Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

An Efficient Two-Stage Pipeline for On-Device Visual Localization

Anastasia Archangelskaya

ceo@navig.io

Skolkovo Institute of Science and Technology (Skoltech)

Ivan Gerasimov

Skolkovo Institute of Science and Technology (Skoltech)

Gonzalo Ferrer

Skolkovo Institute of Science and Technology (Skoltech)

Robert Vasilev

Research Article

Keywords: Visual Localization, Augmented Reality (AR), Structure-from-Motion (SfM), On-Device Processing, Mobile Computing, Pose Estimation

Posted Date: July 20th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-8855017/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

An Efficient Two-Stage Pipeline for On-Device Visual Localization

Anastasia Archangelskaya[1*], Ivan Gerasimov[1], Robert Vasilev[1], Gonzalo Ferrer[1]

[1]Skolkovo Institute of Science and Technology (Skoltech), Moscow, Russia.

*Corresponding author(s). E-mail(s): anastasia.archangelskaya@skoltech.ru;

Abstract

We present an efficient two-stage pipeline for robust, real-time 6-DoF visual localization on resource-constrained mobile devices such as smartphones and AR glasses. Many AR navigation systems rely on cloud-based visual positioning services (VPS), which require network connectivity and may introduce latency. Fully on-device alternatives often face prohibitive compute and memory costs when operating on large-scale 3D maps.

Our approach addresses these limitations by separating coarse spatial initialization from precise pose estimation. Offline, we build a metrically consistent 3D map using an SfM pipeline and prepare a compact visual database that is partitioned into overlapping zones. Online, the device first determines the current zone using inexpensive external cues (e.g., Wi-Fi, BLE, or QR). These signals are used only as discrete identifiers to select and load a zone-specific database shard from local storage. The final pose is then estimated purely from visual data by retrieving candidate keyframes within the active zone, matching local features, and solving PnP with RANSAC.

Experiments across diverse environments show that zone-based database pruning reduces runtime and power consumption while maintaining localization accuracy suitable for offline AR navigation.

Keywords: Visual Localization, Augmented Reality (AR), Structure-from-Motion (SfM), On-Device Processing, Mobile Computing, Pose Estimation

1 Introduction

Accurate, low-latency six-degree-of-freedom (6-DoF) camera re-localization is a core enabler of practical AR/VR/XR experiences, including indoor–outdoor navigation, stable spatial anchoring, and consistent placement of virtual content in large environments. In this paper we address the *map-based re-localization* setting: given a pre-built map of a place, estimate the 6-DoF pose of a resource-constrained mobile device (smartphone or head-worn XR device) from a single query frame.

Deploying map-based re-localization at scale is challenging because large places require large reference databases (images, 3D points, descriptors), while mobile hardware imposes strict constraints on latency, memory footprint, and power consumption. Cloud-assisted localization services for mixed-reality systems can scale well, but they rely on network connectivity and remote compute, which is undesirable for offline or latency-critical XR guidance [1, 2]. On-device SLAM provides accurate real-time tracking, yet re-localization against large pre-built maps remains constrained by memory and compute budgets when the searchable map is large [3]. Hierarchical visual localization pipelines based on global retrieval followed by local matching and geometric verification are state-of-the-art for large-scale localization, but running them over a full, unpartitioned database is still costly on commodity mobile devices [4, 5]. Finally, coarse signals such as Wi-Fi can provide room-level cues, but they are typically used as priors/measurements rather than as an explicit mechanism to control the amount of visual data searched on-device [6, 7].

To bridge this deployment gap for AR/VR/XR, we propose a fully on-device, two-stage pipeline that explicitly separates *coarse zone selection* from *fine visual pose estimation*. A discrete *zone hint* (e.g., Wi-Fi/BLE/QR) is used only to select a compact zone-specific database shard, after which the device executes standard hierarchical localization *inside the selected shard* (global retrieval, local matching, and PnP+RANSAC), producing a 6-DoF pose suitable for XR navigation and stable spatial anchoring.

NavMesh reference frame. In this paper, *NavMesh* denotes the polygonal navigation mesh of walkable space used by the XR application for routing and content placement (i.e., an application asset, not a prior work). We use the NavMesh coordinate system as the global world reference frame and express all reported camera poses in this NavMesh-aligned frame [8].

Figure 1 summarizes the practical XR deployment and evaluation loop considered in this work. For each mapped place, we acquire reference imagery and build a metric 3D reconstruction, then align the reconstruction to the NavMesh coordinate system used by the AR/XR application and partition it into zones to enable zonal filtering and compact per-zone database shards. During operation, the mobile client performs fully on-device localization from a query frame and outputs a NavMesh-aligned 6-DoF pose for stable spatial anchoring and navigation. Finally, we perform offline evaluation of accuracy and performance by replaying queries and analyzing estimated trajectories in the reconstructed environment.

Our main contributions are: (i) a zone-filtered on-device re-localization architecture with predictable latency and memory usage; (ii) an offline map-to-NavMesh alignment and per-zone database packaging pipeline tailored to large places; (iii) an on-device

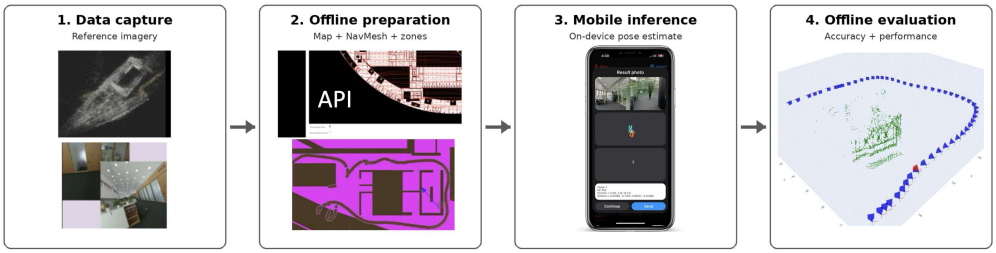


Fig. 1 Complete workflow used in this paper. (1) We capture reference imagery for each place. (2) Offline, we reconstruct a metric map and prepare the deployment database by aligning the reconstruction to the application NavMesh reference frame and annotating zones (for zonal filtering). (3) On-device, the mobile client localizes each query frame and outputs a NavMesh-aligned 6-DoF pose. (4) Offline, we evaluate accuracy and runtime by comparing estimated poses to ground truth and visualizing trajectories in the reconstructed scene.

evaluation across heterogeneous environments and database sizes, quantifying the accuracy–runtime trade-offs relevant to real XR deployments.

2 Related Work

Cloud-assisted visual positioning and spatial anchors.

Porter et al. [1] and Glenn et al. [2] describe industrial AR systems where a remote service and/or shared spatial anchors support localization and location-based content delivery. While such architectures can scale to large environments, they rely on network connectivity and server-side compute, which is undesirable for latency-critical or fully offline navigation.

Hierarchical visual localization as place recognition.

Large-scale visual localization is often formulated as place recognition with learned global descriptors for fast retrieval (e.g., NetVLAD [4]), followed by local matching and geometric verification to recover a precise 6-DoF pose [5]. On-device SLAM systems such as ORB-SLAM [3] provide accurate real-time tracking in small to medium environments; however, re-localization against large pre-built maps can be constrained by mobile memory and compute budgets.

Signal-assisted and multi-stage localization.

Place Lab [6] and the Wi-Fi/vision fusion approach of Guo et al. [7] use Wi-Fi measurements to provide coarse localization cues and improve robustness. For AR navigation, Mulloni et al. [9] use fiducial markers as reliable anchors. A multi-stage AR localization logic that guides users from coarse regions to finer areas using visual anchors and zone-specific logic is described by Pajouh et al. [10]. In contrast to sensor-fusion approaches, our method uses ambient identifiers (Wi-Fi/BLE/QR) only as discrete zone keys to select a compact, zone-specific database shard; the final pose is estimated purely from visual correspondences within that shard (global retrieval + local matching + PnP/RANSAC).

Positioning of our contribution.

Prior work establishes the effectiveness of hierarchical retrieval-plus-matching for large-scale localization (e.g., NetVLAD [4] and coarse-to-fine localization [5]) and the practical value of coarse non-visual cues such as Wi-Fi [6, 7]. Building on these ideas, we focus on a deployment-oriented, fully offline on-device design: we package the map into per-zone shards and use external signals only for *zonal filtering* (database selection), not as inputs to the pose solver. This bounds the on-device search space and yields predictable latency and memory usage on commodity mobile hardware.

3 Proposed Method

We propose a two–stage, zone–aware visual localization pipeline tailored for large indoor–outdoor environments and resource-constrained mobile devices (Figure 2). The core idea is to decouple coarse spatial initialization from fine 6-DoF pose estimation: inexpensive external signals are used only to select a compact subset of the map (a *zone*), while the final camera pose is recovered exclusively from visual data within this subset.

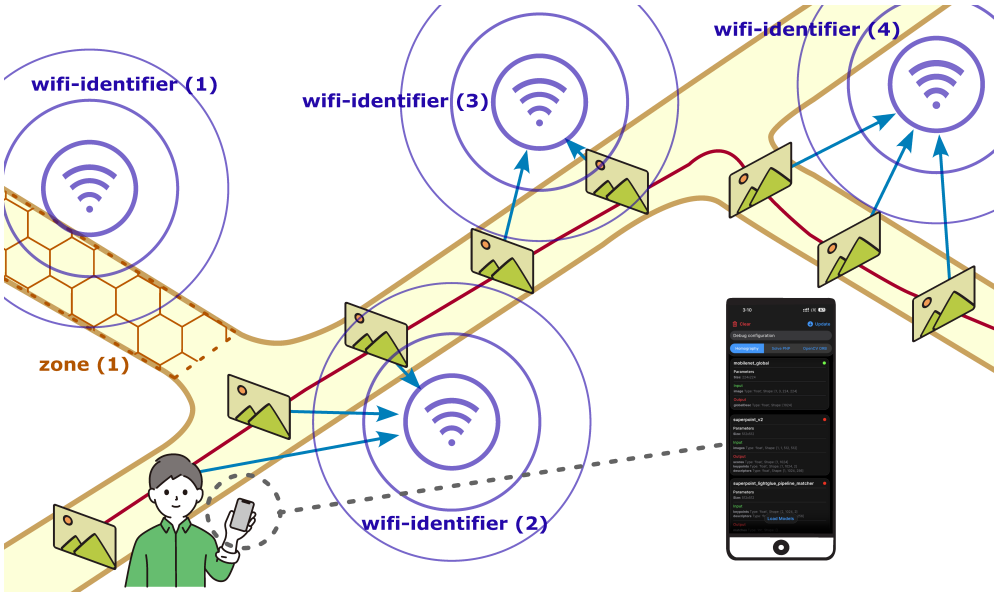


Fig. 2 Zone-hint based sharding for on-device visual localization. Reference keyframes captured along a route are assigned to Wi-Fi labels (zones) during offline mapping. At runtime, the device reads the current Wi-Fi label to select the most likely zone and restrict retrieval and matching to the corresponding zone shard.

3.1 Preliminaries: terminology, notation, and pose definition

We use the term *place* to denote a mapped site—the unit of deployment for which we collect reference imagery and later expect queries. Conceptually, a place can be described by what can be observed from camera poses along traversals (e.g., as a set/graph of reference views), and represented by a metric map and/or a topological structure [11, 12]. Following standard Structure-from-Motion pipelines [12], we reconstruct a metric 3D *map* and derive a compact set of reference *keyframes* and associated 3D points. The place is partitioned into spatial *zones* to enable *zonal filtering*, i.e., restricting on-device retrieval and matching to a small zone-specific database shard in order to reduce latency and memory usage on mobile hardware.

Place. A mapped site (e.g., a building, campus, or outdoor area) represented by (i) a set of reference views (keyframes) acquired from known poses and (ii) the derived metric map. In this paper, a place is therefore defined operationally by what is observable from the available reference viewpoints, rather than by a specific map representation.
Map. A metric 3D representation reconstructed from the place reference imagery, typically as a sparse set of 3D points and calibrated camera poses produced by an SfM pipeline and geometric refinement [12].
Zone. A spatial partition of a place used to restrict on-device search to a compact subset of the reference database (a per-zone *shard*), improving latency and memory usage.
Zone shard D_z. The on-device package associated with zone z, storing only the keyframes, descriptors, and geometric data required to localize within that zone.
Zone hint. A coarse, discrete signal used only to select the most likely zone (not as a metric position measurement). In [6], Wi-Fi identifiers are used to provide coarse location cues indoors.
Zonal filtering. The policy that restricts retrieval and matching to the active shard D_z rather than the full database of the place. This reduces both compute and memory footprint per query.
Ground Control Points (GCPs). In [13], GCPs are reference points with known coordinates used to tie the reconstruction to a metric coordinate system (scale/orientation/absolute alignment).
Global retrieval (place recognition). In [4], compact global image descriptors enable fast retrieval of candidate keyframes for a query image.
Hierarchical localization. In [5], localization is performed in a coarse-to-fine manner: global retrieval proposes candidates, then local matching and geometric verification recover an accurate pose.

Problem definition and pose representation

We consider map-based re-localization in a mapped environment (a *place*) for which an offline pipeline provides a metric 3D map and a compact reference database (keyframes, descriptors, and associated 3D points). The place is partitioned into spatial *zones* to enable *zonal filtering*, i.e., restricting on-device retrieval and matching to a small zone-specific database shard in order to reduce latency and memory usage on mobile hardware.

Coordinate frames and NavMesh alignment. Let w denote the application world frame defined by the NavMesh, m the initial SfM reconstruction frame, and c the camera frame of a query image. Offline, we estimate an aligning similarity transform $\mathbf{S}_{w\leftarrow m} \in \mathrm{Sim}(3)$ from anchor correspondences (e.g., GCPs / manually selected reference points) and apply it to the reconstructed 3D points and camera poses so that the map is expressed in the NavMesh world frame [14]. Therefore, all map points $\mathbf{X}_i$ used below are in the NavMesh-aligned world frame w, and the on-device output pose is directly usable by the AR navigation engine.

Pose and 6-DoF.

We denote by w the (NavMesh-aligned) world frame and by c the camera frame. The rigid transform $\mathbf{T}_{w\leftarrow c}$ maps coordinates from camera to world, i.e., $\mathbf{X}_w = \mathbf{R}_{w\leftarrow c}\mathbf{X}_c + \mathbf{t}_{w\leftarrow c}$ [12, 15]. The SfM reconstruction is initially obtained in an arbitrary map frame; we align it to the NavMesh world frame via a similarity transform estimated from corresponding

Following standard multi-view geometry conventions [15], we represent the camera pose in the world frame as a rigid transform

$$\mathbf{T}_{w\leftarrow c} = \begin{bmatrix} \mathbf{R}_{w\leftarrow c} & \mathbf{t}_{w\leftarrow c} \\ \mathbf{0}^\top & 1 \end{bmatrix} \in SE(3), \quad \mathbf{R}_{w\leftarrow c} \in SO(3),\ \mathbf{t}_{w\leftarrow c} \in \mathbb{R}^3, \tag{1}$$

which maps coordinates from the camera frame to the world frame: $\mathbf{X}_w = \mathbf{R}_{w\leftarrow c}\mathbf{X}_c + \mathbf{t}_{w\leftarrow c}$. We refer to $(\mathbf{R}_{w\leftarrow c}, \mathbf{t}_{w\leftarrow c})$ as a 6-DoF pose (3 DoF translation + 3 DoF rotation). When convenient, we encode the rotation as a unit quaternion $\mathbf{q}_{w\leftarrow c}$ with $\|\mathbf{q}_{w\leftarrow c}\|_2 = 1$ and $\mathbf{R}_{w\leftarrow c} = \mathbf{R}(\mathbf{q}_{w\leftarrow c})$.

Pose estimation (definition and mathematical objective). Given a query image I_q and a zone-specific map shard, local matching produces a set of 2D–3D correspondences $\mathcal{C} = \{(\mathbf{u}_i, \mathbf{X}_i)\}_{i=1}^N$, where $\mathbf{u}_i \in \mathbb{R}^2$ is a pixel coordinate in I_q and $\mathbf{X}_i \in \mathbb{R}^3$ is a 3D map point in the NavMesh world frame w. For projection, we use the inverse (world-to-camera) transform $\mathbf{T}_{c\leftarrow w} = \mathbf{T}_{w\leftarrow c}^{-1}$ and estimate it by minimizing reprojection error:

$$(\hat{\mathbf{R}}_{c\leftarrow w}, \hat{\mathbf{t}}_{c\leftarrow w}) = \arg\min_{\mathbf{R}\in SO(3), \mathbf{t}\in\mathbb{R}^3} \sum_{i\in\mathcal{I}} \|\pi(\mathbf{K}\,(\mathbf{R}\mathbf{X}_i + \mathbf{t})) - \mathbf{u}_i\|_2^2\,, \tag{2}$$

where $\mathbf{K}$ is the camera intrinsics matrix, $\pi(\cdot)$ is the perspective projection, and $\mathcal{I}$ is an inlier set selected by a robust estimator. In practice, we solve (2) using a PnP solver inside RANSAC [16, 17]. The final output reported to the application is the NavMesh-aligned camera pose in the world frame, $\hat{\mathbf{T}}_{w\leftarrow c} = \hat{\mathbf{T}}_{c\leftarrow w}^{-1}$.

3.2 Two-stage pipeline overview

Figure 3 summarizes the proposed two-phase workflow. Offline (once per place), following standard Structure-from-Motion and bundle adjustment pipelines [12, 18] and metric scaling with Ground Control Points [13], we reconstruct a metric 3D map (sparse point cloud and camera poses) from captured images/video. We then perform coverage-preserving database optimization via graph-based keyframe selection, pre-compute compact global descriptors for retrieval and local keypoints/descriptors

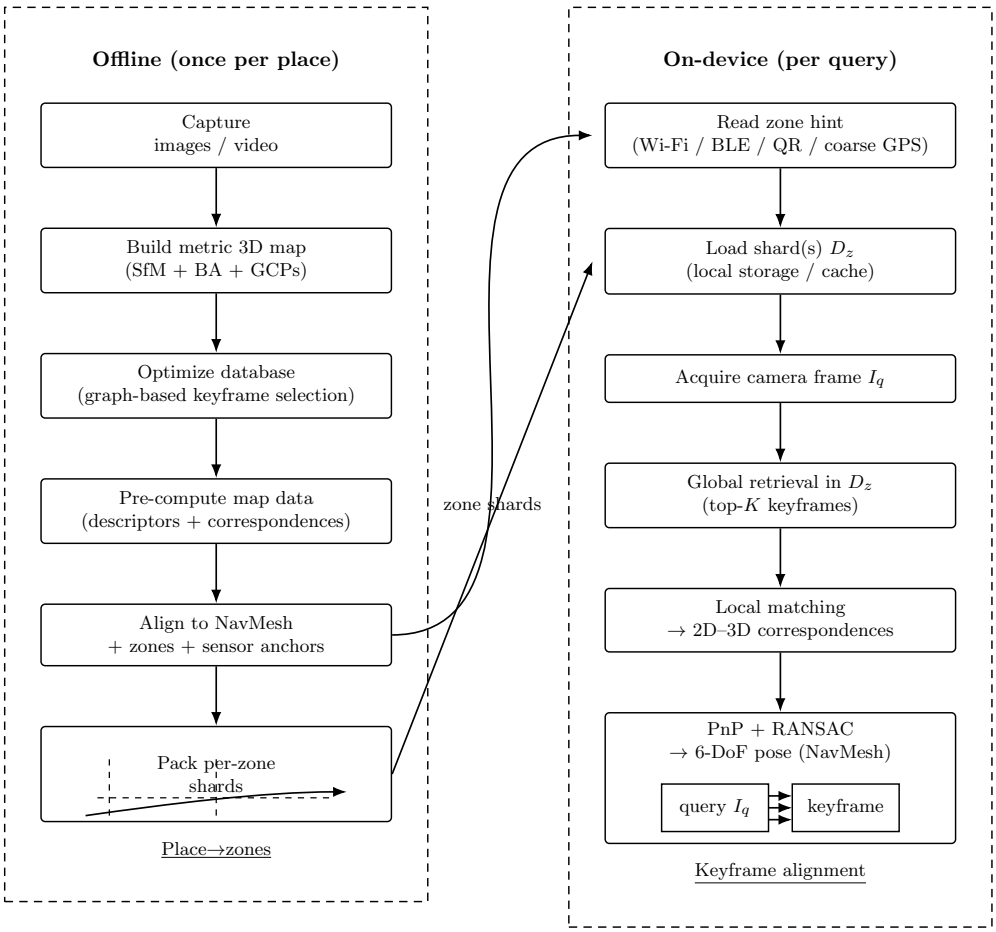


Fig. 3 Conceptual overview: offline map creation and per-zone database preparation (left), and on-device pose estimation per query (right).

for matching (e.g., SuperPoint [19]), and store the geometric associations required for pose estimation (2D observations linked to 3D map points). Next, we align the reconstructed map to the NavMesh coordinate frame used by the AR application [8], annotate zone boundaries and sensor anchors, and pack the reference data into per-zone database shards. Online (on-device, per query), the client determines the active zone from a discrete hint (e.g., Wi-Fi/BLE/QR/coarse GPS), loads the corresponding shard(s) from local storage/cache, and acquires a query frame I_q. In a hierarchical localization paradigm [20], the query global descriptor retrieves the top-K candidate keyframes within the active shard; local feature matching (e.g., LightGlue [21]) establishes 2D–3D correspondences, and a PnP solver (e.g., EPnP [16]) within RANSAC [17] estimates the 6-DoF pose in the NavMesh-aligned world frame for downstream AR navigation.

As illustrated in Fig. 2, we partition each mapped place into overlapping zones anchored by discrete "zone hints" such as Wi-Fi labels (and optionally BLE/QR). During offline map creation, we capture image sequences, reconstruct a metric 3D map, select a compact set of keyframes, and associate each keyframe with one or more nearby zones based on the observed hints. On-device, the system first estimates the current zone from the available hint (independent of the camera query), loads the corresponding per-zone database shard, and then localizes a captured query frame by global retrieval and local feature matching within that shard. Finally, the 6-DoF pose is computed via PnP with RANSAC using the resulting 2D–3D correspondences, yielding a pose in the place/NavMesh coordinate frame.

3.3 Preliminary offline stage: map and per-zone database construction

At the offline stage, each target site is reconstructed into a metrically consistent 3D model using a standard photogrammetric pipeline. From the raw capture we derive a lean representation by (i) selecting a reduced set of keyframes that still covers the navigable space and viewpoint diversity, and (ii) partitioning this representation into overlapping zones that follow the semantic and structural layout of the environment (e.g., floors, wings, streets, park segments). For every keyframe we pre-compute learned global descriptors for image retrieval and local keypoint-based descriptors for geometric matching, and associate them with 3D points and camera poses in the same coordinate system as the navigation mesh. The resulting per-zone bundles are stored in a compact hybrid database suitable for deployment on mobile devices.

(1) Data capture. We acquire images or video covering the navigable space of a place with sufficient overlap and viewpoint diversity.

(2) Metric reconstruction (SfM + BA + GCPs). In [12], SfM recovers camera poses and a sparse 3D structure from multi-view correspondences; in [18], BA refines cameras and points by minimizing reprojection error. To obtain a metric and globally consistent coordinate frame, we incorporate GCPs as in [13].

(3) Database optimization (graph-based keyframe selection). We build a view-graph over reconstructed cameras and select a reduced set of keyframes that preserves spatial coverage and viewpoint diversity while reducing redundancy, which directly decreases memory and compute at runtime.

(4) Pre-computation of retrieval and geometry data. For each keyframe, we compute a compact global descriptor for fast retrieval (place recognition) as in [4], and store the geometry required to later form 2D–3D correspondences for pose estimation.

(5) Alignment to NavMesh, zoning, and sensor anchors. We align the reconstructed map to the NavMesh coordinate system used by the AR navigation engine, then define zones (and optionally overlapping boundaries) and associate each zone with sensor anchors that can provide a zone hint.

(6) Packaging per-zone shards. We pack the reference database into per-zone shards D_z, each containing only the keyframes and data required to localize within that zone, enabling zonal filtering on-device.

3.4 On-device stage: zone-filtered pose estimation per query

At runtime, localization proceeds in a coarse-to-fine manner. In the first stage, the device detects one or more ambient identifiers (Wi-Fi SSID/BSSID, BLE beacons, QR codes or similar infrastructure) and uses them solely as discrete keys to activate the most likely zone and load its corresponding database shard from local storage. These signals are *not* fused as metric measurements; instead, they serve to aggressively prune the search space to a small, zone-specific subset of reference images and 3D points. In the second stage, fine visual localization is performed using only this active subset: a lightweight learned global descriptor computed on the live camera frame retrieves the most relevant keyframes inside the zone, a learned local feature extractor and matcher establish 2D–3D correspondences, and a robust PnP-with-RANSAC solver estimates the 6-DoF camera pose in the global (NavMesh) coordinate system. Standard caching and preloading strategies (e.g., LRU-based zone caching and anticipatory loading of neighboring zones) keep latency low during zone transitions while preserving a small memory footprint.

(1) Zone selection from hints (independent of image capture). The device reads a coarse zone hint (e.g., Wi-Fi/BLE/QR). The hint is used only to select the most likely zone and activate the corresponding shard(s) D_z; it is not used as a metric measurement.

(2) Shard loading and caching. The selected shard(s) D_z are loaded from local storage (or cache). This bounds memory and avoids searching the full place-level database.

(3) Query acquisition. The device acquires a camera frame I_q and computes its global descriptor.

(4) Global retrieval within D_z. We retrieve the top-K most similar keyframes within the active shard, following the hierarchical localization paradigm in [5].

(5) Local matching and correspondence construction. We perform local matching between I_q and the retrieved keyframes to obtain 2D–3D correspondences $\mathcal{C}$.

(6) Robust pose estimation (PnP + RANSAC). We estimate the 6-DoF pose by solving PnP [16] within RANSAC [17], producing $\hat{\mathbf{T}} \in SE(3)$ in the NavMesh frame for downstream AR navigation and content placement.

3.5 Advantages of the proposed approach

This design offers several advantages over single-stage full-map localization and purely cloud-based Visual Positioning Systems. First, zonal pruning and database compression make the approach practical on mid-range mobile hardware: computation and memory scale with the size of a zone rather than the size of the entire map, enabling real-time performance and reduced energy consumption. Second, the system is fully offline-capable, since all models and per-zone databases reside on the device and external signals are used only as symbolic hints. Third, the representation is modular: zones, keyframes and descriptor types can be updated independently via the administration tools without changes to the client code. Finally, the same two-stage pipeline naturally supports different deployment profiles and database sizes, providing explicit

control over the trade-off between storage, runtime and localization robustness that we analyze in the subsequent sections.

3.6 Baseline Hierarchical Localization Pipeline

The core per–frame localization pipeline used in our experiments (global descriptor retrieval, followed by local feature matching and PnP+RANSAC pose estimation) is intentionally aligned with the state–of–the–art hierarchical localization paradigm [20]. Concretely, our "non-zonal" baseline corresponds to a standard hierarchical pipeline that:

- uses learned global image descriptors for coarse retrieval over the *entire* reference database of size N;
- refines the estimate by matching learned local features between the query image and the retrieved keyframes;
- estimates a 6-DoF pose via PnP+RANSAC using 2D–3D correspondences.

The proposed zonal approach does not change this core hierarchy; instead, it reorganizes the map and the database so that the same hierarchical pipeline operates on a compact, zone-specific subset of reference images. In other words, our contribution is orthogonal to the choice of descriptors and matcher: any HLoc–style or HF-Net–style pipeline can run on top of the zonal database.

In this work we systematically evaluated a wide range of global and local backbones (MobileNetV2/V3 [22, 23], GhostNet [24], NetVLAD [4], MixVPR [25], CosPlace [26], AnyLoc [27] for global descriptors; SIFT [28], ORB [29], SuperPoint [19], DISK [30], ALIKED [31] for local descriptors; FLANN [32], BFMatcher [33], LightGlue [34] and SuperGlue [35] for matching). These experiments show that the combination of lightweight convolutional backbones with NetVLAD/GeM aggregation for global descriptors [4, 36] and SuperPoint+LightGlue for local matching [19, 34] offers the best trade-off between accuracy and runtime on mobile devices. The IEEE paper therefore focuses on this strongest configuration as the underlying hierarchical baseline.

3.7 Relation to HLoc and Hierarchical Localization

HLoc [20] is widely regarded as a "gold standard" large-scale localization system: it performs coarse global retrieval followed by fine local matching, typically using SuperPoint and SuperGlue, and estimates poses with PnP+RANSAC on a server-grade GPU. Our pipeline follows the same algorithmic structure and uses the same family of learned features, but is engineered for fully offline, on-device operation:

- The 3D maps used in our system are built with an HLoc-style offline SfM pipeline (SuperPoint + global descriptors + Bundle Adjustment), as detailed in Sec. 3.2.
- At test time, the "non-zonal" configuration in Sec. 6 is equivalent in spirit to running HLoc's hierarchical retrieval directly on the full database for each site.
- The zonal configuration adds an additional coarse stage that uses inexpensive external identifiers (Wi-Fi/BLE/QR) to pre-select a small subset of keyframes before the hierarchical retrieval, thereby reducing the effective database size per query.

Thus, our comparison between zonal and non-zonal configurations should be interpreted as *HLoc-style hierarchical localization with and without zonal pruning*. The fact that we maintain sub-meter accuracy while reducing runtime and memory consumption (Tables 2–3) indicates that the proposed representation and data-management scheme improve the deployability of hierarchical localization on mid-range mobile hardware without changing its core estimation algorithm.

3.8 Relation to ORB–SLAM3 and Atlas–Style SLAM

Another important line of work is keypoint-based SLAM, exemplified by ORB–SLAM3. These systems estimate camera trajectories and dense maps online using handcrafted features (ORB) and bundle adjustment. We experimented with ORB features and ORB–SLAM3 components in our early prototypes. While ORB–based pipelines are computationally attractive, on our indoor–outdoor datasets they produced significantly fewer stable keypoints and were more sensitive to lighting and viewpoint changes than learned features such as SuperPoint. This resulted in lower recall and less robust re-localization, consistent with findings reported in the literature.

Moreover, our target use case is occasional re-localization for AR navigation, rather than continuous odometry and loop-closure as in full SLAM. For this regime, image-based hierarchical localization with learned descriptors provides better robustness, while SLAM-style methods are complementary and could be integrated to improve short-term tracking. A full head-to-head comparison with ORB–SLAM3 in Atlas mode is therefore left as future work, and we explicitly limit our claims to the re-localization scenario.

3.9 Relation to Cloud–Based VPS Systems

Commercial Visual Positioning Services such as Google Cloud Anchor / ARCore VPS and Niantic Lightship VPS provide high-accuracy localization in city-scale environments, but rely on cloud connectivity, proprietary maps and infrastructure. Direct benchmarking against these services on our datasets is not feasible for several reasons:

- Their mapping pipelines and descriptors are closed-source, and their maps cannot be exported to populate our offline database.
- These services require continuous network access and server-side processing, while our design explicitly targets fully offline operation on the client device (Sec. 4).
- Licensing and data-policy restrictions prohibit using them as drop-in baselines on many of the indoor sites where we deploy Navigio (museums, hospitals, university buildings).

Instead, our strategy is to adopt similar architectural principles (global retrieval + local matching + PnP) and to demonstrate, on our own maps, that we can achieve sub-meter accuracy with latencies compatible with interactive AR, while keeping all data and computation on device. In this sense, the proposed system should be viewed as an offline, on-premise alternative to VPS: it follows the same high-level paradigm but replaces the cloud with an optimized zonal database and mobile-friendly models.

Table 1 Qualitative comparison with representative visual localization systems.

Method	On-device	Fully offline	Large-scale map	Uses zones
ORB–SLAM3 (Atlas)	✓	✓	limited by drift	×
Hierarchical Loc. (HLoc)	×	×	city-scale	×
Cloud VPS (ARCore/Lightship)	×	×	city-scale	×
Proposed (zonal)	✓	✓	campus / district	✓

3.10 Qualitative Summary

Table 1 qualitatively compares the proposed system with representative baselines. Our contribution is not a new localization backend per se: we adopt a standard hierarchical pipeline (global retrieval + local matching + PnP/RANSAC) as the core estimator [5, 20]. The novelty lies in how this estimator is made deployable on resource-constrained phones at large scale and fully offline: we introduce a zone-keyed sharding of the reference database, an offline map-to-NavMesh alignment and packaging pipeline, and a deployment layout enabling bounded memory usage and predictable latency per query.

4 System Architecture and Methodology

In this section, we describe the system architecture used in all experiments, the construction of ground-truth poses, the evaluation metrics, the experimental protocol, and the baseline without zonal filtering.

4.1 Baseline Without Zonal Filtering

To isolate the effect of zonal filtering, we define a single-stage baseline that uses the same models and implementation but ignores zones.

For each site and database size N the baseline operates as follows:

- All reference keyframes and descriptors for the site are loaded into a single global database, without partitioning into zones.
- For each query frame we compute the global descriptor and retrieve the top-K nearest neighbors over the *entire* site-level database of size N.
- Local feature extraction, matching, 2D–3D correspondence building and PnP+RANSAC pose estimation are identical to the proposed method.

Thus, the baseline and our method share the same feature extractors, matcher, PnP implementation and success criteria; the only difference is whether the search is performed in a zone-specific subset or in the full site-level database. This guarantees a fair comparison and allows us to attribute performance and runtime differences solely to zonal pruning.

In Section 6 we report detailed results for the proposed zonal method and discuss, qualitatively, how it compares to the non-zonal baseline in terms of speed and robustness for large databases.

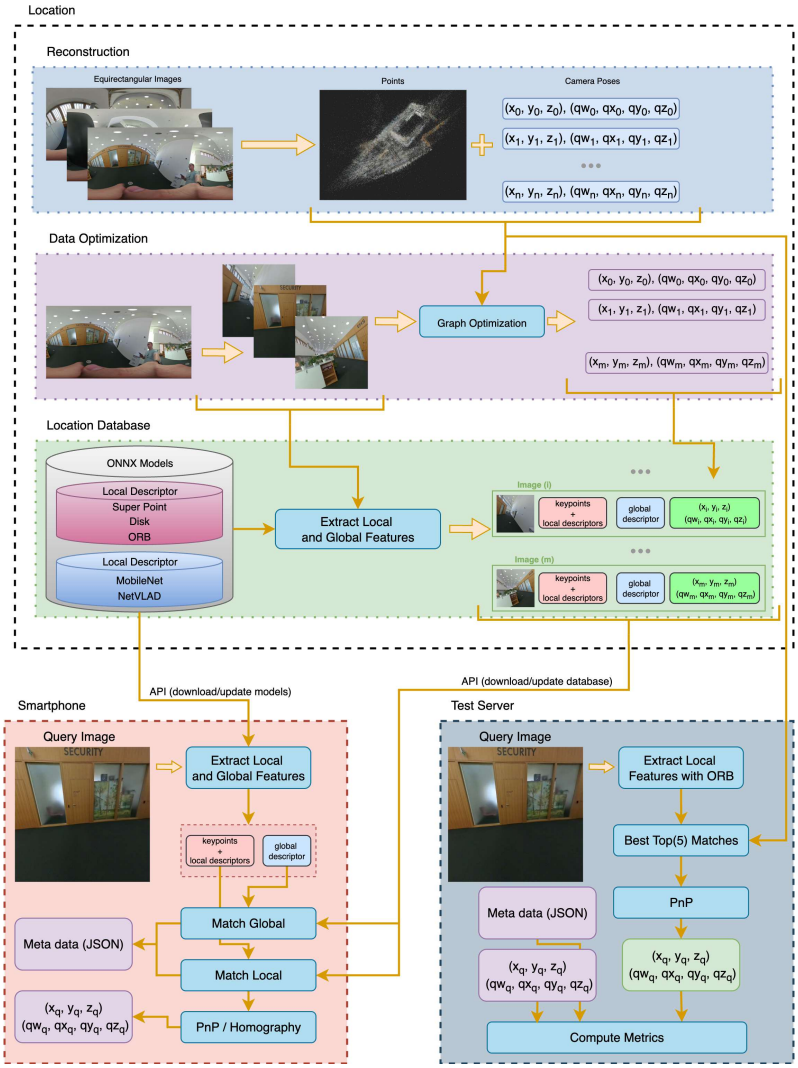


Fig. 4 Baseline Without Zonal Filtering,

4.2 System Architecture and Experimental Testbed

The proposed two-stage localization pipeline is implemented inside a production AR navigation system, deployed in the Navigio mobile application [37] and its associated cloud backend. The architecture comprises three tightly coupled components: (i) an offline data preparation and 3D reconstruction pipeline, (ii) an on-device localization module integrated into a Unity-based mobile client, and (iii) a web-based administration panel with backend services for data management and analytics.

4.2.1 Offline Map Creation and Database Layer

For each target site (e.g., university campus, office, museum, historic street) we acquire dense image sequences using regular or 360° cameras along trajectories that cover the navigable area.

The offline pipeline builds a metrically consistent 3D map using a Structure-from-Motion (SfM) pipeline with Ground Control Points (GCPs). This produces a sparse point cloud, camera intrinsics and 6-DoF camera poses in a common world coordinate frame.

To make the map suitable for resource-constrained mobile devices, we apply a graph-based optimization over camera poses: each image is treated as a node; edges connect views with sufficient field-of-view overlap; a dominating-set–style reduction selects a minimal subset of keyframes that still covers the entire scene, yielding an 80–90% reduction in the number of reference images without degrading coverage.

For each selected keyframe we pre-compute:

- local features and descriptors using a neural local feature extractor, yielding 2D keypoints and local descriptors;
- a compact global image descriptor using a lightweight CNN backbone followed by a learnable aggregation module;
- 2D–3D correspondences linking image keypoints to 3D points in the reconstructed map;
- 6-DoF camera poses aligned with the NavMesh coordinate system used by the navigation engine in Unity.

All reference data are packed into a hybrid database. Lightweight metadata (image IDs, poses, zone labels) are stored in an on-device SQLite table, whereas high-dimensional arrays (keypoints, descriptors) are serialized, compressed (e.g., GZip), and stored in binary files referenced from the database. The global map is partitioned into overlapping zones, each represented by a compact subset of point clouds, keyframes, and descriptors. These zones are linked to coarse identifiers (Wi-Fi SSID/BSSID, BLE beacons, QR codes, or other markers) and to the corresponding NavMesh regions.

4.2.2 On-Device Two-Stage Localization Module

The mobile client is implemented using ONNX models and runs on Android and iOS, Figure 5.

To run controlled on-device experiments, we use a dedicated test build of our mobile client (Fig. 5). The application exposes diagnostic controls to (i) load/unload the deployed inference models (global descriptor, local features, and matcher), (ii) clear or update the on-device per-zone database shards, and (iii) switch between alternative pose-estimation backends (Homography, PnP, ORB) under identical input data. For each captured query frame, the client outputs the estimated 6-DoF pose expressed in the NavMesh-aligned world frame and records detailed logs, including runtime, the number of detected features, the number of inlier correspondences after geometric verification, and solver status. These logs are exported and processed offline to compute accuracy and performance metrics reported in the experimental evaluation.

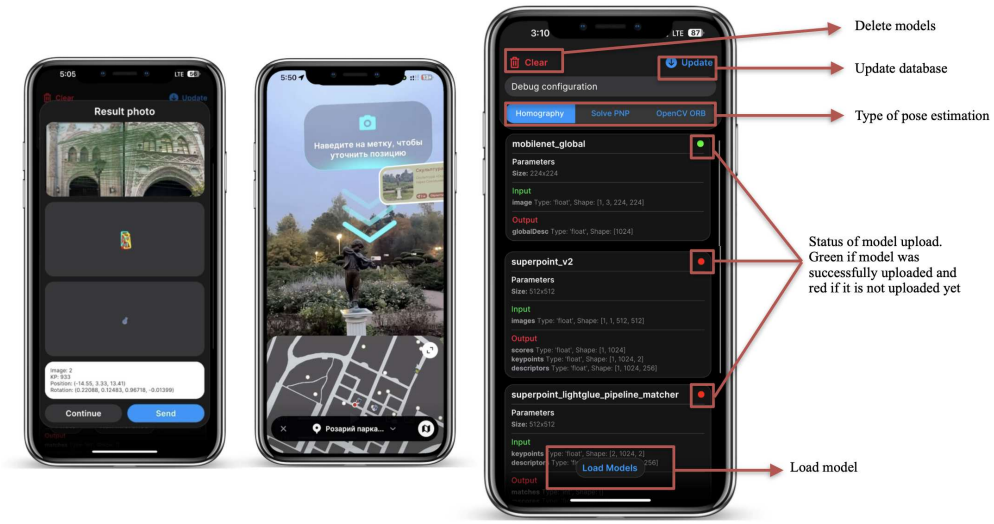


Fig. 5 Experimental mobile client used for on-device re-localization and logging. **Left:** per-query localization output (query frame and estimated pose in the NavMesh frame). **Center:** AR navigation view consuming the estimated pose for stable overlay placement. **Right:** debug panel used to manage local models and databases (clear/update) and to switch the pose-estimation backend (Homography / PnP / ORB), with per-model load status indicators.

The two-stage localization logic is integrated as a modular pipeline:

Stage 1: Coarse Zonal Initialization

The device first detects a zonal identifier from the environment (Wi-Fi/BLE scan, QR code, coarse GPS, or a 2D marker). This signal is used *only* as a non-metric key to select the current zone and load its associated database shard from local storage. No pose is computed at this stage; instead, the search space is pruned from the full map to a small, zone-specific subset of reference images and 3D points.

Stage 2: Fine Visual Localization

Given the active zone, the app receives a live frame from the camera and:

1. computes a global descriptor using an ONNX-exported lightweight CNN + aggregation network;
2. retrieves the top-K nearest neighbor keyframes in descriptor space from the zone-local database;
3. extracts local keypoints and descriptors on the query frame using the same ONNX-based local feature model as in offline processing;
4. matches local features to the selected keyframes using a neural matcher, obtaining 2D–3D correspondences via the pre-associated 3D points of the keyframes;
5. estimates the 6-DoF camera pose with a PnP solver or similar approaches;

6. projects the resulting pose into the NavMesh coordinate system for downstream AR navigation and route planning.

To support large environments while remaining fully offline, the mobile module maintains a small cache of recently used zones and uses an LRU-based eviction policy. Depending on user motion and zone adjacency metadata, neighboring zones can be preloaded to hide loading latency at zone transitions. All heavy computation (feature extraction, matching, PnP) is performed on-device using ONNX Runtime or platform-specific accelerators (e.g., Metal/NNAPI) without sending images to the cloud.

4.2.3 Administration Panel, Backend Services and Test Application

A web-based administration panel allows non-technical staff to manage locations, maps, and AR content without modifying the client application. Through this panel, administrators can:

- upload processed datasets (keyframes, descriptors, poses) as archives for each location;
- edit NavMesh layers, portals (elevators, stairs), and inter-zone connectivity;
- manage AR content (text, images, 3D models, avatars, audio guides) and link it to 3D positions or NavMesh nodes;
- configure zones and associate them with concrete zonal identifiers (Wi-Fi, BLE, QR);
- manage user roles and access levels for different parts of the system.

Backend provides APIs for uploading and validating new datasets, logging experimental runs, and aggregating statistics on localization performance (pose error distributions, recall, precision, latency) across devices and environments.

For the experiments we use a dedicated test configuration of the mobile app that exposes additional diagnostic functionality while reusing the production localization engine. The experimental app allows the operator to select locations and database variants, switch between model combinations, choose the pose estimation method (homography vs. PnP), visualize matches, and log per-frame metrics (errors, inlier counts, runtime).

4.3 Ground-Truth Pose Construction

Accurate ground-truth (GT) poses are essential to evaluate localization error on real devices. We combine physical reference points in the environment, a metrically calibrated NavMesh, and SfM-based reconstruction to obtain consistent 6-DoF GT poses in the same coordinate system used by the AR navigation engine.

4.3.1 Physical Reference Points and NavMesh Annotations

In each site we manually select a set of easily identifiable reference points (corners of walls, door edges, intersections of floor tiles, or other stable landmarks) that are visible to both the human operator and the camera. These points are physically marked to ensure repeatable traversal during data capture.

The same reference points are annotated on the NavMesh used by our navigation system. The NavMesh is derived from architectural plans or precise indoor mapping and defines the world coordinate system of the application. For each reference point p_j we record its 3D position

$$\mathbf{X}_j^{\text{nav}} = (x_j, y_j, z_j)$$

and, when meaningful (e.g., at corridor turns), a canonical reference orientation. All such annotations are stored on the backend alongside the NavMesh.

4.3.2 SfM Reconstruction Anchored at Reference Points

We then capture multi-view image sequences while standing on and moving between the marked reference points. These images are fed into the SfM pipeline to reconstruct a 3D point cloud and estimate camera poses in a reconstruction coordinate frame.

Within this reconstruction, images taken from each physical reference point cluster around a single camera pose. We associate these reconstructed camera centers $\mathbf{C}_j^{\text{sfm}}$ with the corresponding NavMesh-based positions $\mathbf{X}_j^{\text{nav}}$ and estimate a 7-DoF similarity transform (rotation, translation, uniform scale) that aligns the SfM frame to the NavMesh frame via Procrustes analysis.

Applying this similarity transform to all camera poses yields metrically correct 6-DoF poses $\mathbf{T}_i^{\text{gt}}$ for every keyframe and every query frame in the NavMesh coordinate system. The aligned cameras and 3D points are as well visually validated that reconstructed cameras coincide with the NavMesh geometry, with residual offsets at reference points below a few centimeters.

Thus, for each query frame I_q we obtain ground-truth pose

$$\mathbf{T}_q^{\text{gt}} = [\mathbf{R}_q^{\text{gt}}, \mathbf{t}_q^{\text{gt}}],$$

expressed in the same coordinate system that the AR navigation system uses for routing and content placement.

5 Experimental Evaluation

5.1 Evaluation Metrics

We evaluate both geometric accuracy and system-level performance. For each query frame we compare the estimated 6-DoF pose $(\mathbf{t}_{\text{est}}, \mathbf{R}_{\text{est}})$ with the ground-truth pose $(\mathbf{t}_{\text{gt}}, \mathbf{R}_{\text{gt}})$.

5.1.1 Translational Error

The position error is defined as the Euclidean distance between the estimated camera center and the ground truth:

$$e_t = \|\mathbf{t}_{\text{est}} - \mathbf{t}_{\text{gt}}\|_2 \,. \tag{3}$$

5.1.2 Rotational Error

Orientations are compared in quaternion space. Let q_{est} and q_{gt} be unit quaternions corresponding to $\mathbf{R}_{\text{est}}$ and $\mathbf{R}_{\text{gt}}$, respectively. The angular error is

$$e_r = 2 \arccos(|q_{\text{est}} \cdot q_{\text{gt}}|) , \tag{4}$$

measured in radians and converted to degrees as

$$e_r^\circ = \frac{180}{\pi} e_r. \tag{5}$$

5.1.3 Success Criterion, Precision and Recall

We consider a localization successful if both translation and rotation errors are below application-driven thresholds:

$$e_t \leq \tau_t, \quad e_r^\circ \leq \tau_r,$$

with $\tau_t = 0.3$ m and $\tau_r = 5^\circ$ in our experiments, reflecting tolerances for stable AR navigation overlays.

For a given configuration and scene, let:

- N be the number of query frames,
- N_{corr} the number of frames for which a pose is estimated and passes the success criteria (true positives),
- N_{fail} the number of frames for which a pose is estimated but fails the criteria (false positives),
- N_{no} the number of frames for which the solver fails to return a pose (false negatives).

We define:

$$\text{Precision} = \frac{N_{\text{corr}}}{N_{\text{corr}} + N_{\text{fail}}}, \tag{6}$$

$$\text{Recall} = \frac{N_{\text{corr}}}{N_{\text{corr}} + N_{\text{fail}} + N_{\text{no}}} = \frac{N_{\text{corr}}}{N}. \tag{7}$$

Precision measures the fraction of returned poses that are within the tolerances, while recall measures the fraction of all queries that are correctly localized.

5.1.4 Inlier Correspondences

For each successful pose we also record the number of inlier correspondences returned by PnP+RANSAC. Although not used as a primary metric, this value helps interpret failure cases (e.g., scenes with low texture or motion blur).

5.1.5 Runtime

We measure wall-clock runtime on the device for each query frame and decompose it into:

- feature extraction (global and local),
- descriptor matching (global retrieval + local matching),
- pose estimation (PnP+RANSAC),
- total per-frame latency.

For each configuration we report the total time to process a batch of 20 frames (short burst, simulating a quick localization) and 100 frames (longer interaction sequence). These correspond to the timing setup used in our internal test application.

5.2 Experiment

For each site and each database size $N \in \{1500, 800, 300, 100, 25\}$ we evaluate the localization pipeline as follows:

1. **Database preparation.** We construct the corresponding keyframe subset and zone definitions as described in Section 4.2, ensuring that all query trajectories lie within the coverage of the selected keyframes.
2. **Query selection.** From the held-out query trajectories we sample two disjoint sets of frames: (i) 20 frames uniformly along the path, used for fine-grained timing analysis; (ii) 100 frames used for aggregated accuracy and robustness statistics reported in Tables 2 and 3.
3. **On-device evaluation.** Each query frame is processed by the mobile application running on a mid-range Android smartphone (8-core CPU, 4–6 GB RAM, OpenGL ES 3.0) and Iphone 12 pro. All models are executed via ONNX Runtime with no network connectivity.
4. **Metric computation.** For each query we log translational error e_t, rotational error e_r°, whether the pose satisfies the success criterion, the number of inlier correspondences, and per-frame runtime. From these logs we compute median errors, precision, recall, and mean inlier counts.
5. **Comparison across database sizes.** For each site we obtain precision, recall, median e_t and e_r°, average inlier counts, and runtime for each database size. Table 2 summarizes these metrics per scene, while Table 3 reports averages across all five sites.

All experiments use exactly the same visual localization pipeline; only the size and organization of the reference database (zonal vs. non-zonal) vary.

6 Experimental Evaluation

In this section we evaluate the proposed two-stage, zone-aware localization pipeline under realistic conditions on a set of heterogeneous large-scale environments. The goal is to quantify how aggressively we can reduce the size of the visual database per zone while keeping the localization error within a practically acceptable range and preserving real-time performance on a consumer mobile device.

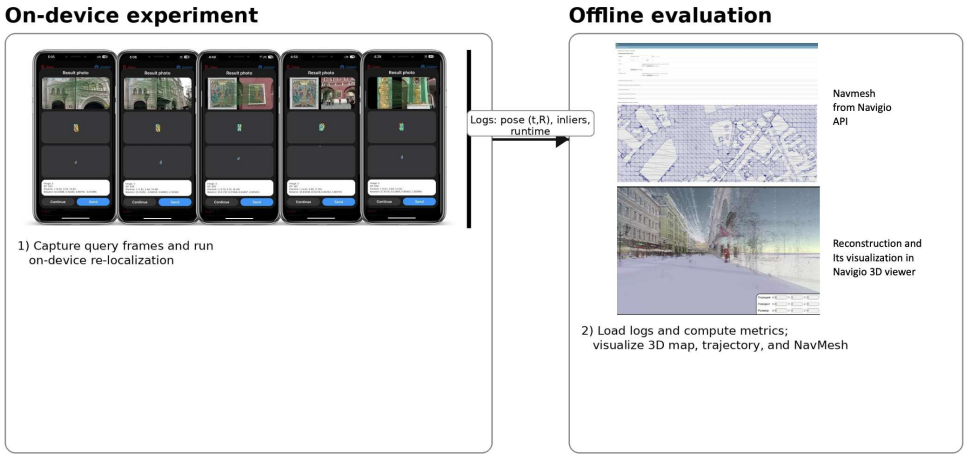


Fig. 6 Visualisation of work inside of the application for Skoltech campus and office

6.1 Environments and Datasets

We consider five representative environments with different geometry, texture and lighting conditions:

- **Skoltech campus** (modern university building with long corridors, glass surfaces, repetitive structure, Figure 7).
- **Office environment** (indoor office or exhibition space with corridors and controlled artificial lighting).
- **Museum 2** (open gallery-style museum with larger halls and mixed lighting).
- **Historic city street** (outdoor pedestrian street with dense façades, signage and heavy occlusions, Figure 6).
- **City park with church and historic buildings** (mixed open-space park and high-texture heritage objects with strong illumination changes).

Figure 6 illustrates the evaluation workflow used in our experiments. First, the mobile client runs the on-device re-localization pipeline on a sequence of captured query frames and logs, for each frame, the estimated 6-DoF pose, the inlier count from geometric verification, and the end-to-end runtime. Second, these logs are imported offline and analyzed together with the place-specific assets, including the NavMesh obtained from the Navigio API and the reconstructed 3D model visualized in our 3D viewer. This offline stage computes accuracy and performance metrics and provides qualitative visualizations of the estimated trajectory with respect to the 3D map and the NavMesh.

For each environment we first build a metrically consistent 3D map and camera trajectory using the offline pipeline described in Section 4.2. The resulting reconstruction is then graph-optimized and pruned so that every environment is covered by a dense set of reference images with known 6-DoF poses and associated local and global descriptors.

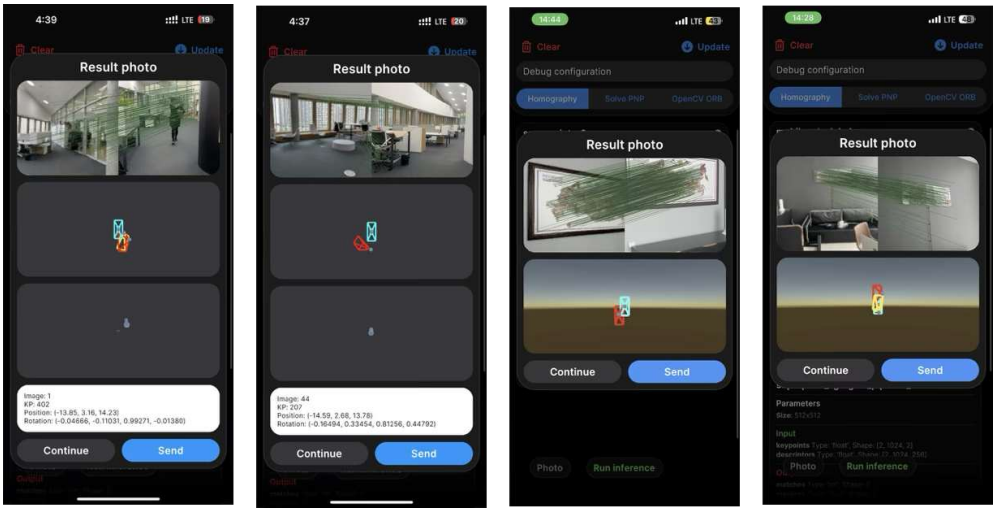


Fig. 7 Illustration of application work in Skoltech and office.

To study the influence of database size on speed and accuracy, we derive five configurations by progressively sub-sampling the reference images in each environment while preserving geometric coverage. The resulting per-environment database sizes are:

$$N \in \{1500,\ 800,\ 300,\ 100,\ 25\} \text{ reference images.}$$

The configuration with $N = 1500$ corresponds to a near-dense database (minimal pruning), while $N = 25$ represents an extremely compact database for highly constrained memory budgets. All other configurations interpolate between these extremes and model different trade-offs between accuracy and resource usage.

For each environment we also collect a separate set of query trajectories that are not used for mapping. Along these paths we record video and later sample 20 and 100 query frames per configuration according to the protocol in Section 5.2.

6.2 Per-Scene and Aggregated Results

Table 2 summarizes the per-scene localization performance for each environment and each database size $N_{\text{ref}} \in \{1500, 800, 300, 100, 25\}$. For every configuration we report:

- median translation error e_t in meters;
- precision P and recall R as defined in Section 5.1;
- qualitative trends in runtime (per-scene timing is reflected in the aggregated statistics).

Table 3 aggregates these results across all five environments. Each row corresponds to a database size N_{ref} and reports the mean median position error, mean precision and recall, and mean processing times for batches of 20 and 100 frames.

Table 2 Per-environment localization accuracy as a function of database size N_{ref}.

Scene	N_{ref}	Med. pos. err [m] ↓	Precision P ↑	Recall R ↑
Skoltech campus	1500	0.25	0.89	0.81
Skoltech campus	800	0.23	0.91	0.84
Skoltech campus	300	0.22	0.94	0.88
Skoltech campus	100	0.24	0.92	0.83
Skoltech campus	25	0.26	0.87	0.77
Office environment	1500	0.29	0.87	0.79
Office environment	800	0.27	0.89	0.82
Office environment	300	0.26	0.92	0.86
Office environment	100	0.28	0.90	0.81
Office environment	25	0.30	0.85	0.75
Museum 2	1500	0.32	0.86	0.78
Museum 2	800	0.30	0.88	0.81
Museum 2	300	0.29	0.91	0.85
Museum 2	100	0.31	0.89	0.80
Museum 2	25	0.33	0.84	0.74
Historic street	1500	0.28	0.88	0.80
Historic street	800	0.26	0.90	0.83
Historic street	300	0.25	0.93	0.87
Historic street	100	0.27	0.91	0.82
Historic street	25	0.29	0.86	0.76
City park & church	1500	0.30	0.87	0.79
City park & church	800	0.28	0.89	0.82
City park & church	300	0.27	0.92	0.86
City park & church	100	0.29	0.90	0.81
City park & church	25	0.31	0.85	0.75

Table 3 Average localization performance over all environments for different database sizes.

N_{ref}	Med. err [m] ↓	P ↑	R ↑	t_{20} [s] ↓	t_{100} [s] ↓
1500	0.29	0.87	0.79	2.6	4.4
800	0.27	0.89	0.82	2.2	3.8
300	0.26	0.92	0.86	1.8	3.0
100	0.28	0.90	0.81	1.6	2.4
25	0.30	0.85	0.75	1.4	2.0

Table 2 shows that for all environments the median position error remains around 0.25–0.33 m across database sizes, i.e., within approximately 10% of the characteristic spatial scale of the scenes, which is sufficient for AR navigation overlays and route guidance. Precision and recall slightly improve when reducing the database from 1500 to 300 images, as redundant or ambiguous keyframes are removed, and then gradually degrade as the database becomes extremely sparse ($N_{\text{ref}} = 25$).

As summarized in Table 3, reducing the database from 1500 to 300 images per environment yields a substantial speedup (from $t_{20} = 2.6$ s and $t_{100} = 4.4$ s down to

$t_{20} = 1.8\,\mathrm{s}$ and $t_{100} = 3.0\,\mathrm{s}$) while slightly *improving* average precision and recall due to the removal of redundant or confusing keyframes. Extremely compact databases ($N_{\mathrm{ref}} = 25$) further reduce runtime but lead to a noticeable drop in recall, as the coverage of challenging viewpoints and lighting conditions becomes insufficient.

Overall, these results indicate that the proposed zonal two-stage design allows us to aggressively prune the per-zone database—and thus reduce memory footprint and computation—while staying within the accuracy budget required by our AR navigation use cases. In additional experiments (not tabulated here), the non-zonal baseline that matches queries against the entire site-level database is consistently slower, especially for large N_{ref}, confirming that zonal pruning is crucial for maintaining real-time performance on resource-constrained mobile devices.

6.3 Discussion

Overall, the experiments confirm that the proposed zonal, two-stage design allows us to aggressively prune the per-zone database—and thus reduce memory footprint and computation—while staying within the accuracy budget required by our AR navigation use cases. The fact that optimal performance is achieved at intermediate database sizes supports our main design choice: combining a compact, well-optimized reference set with zonal pruning is more effective than either using all available images or shrinking the database too aggressively.

In particular, the configuration with around 300 reference images per zone delivers sub–0.3 m median position error and high precision/recall on all evaluated scenes, with total processing times compatible with real-time operation on mid-range smartphones. This demonstrates that a carefully engineered two-stage, on-device visual localization pipeline can provide cloud-comparable accuracy while remaining fully offline-capable.

7 Conclusion

We presented a two-stage, zone-aware visual re-localization pipeline designed for resource-constrained mobile devices and AR/VR/XR deployment constraints. The method uses discrete ambient identifiers (e.g., Wi-Fi/BLE/QR) only to select an active zone and load a compact per-zone database shard, and then performs hierarchical visual localization within this shard to estimate a NavMesh-aligned 6-DoF pose via global retrieval, local matching, and PnP+RANSAC. By making the search space scale with the zone rather than the full map, the proposed design improves on-device practicality for large places and supports fully offline operation.

From an application perspective, the pipeline targets XR scenarios that require stable and low-latency pose estimates: offline AR navigation in large buildings or campuses, robust placement of persistent spatial content, and mixed-reality guidance on smartphones and AR glasses. The same representation (place $\rightarrow$ zones $\rightarrow$ shards) also provides a practical interface between mapping/back-end tooling (NavMesh, zone and sensor annotation) and on-device localization, which is essential for scalable XR deployments.

Future work will consider tighter integration with short-term tracking (e.g., visual-inertial odometry between re-localization events), improved zone-transition handling

and caching policies, and learning-based strategies to infer zones directly from visual cues when ambient identifiers are missing or ambiguous.

References

[1] Porter, S.D., MacCormick, J.S.: Location Based Services in a Multi-User Mixed Reality System. US Patent Application Publication US20210350612A1. Assignee: Microsoft Technology Licensing, LLC (2021). https://patents.google.com/patent/US20210350612A1/en

[2] Glenn, J.P., Duerst, M., Durelli, J.Y.: Cross Reality System with Localization Service and Shared Location-Based Content. US Patent US11386627B2. Assignee: Magic Leap, Inc. (2022). https://patents.google.com/patent/US11386627B2/en

[3] Mur-Artal, R., Montiel, J.M.M., Tardós, J.D.: Orb-slam: A versatile and accurate monocular slam system. IEEE Transactions on Robotics **31**(5), 1147–1163 (2015) https://doi.org/10.1109/TRO.2015.2463671

[4] Arandjelović, R., Gronat, P., Torii, A., Pajdla, T., Sivic, J.: Netvlad: Cnn architecture for weakly supervised place recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5297–5307 (2016)

[5] Sarlin, P.-E., Cadena, C., Siegwart, R., Dymczyk, M.: From coarse to fine: Robust hierarchical localization at large scale. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 12716–12725 (2019)

[6] LaMarca, A., *et al.*: Place lab: Device positioning using radio beacons in the wild. In: Proceedings of the Third International Conference on Pervasive Computing (Pervasive), pp. 116–133 (2005)

[7] Guo, S., Gu, Y., Liu, B., Wu, W., Yang, Y.: Smartphone-based indoor localization using wi-fi and vision data fusion. Sensors **18**(11), 3604 (2018) https://doi.org/10.3390/s18113604

[8] Toll, W., Cook, A.F., Geraerts, R.: A navigation mesh for dynamic environments. Computer Animation and Virtual Worlds **23**(6), 535–546 (2012)

[9] Mulloni, A., Seichter, H., Schmalstieg, D.: Handheld augmented reality indoor navigation with activity-based instructions. In: Proceedings of the 13th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI), pp. 211–220 (2011)

[10] Pajouh, M., Beitollahi, A., Amirghodsi, S.: Systems and Methods for Augmented Reality Localization. US Patent Application Publication US20230234567A1. Published Aug. 2023 (2023)

[11] Thrun, S.: Learning metric-topological maps for indoor mobile robot navigation. Artificial Intelligence **99**(1), 21–71 (1998)

[12] Szeliski, R.: Computer Vision: Algorithms and Applications. Springer, Nature (2010)

[13] Luhmann, T., Robson, S., Kyle, S., Harley, I.: Close-Range Photogrammetry and 3D Imaging, 2nd edn. De Gruyter, KG (2014)

[14] Umeyama, S.: Least-squares estimation of transformation parameters between two point patterns. IEEE Transactions on pattern analysis and machine intelligence **13**(4), 376–380 (1991)

[15] Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press, Cambridge, UK (2003)

[16] Lepetit, V., Moreno-Noguer, F., Fua, P.: EPnP: An accurate O(n) solution to the PnP problem. In: Proc. IEEE International Conference on Computer Vision (ICCV) (2009)

[17] Fischler, M.A., Bolles, R.C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM **24**(6), 381–395 (1981)

[18] Triggs, B., McLauchlan, P.F., Hartley, R.I., Fitzgibbon, A.W.: Bundle adjustment—a modern synthesis. In: International Workshop on Vision Algorithms, pp. 298–372 (1999)

[19] DeTone, D., Malisiewicz, T., Rabinovich, A.: Superpoint: Self-supervised interest point detection and description. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 224–236 (2018). https://arxiv.org/abs/1712.07629

[20] Sarlin, P.-E., Cadena, C., Siegwart, R., Dymczyk, M.: From coarse to fine: Robust hierarchical localization at large scale. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 12716–12725 (2019)

[21] Lindenberger, P., Sarlin, P.-E., Pollefeys, M.: Lightglue: Local feature matching at light speed. arXiv preprint arXiv:2306.13643 (2023)

[22] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2018). https://arxiv.org/abs/1801.04381

[23] Howard, A., Sandler, M., Chu, G., Chen, L., Chen, B., Tan, M., Wang, W., Zhu,

Y., Pang, R., Vasudevan, V., Le, Q.V., Adam, H.: Searching for MobileNetV3. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (2019). https://arxiv.org/abs/1905.02244

[24] Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., Xu, C.: Ghostnet: More features from cheap operations. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2020). https://arxiv.org/abs/1911.11907

[25] Ali-Bey, A., Chaib-Draa, B., Giguère, P.: Mixvpr: Feature mixing for visual place recognition. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) (2023)

[26] Berton, G., Masone, C., Caputo, B.: Rethinking visual geo-localization for large-scale applications. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2022). https://arxiv.org/abs/2204.02287

[27] Keetha, N., Cheema, S., Kumari, U., Olsson, C., Trulls, E., Karlsson, P.-E., Tardos, J.D., Hofer, M.: Anyloc: Towards universal visual place recognition. arXiv preprint arXiv:2308.00688 (2023)

[28] Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision **60**(2), 91–110 (2004)

[29] Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: ORB: An efficient alternative to SIFT or SURF. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV) (2011)

[30] Tyszkiewicz, M., Fua, P., Trulls, E.: DISK: Learning local features with policy gradient. In: Advances in Neural Information Processing Systems (NeurIPS) (2020). https://arxiv.org/abs/2006.13566

[31] Zhao, X., Wu, X., Chen, W., Chen, P., Xu, Q., Li, Z.: ALIKED: A lighter keypoint and descriptor extraction network via deformable transformation. arXiv preprint arXiv:2304.03608 (2023)

[32] Muja, M., Lowe, D.G.: FLANN: Fast library for approximate nearest neighbors. In: Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP) (2009)

[33] Bradski, G.: The OpenCV library. Dr. Dobb's Journal of Software Tools (2000)

[34] Lindenberger, P., Sarlin, P.-E., Pollefeys, M.: Lightglue: Local feature matching at light speed. arXiv preprint arXiv:2306.13643 (2023)

[35] Sarlin, P., DeTone, D., Malisiewicz, T., Rabinovich, A.: Superglue: Learning feature matching with graph neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2020). https://arxiv.org/abs/1911.11763

[36] Radenović, F., Tolias, G., Chum, O.: Fine-tuning CNN image retrieval with no human annotation. IEEE Transactions on Pattern Analysis and Machine Intelligence (2019)

[37] Navigio Team: Navigio: Mobile AR Navigation Application. Software. Accessed: 2026-01-20 (2026). https://navig.io