OpenFLAME: Federated Visual Positioning System to Enable Large-Scale Augmented Reality Applications

Sagar Bharadwaj*† Harrison Williams*† Luke Wang*† Michael Liang† Tao Jin†
Srinivasan Seshan† Anthony Rowe†

Carnegie Mellon University

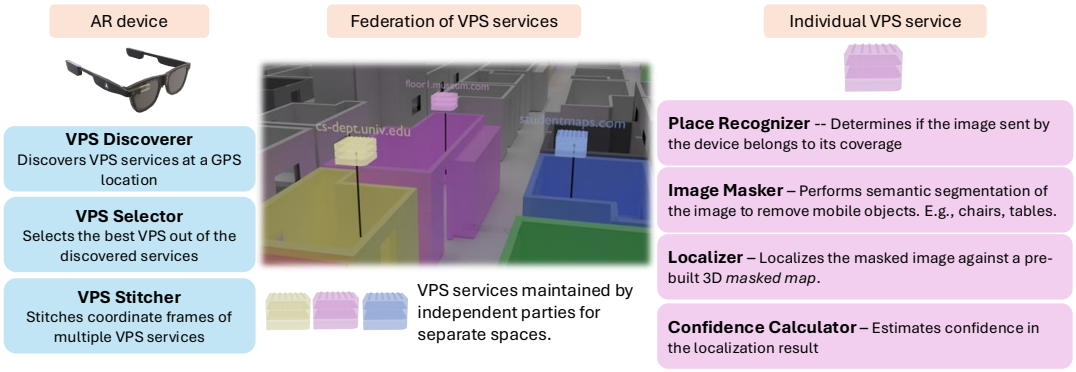


Figure 1: OpenFLAME federates the creation and maintenance of Visual Positioning Systems (VPS) allowing independent organizations to support VPS services for their own spaces enabling privacy and distributed maintenance. Its client-side components enable VPS discovery, selection and seamless transition across service boundaries.

Abstract

World-scale augmented reality (AR) applications need a ubiquitous 6DoF localization backend to anchor content to the real world consistently across devices. Large organizations such as Google and Niantic are 3D scanning outdoor public spaces in order to build their own Visual Positioning Systems (VPS). These centralized VPS solutions fail to meet the needs of many future AR applications—they do not cover private indoor spaces because of privacy concerns, regulations, and the labor bottleneck of updating and maintaining 3D scans. In this paper, we present OpenFLAME, a federated VPS backend that allows independent organizations to 3D scan and maintain a separate VPS service for their own spaces. This enables access control of indoor 3D scans, distributed maintenance of the VPS backend, and encourages larger coverage. Sharding of VPS services introduces several unique challenges—coherency of localization results across spaces, quality control of VPS services, selection of the right VPS service for a location, and many others. We introduce the concept of federated image-based localization and provide reference solutions for managing and merging data across maps without sharing private data.

Index Terms: Localization, augmented reality.

1 Introduction

Building world-scale persistent Augmented Reality (AR) applications has been the goal of the community for several decades. Such applications require a ubiquitous, continually updated localization backend [33, 49, 41, 42] so that 3D content can be anchored anywhere in a continuously changing world. Furthermore, such anchoring must be consistent across devices to enable collaborative AR applications. To support such applications, Visual Positioning Systems (VPS) are being increasingly deployed in public spaces by many organizations (e.g., Google [12], Niantic [16], and Apple [1]).

VPS determines the location and orientation of a device with respect to a 3D map using visual features. Existing VPS solutions are usually provided by organizations that have the resources to scan and host vast spans of public spaces. For example, Geospatial API [12], Google's VPS solution, is based on the 3D world map created by Google's street view imagery collected over 15 years [14]. These types of VPS solutions fail to meet the expectations of future AR applications in two important ways:

- **Lack of ubiquitous coverage** – Private spaces would not host their scans on centralized servers due to privacy concerns and lack of fine-grained access control of their data. As a result, existing VPS solutions are limited to public spaces that such organizations can access and scan [16, 12].
- **Staleness of visual data** – Real-world scenes evolve over time. Constantly updating scans of dynamic environments requires massive cartography efforts, which are cost-prohibitive for centralized organizations. As a result, the scans get stale, reducing localization accuracy.

A crowdsourced VPS solution accepting contributions from independent parties could perhaps address these limitations [13, 17]. However, not all spaces (e.g., sensitive spaces like private offices and national labs) would consider uploading their maps to crowdsourced VPS services. Furthermore, crowdsourcing all of the data into a handful of large VPS services makes future AR applications reliant on these central organizations. This introduces central points

*Equal contribution.
†e-mail addresses:{skalasib, hwillia2, ainiuw, mliang4, taojin, srini, agr}@andrew.cmu.edu

of failure that could bring down a large number of AR applications and concentrates control over future AR applications.

A federated VPS infrastructure that allows independent organizations to collect and host their own scans can overcome these limitations. Federation encourages larger coverage as private data can be independently hosted and access-controlled. It also unlocks the ability to maintain and frequently update a ubiquitous VPS solution in a distributed fashion. Federation enables the system to scale organically, as new VPS services can be added independently without the need for centralized coordination or shared infrastructure. We present OpenFLAME[1], a federated VPS solution that can enable world-scale 6 Degrees-of-Freedom (DoF) localization across dynamically changing private and public spaces. In OpenFLAME, a device that wants to localize itself first discovers VPS services available to it at the given GPS location. The device then contacts each of these VPS services, potentially hosted by independent organizations, and sends visual cues to all of them. Once localization results are received from these VPS services, the device will select the best VPS service for the location. The device also calculates and applies transformations to VPS results to ensure they are coherent across service boundaries. Federation of VPS services introduces some unique challenges absent in today's centralized solutions:

- **Coherent localization across spaces** – The localization trajectory exposed to AR applications has to be coherent across spaces, even if the individual 3D scans of spaces are in their own separate coordinate frames of reference. To maintain privacy, the VPS system cannot rely on sharing visual features between different VPS services to stitch these disparate coordinate frames.
- **High variability in service quality** – The quality of VPS services hosted on OpenFLAME could have a wide range from scans made using low-quality RGB images to high-quality lasers. We need a solution for devices to dynamically pick the highest-quality VPS service at a location.
- **Irrelevant queries to services** – Individual VPS services in OpenFLAME might receive localization requests from devices outside of their coverage, resulting in a waste of resources used for localization. We need a lightweight solution to weed out requests that are outside the coverage area of a server before triggering the localization pipeline.

In this paper, we introduce the concept of federated VPS and provide effective solutions to the above challenges. To summarize, our contributions are as follows:

- We present the design of a distributed federated VPS infrastructure called OpenFLAME, where independent organizations maintain their own VPS services for their own spaces.
- We present simple solutions to solve challenges unique to federated VPS—a *VPS Stitcher* to enable coherent trajectories across VPS services without exposing visual features, a *VPS Selector* to dynamically select the highest quality VPS service at a location, a *Place Recognizer* to filter out requests outside of a server's VPS coverage, and a *Pose Confidence Calculator* to estimate the confidence of pose estimates. While the solutions to these individual components are borrowed from existing fields of study, their integration enables our vision of a federated VPS backend.
- At individual VPS services, we present a localization pipeline based on scene understanding that enables the handling of dynamically changing scenes.

[1]OpenFLAME stands for Open Federated Localization and Mapping Engine. https://www.open-flame.com/

We evaluate each of the proposed components in dynamically changing large-scale indoor environments in § 8. We implement an AR 3D indoor navigation application on top of OpenFLAME to show that it can support the development of large-scale AR applications. The implementations of the VPS pipeline, AR application, client-side library, and supplementary tools will be open-sourced after publication.

2 Example Workflow

Consider an AR cultural heritage guided tour application that displays infographics overlaid on real-world objects indoors and outdoors. For example, outdoors, it might show fact bubbles and timelines of landmark historical buildings, while indoors, it might describe museum artifacts and paintings. To accurately anchor 3D content against the real world, such an application would need a localization solution that works across indoor and outdoor spaces while seamlessly transitioning between environments.

Existing VPS solutions, mostly limited to public areas, cannot support such applications. Even if some museums have 3D-scanned their floors, no convenient system today can integrate these scans with outdoor 3D localization systems (e.g., Google Geospatial APIs) to provide coherent VPS solutions to applications. While the museum could request one of the large providers to integrate their scans with the larger system, it would deter the museum from implementing fine-grained access control to their scans (e.g., only people with tickets can view and localize against museum scans). Furthermore, the process of updating the map would be bottlenecked at the large VPS provider.

In OpenFLAME, the VPS services for specific regions are maintained by independent organizations. In our example, the museums would maintain their own 3D scans, and the localization service would be provided against those scans. When outdoors, the AR application would use a large VPS provider, such as Google, to localize and anchor 3D content. Once the user enters the museum, OpenFLAME will discover and switch to the VPS service maintained by the museum. The application would rely on OpenFLAME to discover VPS services, select the best VPS service for a location, get 6DoF poses, and stitch results from multiple VPS services.

Furthermore, individual VPS services in OpenFLAME implement a pipeline to support dynamically changing environments. We use simple image segmentation to remove frequently moving objects from the scan and visual cues while localizing. For example, the pipeline allows furniture and people to be removed from museum scans while retaining the predominantly stationary artifacts, increasing the scan's longevity and relevance.

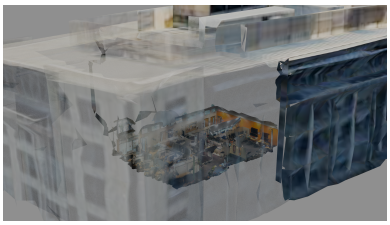
3 Background

3.1 Visual positioning

Visual Positioning Systems aim to estimate the 6 DoF pose of a device camera using visual input. Early VPS methods often rely on visual markers such as AprilTags [40] and ArUco [7], where known 2D templates with pre-defined geometry are detected in the image and mapped to corresponding 3D points. This enables 6-DoF pose estimation by solving the Perspective-n-Point (PnP) problem [35]. However, deploying visual markers throughout a space is often impractical. Hence, extensive work has been focusing on markerless visual localization using hand-crafted feature descriptors such as SIFT[37], SURF[26], and ORB[45] to detect and describe visual features. These features are typically matched against a database of 3D points constructed via Structure-from-Motion [49] or Simultaneous Localization and Mapping [39]. Once 2D-3D feature correspondences are established, the camera pose can be estimated by solving the PnP problem. Learning-based methods have recently improved the localization pipeline, with keypoint detectors and descriptors like SuperPoint [28], and matchers such as SuperGlue [48] and LightGlue [36] boosting reliability and accuracy.

(a) Outdoor scan by Google.

(b) Indoor scan by a university.



(c) OpenFLAME integrates multiple VPS services.

Figure 2: OpenFLAME integrates independent VPS services.

3.2 VPS services with ubiquitous coverage

VPS services with ubiquitous coverage aim to localize users in expansive environments such as city streets, parks, and commercial venues. Unlike indoor VPS, which can rely solely on visual input due to its relatively small search space, large-scale VPS typically adopts a hybrid approach—using GPS and IMU data to obtain a coarse initial 6 DoF pose estimate, which is then refined through visual observations captured by the device's camera. Several commercial systems have successfully implemented this approach. Google's Geospatial API [12] allows users to determine precise device poses by combining GPS data with visual localization against Google Street View imagery as a global reference. Similarly, Apple's ARKit ARGeoAnchor [3] also leverages location and camera input to anchor AR content in real-world locations. Niantic Lightship [16] has taken a crowdsourced approach by asking users to scan public spaces at various times of day and from multiple viewpoints, building a rich and diverse database that enables robust localization under varying lighting conditions. These large-scale VPS services demonstrate the feasibility of global visual localization, enabling persistent AR experiences and navigation assistance in complex real-world environments.

4 Characteristics of Individual VPS Services

Figure 2 shows two 3D scans from independent organizations—Google (2a) and a private university (2b). The two scans, and as a result, the VPS services provided on top of these scans, vary in multiple ways, such as reconstruction quality (which in turn affects pose estimation accuracy) and the coordinate system used. In this section, we explore the characteristics of individual VPS services and how they differ from one another.

Quality – OpenFLAME supports VPS services of varying qualities. For example, in Figure 2a, Google's scan is created with data from high-quality cameras and LiDARs mounted on Google Street View cars and aerial imagery. In contrast, the university's indoor scan in Figure 2b is created using the Polycam application [18] on an iPhone. Admitting VPS services of varying quality encourages incremental deployment of VPS without expecting perfection from the start, thereby increasing VPS coverage.

Disparate coordinate systems – OpenFLAME allows individual 3D scans to be in their own coordinate systems and does not require alignment with the global geographic coordinate system. Precise alignment with latitude, longitude, and altitude requires expensive survey equipment such as Total Stations [21] and RTK GNSS [20], increasing the barrier to deployment. For example, the university indoor scan in 2b is in its own coordinate system, unlike Google's scan, which is laid out in the system of latitudes and longitudes.

Overlap of VPS coverages – We do not enforce spatial exclusivity—different VPS services can cover the same area.

5 Localization Pipeline

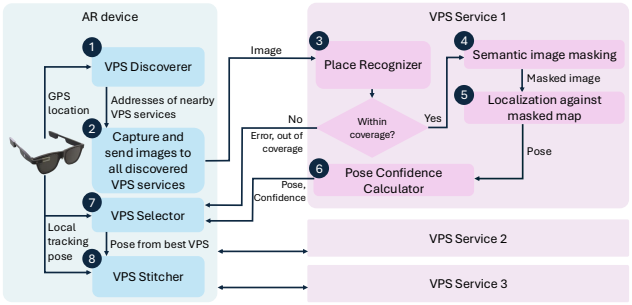


Figure 3: Localization pipeline on OpenFLAME.

Figure 3 shows the localization pipeline of OpenFLAME. The device first uses OpenFLAME's *VPS Discoverer* module to identify all VPS services offered at the device location. The device then sends visual cues to a select subset of these services. Individual OpenFLAME VPS services perform image-based localization while ignoring dynamic parts of the map, such as furniture and people. Individual VPS services return the estimated pose and confidence scores to the device. The device then selects the best localization results and transforms it to stitch it with the coordinate system of previous pose estimates from other VPS services.

The localization pipeline shown in Figure 3 is run periodically at a pre-configured interval (1 second by default). Results from local tracking libraries (e.g., ARCore [11], ARKit [2], and WebXR [23]) are used to anchor 3D elements in the intermediate frames. Furthermore, we do not trigger the entire localization pipeline every cycle. Since devices are not expected to frequently cross VPS service boundaries, we assume that a selected VPS service can be used for multiple consecutive localization cycles. The device continues to use the current service until the pose confidence scores drop; at this point, a new discovery query is triggered to identify a more suitable service. The remainder of this section describes in detail each of the components in Figure 3.

5.1 VPS Discoverer

The VPS Discoverer module is responsible for identifying the VPS services available at a given GPS location. To accomplish this, it requires access to data that maps geographic regions to corresponding lists of VPS services. This data can be stored and queried in various ways using existing systems, including spatial databases (e.g., GeoFire [10], PostGIS [19], MongoDB [15]), Geographic Information Systems (GIS) such as ArcGIS [4] or Carto [5], or even through DNS-based approaches [32, 27, 31]. In our implementation, we repurpose the Domain Name System (DNS) to function as the VPS discoverer. This approach offers several advantages: it leverages widely available infrastructure, supports caching mechanisms, is straightforward to implement, and naturally enables federation. In this paper, we specifically focus on the challenges of image-based localization in a federated setting. A detailed treatment of the organization and querying of spatial data for VPS service discovery is out of scope of this paper.

5.2 Capture and send images

Once the VPS services at a given location are discovered, OpenFLAME requests the device to capture an image. It then broadcasts

the image to a subset of the discovered VPS services. The filtering mechanisms used to select a subset of VPS services are configurable by the AR applications using OpenFLAME. For example, the application can whitelist a subset of Top Level Domains (TLDs) that are acceptable. An AR campus navigation application, for instance, can choose to use only VPS services hosted on `.edu` domains. The application can also choose to limit the number of VPS services that are being contacted every discovery cycle, in which case a subset is arbitrarily chosen. After a few discovery and localization cycles, OpenFLAME eventually locks in on the VPS service that is the most accurate for a location using the *VPS Selector* module described later. OpenFLAME provides parameter settings that the applications can use to configure the method of filtering VPS services.

5.3 Place Recognizer

Unlike centralized VPS services that have to satisfy all localization requests from devices, the VPS services on OpenFLAME might receive requests from devices outside of their VPS coverage. This is especially true indoors, where GPS errors are high, and the device might discover and make requests to multiple maps before narrowing down its location to a single map. In OpenFLAME, VPS services include the Place Recognizer module that takes in an image and determines if it belongs to a place that is within the VPS coverage of the server. This is run before triggering the localization pipeline to conserve system resources.

The Place recognizer module is primarily based on the CLIP [6] model—a neural network trained on a large dataset of image-text pairs. We specifically use the image encoder layer of CLIP, which converts an image to an *image embedding* (i.e., a vector), expected to represent the semantic information in the image. To ensure discriminability between spatially close and visually similar spaces (e.g., office areas used by different groups in a university), we fine-tune CLIP for specific VPS services. We construct a dataset of images from such neighboring locations. We modify the original CLIP model by appending a projection head—a fully connected layer—that maps CLIP's high-dimensional image embeddings to a new 256-dimensional embedding space. During training, we freeze CLIP's parameters and train only the projection layer using a triplet loss. This encourages embeddings of images from the same room to have lower cosine distance, while pushing apart embeddings from different rooms. The result is a model that produces more discriminative representations suited for fine-grained place recognition. While such fine-tuning enhances performance in areas where data sharing between VPS services is possible, we observe that even the pre-trained CLIP model—without any fine-tuning—performs well in settings where cross-service data access may be restricted.

A CLIP image embedding data set is constructed offline for all the images in the 3D map database—the *database embeddings*. To determine if a given query image lies within the VPS service's coverage, we first get the CLIP embedding of the query image. The query embedding is compared to the database embeddings to find the one with the smallest cosine distance. If the smallest cosine distance exceeds a pre-configured threshold, the image is discarded, and the rest of the localization pipeline is skipped. The threshold can be changed based on how resource-conserving the VPS service wants to be. At low thresholds, most images are discarded. If the minimum Euclidean distance is high, the VPS service responds with an error code that indicates to the device that the queried image is out of coverage. § 8.5 evaluates the performance of the Place Recognizer module.

5.4 Semantic image masking

Individual VPS services might be hosted in dynamically changing spaces with many objects that frequently change their positions, such as chairs, keyboards, backpacks, and people. As visual positioning is sensitive to the position of visual features, the localization quality drastically reduces because of the frequent motion of such objects. In our pipeline, we address this issue by masking out classes of objects that frequently move in an environment and not considering them in pose estimation.

Figure 4: Semantic image masking is used to ignore features from typically dynamic objects, making localization more robust to changes in the environment.

Figure 4 shows two examples of semantic masking. The left column shows the original image. To generate the *semantic mask* in the middle column, we use YOLOv8 [24]—a real-time object detection system. The classes of objects to be detected are configurable and can be set by individual VPS services. The figure shows a typical university office environment, and objects such as chairs, keyboards, and backpacks are detected. In a different environment, a different set of object classes can be used. Once 2D image features are detected in the *pose estimation* step (§ 5.5), the features inside the boundaries of the detected objects are removed and are not considered in the rest of the localization pipeline. The rightmost column in Figure 4 shows the set of 2D features that are removed. § 8.2 evaluates the benefits of masking dynamic features.

5.5 Pose estimation

We use `hloc` (Hierarchical Localization) [47] to estimate the pose of the given query image against a 3D map. The process of constructing the masked 3D map is described in § 6. `hloc` uses SuperPoint [28] feature detector and descriptor, and SuperGlue [48] feature matcher. The combination of learned feature detector, descriptor, and matcher makes the pose estimation robust to lighting changes. To support large maps, hloc also adds a *global retrieval* layer that uses NetVLAD [25] to isolate pose estimation to a small portion of the map that is relevant to the query image.

We optimize the implementation of hloc to integrate well with OpenFLAME. Specifically, the default implementation of the localization pipeline on hloc loads the neural network weights of all models (i.e., SuperPoint, SuperGlue, and NetVLAD) into memory for every localization request. We persist the weights in memory across multiple localization queries to speed up pose estimation. Furthermore, the implementation was not written with support for concurrent executions. We made modifications to the way the query image is loaded to enable concurrent pose estimations. While our implementation uses hloc, this component can be replaced by any vision-based localization method, including the recent synthesize-and-localize [46, 44, 34, 56] methods. We build the rest of the components in OpenFLAME to be agnostic to the method used for pose estimation, which enables VPS services to take advantage of the rapid advances in the field of vision-based localization.

5.6 Pose Confidence Calculator

Individual VPS services return a confidence score along with their pose estimates to enable the devices to make an informed selection

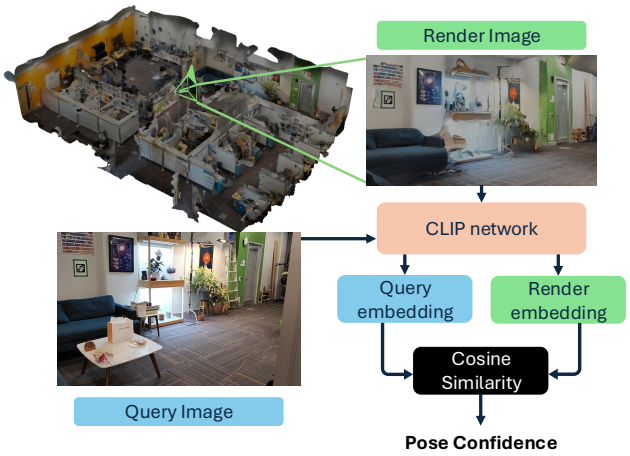


Figure 5: The Pose Confidence component estimates how reliable the calculated pose is by comparing the query image with one rendered from that pose.

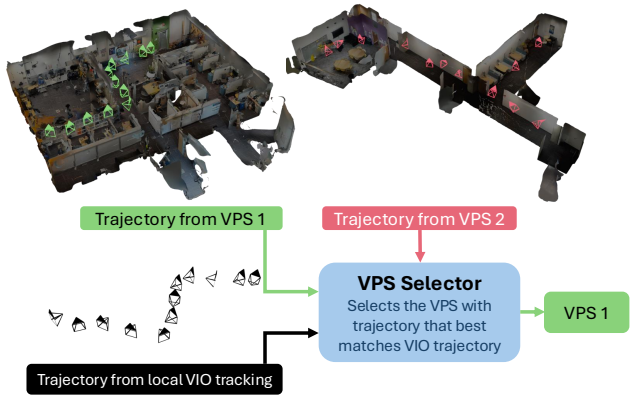


Figure 6: VPS Selector chooses the best VPS service by comparing different VPS trajectory results against on-device VIO tracking.

and switch to a different VPS service once the device moves outside of the coverage area. The *Pose Confidence Calculator* module helps estimate the confidence of the computed pose.

In image-based localization, common pose confidence metrics include the number of feature inliers [30], reprojection error [22], and the ratio of inliers to detected keypoints. However, these feature-based metrics are tightly coupled to the specific pose estimation method and cannot be reliably compared across different VPS services. This coupling would also limit the ability of VPS services to independently adopt new localization techniques, such as emerging synthesize-and-localize methods [44, 34], which do not rely on feature matching. Additionally, feature-based metrics can fail in indoor environments with repetitive textures (e.g., floor tiles, carpets), where incorrect poses may still yield high inlier counts. To address these limitations, we instead use image similarity between the query image and a rendered view from the 3D scan, enabling consistent and technology-agnostic confidence estimation across VPS services.

Figure 5 shows our pipeline to estimate pose confidence. Once we estimate the pose of the query image, we render the corresponding pose from our 3D scan at the estimated pose. CLIP embeddings for both the rendered image and the query image are generated. The cosine similarity between the embeddings of the rendered and query images is returned as pose confidence. Our pose confidence scores are independent of repeating textures and the number of features as CLIP compares semantic differences between the images. To improve performance, we extend CLIP with a projection layer and fine-tune it so that the final embeddings of the query image and the 3D render at the estimated pose are close in the embedding space. As in § 5.3, we train this model using triplet loss on a dataset of query and rendered image pairs. In § 8.3, we justify our choice of using a fine-tuned CLIP model by comparing it to other similarity metrics.

5.7 VPS Selector

Once the device has all pose estimations from all the VPS services that it sent images to, it has to select one VPS service that is the most accurate for the current location. If all the VPS services implement the pose confidence module (§ 5.6), and the device trusts these VPS services, it can use these scores for its selection (e.g., a museum AR application would trust the VPS service hosted by the museum administrators and use its pose confidence scores). However, in our federated setting, not all VPS services can be trusted. Some services might return high confidence—either accidentally or with malicious intent—to entice devices to continue using them. Some might only return pose estimates without an attached confidence score. Therefore, devices need a VPS selection method that is independent of the server estimated confidence. If the device consistently finds discrepancies between the device-calculated score and the VPS service returned score, it can mark the VPS service as undesirable and avoid it in future localization cycles.

Figure 6 shows our server-independent VPS selection method. This method can work after the device has captured and collected pose estimates for at least 3 images. The trajectories estimated by all the VPS services are first aligned with the local device trajectory calculated using VIO (Visual Inertial Odometry) algorithms. The trajectories need alignment, as local tracking and tracking with the VPS services each run in their own coordinate systems. Once trajectories are aligned, we calculate the Absolute Trajectory Error (ATE) between server-provided and on-device trajectories. The VPS service with the least trajectory error is selected. § 8.4 shows that this technique selects the correct VPS service in almost all cases.

5.8 Visual features-free dynamic VPS Stitcher

Once the device obtains a pose from the selected VPS service, it must align this result with poses from other VPS services encountered in previous cycles, as each service operates in its own coordinate frame. Although stitching 3D scans using visual features at region boundaries is a well-studied problem [43, 52], we cannot rely on such feature sharing between VPS services due to privacy constraints and pose estimation technology variations.

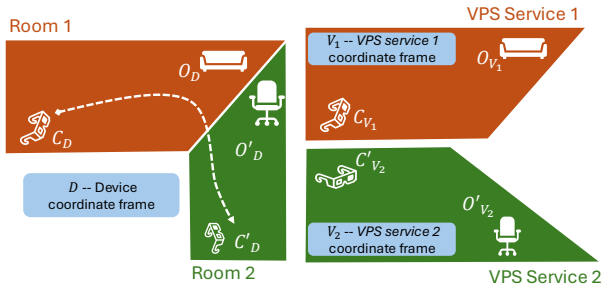


Figure 7: Dynamically stitching coordinate systems.

To understand how OpenFLAME dynamically stitches coordinate systems together without requiring visual features, consider two rooms, Room 1 and Room 2 shown in Figure 7. Room 1 and 2 are covered by different VPS services (VPS 1 and VPS 2). The two VPS services have their own scans of the respective rooms in their own coordinate systems—V_1 and V_2 (shown on the right side of Figure 7.) A device running an AR application is also building its own local coordinate system, D. For example, such a coordinate system is built by ARCore [11] in Android applications, ARKit [2] in Apple devices, or WebXR [23] in web applications. The left side of Figure 7 shows the layout of the rooms in the real world and the trajectory of the device determined by local AR tracking in the coordinate frame D.

Consider an object O in Room 1. Let the $4X4$ pose matrix of the object in the V_1 coordinate frame be O_{V_1}. Similarly, the pose matrix of the same object O in the local coordinate frame D is O_D. A device camera moves from Room 1 to Room 2 as shown on the left side of Figure 7. The initial position of the device, as calculated in the local tracking coordinate frame, is C_D. The device also sends a localization request to VPS server 1 to get its pose in the V_1 coordinate frame, C_{V_1}. The pose of the object O with respect to the camera is the same irrespective of which coordinate system is considered. Therefore, we have:

$$C_D^{-1} O_D = C_{V_1}^{-1} O_{V_1} \implies O_D = C_D C_{V_1}^{-1} O_{V_1} \tag{1}$$

Let the final position of the device camera in Room 2, in the local tracking coordinate frame D, be C'_D. The pose of the camera as calculated by VPS service 2, in the V_2 coordinate frame is C_{V_2}. Following the same argument as above, for a different object O' in Room 2, we have:

$$C_D'^{-1} O'_D = C_{V_2}'^{-1} O'_{V_2} \implies O'_D = C'_D C_{V_2}'^{-1} O'_{V_2} \tag{2}$$

Now we know the poses of the objects O and O' in the same frame, D. The relative positions of these objects will remain the same in any coordinate frame of reference:

$$O_D^{-1} O'_D = O_M^{-1} O'_M, \forall M \tag{3}$$

Although the objects O and O' are in different rooms and scanned by different VPS services, we now know the relative pose of one with respect to the other.

$$\begin{aligned} O_D^{-1} O'_D &= (C_D C_{V_1}^{-1} O_{V_1})^{-1} (C'_D C_{V_2}'^{-1} O'_{V_2}) \\ &= O_{V_1}^{-1} C_{V_1} C_D^{-1} C'_D C_{V_2}'^{-1} O'_{V_2} \end{aligned}$$

As the locations of objects O and O' in V_1 and V_2 is arbitrary, we choose $O_{V_1} = O'_{V_2} = I$. In other words, the relative transform between the origins of the two coordinate frames V_1 and V_2 is:

$$C_{V_1} C_D^{-1} C'_D C_{V_2}'^{-1} \tag{4}$$

The matrices C_{V_1}, C_{V_2}, and C_D can be repeatedly measured by the device and collected from the VPS services as the device moves between rooms 1 and 2. As the matrices are noisy because of localization errors, it might take a few iterations of measurements to get the right transform. Figure 8 shows the relative transform (i.e., Transform 4) between the 3D scans of two independent VPS services. We show a green outline around one of the scans. In § 8.1, we show that only one observation from the second room is sufficient, in most cases, to align the two coordinate frames of reference, resulting in seamless transitions between VPS services.

It is important to note that the transformation described in this section applies only to linear transformations between 3D scans and does not account for non-linear warps. However, in our 3D map construction pipeline, we remove distortion from our input images and use the *corrected* images to generate the maps, effectively removing non-linear distortions.

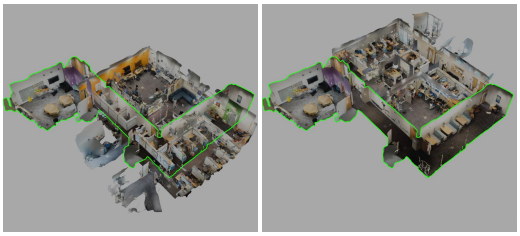


Figure 8: Example of stitching coordinate systems.

6 3D Scanning Pipeline

This section outlines the 3D scanning pipeline used for constructing the VPS map in OpenFLAME. Our pipeline consists of data acquisition with mobile devices, filtering dynamic elements, and building a feature database for localization.

Collecting posed images: We begin by scanning the target environment using an iPad equipped with a LiDAR sensor, capturing data through the Polycam app. This provides us with RGB-D images alongside 6-DoF camera poses. To ensure geometric accuracy, the iPad's camera intrinsics are calibrated using Zhang's method [55]. The RGB-D images exported by Polycam are already undistorted to conform to the pinhole camera model, which is essential for accurate 3D to 2D projection.

Dynamic object removal: To improve map stability over time, we remove dynamic elements from the scene. We use YOLO-v8 [24] for semantic segmentation and classification, generating masks that identify potentially dynamic objects such as people, chairs, and cars. These masks are applied to the RGB images to exclude dynamic regions from the mapping process, ensuring the VPS map captures only the static structure of the scene.

Feature database construction: The masked images are then processed using COLMAP [49] to generate a 3D reconstruction and image feature database. We replace COLMAP's default feature pipeline with SuperPoint for keypoint extraction and SuperGlue for matching, which provides more robust performance. Additionally, we compute a NetVLAD global descriptor for each image to accelerate image retrieval during localization.

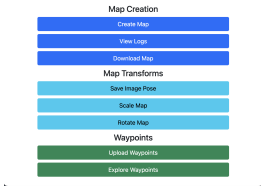
7 Implementation and Support Tools

We will open-source OpenFLAME—both device-side and VPS service implementations. We also present supplementary tools that help with 3D scan visualization and verification of pose estimation. Additionally, we have implemented a 3D indoor navigation application that uses OpenFLAME as the underlying localization backend to show that it can support large-scale AR applications.

VPS service (9a): Our implementation of VPS service uses an optimized version of hloc [47] for pose estimation (§ 5.5), along with CLIP and YOLO models for other components described in § 5. In our evaluations, it runs on an Intel Core i9-13900K CPU and an NVIDIA GeForce RTX 4090 GPU.

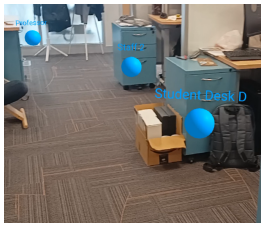
Scan visualization and tagging (9b): We present a web-based tool built using the A-Frame Inspector [9] that can visualize 3D scans generated for the VPS service (§ 6). It also allows tagging regions in these scans. These tags can be downloaded and rendered against the real-world using our AR verifier tool below.

AR localization verifier (9c): An A-Frame [8] application that uses camera RGB images and the selected VPS service to localize. It overlays the tags created using the above tool onto the real world.

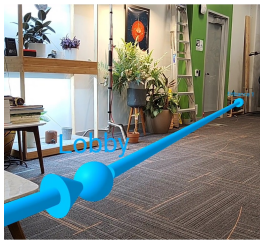


(a) VPS Service landing page.

(b) Scan Visualizer and tagging.

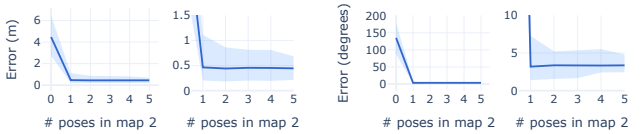


(c) AR localization verifier.



(d) AR Navigation application.

Figure 9: Supplementary tools.



(a) Translation Error.

(b) Rotation Error.

Figure 10: Error in the stitching transform estimated by the VPS Stitcher component compared to the *true stitching transform*—transform calculated by manually stitching the two maps.

(a) Image used for map creation.

(b) Query image.

Figure 11: Differences between the query image and the image used for map creation to evaluate our masked localization pipeline.

It enables quick verification that the VPS service can correctly anchor virtual objects to the physical environment. Figure 9c shows the tags in 9b overlaid on the real-world.

AR 3D indoor navigation application(9d): A-Frame application that uses OpenFLAME as the localization backend to navigate large indoor spaces. Between remote localization calls to OpenFLAME, it uses WebXR for local VIO tracking. The data for route calculation is generated using the scan visualizer tool above. Details on routing and navigation in our application is out of scope of this paper.

8 Evaluation

In this section, we evaluate our techniques to demonstrate the feasibility of a federated VPS system. While each method can be further optimized in future work, we believe our results show that a federated VPS solution is viable and can support the needs of future augmented reality applications.

Dataset: To evaluate OpenFLAME, we 3D-scanned and set up 30 VPS services for various indoor locations on a university campus. The locations are diverse, including conference rooms, office cubicles, building lobbies, and classrooms. We used the raw data export from the Polycam application on an iPad Pro to collect our scans. We avoid using expensive scanning equipment to demonstrate that our techniques are effective, even with low-quality, easily obtainable scans. Each indoor location contains many movable objects, such as chairs, tables, keyboards, backpacks, etc., that change position over time. For each location, we collect two sets of query images—one immediately after the scan, and another after moving the objects in the scene. We use these query sets to show that our technique works for dynamic spaces without imposing the need to re-scan them. Each query image has an AprilTag [53] to get baseline pose estimates. To evaluate our techniques that rely on local VIO traces, we record WebXR poses in our AR application as a test device moves through spaces.

8.1 Visual features-free dynamic VPS stitcher

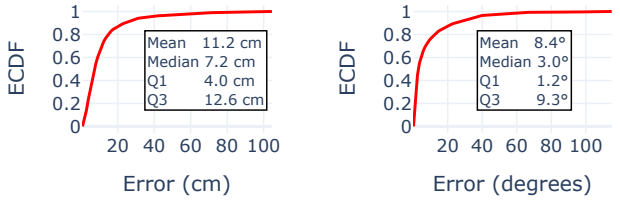
In § 5.8, we described our technique for dynamically stitching coordinate systems from independent VPS services, without requiring the VPS services to share visual features with each other. This technique requires pose matrices from three sources—the two VPS services being stitched and the local VIO tracking results.

Ideally, the device should determine the correct transform between two VPS systems as soon as it transitions from one service to the other, without needing numerous localizations in the new service. We use the VIO traces and localization results on images in our query dataset, and apply Transform Expression 4 on them, and compare the result with the true transform—the transform we get by manually aligning independent VPS maps. Figure 10 shows the translation and rotation errors between the true transform and the transform calculated using (4) against the number of pose estimations in the second VPS map. The shaded area shows 5^{th} and 95^{th} percentiles. We see that the device fixes on a good transform just after a single query to the second VPS service. The translation error on average goes down from more than 4 m to 0.5 m, and the rotation error (i.e., angle-axis distance: $cos^{-1}\frac{\text{trace}R_1R_2-1}{2}$, where R_1 and R_2 are 3X3 rotation matrices) goes down from 180° in the worst case to about 5° immediately after the first localization result. In this evaluation, we consider the 5 best poses from the first VPS service when calculating the transform in (4). Concretely, we use 5 values of C_{V_1} and C_D, resulting in 5 different transformations which are then averaged to get the required transform. While considering more localization results after entering the second VPS service does not significantly reduce the median errors, the worst-case error (i.e., 95^{th} percentile in the figures) decreases.

8.2 Masked localization in dynamic environments

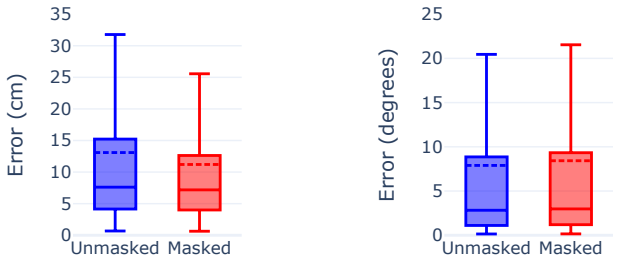
To evaluate our masked localization pipeline, we compare poses estimated by our pipeline against ground-truth poses obtained using AprilTags. The evaluation uses query images captured in environments where objects such as chairs and backpacks were moved since the map was created. Figure 11 shows an example comparison between a query image (11b) and the corresponding map image (11a). Notice that the chairs and backpacks are in different positions in the two images. Such queries allow us to evaluate how our masked localization pipeline handles dynamism.

Each query image contains an AprilTag, allowing us to compute two poses: one using the AprilTag and another using our localization pipeline. We generate two trajectories per query set, one from AprilTag poses and one from our method, and align them before computing the relative pose error (RPE). Figure 12 shows the empirical cumulative distribution function (ECDF) of the translation and rotation RPEs compared to AprilTag poses. The median translation error is 7.2 cm, and the median rotation error is 3°, which is acceptable for most AR applications.



(a) Translation RPE (b) Rotation RPE

Figure 12: Relative Pose Error (w.r.t. AprilTags).



(a) Translation errors. (b) Rotation errors.

Figure 13: Performance of masked localization.

Figure 13 compares errors observed with and without masking. The box plots (whiskers extend to 1.5 times the interquartile range beyond the first and third quartiles) shows the distribution of errors. Masking reduces translation errors, especially in worst-case scenarios—the 95th percentile error is reduced by 21.1% and the mean error decreases by 14.4%. In case of rotation, we see a slight increase in worst case errors—the 95th percentile error increases by 2.5%, while the mean stays about the same.

8.3 Pose confidence

As described in § 5.6, we require a pose confidence metric that is comparable across VPS services, agnostic to individual service configurations, and capable of distinguishing correctly localized images from potentially inaccurate ones.

In this section, we compare several confidence metrics and justify our choice of a fine-tuned CLIP model. Figure 14 shows the cumulative distribution functions (CDFs) of confidence scores for various metrics we evaluated. The green line represents scores when queries are made to the correct VPS service, while the red line corresponds to queries sent to an incorrect service. All metrics are normalized to the range [0, 1], and we invert LPIPS so that higher values indicate greater confidence. A good confidence metric must easily distinguish between correct and wrong VPS services. In other words, good metrics have large gaps between the green and the red line in Figure 14.

The first two rows show feature-based metrics: the number of SuperPoint inliers and the ratio of inliers to the total number of detected SuperPoint keypoints. These metrics are tightly coupled to the specific feature detector (SuperPoint) and matcher (SuperGlue) used in our pose estimation pipeline and are not directly comparable across VPS services; they are shown here for reference. We observe that, within the same pose estimation setup, these feature-based metrics provide good discriminability between correct and incorrect matches. The second row presents image-based comparison metrics between the rendered and query images, as described in § 5.6. Metrics such as SSIM, PSNR, and LPIPS show limited ability to distinguish between true and false matches when compared to feature-based approaches. The third row shows the cosine similarity of CLIP image encodings. 'CLIP-finetuned' refers to our version of CLIP, fine-tuned on our dataset to produce similar embeddings for corresponding rendered and query images. Among all metrics, the fine-tuned CLIP model demonstrates the best discriminability.

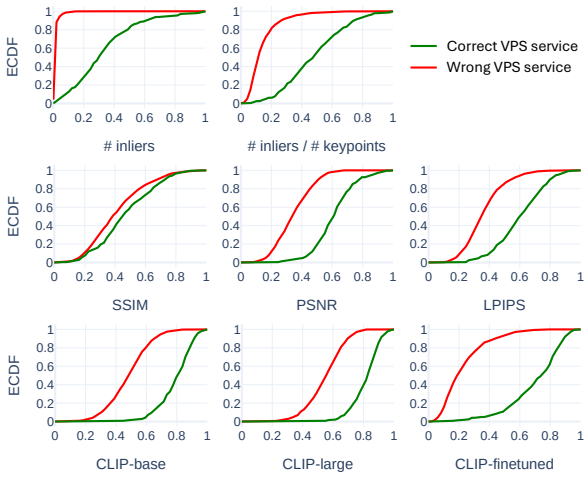


Figure 14: CDFs of metrics estimating pose confidences. The CLIP-finetuned model shows the best discriminability between correct and wrong VPS services.

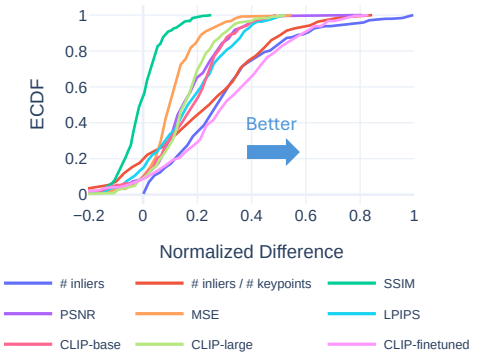


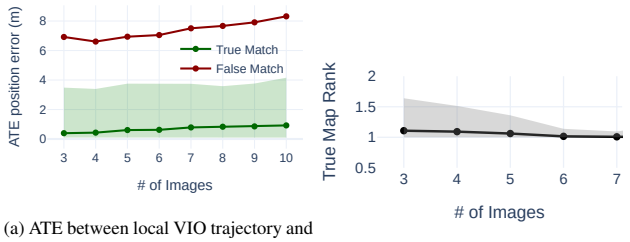
Figure 15: Discriminability of pose confidence metrics.

Figure 15 presents the empirical cumulative distribution functions (ECDFs) of normalized confidence score differences between correct and incorrect VPS queries across various pose confidence metrics. A higher curve indicates better discriminability between true and false localizations, as more samples exhibit a larger difference in confidence scores. Notably, the CLIP-finetuned method demonstrates the highest discriminative power supporting our decision to use it as our pose confidence metric.

8.4 VPS Selector

In § 5.7, we described how the device compares a remote trajectory from a VPS service with its own local VIO trajectory to select the appropriate VPS service. In this section, we show that this approach enables correct VPS selection after only a few localization cycles.

In this evaluation, we group five VPS services located in close proximity—such that they are all discovered by a single discovery query—and assess whether our VPS selector can correctly identify the appropriate service using only local VIO traces. Figure 16a



(a) ATE between local VIO trajectory and remote service trajectory.

(b) Mean rank of the correct VPS service.

Figure 16: Performance of VPS Selector. The correct map is reliably ranked at the top even with few localization cycles.

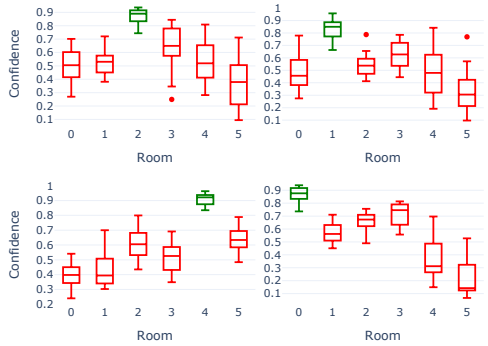


Figure 17: Performance of Place Recognizer module.

shows the absolute trajectory error (ATE) in position between the local VIO trajectory and the VPS-estimated trajectory, for both correct and incorrect service selections. The green shaded region represents the spread between the 5th and 95th percentile errors for correct matches. We observe that the ATE for incorrect matches increases as more images are localized. The 95th percentile error for true matches is lower than the median error for false matches, showing our technique's reliability in selecting the right candidate.

Figure 16b illustrates the mean rank of the correct VPS service against the number of localization cycles. A rank of one indicates that the service was selected as the top candidate by OpenFLAME on the device. Even with a small number of localization cycles, the correct VPS service consistently achieves a mean rank close to one. As the length of the compared trajectory increases, the variability in the estimated rank decreases, indicating more reliable selection.

8.5 Place recognizer

To evaluate the Place Recognizer module (§ 5.3), we consider a group of five VPS services covering places that are close and look visually similar to one another. Figure 17 shows the confidence scores produced by the Place Recognizer module for various rooms. For each query, a CLIP model fine-tuned to each individual room is used to compute a confidence score, and the results are visualized as box plots across six rooms. The green box in each subplot represents the correct room for the query, while the red boxes indicate other (incorrect) rooms.

Across all four subplots, the correct room consistently achieves the highest confidence score, clearly separated from the others. This demonstrates the discriminative power of the fine-tuned CLIP models in correctly identifying the room associated with a given query. The spread of the red boxes also highlights that incorrect rooms receive significantly lower and more varied confidence scores, reinforcing the reliability of the module in distinguishing between similar indoor environments.

9 Limitations and Future Directions

The infrastructure presented in this paper is only a step towards our vision of a widely deployed federated VPS infrastructure. Several important challenges require further careful consideration. Two that we consider briefly below are privacy and application design.

Privacy considerations: As discussed in § 5.7, the VPS Selector module requires poses for at least 3 images to identify the right VPS service for its current location. Therefore, initially, the client device has to broadcast the images of its current location to VPS Services that might not have coverage over the location. This opens the possibility for snooping attacks where a VPS Service might register itself adjacent to a sensitive private space (e.g., a secret lab), collect images sent to it during the discovery phase, and reconstruct the map of the private space. This is a violation of privacy. Past work has proven that it is possible to extract information from just extracted features without raw images [54, 29].

To overcome this challenge, VPS Services hosted for sensitive spaces can can adopt existing work on *privacy preserving localization* [51, 38, 50]. They perform localization on *secure features* (e.g., *feature lines* [51]) sent by the client instead of raw images. These secure features reveal little or no intelligible visual or location information to any party except an authorized server that possesses the correct map of the space. Therefore, even if the client broadcasts these secure features to multiple VPS services, only the service that has the correct map of the private space can decode these features and perform localization. Once the Selector has determined the correct VPS service for the location, the client no longer needs to broadcast visual data to multiple services and can continue interfacing with a single VPS service. In this phase, the VPS service can switch to using raw images so that it can take advantage of features such as semantic image masking running on the VPS service. Integrating existing work on privacy preserving localization with OpenFLAME is one of our future goals.

Application Design: Applications that use one of the existing ubiquitous VPS providers can anchor their content using latitudes, longitudes, and altitudes as their scans are laid out in the geographic coordinate system. As OpenFLAME does not have a unified global coordinate system, designing applications on top of OpenFLAME is tricky. A straight-forward solution is for applications to position their content using local coordinates of a VPS service and store a pointer to the corresponding VPS service. However, this makes the content tightly coupled with the VPS service, making it unusable when the VPS service is upgraded or replaced. Avoiding such tight-coupling in application design is an interesting future direction. An example solution would be for the VPS services to expose an additional interface that informs applications of the location of some *landmarks* with respect to their local coordinate system. Hallways, doors, elevators and stairs are examples of landmarks in indoor spaces. The application can then author and store content using landmarks as references. As landmarks do not change when VPS services are upgraded or replaced, content will no longer be coupled with specific VPS services.

10 Conclusion

We present OpenFLAME, a federated VPS system that allows independent organizations to support VPS for their private spaces. Federation of VPS introduces several challenges, such as selection of the right VPS service at a given location, coherence of localization results across service boundaries, and handling dynamic indoor spaces. In this paper, we provide some effective solutions to these challenges and show that the resulting system can provide efficient and accurate localization. Federation of VPS services paves the path for a ubiquitous localization backend, which will enable world-scale augmented reality applications of the future.

REFERENCES

[1] Apple AR Anchor. https://developer.apple.com/documentation/arkit/aranchor. Online. Accessed: April 2025. 1

[2] Apple ar kit. https://developer.apple.com/augmented-reality/arkit/. Online. Accessed: April 2025. 3, 6

[3] Apple argeoanchor. https://developer.apple.com/documentation/arkit/argeoanchor. Online. Accessed: April 2025. 3

[4] Arcgis. https://www.arcgis.com/index.html. Online. Accessed: April 2025. 3

[5] Carto. https://carto.com/. Online. Accessed: April 2025. 3

[6] Clip: Connecting text and images. https://openai.com/index/clip/. Online. Accessed: April 2025. 4

[7] Detection of aruco markers. https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html. Online. Accessed: April 2025. 2

[8] A frame. https://aframe.io/. Online. Accessed: April 2025. 6

[9] A frame inspector. https://github.com/aframevr/aframe-inspector. Online. Accessed: April 2025. 6

[10] Geofire for javascript. https://github.com/firebase/geofire-js. Online. Accessed: April 2025. 3

[11] Google ar core. https://developers.google.com/ar. Online. Accessed: April 2025. 3, 6

[12] Google Geospatial API. https://developers.google.com/ar/develop/geospatial. Online. Accessed: April 2025. 1, 3

[13] Google Indoor Maps. https://www.google.com/maps/about/partners/indoormaps/. Online. Accessed: July 2025. 1

[14] Google Street View. https://www.google.com/streetview/. Online. Accessed: April 2025. 1

[15] Mongodb. https://www.mongodb.com/. Online. Accessed: April 2025. 3

[16] Niantic Lightship. https://www.nianticspatial.com/products/niantic-sdk. Online. Accessed: April 2025. 1, 3

[17] OpenStreetMap. https://www.openstreetmap.org/. Online. Accessed: July 2025. 1

[18] Polycam. https://poly.cam/. Online. Accessed: April 2025. 3

[19] Postgis. https://postgis.net/. Online. Accessed: April 2025. 3

[20] Real-time kinematic positioning. https://en.wikipedia.org/wiki/Real-time_kinematic_positioning. Online. Accessed: April 2025. 3

[21] Total station. https://en.wikipedia.org/wiki/Total_station. Online. Accessed: April 2025. 3

[22] Understanding reprojection error. https://calib.io/blogs/knowledge-base/understanding-reprojection-errors?srsltid=AfmBOorYWYUp59LYtOaajmmLxoYMshGx7VDOUSaYdxJlxzEP6BM-eJ7A. Online. Accessed: April 2025. 5

[23] Webxr. https://immersiveweb.dev/. Online. Accessed: April 2025. 3, 6

[24] Yolov8. https://yolov8.com/. Online. Accessed: April 2025. 4, 6

[25] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5297–5307, 2016. 4

[26] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pp. 404–417. Springer, 2006. 2

[27] S. Bharadwaj, A. Rowe, and S. Seshan. Uniting the world by dividing it: Federated maps to enable spatial applications. In *Proceedings of the 20th Workshop on Hot Topics in Operating Systems*, 2025. 3

[28] D. DeTone, T. Malisiewicz, and A. Rabinovich. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 224–236, 2018. 2, 4

[29] A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4829–4837, 2016. 9

[30] M. FISCHLER AND. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981. 5

[31] R. Gibb, A. Madhavapeddy, and J. Crowcroft. Where on earth is the spatial name system? In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, pp. 79–86, 2023. 3

[32] R. T. Gibb. Spatial name system. *arXiv preprint arXiv:2210.05036*, 2022. 3

[33] T. Jin, S. Wu, M. Dasari, K. Apicharttrisorn, and A. Rowe. Stagear: Markerless mobile phone localization for ar in live events. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pp. 1000–1010. IEEE, 2024. 1

[34] N. Keetha, J. Karhade, K. M. Jatavallabhula, G. Yang, S. Scherer, D. Ramanan, and J. Luiten. Splatam: Splat track & map 3d gaussians for dense rgb-d slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21357–21366, 2024. 4, 5

[35] S. Li, C. Xu, and M. Xie. A robust o (n) solution to the perspective-n-point problem. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1444–1450, 2012. 2

[36] P. Lindenberger, P.-E. Sarlin, and M. Pollefeys. Lightglue: Local feature matching at light speed. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17627–17638, 2023. 2

[37] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110, 2004. 2

[38] H. Moon, C. Lee, and J. H. Hong. Efficient privacy-preserving visual localization using 3d ray clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9773–9783, June 2024. 9

[39] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: A versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015. 2

[40] E. Olson. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE international conference on robotics and automation*, pp. 3400–3407. IEEE, 2011. 2

[41] L. Pan, D. Barath, M. Pollefeys, and J. L. Schönberger. Global Structure-from-Motion Revisited. In *European Conference on Computer Vision (ECCV)*, 2024. 1

[42] W. Pang, C. Xia, B. Leong, F. Ahmad, J. Paek, and R. Govindan. Ubipose: Towards ubiquitous outdoor ar pose tracking using aerial meshes. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, pp. 1–16, 2023. 1

[43] M. Rizk, A. Mroue, M. Farran, and J. Charara. Real-time slam based on image stitching for autonomous navigation of uavs in gnss-denied regions. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 301–304. IEEE, 2020. 5

[44] A. Rosinol, J. J. Leonard, and L. Carlone. Nerf-slam: Real-time dense monocular slam with neural radiance fields. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3437–3444. IEEE, 2023. 4, 5

[45] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pp. 2564–2571. Ieee, 2011. 2

[46] E. Sandström, Y. Li, L. Van Gool, and M. R. Oswald. Point-slam: Dense neural point cloud-based slam. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023. 4

[47] P.-E. Sarlin, C. Cadena, R. Siegwart, and M. Dymczyk. From coarse to fine: Robust hierarchical localization at large scale. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12716–12725, 2019. 4, 6

[48] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4938–4947, 2020. 2, 4

[49] J. L. Schonberger and J.-M. Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4104–4113, 2016. 1, 2, 6

[50] M. Shibuya, S. Sumikura, and K. Sakurada. Privacy preserving visual slam. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII 16*, pp. 102–118. Springer, 2020. 9

[51] P. Speciale, J. L. Schonberger, S. N. Sinha, and M. Pollefeys. Privacy preserving image queries for camera localization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1486–1496, 2019. 9

[52] Q. Tang, K. Zhang, P. Xu, J. Zhang, and Y. Cui. Map fusion method based on image stitching for multi-robot slam. In *Advances in Swarm Intelligence: 12th International Conference, ICSI 2021, Qingdao, China, July 17–21, 2021, Proceedings, Part II 12*, pp. 146–154. Springer, 2021. 5

[53] J. Wang and E. Olson. Apriltag 2: Efficient and robust fiducial detection. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4193–4198. IEEE, 2016. 7

[54] P. Weinzaepfel, H. Jégou, and P. Pérez. Reconstructing an image from its local descriptors. In *CVPR 2011*, pp. 337–344. IEEE, 2011. 9

[55] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2002. 6

[56] Z. Zhu, S. Peng, V. Larsson, W. Xu, H. Bao, Z. Cui, M. R. Oswald, and M. Pollefeys. Nice-slam: Neural implicit scalable encoding for slam. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12786–12796, 2022. 4