares them with a secondary that has lost track of the object. The latter uses $H_{p\to s}$ (from §3.1) to transform the 3D points into its own coordinate system (x_s, y_s, z_s). It then uses its recent device pose R, t (updated by § 3.3) and Eq. 2 to project the 3D points onto its 2D view and draw the virtual overlay. Fig. 9 illustrates PCR's processes.

3.5 Fast and seamless global fallback

In extremely rare cases, all of the repair methods may fail, and here FreeAR will start a new time slice and fallback to the synchronization phase, *i.e.,* all devices will execute SLAM and DNNs again. This is outside normal invocations of this phase either periodically or when the primary's battery drops by a certain threshold.

Challenges: Re-initializing SLAM naively can either cause it to reset, or to fail to reconnect with its previous state and crash. Resetting SLAM from a cold state clearly misses on opportunities to leverage previously stored data, and incurs high latency. However, naively attempting to merge with SLAM's previous state usually fails because SLAM expects a continuous stream of data from the camera and IMUs, and the secondary device has not been running SLAM during a steady-state phase.

Key ideas: We use an existing technique in SLAM, called loop closure [33, 42] in FreeAR, to "trick" SLAM into merging the information from the current and previous synchronization phases. Loop closure is normally used to determine when a user re-visits a previously seen area (*e.g.,* by walking in a loop). We exploit loop closure to give SLAM the impression that the device was simply lost for

a while (*i.e.,* during the steady state phase), and is now re-visiting the area from the previous synchronization phase. Loop closure here helps stitch these two worlds (from the current and previous synchronization phases) together, allowing for their smooth re-connection. This speeds up FreeAR's re-synchronization, the start of a new time slice, and the transition to the next steady-state phase.

Primary rotation policy. At the beginning of each time slice, all AR devices share their residual energy = remaining battery (%) × battery capacity (Ah), and the device with the maximum residual energy is chosen as the primary (an election can be used [63]). When the primary's residual energy drops significantly (*e.g.,* compared to the secondaries), a new time slice is initiated to choose a new primary device using the maximum residual energy criterion. We defer more sophisticated policies (*e.g.,* considering computational resources, network topology or objects in the FoVs) to future work.

Handling group change dynamics. A new device joining an existing AR session in the middle of a time slice has to synchronize its coordinate system only with the primary device. After the synchronization phase is done, the new device can enter the low-power mode. A secondary device can leave the AR group at any time; when a primary device leaves the AR ecosystem, a new primary is chosen (based on residual energy as before), and a time slice initiated.

4 Implementation

Platforms: FreeAR is implemented on smartphones running Android 11 (Google Pixel 4, Google Pixel 4a 5G, Google Pixel 5, and Samsung S21). We use VINS-Mobile [33], TensorFlow [60], and OpenCV [11] libraries to implement SLAM, object detection and tracking, and PnP synchronization and image processing, respectively. Our code is available at the FreeAR website [6].

Module implementation: There are two main parts of FreeAR viz., the main UI in Android (MainActivity) in Java, and the key SLAM class (ViewController) in C++. The *coordinate system synchronization* and *SLAM and DNN coordination* work inside ViewController to retrieve and match keyframes, and share object information, respectively. *Lightweight localization*'s IMU tracking is implemented inside MainActivity to access and process IMU sensor inputs. *Repair methods and global fallback* are implemented inside ViewController to access required inputs.

Inter-device information sharing: We use Android's WiFi P2P [3] to implement inter-device communications. We use Java's client-server TCP sockets for unicast, and UDP sockets for broadcast.

Logging: To estimate the power consumption, we read voltage and current variables of Andorid's BatteryManager to obtain the battery's voltage (mV) and current (μA), then calculate power as $Power = Voltage * Current$ in Watts. To compute the virtual overlay's IOU, we log information about each tracked object shared by the primary: timestamp, bounding box, class. At the secondaries if there is a matched local virtual overlay, we also log: (matched) local object id and (matched) local bounding box.

Baselines: We implement the following baselines:

MARVEL [13] Since MARVEL is not open sourced, we implement a faithful reproduction as follows. We build an offline map of the ecosystem with its objects using SLAM [33]. Then, the lightweight AR client app localizes itself online in this pre-built map on the edge server using our implementation of Eqs. 6-8 from [13]. We call this *Centralized Localization.*

MARLIN [7] We obtained MARLIN's source code, but replaced the DNN models (*e.g.,* Tiny YOLO) with the newer and more accurate EfficientDet model [58] trained on the COCO 2017 dataset [35]. FreeAR also uses this DNN model for fair comparison. We modify MARLIN slightly to achieve collaborative AR among multiple devices as follows. The primary device runs MARLIN and shares the 2D virtual overlays (rather than 3D coordinates as in FreeAR) and object classes with the secondary devices, which also run MARLIN. The secondaries only display a virtual overlay if a locally detected object matches the object class sent by the primary (IOU > 0.3).

Multi-user SLAM or MU-SLAM runs SLAM on all the AR devices all the time, but only the primary runs DNNs to get the virtual overlay positions and shares these with the secondaries. The secondaries also perform coordinate system synchronization with the primary, similar to FreeAR. Inspired by SPAR [50], this represents a direct application of collaborative SLAM [9, 28, 39, 52] to AR.

Vanilla runs SLAM and DNN continuously on all the devices and performs coordinate systems synchronization, similar to FreeAR, to achieve collaborative AR without considering energy issues.

5 Evaluation

In this section, we present our evaluations of FreeAR. We first list our metrics of interest and then present our results.

5.1 Evaluation metrics

Power consumption: We log the power consumption on each phone, when it is running only the Android OS, the camera, and screen display (brightness set to 70%); this is called the base power, $power_{base}$. In our evaluations, we run FreeAR or a baseline and log the total power $power_{total}$. We estimate $power_{app} = power_{total} - power_{base}$, thereby isolating the power consumed by AR.

IOU accuracy: The Intersection Over Union (IOU) [12] lies between 0 and 1 and captures whether a virtual overlay (bounding box) is where it should be on a display. The IOU is defined as $\frac{O \cap G}{O \cup G}$, where G is the ground truth bounding box and O is the bounding box displayed by FreeAR or a baseline. The larger the IOU, the better. We report the average IOU over all analyzed frames. To obtain the ground truth bounding boxes and object classes, we execute the largest EfficientDet DNN model, EfficientDet-7x. The IOU is non-zero only if the ground truth object class matches the class output by the primary/secondary's DNN. This measures whether the secondary is indeed highlighting the correct object.

5.2 End-to-end evaluations of FreeAR

We first provide our holistic evaluations of FreeAR in various scenarios and compare its performance with the baselines.

5.2.1 Holisitic results. We run 15 AR experiments, each with 2 to 5 volunteers using the devices mentioned in § 4. In each experiment, AR users follow different trajectories, move devices differently (semi-stationary to constant motion), with one or two (bottle and/or cup) objects of interest in a 20 m × 20 m space, where they are always in proximity (*e.g.,* < 10 m from each other); in this space, objects do not appear too small in the FoVs. These experiments were done on weekends to avoid background WiFi traffic from non-AR users and encompass all scenarios described in §5.2.2 to §5.2.7.

Fig. 10 shows the CDF of FreeAR and the four comparison baselines for power consumption and IOU accuracy. As can be seen, the

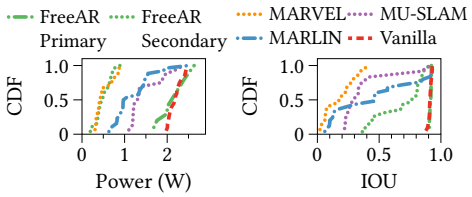


Figure 10: On average, FreeAR consumes low power (≈ 0.5 W), comparable to MARVEL's and only one-fourth of the vanilla scheme's power. FreeAR also improves IOU accuracy by 78%, 64% and 43% compared to MARVEL, Multi-user SLAM and MARLIN, respectively.

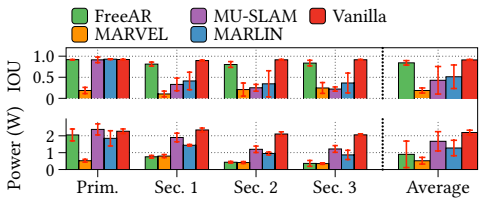


Figure 11: Four semi-stationary users track two objects from similar FoVs: FreeAR improves 78% IOU but increases 42% power over MARVEL which has edge infrastructure. FreeAR both reduces 30% (46%) power *and* improves 39% (49%) IOU compared to MARLIN (Multi-user SLAM or MU-SLAM).

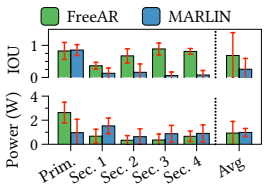


Figure 12: When five users track an object from different FoVs, FreeAR outperforms MARLIN with 63% better IOU, with comparable power.

primary device consumes more power (similar to Vanilla), while the secondary devices save significant power.

Compared to MARLIN, FreeAR reduces power by 46% *and* improves IOU accuracy by 43%. MARLIN consumes high power (≈ 1.0 W) because AR devices continuously execute DNNs. Because of its low-power mode, FreeAR consumes only 0.5 W on average. Furthermore, MARLIN only achieves an average IOU of 0.46 because it only shares 2D virtual overlay positions; since users have different views, the overlays can easily be dislocated from their associated physical objects. On the other hand, FreeAR leverages both DNNs and SLAM to share 3D virtual overlays for consistency among different views, leading to a 43% increase in IOU over MARLIN.

Compared to MARVEL, FreeAR consumes 18% more power but improves IOU by 78%. MARVEL consumes low power because it offloads heavy computations to the edge, unlike FreeAR. However, MARVEL only achieves an IOU accuracy of 0.2 because it fully depends on centralized localization at the edge, and has no repair mechanisms. If many devices send frames to the edge for localization, they experience long delays and thus misplace their virtual overlays. In contrast, FreeAR's distributed localization quickly updates devices' poses or triggers fast local repair upon failure, achieving an IOU accuracy of 0.8, on average.

Compared to Multi-user SLAM, FreeAR consumes 60% less power and improves IOU by 64%. With MU-SLAM, secondaries indirectly compute the virtual overlay poses from object information shared by the primary. It is thus susceptible to synchronization or localization errors which result in a low IOU accuracy of 0.3, on average. FreeAR employs effective repair mechanisms, leading to an IOU accuracy of 0.8. Further, secondaries in FreeAR run neither SLAM nor DNNs extensively, thus consuming much lower power.

Compared to vanilla, FreeAR consumes just one-fourth of the power but takes only a 12% hit in terms of IOU accuracy.

FreeAR's synchronization phase takes ≈ 2 - 4 minutes and consumes 2.3 - 3.0 W of power. During this phase, the secondary devices collect object templates for later use in VLR. These devices also repeatedly send key frames to the primary to perform coordinate system synchronization, until successful. The greater the number of AR devices, the longer this phase takes. This phase consumes high power, because AR devices have to run SLAM, DNNs, and P2P communications, but will last only for a short duration. The primary consumes an additional 0.5 W of power during synchronization (*e.g.,* running PnP solver [32]).

FreeAR's communication overhead is low. Because the AR devices are in proximity, the WiFi P2P links are measured (using iPerf [18]) to operate at bandwidths of 100 - 300 Mbps. During synchronization, key frames (each with ≈ 100-200 KB size taking ≈ 50ms transmission time) are periodically sent from each secondary to the primary every $N \times 300$ ms where N is the number of AR devices; as N increases, the frequency is decreased to avoid contention at the primary. $H_{p \to s}$ (a 4x4 matrix defined in §3.1 and of size ≈ 170 B), and the 3D object representation anchoring the virtual overlay (≈ 600 B per object), are much smaller and take < 1 ms of transmission time. Notification messages (*e.g.,* primary device selection) are carried via UDP broadcasts to all N devices. Therefore, for moderate values of N (*e.g.,* < 20), FreeAR can perform synchronization and allow devices to exchange information reasonably quickly.

FreeAR's memory footprint is small. The memory requirements of FreeAR are predominantly due to the baseline SLAM framework [33], which primarily stores a collection of key frames along the AR user trajectory. Continuously walking in a 20 m × 20 m space for 2 minutes results in 250 key frames and 0.6 GB of memory. This would be incurred in any AR system that runs SLAM. In addition to this memory consumption due to SLAM, FreeAR stores $H_{p \to s}$ (≈ 170 B) for each secondary, and the 3D representation (≈ 600 B) and multiple templates (≈ 50 B each) for each object. For all the scenarios, FreeAR's memory footprint beyond SLAM is < 5 KB (which is negligible in comparison to SLAM).

5.2.2 Semi-stationary scenarios. In this experiment, four volunteers holding devices (the primary is a Samsung S21, Secondary 1 is a Google Pixel 5, and Secondaries 2 and 3 are Google Pixel 4), walk around an area of 20 m × 20 m and after synchronization, point their devices with similar FoVs to look at two objects (bottle and cup). At steady state, remaining for 2 minutes, they move the devices around 5-10cm and/or rotate them by 5-10 degrees, keeping the objects in their FoVs. This reflects AR use cases where the users are semi-stationary to interact with the virtual overlays. We perform 5 trials, each with FreeAR and then with the four baselines. Fig. 11 shows the power consumed by and IOU accuracy of each user.

FreeAR **improves the IOU by 78% over MARVEL.** On the secondary devices, FreeAR consumes power comparable to MARVEL because with both methods, these devices operate in the low-power mode, mostly performing IMU-based tracking. As mentioned earlier, FreeAR achieves significantly better IOU because it utilizes the object locations found by the DNNs during the sync phase, rather than relying on MARVEL's centralized localization on the edge. The

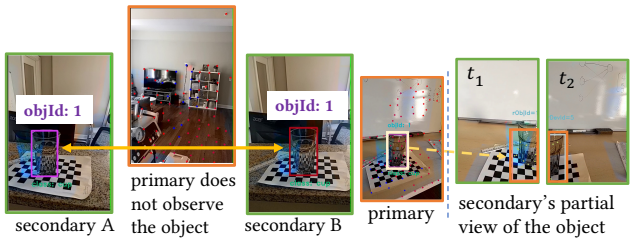


Figure 13: In scenarios (a) where primary does not observe the object, or (b) where secondary partially observes the object, the secondaries can still have consistent virtual overlays.

average power, however, is higher with FreeAR, since in MARVEL the edge does the heavy computation (in our implementation of MARVEL, both the primary and secondaries offload the computation). We again point out (a) FreeAR eliminates the need for the edge server, and (b) we only have four devices in our experiments. If the number of devices leveraging the primary's computations is higher, this power is amortized and the penalty will be much smaller.

Compared to MARLIN, FreeAR both reduces the power (by 30%) and improves IOU (by 39%). By not considering the 3D positions of the devices and objects, MARLIN frequently fails to place the virtual overlays consistently at the primary and secondaries (*e.g.,* when the users view objects from slightly different distances). Further, although MARLIN can save power by not running SLAM, it often triggers DNNs, consuming higher power than FreeAR's secondaries, which run neither DNNs nor SLAM in steady states.

Compared to Multi-user SLAM, FreeAR both reduces power by 46% and improves IOU by 49%. Secondaries in MU-SLAM draw virtual overlays simply based on information from the primary. In contrast, FreeAR's secondaries use effective repair mechanisms to quickly recover lost virtual overlays, leading to significant improvements in IOU accuracy compared to MU-SLAM. Secondaries in MU-SLAM run SLAM consuming high power, while those in FreeAR run only lightweight methods (no SLAM or DNNs).

Compared to vanilla, FreeAR has a marginally lower IOU (by 8%) but reduces significant power consumption (by 59%).

5.2.3 Scenarios where users have different FoVs. In this experiment, we add a fifth volunteer with a Google Pixel 4a 5G (as Secondary 4) to the previous setup; after the sync phase, the users track a single object on a table from 5 different FoVs with semi-stationary motion. We run 1 trial and compute the average power and IOU for the 10-minute duration of the experiment (with samples at the granularity of each frame). We focus on comparing FreeAR with MARLIN because the latter shows acceptable performance in terms of power and IOU, and both are infrastructure-free.

Compared to MARLIN, when users have different FoVs, FreeAR improves the IOU by 63%. Fig. 12 shows that MARLIN takes a significant hit due to mismatches between the virtual overlays on the primary and secondaries, arising due to the different FoVs of the users; we see that users are disconnected from the AR experience for significant times (as shown by their low IOUs). In contrast, FreeAR tracks the object with high IOU, because it (a) synchronizes the 3D coordinate systems to ensure consistency in spite of the different FoVs and (b) adapts quickly to user motion through device localization (§ 3.3) and repair methods (§ 3.4).

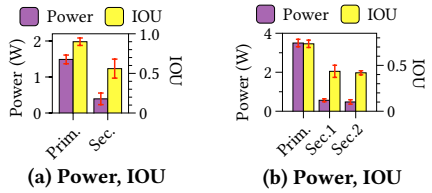
From Fig. 12, FreeAR's user Secondary 1 (Sec. 1) is seen to experience a lower IOU than the others. A log analysis shows that this user suddenly moves the device during the sync phase, causing the object ID to be changed; thus, VLR does not create the proper object templates and later cannot recover from failure. However, this user uses LLR and PCR (object 3D coordinates) to recover the virtual overlay (bounding box), and achieves an IOU of 0.36, on average. This is lower than that of the other users who run VLR and thus achieve IOUs of 0.7-0.8; however, importantly, we see that FreeAR achieves a higher IOU than MARLIN, even for this user.

5.2.4 Secondary Device in Constant Motion. In this experiment, we run FreeAR with one primary and one secondary device, which track two objects in their FoVs. The primary user is semi-stationary but the secondary moves or rotates back and forth (*i.e.,* until the left object nears the left screen edge or right object reaches the right screen edge); at this point, the user immediately moves/rotates the device in the opposite direction. We run 3 trials, each lasting 2 minutes. In Figs. 14a, we see that in this challenging scenario of a constantly moving secondary device, the IOU drops to 0.56 (compared to 0.7-0.8 in the previous semi-stable experiments) which is still considered to be very good for object tracking [34]. Because of the motion, stable object templates can rarely be collected, and VLR is mostly unsuccessful. However, because the secondary user still has the 3D coordinates of the object used by LLR or PCR, it achieves good IOU accuracy with its virtual overlays.

5.2.5 Collaborative AR with users walking in circle. In this experiment, three users (one primary and two secondary) slowly (1m/s) and continuously (walk 1m and momentarily stop and then continue), walk around a table while always keeping one object in the FoVs. This is one of the most challenging scenarios because (a) object tracking can easily fail because the object appearance changes quickly due to changing FOVs, invoking local repairs often and (b) drift accumulates in the IMU-based translational tracker. Still, Fig. 14b shows that FreeAR's performance is quite good, as the secondaries achieve IOUs of ≈ 0.4 (considered satisfactory [34]) with approximately 0.4 W of power. Compared to MARVEL and MARLIN in § 5.2.2, in this more complex scenario, FreeAR (i) achieves a better average IOU of 56% compared to MARVEL, and (ii) consumes 52% less power and achieves 12% higher IOU compared to MARLIN.

5.2.6 Scenarios where the primary and secondaries do not observe the same set of objects. In this experiment, one primary and two secondary devices track one cup in their views. In Fig. 13a, when the primary changes its FoV, away from the object, the secondaries can still have consistent overlays. Towards this, secondary A (s_A) registers the object in its own coordinate system and shares the object information with the primary, which applies $H_{s_A \rightarrow p}$ to compute (x_p, y_p, z_p) of the object in the primary's coordinate system. Then, these 3D coordinates are forwarded to secondary B (s_B), who applies $H_{p \rightarrow s_B}$ to have the object's 3D coordinates in its own coordinate system. Finally, s_B maps the 3D points onto its own view using its device pose (R_{s_B}, t_{s_B}) and Eq. 2 to place the virtual overlay.

5.2.7 Scenarios where the secondary partially observes the object. This experiment setup is similar to the previous one, except the secondary only partially observes the cup. These are challenging scenarios for VLR as the collected templates likely cover a full view of the object (*e.g.,* DNNs usually consider full visual features of an



(a) Power, IOU (b) Power, IOU

Figure 14: In FreeAR, (a) a constantly mobile secondary and (b) three users constantly walking around achieve good IOU accuracy due to efficient repair methods.

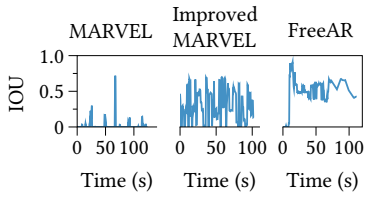
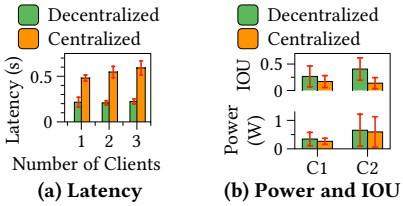


Figure 15: FreeAR outperforms baselines in coordinate system synchronization with improper frame avoidance and variance suppression.



(a) Latency (b) Power and IOU

Figure 16: FreeAR's decentralized localization reduces latency and improves accuracy with low power. C1 and C2 are the clients.

object [56]). Therefore, the templates used to match the patches of an input image will likely fail because of the mismatched appearances. Fig. 13b shows that with LLR and PCR, FreeAR can still function despite the partial views. At t_1 using PCR, the primary shares 3D "cup" locations with the secondary which then uses Eq. 2 to project the 3D coordinates into its 2D view. Eq. 2 automatically determines that some of those 3D coordinates (*e.g.,* right parts of the cup) appear outside the view, and thus should not project them onto the display. The virtual overlay (orange bounding box) is then drawn accordingly by considering only those 3D points that are inside the FoV. Subsequently, the secondary registers these 3D coordinates of the object, and at t_2 uses LLR to recover the virtual overlay after the device has moved to the right. The virtual overlay then is drawn, again, over the 3D points that are inside the FoV.

5.3 Component-wise benchmarks

Here, we evaluate FreeAR's four components from § 3 individually.

FreeAR's coordinate system synchronization improves IOU by 40% compared to MARVEL. In this experiment, we use two smartphones (Google Pixel 4 and Samsung S21) which establish SLAM, synchronize their coordinate systems, and track one object in their FoVs. Specifically, one (primary) device runs DNNs and shares a virtual overlay to another (secondary, not running DNNs) device, in order for the latter to project the overlay onto its view. Fig. 15 shows a timeline of the secondary's IOUs. We see that MARVEL exhibits very low IOU accuracy, because it does not consider the freshness of the keyframes. It repeatedly picks improper keyframes from the primary's history, based only on visual feature similarity, leading to poor coordinate system synchronization.

To show how considering freshness improves coordinate system synchronization, we incorporate the exact improper frame avoidance technique (§ 3.1) of FreeAR, within MARVEL; we call this *Improved MARVEL*. We find that the IOU accuracy improves significantly in Improved MARVEL, and there are very few outliers in coordinate system synchronization. However, Improved MARVEL finds a good synchronization result initially, but gives it up too quickly and replaces it with a worse result on subsequent synchronization attempts, lacking FreeAR's variance suppression. In contrast, FreeAR (on the right side of Fig. 15) quickly achieves a high IOU by using improper frame avoidance, and retains this high level for a long time using variance suppression, thereby improving IOU accuracy by 40% on average over Improved MARVEL.

FreeAR's lightweight localization copes better with user motion and achieves up to 66% better IOU, compared to centralized localization (MARVEL). In this experiment, multiple (up to 3) devices move slowly from left to right ≈ 5 cm, stop, and then move right to left, and keep going for 2 minutes. This reflects a case where users stop to interact with virtual overlays and move to change FoVs and interact with them again. At steady state, with FreeAR, the secondary devices perform lightweight, distributed localization (§ 3.3). The baseline is MARVEL's centralized localization, wherein all users offload visual data to the edge server, to find their locations in the offline map built a priori. We measure the latency from when the localization request is made to when the result is received. Fig. 16a shows that FreeAR experiences ≈ 0.22*s* latency on average per device, when there are 1 to 3 users. On the other hand, MARVEL latency increases from 0.48*s* to 0.59*s* as the number of users grows from one to three, respectively. This is because MARVEL sends camera frames to the edge server over a WiFi P2P link, which can become a bottleneck due to congestion, whereas FreeAR performs lightweight localization locally across devices in parallel. In Fig. 16b, we run experiments with two clients and see that FreeAR's low latency improves the IOU accuracy by 36% and by 66%, for secondaries 1 (C1) and 2 (C2) respectively, with negligible power overhead, compared to centralized localization.

FreeAR's VLR recovers lost tracked objects 29× faster than using DNNs directly for local repair. In this experiment, we have a pair of primary and secondary devices tracking one object in their FoVs. At steady state, the secondary suddenly changes its FoV (*e.g.,* turns away) and then returns to the original FoV. We run 10 trials for each method. We measure the time from when the object first re-appears in the FoV, to when its virtual overlay appears on the screen, *i.e.,* how long local repair takes.

From Fig. 17, we find that FreeAR's VLR spends (A) 0.32s waiting for the device to become quasi-stationary ($t_3 - t_2$ in Fig. 17 (top)) , a portion of which (0.06s) is spent on performing template matching, and (B) 0.08 s on coordinating the object between the primary and secondary's views ($t_4 - t_3$ in Fig. 17 (top)). In contrast, a baseline of triggering DNNs repeatedly to search for the re-appearance of the object takes 1.85*s* ($t_3 - t_2$ in Fig. 17 (bottom)). Subsequently, because DNN only understands 2D object coordinates, it waits until the object's 2D position in the secondary's display is similar to the 2D coordinates mapped from the primary, for confidence (IOU > 0.3) that it is highlighting the same object. These two processes take about 14.95*s* in total ($t_4 - t_2$ in Fig. 17 (below)) in this trial. We observe similar behaviors over the 10 trials and find that on average $t_4 - t_2$ takes 16.38*s* for DNNs which is 29× over FreeAR (only 0.57*s*).

However, as one might expect, DNN offers object detection with higher IOU both before (≈ 0.9 before t_1) and after repair (≈ 0.9 after t_4) than FreeAR's VLR which runs object template matching (≈ 0.8

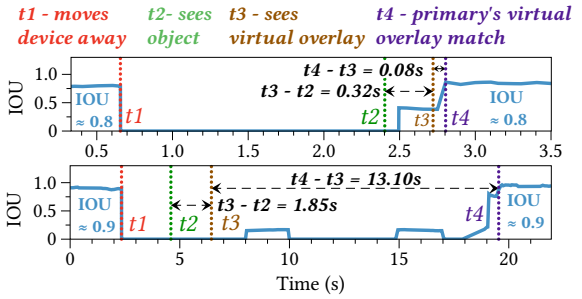


Figure 17: FreeAR**'s VLR (top) quickly recovers the virtual overlay after an object re-appears in the user's FoV (t_2), 29× faster than direcly using a DNN for local repair (bottom).**

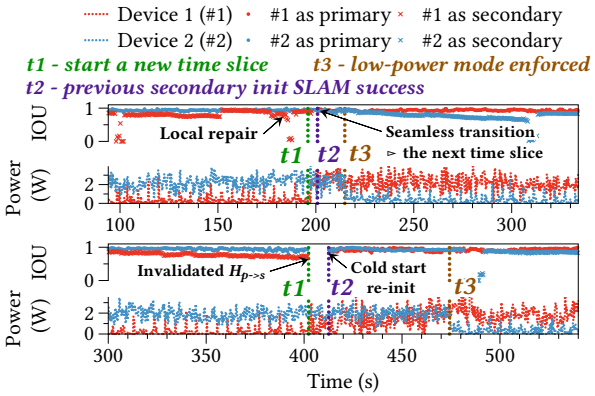


Figure 18: FreeAR**'s global fallback (top) transitions to a next time slice 4.4 × faster than using a cold start (bottom).**

before t_1 and after t_4) (Fig. 17). Our measurements also show that FreeAR's VLR consumes less power than DNN (0.84 W vs 1.41 W).

FreeAR**'s seamless global fallback enables transition to a new low-power steady state** 4.4× **faster than with a cold-start.** To show this, with two devices, we highlight what happens during global fallback in Fig. 18. The primary (Device 2) initiates a new time slice because of a present energy drop. With FreeAR's seamless global fallback, the secondary (Device 1) re-instantiates SLAM and quickly succeeds in coordinate system synchronization with Device 2, leveraging stored data from the previous synchronization instance. Now, Device 1 is promoted to be the new primary in the new time slice, and Device 2 transits to a low-power mode.

From Fig. 18 (top), we see that with FreeAR, from when a new time slice is initiated (t_1) to when the secondary device re-instantiates SLAM (t_2), both the devices still have high IOUs because they maintain coordinate system synchronization and thus, are able to use shared object information. Furthermore, the time from t_2 to until the low-power transition of Device 2 (t_3) is 13.9s. In contrast, with a cold start of SLAM, we see in Fig. 18 (bottom) that from t_1 to t_2 the coordinate systems are not synchronized and IOUs fall to zeros, and the time from t_2 to t_3 is 61.7s. To summarize, FreeAR's transition to a new time slice is seamless and 4.4× faster.

6 Related work

Single-user AR: Several works study cloud or edge-based AR for a single user [14, 24, 26, 37, 49, 67]. They mainly focus on virtual overlay placement using DNNs or other computer vision methods, without concerns of power. MARVEL [13] and MARLIN [7] do focus on energy of mobile AR. Since we discussed them extensively in § 5, we omit further discussion here in the interest of space.

Multi-user AR: CARS [68] and COllabAR [38] rely on the cloud/edge for collaborative AR. AVR [48] and SPAR [50] use SLAM for localization like we do. AVR shares sparse point clouds between multiple vehicles. SPAR shares environment data between multiple mobile devices, but runs SLAM continuously on all devices and assumes that the virtual overlay locations are provided in advance (*i.e.,* no DNNs are running). Neither considers energy drain; in AVR's vehicles, for example, energy is not a major concern due to plentiful on-board power sources. In contrast, FreeAR focuses on infrastructure-free AR with energy limitations.

Localization: SLAM-based localization is used in off-the-shelf AR systems [8, 21, 41] to enable sharing of virtual overlay positions. Edge-SLAM [9] and EdgeSharing [36] rely on edge infrastructure for SLAM processing, unlike in FreeAR. Research on multi-user SLAM [1, 70] neglect AR aspects such as virtual overlay positioning. We significantly go beyond SLAM used in the robotics community [33] by adding multi-user capabilities with energy savings for FreeAR.

7 Discussion and future work

FreeAR is a practical collaborative mobile AR framework even with network infrastructure (*e.g.,* 5G), because it leverages peer to peer (P2P) low traffic (occasional frames and 3D coordinates only) connections between AR users. Congestion, network induced delays, or overload on the infrastructure can cause high latency that disrupt the AR experience among users [5]; therefore, augmenting such experiences with P2P links can help. In the future, we will consider cases where there could be multiple primaries, and hybrid edge-P2P systems to expand the spatial range of the AR experience. We also note that FreeAR can perform additional optimizations on the primary, such as those in MARLIN [7], to further save power.

8 Conclusions

Our work sets out to answer a question applying to many practical cases: Can we enable a rich AR experience in infrastructure-free settings, running natively on user devices, without significant energy drain? Our system FreeAR is proof that this goal can be within our reach using collaborative time slicing to reuse/reduce compute heavy tasks such as DNNs/SLAM. While conceptually easy to explain, achieving this in practice induces key synchronization, consistency, and recovery challenges in decentralized AR operations that we address in FreeAR. We showed that FreeAR reduces the power consumption of users by up to 60% compared to state-of-the-art AR systems, while also improving the detection accuracy of objects in the real world by nearly 78%. FreeAR thus can enable a low power framework that can allow users to engage in an AR experience on the fly, without needing infrastructure support.

Acknowledgments

We thank the anonymous reviewers and shepherd for their valuable comments, from which this paper greatly benefited. We also thank the volunteers who participated in our user study. This work was partially supported by the NSF grants CPS 1544969, CAREER 1942700, CNS 2106214. We extend special thanks to CMU's CyLab and WiSE Lab for the support during the development of this work.

References

[1] Fawad Ahmad, Hang Qiu, Ray Eells, Fan Bai, and Ramesh Govindan. 2020. Carmap: Fast 3d feature map updates for automobiles. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. USENIX Association, Santa Clara, CA, 1063–1081.

[2] Adel Ahmadyan, Liangkai Zhang, Artsiom Ablavatski, Jianing Wei, and Matthias Grundmann. 2021. Objectron: A Large Scale Dataset of Object-Centric Videos in the Wild With Pose Annotations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[3] Android. [n. d.]. Wi-Fi Direct (peer-to-peer or P2P) overview. https://developer.android.com/guide/topics/connectivity/wifip2p.

[4] Android. 2022. Android Sensor: Linear Acceleration. https://developer.android.com/reference/android/hardware/Sensor#TYPE_LINEAR_ACCELERATION. Accessed: 2022-09-30.

[5] Kittipat Apicharttrisorn, Bharath Balasubramanian, Jiasi Chen, Rajarajan Sivaraj, Yi-Zhen Tsai, Rittwik Jana, Srikanth Krishnamurthy, Tuyen Tran, and Yu Zhou. 2020. Characterization of Multi-User Augmented Reality over Cellular Networks. In *2020 17th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. 1–9. https://doi.org/10.1109/SECON48991.2020.9158434

[6] Kittipat Apicharttrisorn, Jiasi Chen, Vyas Sekar, Anthony Rowe, and Srikanth V. Krishnamurthy. 2022. FreeAR Website. https://sites.google.com/view/infra-free-ar/home.

[7] 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 *Conference on Embedded Networked Sensor Systems* (New York, New York) *(SenSys)*. ACM, New York, NY, USA.

[8] Apple. [n. d.]. Creating a Multiuser AR Experience. https://developer.apple.com/documentation/arkit/creating_a_multiuser_ar_experience.

[9] Ali J. Ben Ali, Zakieh Sadat Hashemifar, and Karthik Dantu. 2020. Edge-SLAM: Edge-Assisted Visual Simultaneous Localization and Mapping. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services (MobiSys '20)*. ACM, New York, NY, USA.

[10] S. Benhimane and E. Malis. 2004. Real-time image-based tracking of planes using efficient second-order minimization. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[11] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[12] L. Cehovin, A. Leonardis, and M. Kristan. 2016. Visual Object Tracking Performance Measures Revisited. *IEEE Transactions on Image Processing* 25, 3 (March 2016), 1261–1274. https://doi.org/10.1109/TIP.2016.2520370

[13] Kaifei Chen, Tong Li, Hyung-Sin Kim, David E Culler, and Randy H Katz. 2018. MARVEL: Enabling Mobile Augmented Reality with Low Energy and Low Latency. In *Conference on Embedded Networked Sensor Systems (SenSys)*. ACM, 292–304.

[14] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. 2015. Glimpse: Continuous, real-time object recognition on mobile devices. *ACM SenSys* (2015).

[15] Titus Cieslewski, Siddharth Choudhary, and Davide Scaramuzza. 2018. Data-Efficient Decentralized Visual SLAM. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2466–2473. https://doi.org/10.1109/ICRA.2018.8461155

[16] Alexander Cunningham, Vadim Indelman, and Frank Dellaert. 2013. DDF-SAM 2.0: Consistent distributed smoothing and mapping. In *2013 IEEE International Conference on Robotics and Automation*. 5220–5227. https://doi.org/10.1109/ICRA.2013.6631323

[17] Alexander Cunningham, Manohar Paluri, and Frank Dellaert. 2010. DDF-SAM: Fully distributed SLAM using Constrained Factor Graphs. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 3025–3030. https://doi.org/10.1109/IROS.2010.5652875

[18] Jon Dugan, Seth Elliott, Jeff Mah, Bruce A.and Poskanzer, and Kaustubh Prabhu. 2022. iPerf - The ultimate speed test tool for TCP, UDP and SCTP. https://iperf.fr/.

[19] Dorian Galvez-López and Juan D. Tardos. 2012. Bags of Binary Words for Fast Place Recognition in Image Sequences. *IEEE Transactions on Robotics* (2012).

[20] Google. [n. d.]. MediaPipe Objectron. https://google.github.io/mediapipe/solutions/objectron.html.

[21] Google. 2018. Share AR Experiences with Cloud Anchors. https://developers.google.com/ar/develop/java/cloud-anchors/cloud-anchors-overview-android.

[22] Google. 2022. Google Just a Line. https://justaline.withgoogle.com/. Accessed: 2022-09-30.

[23] Yongjie Guan, Xueyu Hou, Nan Wu, Bo Han, and Tao Han. 2022. Realtime 3D Object Detection for Headsets. *arXiv preprint arXiv:2201.08812* (2022).

[24] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2014. Towards wearable cognitive assistance. *ACM MobiSys* (2014).

[25] Seongwon Han, Sungwon Yang, Jihyoung Kim, and Mario Gerla. 2012. EyeGuardian: A Framework of Eye Tracking and Blink Detection for Mobile Device Users. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems and Applications* (San Diego, California) *(HotMobile '12)*. Association for Computing Machinery, New York, NY, USA, Article 6, 6 pages. https://doi.org/10.1145/2162081.2162090

[26] Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. 2016. Low Bandwidth Offload for Mobile AR. *ACM CoNEXT* (2016).

[27] Kaleb. 2022. FSensor. https://github.com/KalebKE/FSensor. Accessed: 2022-09-30.

[28] Marco Karrer, Patrik Schmuck, and Margarita Chli. 2018. CVI-SLAM—Collaborative Visual-Inertial SLAM. *IEEE Robotics and Automation Letters* 3, 4 (2018), 2762–2769. https://doi.org/10.1109/LRA.2018.2837226

[29] Kris Kitani. [n. d.]. Camera Matrix. http://www.cs.cmu.edu/~16385/s17/Slides/11.1_Camera_matrix.pdf.

[30] Laurent Kneip, Hongdong Li, and Yongduek Seo. 2014. UPnP: An Optimal O(n) Solution to the Absolute Pose Problem with Universal Applicability. In *Computer Vision – ECCV 2014*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.).

[31] Steven LaValle. [n. d.]. *Virtual Reality*. Cambridge University Press.

[32] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. 2009. EPnP: An Accurate O(n) Solution to the PnP Problem. *Int. J. Comput. Vision* (2009).

[33] Peiliang Li, Tong Qin, Botao Hu, Fengyuan Zhu, and Shaojie Shen. 2017. Monocular Visual-Inertial State Estimation for Mobile Augmented Reality. In *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*.

[34] Pengpeng Liang, Yifan Wu, Hu Lu, Liming Wang, Chunyuan Liao, and Haibin Ling. 2018. Planar Object Tracking in the Wild: A Benchmark. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*.

[35] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. *CoRR* abs/1405.0312 (2014). arXiv:1405.0312 http://arxiv.org/abs/1405.0312

[36] Luyang Liu and Marco Gruteser. 2021. EdgeSharing: Edge Assisted Real-time Localization and Object Sharing in Urban Streets. In *IEEE INFOCOM 2021*.

[37] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge Assisted Real-time Object Detection for Mobile Augmented Reality. *ACM MobiCom* (2019).

[38] Zida Liu, Guohao Lan, Jovan Stojkovic, Yunfan Zhang, Carlee Joe-Wong, and Maria Gorlatova. 2020. CollabAR: Edge-assisted Collaborative Image Recognition for Mobile Augmented Reality. In *2020 19th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*.

[39] Giuseppe Loianno, Yash Mulgaonkar, Chris Brunner, Dheeraj Ahuja, Arvind Ramanandan, Murali Chari, Serafin Diaz, and Vijay Kumar. 2016. A swarm of flying smartphones. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 1681–1688. https://doi.org/10.1109/IROS.2016.7759270

[40] Microsoft. [n. d.]. HoloLens 2. https://www.microsoft.com/en-us/hololens/hardware.

[41] Microsoft. 2018. Shared experiences in Unity. https://docs.microsoft.com/en-us/windows/mixed-reality/shared-experiences-in-unity.

[42] Raúl Mur-Artal and Juan D. Tardós. 2017. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *IEEE Transactions on Robotics* 33, 5 (2017), 1255–1262. https://doi.org/10.1109/TRO.2017.2705103

[43] Jannis Möller. 2019. VINS-Mobile-Android. https://github.com/jannismoeller/VINS-Mobile-Android.

[44] OpenCV. [n. d.]. Cascade Classifier. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.

[45] OpenCV. 2022. Basic concepts of the homography explained with code. https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html. Accessed: 2022-09-30.

[46] OpenCV. 2022. Object Detection with Template Matching. https://docs.opencv.org/3.4.16/df/dfb/group__imgproc__object.html. Accessed: 2022-09-30.

[47] Alexander Pacha. [n. d.]. Sensor Fusion Demo. https://github.com/apacha/sensor-fusion-demo.

[48] Hang Qiu, Fawad Ahmad, Fan Bai, Marco Gruteser, and Ramesh Govindan. 2018. AVR: Augmented vehicular reality. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. 81–95.

[49] Xukan Ran, Haoliang Chen, Zhenming Liu, and Jiasi Chen. 2018. DeepDecision: A Mobile Deep Learning Framework for Edge Video Analytics. *IEEE INFOCOM* (2018).

[50] Xukan Ran, Carter Slocum, Yi-Zhen Tsai, Kittipat Apicharttrisorn, Maria Gorlatova, and Jiasi Chen. 2020. Multi-user augmented reality with communication efficient and spatially consistent virtual objects. In *ACM CoNEXT*. 386–398.

[51] Matthew Reynolds. [n. d.]. Pokemon Go Buddy Adventure explained - how to get hearts, excited Buddies, and all Buddy level rewards including Best Buddy explained. https://www.eurogamer.net/articles/2019-12-19-pokemon-go-buddy-adventure-play-excited-6002.

[52] Patrik Schmuck and Margarita Chli. 2019. CCM-SLAM: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams. *Journal of Field Robotics* (2019).

[53] Sheng Shen, Mahanth Gowda, and Romit Roy Choudhury. 2018. Closing the Gaps in Inertial Motion Tracking. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18)*. ACM, New York, NY, USA.

[54] Arno Solin, Santiago Cortes, Esa Rahtu, and Juho Kannala. 2018. Inertial Odometry on Handheld Smartphones. In *2018 21st International Conference on Information Fusion (FUSION)*.

[55] Spectacles. [n. d.]. New Spectables AR Glasses. https://www.spectacles.com/new-spectacles/.

[56] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. 2013. Deep neural networks for object detection. *Advances in neural information processing systems* 26 (2013).

[57] Richard Szeliski. 2010. *Computer vision: algorithms and applications*. Springer Science & Business Media.

[58] Mingxing Tan, Ruoming Pang, and Quoc V. Le. 2020. EfficientDet: Scalable and Efficient Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[59] Zhenjun Tang, Y. Dai, and X. Zhang. 2012. Perceptual hashing for color images using invariant moments. *Applied Mathematics and Information Sciences* 6 (04 2012), 643S–650S.

[60] TensorFlow. [n. d.]. TensorFlow Lite Object Detection Example. https://www.tensorflow.org/lite/examples/object_detection/overview.

[61] TFHub.dev. [n. d.]. Image object detection. https://tfhub.dev/tensorflow/efficientdet/lite2/detection/1.

[62] Ramona Trestian, Arghir-Nicolae Moldovan, Olga Ormond, and Gabriel-Miro Muntean. 2012. Energy consumption analysis of video streaming to Android mobile devices. In *2012 IEEE Network Operations and Management Symposium*.

[63] Maarten Van Steen and Andrew S Tanenbaum. 2017. *Distributed systems*. Maarten van Steen Leiden, The Netherlands.

[64] Ana Villanueva, Zhengzhe Zhu, Ziyi Liu, Kylie Peppler, Thomas Redick, and Karthik Ramani. 2020. *Meta-AR-App: An Authoring Platform for Collaborative Augmented Reality in STEM Classrooms*. ACM.

[65] VUZIX. [n. d.]. VUZIX BLADE UPGRADED SMART GLASSES. https://www.vuzix.com/products/vuzix-blade-smart-glasses-upgraded.

[66] Yihong Wu and Zhanyi Hu. 2006. PnP problem revisited. *Journal of Mathematical Imaging and Vision* 24, 1 (2006), 131–141.

[67] Wenxiao Zhang, Bo Han, and Pan Hui. 2018. Jaguar: Low Latency Mobile Augmented Reality with Flexible Tracking. In *International Conference on Multimedia*. ACM, 355–363.

[68] Wenxiao Zhang, Bo Han, Pan Hui, Vijay Gopalakrishnan, Eric Zavesky, and Feng Qian. 2018. CARS: Collaborative Augmented Reality for Socialization. *ACM HotMobile* (2018).

[69] Yinqiang Zheng, Yubin Kuang, Shigeki Sugimoto, Kalle Åström, and Masatoshi Okutomi. 2013. Revisiting the PnP Problem: A Fast, General and Optimal Solution. In *2013 IEEE International Conference on Computer Vision*.

[70] Danping Zou and Ping Tan. 2012. Coslam: Collaborative visual slam in dynamic environments. *IEEE transactions on pattern analysis and machine intelligence* 35, 2 (2012), 354–366.

[71] Longhao Zou, Ali Javed, and Gabriel-Miro Muntean. 2017. Smart mobile device power consumption measurement for video streaming in wireless environments: WiFi vs. LTE. In *2017 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. 1–6. https://doi.org/10.1109/BMSB.2017.7986151