electronics

MDPI

Article

Hardware-Aware Acceleration of Open-Vocabulary Multi-Object Navigation on Edge GPUs

Michael Chibudom Akor and Heoncheol Lee *

Department of IT Convergence Engineering, Kumoh National Institute of Technology, Gumi 39177, Gyeongbuk, Republic of Korea; akormichael@kumoh.ac.kr
* Correspondence: hclee@kumoh.ac.kr

Abstract

Persistent open-vocabulary navigation enables robots to search for sequential language-specified objects using reusable visual semantic evidence. On edge GPUs, the pipeline is constrained by foundation-model inference, semantic projection, persistent map movement, frontier processing, and repeated target detection. Using OneMap as a controlled reference, we reorganize this workload through compiled mixed-precision perception, direct-to-map confidence-weighted feature aggregation, accelerator-resident semantic maps, compiled navigation operations, and language-conditioned detector scheduling. The resulting dataflow removes the dense image resolution feature intermediate and schedules target confirmation detection from current-view target similarity. On the complete Habitat-Matterport 3D benchmarks, our implementation retains 92.6% and 96.1% of the reproduced OneMap single- and multi-object success rates. Across 36 paired hardware-in-the-loop comparisons on a Jetson AGX Orin at four power modes, it achieves geometric mean speedups of 4.64× for on-device computation and 3.55× for the complete episode duration. It also reduces peak system random access memory (RAM) by 51.21% and a summed board-rail energy proxy by 75.32% on average while preserving every paired outcome. These results establish the coordinated dataflow and current-view target-conditioned execution as effective mechanisms for efficient persistent foundation-model navigation on power-constrained edge robots.

Keywords: edge artificial intelligence; embedded systems; GPU computing; hardware acceleration; computer vision; mobile robotics; semantic navigation; deep learning inference

1. Introduction

Foundation models have enabled mobile robots to search unfamiliar environments for objects specified through unrestricted language. Instead of learning a navigation policy for a fixed set of semantic categories, recent systems combine pretrained vision-language representations, open-vocabulary object detectors, and geometric mapping to infer where a requested object is likely to occur [1]. Vision–Language Frontier Maps (VLFM), for example, grounds vision–language similarity in a spatial value map to guide frontier exploration without task-specific semantic navigation training [2]. OneMap extends this direction to persistent, dense open-vocabulary mapping and sequential multi-object navigation [3]. These methods demonstrate that foundation-model perception can provide useful semantic priors for embodied agents without requiring a new model to be trained for every target category or environment.

Check for updates

Academic Editor: Tee Hui Teo

Received: 30 July 2026
Revised: 7 September 2026
Accepted: 10 September 2026
Published: 11 September 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

The computational organization of these systems, however, remains poorly matched to the edge hardware carried by mobile robots. Robotics workloads combine heterogeneous perception, mapping, planning, and control stages whose end-to-end behavior is not represented by neural-model throughput alone [4]. Their implementations repeatedly execute dense visual encoders and object detectors, project high-dimensional image features into large spatial maps, update navigation structures, and transfer intermediate data among GPU kernels, CPU routines, and general-purpose framework objects. Hardware studies show that data movement and operational intensity can limit both performance and energy efficiency even when the arithmetic throughput is high [5,6].

Repeated execution makes this hardware cost particularly consequential in multi-object navigation. The same perception and mapping pipeline runs for hundreds or thousands of control steps. Even a small per-step delay can therefore add substantial task time and energy consumption. Completion-time-aware navigation metrics similarly recognize that the path length alone does not capture the execution behavior of a physical robot [7]. Restricted embedded power modes intensify the mismatch by constraining the memory bandwidth, GPU resources, and CPU throughput simultaneously.

Existing acceleration techniques address only parts of this execution path. Neural network compilers optimize graphs and operators across hardware backends [8]. Mixed-precision and hardware-aware adaptation instead trade numerical precision or model structure against latency, memory, and energy [9–11]. Applying these techniques independently is insufficient because open-vocabulary navigation is a heterogeneous closed-loop workload. Its runtime spans neural inference, feature projection, map fusion, frontier extraction, target detection, planning, and action selection. Changing one stage can also change the observations or decisions received by later stages.

Reduced feature precision and conditional visual processing can improve the measured hardware efficiency while changing model outputs or the execution frequency [10,11]. In a closed-loop navigation system, such local changes can alter the resulting trajectory. Hardware optimization must therefore address the complete dataflow and evaluate both execution efficiency and closed-loop task quality. This makes persistent open-vocabulary navigation a cross-layer dataflow problem spanning perception, mapping, and planning.

The central question is therefore how to reorganize a persistent open-vocabulary navigation pipeline for edge GPU execution while retaining its semantic representation and navigation formulation. We characterize this problem by profiling OneMap as a representative multi-object pipeline. The profile shows that the principal costs extend beyond the foundation model. Dense vision inference dominates the runtime, but semantic map projection also creates a large per-pixel feature tensor. Frontier processing adds CPU overhead, and unconditional detection consumes resources even when target evidence is weak. These costs motivate the coordinated optimization of perception, map construction, data movement, and conditional execution.

We present a hardware-aware accelerated implementation of OneMap that retains its open-vocabulary representation, mapping formulation, and navigation logic while restructuring execution for edge GPUs. The method reuses each language-aligned image feature grid through two parallel paths: fused direct-to-map projection updates the persistent semantic map without a dense intermediate at image resolution, while current-view target similarity regulates future target confirmation detection. Compiled mixed-precision perception, a persistent accelerator-resident state, and compiled navigation operations support this shared-feature execution model. Together, these changes reduce the inference, memory transfer, and map processing overhead across the existing navigation pipeline.

On the complete HM3D benchmarks, the implementation retains more than 92% of the reproduced OneMap success rates while achieving a 4.64× geometric mean compute speedup across four Jetson AGX Orin power modes.

The proposed implementation treats open-vocabulary navigation as an integrated hardware workload rather than a collection of independent models and routines. Its main contributions are as follows:

- We characterize persistent open-vocabulary navigation as a heterogeneous edge workload. The profile identifies semantic projection, frontier processing, memory movement, and unconditional detection as substantial costs beyond foundation-model inference.
- We develop a fused direct-to-map projection that preserves OneMap's confidence-weighted mapping equations while eliminating the dense intermediate at depth resolution. On the desktop, it accelerates isolated fixed-input projection by 2.34× and the projection stage in the closed-loop profile by 2.53×, while lowering the temporary memory demand.
- We design a hardware-aware runtime that combines stage-specific mixed precision, concurrent compiled perception, persistent accelerator buffers, compiled point-of-interest clustering and planning mask operations, and language-conditioned detector scheduling.
- We evaluate the resulting quality–efficiency trade-off on the complete single- and multi-object HM3D benchmarks and through closed-loop Jetson AGX Orin experiments across four power modes.

2. Related Work

This work lies at the intersection of open-vocabulary semantic navigation and hardware-aware edge inference. The first research stream establishes the representations and closed-loop behavior required for language-directed navigation, including persistent semantic memory and sequential object searching. The second examines how neural inference, spatial transformation, data movement, and runtime scheduling can be reorganized for constrained hardware. Together, they define the central systems question of this study: how to accelerate a persistent open-vocabulary navigation workload without separating hardware efficiency from navigation quality.

2.1. Open-Vocabulary Navigation and Persistent Semantic Mapping

Open-vocabulary object navigation has developed from single-goal semantic searching toward systems that retain and reuse spatial knowledge over longer tasks. CoW combines CLIP-based localization with classical exploration to provide a direct training-free baseline for language-driven object searching [1]. VLFM assigns target-conditioned semantic values to observed views and projects them onto a value map for frontier selection [2]. These methods establish a modular pattern in which pretrained visual–language evidence guides geometric exploration without navigation-specific semantic training.

A related line of work represents this evidence explicitly in spatial memory. VLMaps projects language-aligned visual features into a geometric map that supports free-form spatial queries [12]. OpenScene, ConceptFusion, and Open-Fusion similarly preserve open-vocabulary features in persistent 3D representations, while ConceptGraphs organizes observations as an object-centric scene graph [13–16]. Although their map structures differ, these systems share a common computational sequence: visual features are encoded, associated with geometry, accumulated in a spatial representation, and compared with a language query.

Recent work extends this foundation to sequential and more expressive navigation. GOAT-Bench evaluates agents across sequences of category, language, and image goals and shows that retaining scene memory improves lifelong navigation efficiency [17]. OneMap constructs a dense, confidence-weighted feature map that persists across successive object queries, enabling semantic evidence gathered for one target to support later searches [3]. DIV-Nav builds on this persistent map family by decomposing spatially constrained instructions, intersecting the resulting similarity maps, and validating candidate regions during online exploration [18]. VL-Nav addresses abstract multi-target instructions through neural reasoning, symbolic memory, and a lightweight exploration system [19]. Together, these studies show the field progressing from isolated object searching toward reusable semantic states, sequential goals, relational queries, and longer-horizon reasoning.

Despite differences in task formulation and map representation, modular open-vocabulary navigation systems share a common execution pattern. They encode visual features, project them geometrically, update semantic states, retrieve language-conditioned evidence, confirm targets, process frontiers, and compute paths. Their evaluations primarily establish semantic capabilities, navigation quality, or memory reuse. OneMap provides the controlled implementation studied here because it exposes all of these stages within a persistent multi-object pipeline. The resulting hardware principles concern structures shared by a broader class of map-based systems rather than a target category or query type specific to OneMap.

2.2. Hardware Acceleration of Open-Vocabulary Navigation

Open-vocabulary navigation is a modular workload whose principal costs have each been studied in adjacent hardware research. TVM compiles neural graphs and operators for heterogeneous processors, NetAdapt adapts the network structure using the measured platform latency, and HAQ selects layerwise precision from hardware feedback [8,10,11]. Spatial transform methods address the movement of image features into geometric representations: BEVPoolv2 avoids a large frustum intermediate, whereas MatrixVT factorizes camera-to-BEV transformation into standard matrix operations [20,21]. Runtime systems reduce unnecessary inference by selecting detector configurations or task schedules from image content, resource contention, latency requirements, or environmental workload changes [22–24]. These studies establish effective techniques for neural inference, spatial transformation, and adaptive execution but optimize these components independently of the language-conditioned navigation state.

Recent embodied systems bring hardware efficiency closer to the complete task. OneMap demonstrates persistent multi-object navigation as a complete physical robot system on a Jetson AGX Orin [3]. VL-Nav uses a lightweight navigation stack on a Jetson Orin NX while executing its high-level vision–language planner on a remote RTX 4090 [19]. These studies demonstrate embedded deployment but do not report acceleration against a defined implementation baseline; they therefore provide a deployment context rather than entries in Table 1. EfficientNav reduces the on-device language-model planning cost through semantics-aware memory retrieval and reusable key–value caches [25]. FindAnything aggregates visual–language features into object-centric volumetric submaps, reducing semantic map processing and storage while supporting deployment on a Jetson Orin NX [26]. SemanticXR organizes mapping, communication, and device memory around objects in a device cloud architecture [27]. Table 1 compares the methods mentioned in this subsection whose primary contribution is hardware or execution efficiency.

Our method combines direct-to-map projection with current-view target-conditioned scheduling for persistent open-vocabulary navigation. The direct-to-map operator combines bilinear sampling, RGB-D transformation, map cell indexing, and confidence-

weighted aggregation. It writes low-resolution language-aligned features directly into the persistent map without materializing the dense $H \times W \times d$ intermediate. In parallel, the detector scheduler compares the same low-resolution image feature grid with the current target embedding. The similarity level and its temporal rise initiate future confirmation bursts, while a periodic safety interval bounds detector-free execution. The two mechanisms therefore reuse one language-aligned feature grid through parallel mapping and scheduling paths. Prior spatial transform methods do not produce confidence-weighted persistent open-vocabulary map updates. Prior adaptive inference systems do not schedule target confirmation from current-view target-language similarity. Compiled inference, mixed precision, persistent accelerator states, and compiled CPU routines support this execution model but are not separate method novelties.

Table 1. Reported hardware acceleration methods. Each value retains its baseline and measurement scope.

Name	Method	Platform	Reported Speedup
TVM [8]	Compiles neural graphs and operators for heterogeneous hardware backends.	NVIDIA Titan X.	Up to 3.8× over MXNet, TensorFlow, and TensorFlow XLA on Titan X.
NetAdapt [11]	Adapts network structure using latency measured directly on the target platform.	Google Pixel 1 CPU.	Up to 1.7× over unadapted MobileNet.
HAQ [10]	Selects layerwise mixed precision using accelerator latency and energy feedback.	BitFusion and BISMO accelerators.	1.4–1.95× latency over fixed 8-bit quantization.
BEVPoolv2 [20]	Removes the frustum feature intermediate from camera-to-BEV projection.	NVIDIA RTX 3090.	15.1× over the preceding fastest BEV-pooling implementation.
MatrixVT [21]	Factorizes camera-to-BEV transformation into compact matrix operations.	CUDA GPU; Intel Xeon CPU.	2–8× over prior view transformations.
ApproxDet [22]	Selects detector and tracker configurations from content, contention, and latency requirements.	Jetson TX2.	2.08× over YOLOv3, derived from the reported 52% latency reduction.
LiteReconfig [23]	Reduces the switching cost among content-aware mobile detection branches.	Jetson TX2.	74.9×, 30.5×, and 20.0× over SELSA, MEGA, and REPP, respectively.
RED [24]	Schedules robotic inference tasks under workload changes and real-time constraints.	Jetson Nano, TX2, AGX Xavier, and AGX Orin.	1.33× over earliest-deadline-first scheduling (24.7% lower mean response time).
EfficientNav [25]	Retrieves semantic map memory and reuses clustered key–value caches to reduce language model planning cost.	RTX A6000; Jetson AGX Orin deployment.	6.7× real-time and 4.7× end-to-end latency reduction over a GPT-4 planner.
FindAnything [26]	Aggregates visual–language features by object and stores them in volumetric submaps.	NVIDIA RTX 4500 Ada Generation; Jetson Orin NX deployment.	2.30× over OVO-SLAM when both process every tenth frame.
SemanticXR [27]	Uses objects as the units of mapping, communication, and device-side semantic memory.	RTX 6000 Ada server; Jetson Orin device proxy.	2.2× server mapping over its same-model device cloud baseline.
Ours	Projects features directly into persistent map cells and schedules target confirmation from current-view target similarity.	Jetson AGX Orin.	4.64× compute; 3.55× episode duration over OneMap.

OneMap provides the closest controlled reference among the reviewed systems. It combines dense vision–language encoding, persistent semantic mapping, target confir-

mation, frontier selection, and path computation in one sequential pipeline on the same class of Jetson hardware. Other edge-oriented systems accelerate language model planning, object-centric mapping, device cloud execution, or navigation stacks with remote reasoning. Their results provide an architectural context but cannot form matched end-to-end baselines without changing the task, execution boundary, or semantic representation. Pairing our implementation with OneMap holds these factors fixed and isolates the effects of the proposed dataflow and scheduler.

3. Materials and Methods

Our method retains OneMap's navigation formulation and reorganizes its computation for edge GPU execution. We first describe the reference observation-to-action pipeline and identify its principal runtime costs. We then present four changes in dataflow order: compiled mixed-precision perception, fused map projection, accelerator-resident mapping and navigation, and adaptive object detection. The integrated pipeline combines these changes into one navigation step, after which we analyze how its time and memory requirements scale. Experimental protocols are given separately in Section 4.

3.1. OneMap Reference Pipeline

We consider OneMap's zero-shot multi-object navigation formulation [3]. At time step t, the agent receives an RGB image I_t, depth D_t, pose P_t, and a language query q_k naming the current target. It updates a persistent semantic map M_t and returns a discrete action a_t. When the target is reached, the next query is issued without resetting the map. Figure 1 summarizes the components of this pipeline.

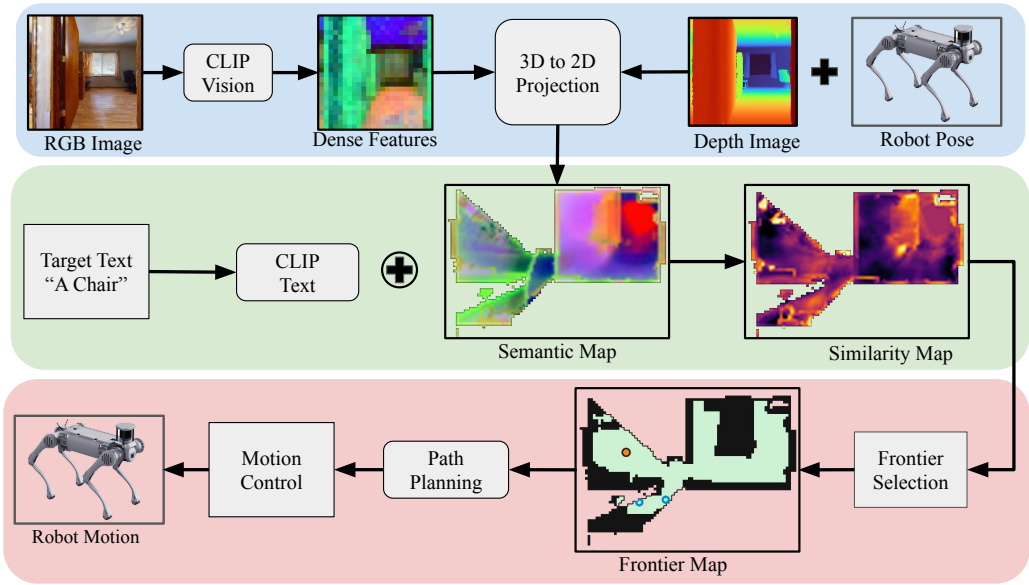


Figure 1. OneMap observation-to-action pipeline. The block labeled CLIP Vision denotes the SED image encoder. Its RGB features are projected into a persistent semantic map using depth and pose. A target-text embedding queries the map, and frontier selection, planning, and control produce the next action. Dense features and maps use PCA pseudocolor.

OneMap constructs its semantic map by encoding each RGB observation with a CLIP-aligned SED image encoder [3,28,29]. Depth and pose project the resulting feature grid into a top-down map, where repeated observations are combined by confidence-weighted fusion. The map stores visual features rather than target-specific scores, so evidence gathered for one object remains available when the query changes.

For retrieval, the current query is encoded in the same feature space. Cosine similarity between the text embedding and each observed map cell produces a target-specific similarity map without modifying the persistent representation. Similarity peaks and unexplored frontiers then provide candidate goals. OneMap selects a goal, computes a graph-based path, and applies its motion controller. YOLOv7 confirms visible HM3D targets, while MobileSAM segments each detection candidate for depth-based localization [30,31].

This pipeline exposes four costs that are addressed in the following subsections. First, the SED image encoder and detector execute as separate framework models in OneMap [3]. Second, projection materializes a dense feature tensor at image resolution before map aggregation. Third, persistent state and navigation masks repeatedly cross processor boundaries. Finally, YOLO executes at every observation. The proposed implementation changes these four execution paths while preserving the map coordinates, target confirmation rule, frontier scores, graph search, controller, and action interface.

3.2. Compiled Perception

OneMap evaluates the SED image encoder and YOLOv7 as separate PyTorch models. We retain their pretrained checkpoints, preprocessing, and output formats but compile their fixed-shape inference paths with TensorRT, which provides graph optimization and mixed-precision inference for NVIDIA GPUs [32]. The ConvNeXt-Large SED image encoder receives a 768×768 RGB image and returns the same 768-dimensional feature grid used by OneMap. It executes in FP16. The CLIP text embedding is computed only when the query changes and is then reused.

YOLOv7-E6E receives a tensor of shape $1 \times 3 \times 448 \times 640$. Its convolutional layers execute in FP16, whereas exponential, power, and sigmoid operations in the decoder remain in FP32 because their outputs directly affect box coordinates and confidence scores. The image encoder and detector use separate TensorRT contexts and CUDA streams. When detection is requested, both models process the same observation concurrently because neither requires the other's output.

The compiled contexts remove repeated framework dispatch and permit the concurrent execution of the two independent perception branches. Their outputs retain OneMap's original interfaces: the image encoder produces a dense semantic grid, and YOLO produces decoded boxes and confidence scores. The next stage projects the semantic grid into map coordinates.

The downstream stages require only a dense visual grid $F_t \in \mathbb{R}^{d \times h \times w}$ and a query embedding $e(q_k) \in \mathbb{R}^d$ in the same representation space. Section 6.5 discusses the substitution of other paired vision–language encoders.

3.3. Fused Map Projection

OneMap projects dense visual features into the persistent map in three steps [3]. It first expands the low-resolution feature grid to the depth image resolution and then transforms valid depth pixels into map coordinates; finally, it combines pixels assigned to the same map cell. The first step materializes an $H \times W \times d$ tensor for image height H, width W, and feature dimension d, requiring $\Theta(HWd)$ temporary storage.

Our projection produces the same cell-level features and confidence values without constructing this tensor. Let valid depth pixel p have depth z_p and reliability s_p. As in OneMap [3], s_p depends on the local depth gradient and the difference between z_p and the encoder's preferred observation distance. The camera intrinsics and pose transform the pixel into episodic frame point x_p, which is assigned to the map cell with index $g(p)$:

$$g(p) = \left\lfloor \frac{x_p^{xy}}{r} \right\rfloor + o, \tag{1}$$

where x_p^{xy} contains the horizontal coordinates, $r = 0.1$ m is the map resolution, and o is the map origin offset. For each occupied cell i, the output feature and confidence are

$$f_i^t = \frac{\sum_{p:g(p)=i} s_p \tilde{f}_p}{\sum_{p:g(p)=i} s_p}, \qquad s_i^t = \frac{1}{|\mathcal{P}_i|} \sum_{p:g(p)=i} s_p, \tag{2}$$

where $\mathcal{P}_i$ is the set of valid pixels assigned to cell i, and $\tilde{f}_p$ is obtained by bilinear interpolation from the low-resolution feature grid. Equations (1) and (2) define the output preserved by the accelerated implementation.

The accelerated path performs coordinate transformation, map indexing, grouping, and reduction on the GPU. Each valid pixel reads only the four neighboring feature vectors required for interpolation. Pixels are grouped by map index, and a segmented weighted reduction writes one feature vector, confidence value, and obstacle value for each updated cell. The desktop evaluation uses Triton, a tile-oriented language and compiler for custom GPU tensor operations [33], whereas the Jetson uses a CUDA extension with the same inputs and reduction order. In a fixed-seed equivalence test over 50 randomly generated $24 \times 24 \times 768$ FP16 feature grids and 640×640 output grids, the two backends produced a maximum absolute difference of zero.

The resulting temporary storage is $\Theta(P + Ud)$, where P is the number of valid depth pixels and U is the number of updated cells. Here, U is much smaller than the number of image pixels HW and the total number of map cells N. The fused order therefore avoids the dense feature intermediate at image resolution and reduces allocation and global-memory traffic. In the evaluated configuration, OneMap's spatial uncertainty kernel spans one map cell and is therefore an identity; we omit it only for this configuration. A finer map or larger uncertainty radius would require the corresponding filtering operation.

3.4. Mapping and Navigation Runtime

Projected features enter a persistent map that is updated at every observation. For map cell i, let m_i^{t-1} and c_i^{t-1} denote its previous feature and accumulated confidence, and let f_i^t and s_i^t denote the feature and confidence from the current observation. OneMap's confidence-weighted update can be written as

$$m_i^t = \frac{c_i^{t-1} m_i^{t-1} + s_i^t f_i^t}{c_i^{t-1} + s_i^t}, \qquad c_i^t = c_i^{t-1} + s_i^t. \tag{3}$$

With confidence interpreted as inverse variance, Equation (3) is equivalent to OneMap's Kalman gain form [3]. We perform the accumulation in FP32 and store the resulting semantic features in FP16. This retains higher precision for repeated fusion while approximately halving persistent feature storage; Section 5 evaluates lower-precision alternatives.

Equation (3) preserves OneMap's continuously accumulative rule and applies neither confidence saturation nor temporal fading. The current-observation gain is $\alpha_i^t = s_i^t / (c_i^{t-1} + s_i^t)$, so the accumulated confidence reduces the influence of later evidence. This rule is held fixed to isolate execution changes; Section 6.6 discusses its implications in changing environments and a cell-local fading extension.

The semantic map, confidence map, fusion operation, and map text similarity remain on the GPU across observations. Persistent buffers replace repeated allocation. The runtime packs the obstacle, observed, explored, checked, and navigability masks into one array before the CPU transfer. It transfers the similarity map at most once per observation and reuses it for frontier scoring, target confirmation validation, and planning.

Frontier extraction, point-of-interest processing, graph search, and control remain CPU-resident because they operate on compact two-dimensional structures. Their repeated

Python-level work is reduced without changing the navigation rules. Point-of-interest region growing operates on contiguous arrays in C++, while frontier extraction retains OneMap's Python/OpenCV implementation. Planning mask dilation uses an integral image so that each rectangular-window test requires four indexed reads rather than repeated neighborhood traversal. The resulting masks and candidate goals are passed to OneMap's graph search and controller. After reducing these per-step data movement and control costs, detector invocation remains the largest avoidable computation.

3.5. Adaptive Detection

In OneMap, both the SED image encoder and YOLO run for every observation [3]. The image encoder is required continuously because its features update the map. YOLO serves a narrower role: it confirms and localizes the current target. We therefore keep image encoding at every step and schedule only the detector. Its checkpoint, decoded detections, and OneMap target confirmation rule are unchanged.

The scheduler uses current-view semantic evidence already produced by the image encoder. For normalized image features F_t and target embedding $e(q_k)$, the current-view evidence is

$$v_t = \max_p \langle F_t(p), e(q_k) \rangle. \tag{4}$$

where p indexes locations in the dense feature grid. Because both vectors are normalized, the inner product is the cosine similarity.

Equation (4) obtains target-conditioned spatial evidence by comparing the current image feature at each location with the target-text embedding. This current-view score is distinct from the persistent map text similarity used for semantic goal selection. The temporal scheduler then evaluates the scalar difference $v_t - v_{t-1}$ between successive target-conditioned maxima.

The scheduler has two explicit phases. *Before encoding step* t, it decides whether to run YOLO from states carried from earlier observations. The detector runs if the periodic safety counter has expired or a detection burst is active. This decision is available before either neural model starts, so YOLO can execute concurrently with the image encoder.

After encoding step t, the scheduler computes v_t and updates the state used by future observations. It compares v_t with threshold τ and the rise $v_t - v_{t-1}$ with threshold δ. If either threshold is exceeded, a burst of b calls begins at $t+1$. Thus, evidence first observed at step t can trigger detection at steps $t+1$ and $t+2$ when $b = 2$. The periodic invocation bounds the detector-free interval, and all scheduler states are reset when the query changes.

The evaluated operating point uses a four-observation safety interval, $\tau = 0.16$, $\delta = 0.02$, and $b = 2$. Section 4 describes parameter selection and the independent sensitivity protocol. The score indicates semantic relevance, not distance; depth continues to provide geometric localization. Because score calibration depends on the image encoder and checkpoint, changing either would require recalibrating τ and δ on validation data.

If p_{det} is the fraction of observations on which YOLO runs, the perception cost lies between the encoder-only path and the concurrent encoder-YOLO path. The mean step cost is

$$T_{\text{acc}} = (1 - p_{\text{det}}) T_{\text{enc}}^{\text{TRT}} + p_{\text{det}} T_{\text{conc}}^{\text{TRT}} + T_{\text{proj}}^{\text{fused}} + T_{\text{fuse}} + T_{\text{nav}}^{\text{compiled}} + T_{\text{ctrl}}. \tag{5}$$

In Equation (5), $T_{\text{enc}}^{\text{TRT}}$ is the measured encoder-only latency and $T_{\text{conc}}^{\text{TRT}}$ is the measured concurrent encoder-detector latency, including shared-GPU contention. The remaining terms denote fused projection, map fusion, frontier and point-of-interest processing, and control. Unlike fixed frame skipping, p_{det} changes with the current-view target evidence observed during navigation.

3.6. Integrated Pipeline

Algorithm 1 combines the four changes into one observation-to-action step. In addition to map M_t, the implementation maintains scheduler state S_t, comprising the safety counter and remaining burst length. The algorithm keeps dense encoding and map updates continuous while making detection conditional.

Algorithm 1: Hardware-aware open-vocabulary navigation step

Input: RGB image I_t, depth D_t, pose P_t, query q_k, map M_{t-1}, scheduler state S_{t-1}
Output: Action a_t, updated map M_t, updated scheduler state S_t

1. If q_k changed, compute and cache its text embedding and reset scheduler state.
2. Determine whether YOLO is required from the safety interval and active burst in S_{t-1}.
3. Launch the SED image encoder and, when required, YOLO on separate CUDA streams.
4. Compute current-view target similarity v_t from the image features and update S_t for subsequent observations.
5. Project valid RGB-D features directly into map cell feature, confidence, and obstacle accumulators.
6. Apply FP32 confidence-weighted fusion and store the persistent semantic map in FP16.
7. Compute map text similarity and generate the packed navigation masks.
8. For each YOLO candidate, decode its MobileSAM mask, localize it using depth, and apply OneMap's target confirmation rule.
9. If no target is confirmed, execute frontier extraction, compiled point-of-interest clustering, and exact mask dilation, followed by OneMap's graph search and controller.
10. If the target is reached, advance to q_{k+1} without resetting M_t; return a_t, M_t, and S_t.

Figure 2 compares processor residency for 25 defined operations. Each implementation executes 24 rows: OneMap has no adaptive detector scheduler, whereas ours omits the one-cell spatial uncertainty identity. OneMap contains 13 CPU-only and 11 GPU-only executed operations. Our implementation contains eight CPU-only, 15 GPU-only, and one hybrid executed operations. The principal residency changes are semantic map storage, confidence map storage, map fusion, map text similarity, and planning mask generation, which move from the CPU to the GPU.

3.7. Computational Complexity

We characterize complexity in terms of the data structures processed by the navigation pipeline. Let $N = n_x n_y$ be the number of cells in the persistent map, d the semantic feature dimension, and R the number of pixels in one depth observation. At observation t, let $P_t \leq R$ denote the valid projected pixels, $U_t \leq \min(P_t, N)$ the unique map cells receiving updates, and $V_t \leq N$ the traversable cells examined by graph search. Table 2 gives the worst-case sequential work and additional working memory. Fixed neural network parameters and backend-specific workspaces are excluded because their sizes do not vary with N, d, or the observation count for a selected backend.

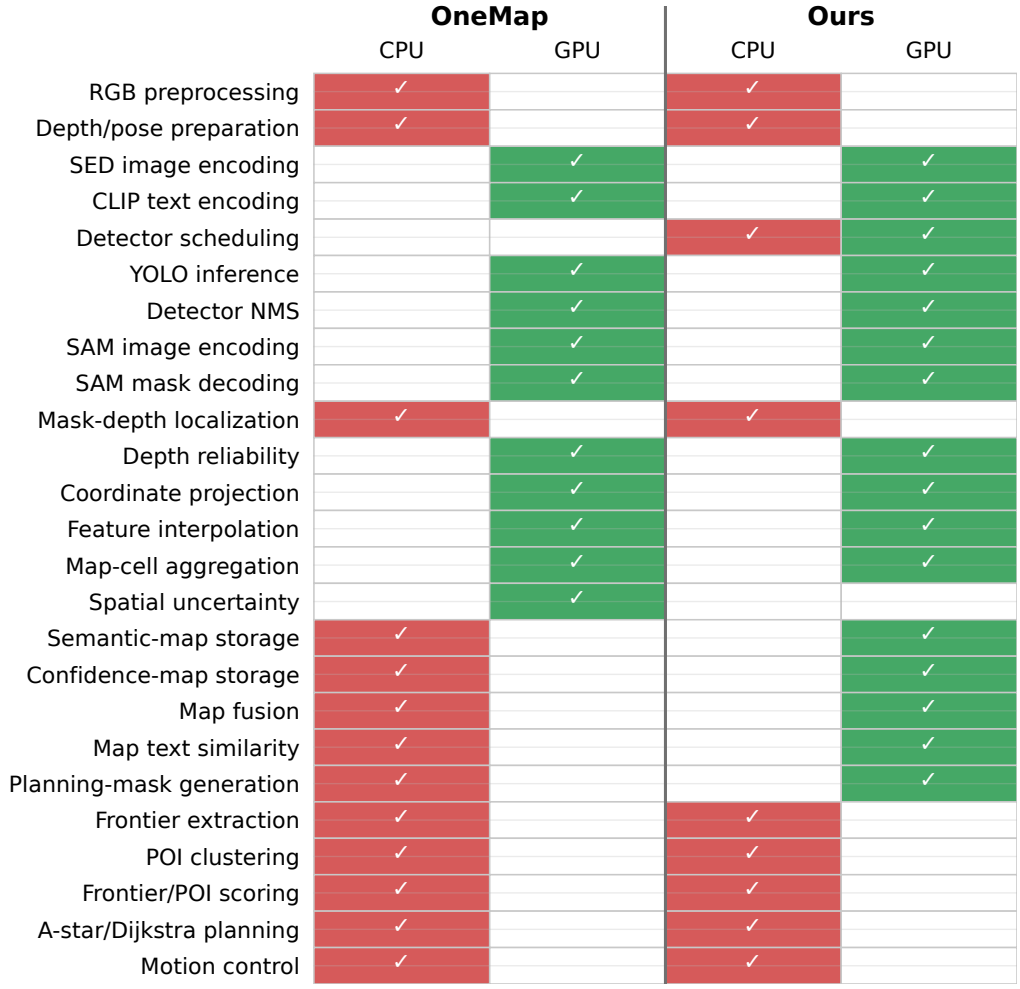


Figure 2. CPU-GPU residency for 25 defined operations. Each implementation executes 24 rows: OneMap lacks adaptive detector scheduling, whereas ours omits the configured one-cell spatial uncertainty identity. Filled cells identify the executing processor; blank cells indicate non-execution. The proposed scheduler spans both processors because GPU current-view target similarity updates the CPU scheduler state.

Table 2. Time and working-memory complexity of the mapping and navigation stages for one observation. The OneMap projection row is included to expose the dense intermediate removed by the proposed fused projection.

Operation	Time	Additional Working Memory
OneMap projection	$O(Rd + P_t \log P_t + P_t d)$	$\Theta(Rd + U_t d)$
Fused projection	$O(R + P_t \log P_t + P_t d)$	$O(P_t + U_t d)$
Map fusion	$O(U_t d)$	$O(U_t d)$
Map text similarity	$O(U_t d)$; $O(Nd)$ after a query change	$O(U_t)$; $O(N)$ after a query change
Masks and frontiers	$O(N)$	$O(N)$
Graph search	$O(V_t \log V_t)$	$O(V_t)$
Persistent map	-	$\Theta(Nd + N)$

Let T be the number of observations, $K \leq T$ the number that invoke the detector, and $Q \leq T$ the number of target query changes. For fixed neural models with image encoder cost C_{enc} and detector cost C_{det}, a worst-case sequential bound for an episode is

$$O\left(TC_{\text{enc}} + KC_{\text{det}} + \sum_{t=1}^{T}[R + P_t \log P_t + P_t d + U_t d + N + V_t \log V_t] + QNd\right). \tag{6}$$

In Equation (6), the QNd term accounts for evaluating a new text query against the complete persistent map; between query changes, the similarity is recomputed only for the U_t updated cells. Frontier and path computations are event-driven in the implementation, so the expression is a conservative bound that permits them at every observation. Adaptive detection replaces the full detector term TC_{det} with $KC_{\text{det}} = \rho TC_{\text{det}}$, where $\rho = K/T$, reducing repeated work without changing the worst-case linear dependence on T.

Excluding fixed model storage and optional trajectory logging, the peak runtime memory is

$$O\left(Nd + N + \max_{1 \le t \le T}\{P_t + U_t d + V_t\}\right). \tag{7}$$

Equation (7) shows that memory does not grow with the observation count because the map and temporary buffers are reused. It grows linearly with the feature dimension and with the number of map cells; doubling both map axes quadruples N, persistent semantic storage, and full-map mask processing. The fused projection removes the $\Theta(Rd)$ intermediate at depth resolution. When $P_t \approx R$, both projection paths still perform work proportional to Rd, so the principal theoretical improvement is the smaller working set and reduced memory traffic rather than a lower worst-case time class. In the evaluated configuration, the $600 \times 600 \times 768$ FP16 semantic map occupies approximately 527 MiB, while a $640 \times 640 \times 768$ dense FP16 image intermediate would require approximately 600 MiB for one observation.

For a square physical map of width L and cell width r, $N = (L/r)^2$ and the persistent semantic storage is $\Theta(L^2 d/r^2)$. Decreasing r therefore increases the capacity and full-map processing requirements for both implementations, which retain the same persistent grid. By contrast, the sensor and encoder fix the image resolution R and low-resolution feature grid. The proposed interpolation operator therefore has no additional dependence on the total number of map cells. The map resolution affects each observation only through the number and spatial distribution of its updated cells U_t.

4. Experimental Setup

This section defines the navigation benchmarks, profiling protocol, and embedded hardware evaluation used to assess task quality and computational efficiency. Complete HM3D benchmark results quantify the navigation quality, while synchronized desktop profiles and paired Jetson hardware-in-the-loop runs characterize latency, memory, power, and energy-to-completion.

4.1. Navigation Benchmarks

Navigation quality is evaluated in Habitat using the HM3D validation environments [34,35]. We follow the ObjectNav evaluation conventions [36] and the benchmark configurations defined by OneMap [3]. The single-object benchmark contains 2000 episodes across 20 scenes and six target categories, with at most 1000 actions per episode. The multi-object benchmark contains 236 episodes over the same scene collection; each episode requests three sequential objects and permits up to 1500 actions. RGB-D observations are square and use the OneMap action space.

For single-object navigation, we report the success rate (SR) and success weighted by path length (SPL), following the standard embodied navigation definitions [37]. For multi-object navigation, we additionally report progress, the fraction of requested objects reached, and progress weighted by path length (PPL), following OneMap [3]. We reran the OneMap

reference implementation using the same checkpoints, benchmark configurations, official splits, simulator, software environment, and evaluation code used for the accelerated implementation. The reproduced aggregate scores matched the values reported by OneMap to the displayed precision; the reproduced values form the baseline for all quality retention calculations. Controlled component comparisons use identical profiling traces and execution settings except for the component under test.

4.2. Runtime Profiling Protocol

The profiler records initialization and per-step events using a monotonic nanosecond clock. CUDA synchronization is performed immediately before and after GPU stages so that asynchronous kernel launches are attributed to the stage that issued them. The first ten observations of each run are excluded from aggregate stage statistics. For every component, we record the invocation count, mean, median, 95th percentile, minimum, maximum, standard deviation, and cumulative latency. The instrumented stages include SED image encoding, YOLO inference, concurrent perception, conditional MobileSAM encoding and decoding, projection, map fusion, similarity evaluation, frontier and point-of-interest extraction, path planning, and control.

Projection equivalence is evaluated by replaying 601 recorded observations. For each observation, one TensorRT SED feature grid is computed and supplied with the same depth image, camera pose, camera calibration, and mapping configuration to the original OneMap dense projection and the proposed fused projection. The comparison is performed on their sparse outputs before persistent map fusion and measures destination cell indices, confidence values, semantic cell features, obstacle labels, and obstacle confidence values.

Desktop profiling records process-level host memory independently of device utilization. Jetson hardware measurements use the board-level random access memory (RAM) value reported by NVIDIA's `tegrastats` utility [38]. Habitat rendering, network communication, and on-device OneMap compute are timed separately, allowing kernel and pipeline speedups to be attributed to the native and accelerated processing paths.

4.3. Software and Compilation

The desktop experiments use Python 3.10.19, PyTorch 2.11.0.dev20260105 with CUDA 12.8, ONNX 1.22.0, TensorRT 11.1.0.106, and TensorRT RTX 1.5.0.114. The evaluation uses OneMap source revision 0b565fd, Habitat-Sim 0.2.4, Habitat-Lab 0.2.420230405, OpenCV 4.8.0, Triton 3.6.0, pybind11 3.0.4, and CMake 4.4.0. Both neural networks are exported with ONNX opset 17. TensorRT uses optimization level 5 and a 4 GiB workspace for the standard builder. The YOLO engine constrains exponential, power, and sigmoid operations to FP32 while permitting FP16 elsewhere. Jetson engines are compiled on the target under Jetson Linux R36.4.4 and CUDA 12.6.

The desktop host runs Ubuntu 22.04 with an NVIDIA GeForce RTX 5070 Ti (NVIDIA Corporation, Santa Clara, CA, USA), NVIDIA driver 580.173.02, cuDNN 9.10.2, and GCC 11.4.0. The Jetson environment uses Ubuntu 22.04, Python 3.10.12, PyTorch 2.8.0, TensorRT 10.3.0, cuDNN 9.3.0, and GCC 11.4.0. TensorRT engines are built separately on each execution platform because serialized engines are specific to the target TensorRT version and GPU architecture. Table 3 reports the remaining conversion and compilation settings.

NVIDIA identifies the Jetson AGX Orin GPU as compute capability 8.7, an architecture designation that specifies its supported GPU instruction and precision features. The TensorRT hardware matrix supports FP16 and INT8, but not native FP8 execution, on this architecture [39]. Consequently, the FP8 candidate in Table 4 evaluates persistent map storage compression rather than FP8 Tensor Core acceleration; map fusion and map text similarity require conversion to a supported arithmetic precision.

The accelerated Jetson runtime loads the TensorRT engines for SED and YOLO. When detection is requested, it launches the engines on separate CUDA streams. The runtime stores image and map features in FP16, accumulates map updates in FP32, and uses the CUDA projection backend. The evaluated scheduler uses a four-observation safety interval, $\tau = 0.16$, $\delta = 0.02$, and a two-call burst. The OneMap baseline uses the corresponding PyTorch models and FP32 semantic map. It invokes YOLO at every observation and uses the dense projection path. Profiling excludes ten warm-up observations and synchronizes CUDA immediately before and after each measured GPU stage. No INT8 quantization or structured sparsity is enabled.

Table 3. Model conversion and native extension settings used in the experiments.

Component	Export or Input	Compilation Configuration
SED encoder	Static $1 \times 3 \times 768 \times 768$ FP16; ONNX opset 17; constant folding	Strongly typed network; TensorRT optimization level 5; 4 GiB workspace with the standard builder
YOLOv7-E6E	Static $1 \times 3 \times 448 \times 640$; ONNX opset 17; constant folding	FP16 except FP32 `Exp`, `Pow`, `Sigmoid`, and resize-scale constants; optimization level 5; 4 GiB workspace
CUDA projection	PyTorch CUDA extension; Jetson compute capability 8.7	C++ `-O3`; NVCC `-O3 -use_fast_math`; built on the Jetson with CUDA 12.6
CPU navigation	pybind11 and Eigen 3.3 interface	C++17 CMake release build using GCC 11.4.0

Table 4. Isolated fixed-input desktop component results and correctness checks. Latency entries are mean component times. The first five rows compare the OneMap reference with the proposed implementation; the FP8 row compares the accepted FP16 map with the rejected FP8 storage candidate.

Component	Reference	Candidate	Gain	Check
Dense encoder	16.79 ms	11.15 ms	1.51×	Cosine: 0.999983
YOLO inference	9.04 ms	2.96 ms	3.05×	Cosine: 0.999998
Map projection	21.36 ms	9.14 ms	2.34×	Cell sets: 601/601; min. cosine: 0.99999976
Frontier and POI	33.93 ms	5.56 ms	6.10×	POI output match
Obstacle dilation	9.16 ms	1.59 ms	5.76×	40/40 paths
FP8 map storage	527 MiB	264 MiB	2.00×	Terminal outcome changed

The two projection backends are checked separately from the navigation evaluation. A fixed-seed, 50-trial test compared Triton and CUDA segmented reductions using $24 \times 24 \times 768$ FP16 feature grids and 640×640 output grids. The test produced a zero maximum and mean absolute error.

4.4. Jetson Hardware-in-the-Loop Evaluation

Embedded measurements use an NVIDIA Jetson AGX Orin Developer Kit (NVIDIA Corporation, Santa Clara, CA, USA) with 32 GB unified memory, Jetson Linux R36.4.4, and CUDA 12.6. The platform is evaluated in its 15 W, 30 W, 50 W, and MAXN power modes. Following NVIDIA's platform power and performance controls [40], `nvpmodel` selects the power envelope and `jetson_clocks` fixes the maximum clocks allowed within this envelope. Native and accelerated runs are alternated across cases to reduce ordering bias.

All HIL runs use the multi-object evaluation configuration with a 1500-action limit, 640×640 square RGB-D observations, and a 600×600 map covering 60×60 m at 0.1 m per cell. The Jetson power configurations correspond to `nvpmodel` IDs 1, 2, 3, and 0 for 15 W, 30 W, 50 W, and MAXN, respectively.

Habitat and HM3D execute on the workstation, while the complete OneMap actor, including perception, mapping, frontier selection, planning, and control, executes on the Jetson. At each closed-loop step, the workstation sends RGB, depth, pose, and the query state. The Jetson returns the selected action and target-found state. Habitat then applies this action before generating the next observation. This closed-loop protocol allows each implementation to determine its own trajectory. The communication round-trip time and Jetson compute time are logged separately.

The protocol provides a controlled closed-loop evaluation of the complete embedded navigation actor while Habitat supplies sensing, localization, environment dynamics, and actuation. This boundary separates Jetson computation, host device communication, and the complete episode duration, enabling the direct attribution of the accelerated actor's contribution. The integrated physical platform evaluation extends this measurement boundary to camera acquisition, sensor driver and direct memory access latency, depth noise, localization drift, motion blur, actuator responses, and concurrent onboard processes.

The hardware study uses episodes 78, 225, and 65, selected before execution as the episodes nearest to the 10th, 50th, and 90th percentiles of the annotated optimal three-target path length. The clustered case has an optimal distance of 5.55 m, the median-length case 11.95 m, and the distributed case 22.87 m. Each case is executed three times for both implementations in every power mode, producing 72 closed-loop runs arranged as 36 native-accelerated pairs. The native and accelerated execution order is alternated across repetitions to reduce ordering and thermal bias. The comparison reports the navigation outcome, on-device step latency, end-to-end episode duration, peak system RAM use, sampled summed rail power, and summed rail energy

$$E_{\text{rail}} = \bar{P}_{\text{rail}} T_{\text{episode}}, \tag{8}$$

where $\bar{P}_{\text{rail}}$ is the sum of the episode-mean values of the available Jetson power rails, and T_{episode} is the measured end-to-end episode duration. Peak system RAM use is the maximum board-level RAM-used value reported by `tegrastats`; the reported memory, power, and rail fields follow NVIDIA's utility documentation [38]. We report E_{rail} as a summed board-rail energy proxy, which precisely identifies its `tegrastats`-based measurement boundary.

4.5. Controlled Evaluation Protocols

Four controlled evaluations complement the closed-loop benchmarks. They use matched inputs to attribute component effects, test projection scaling, measure scheduler sensitivity, and isolate the scheduler's response to motion blur. Unless stated otherwise, the Jetson configuration and profiling boundaries follow Section 4.4.

For the component ablation, the Jetson replays a fixed 601-step trace containing RGB, depth, pose, and target query observations from one multi-object episode. We compare the complete implementation with four leave-one-out variants. The variants replace TensorRT with PyTorch, fused projection with dense projection, and FP16 map storage with FP32 storage. The fourth variant runs detection at every eligible observation. We evaluate each configuration once in the 15 W, 30 W, 50 W, and MAXN modes, producing 20 runs. The replay records actions and target-found states without feeding them back into the trace. Every configuration therefore processes the same observations in the same order.

We first screened scheduler parameters on 16 single-object development episodes against always-on detection. The selected setting uses a four-observation safety interval, $\tau = 0.16$, $\delta = 0.02$, and $b = 2$. It matched the always-on result of 7/16 successes and a mean SPL of 0.250 while invoking the detector on 47.1% of observations. We then fixed these values for the primary evaluations. A separate sensitivity analysis uses 16 multi-object

episodes selected from equal quantile bins of the total optimal three-target path length. Their distances range from 3.61 to 29.97 m. The analysis varies one parameter at a time around the selected setting. It tests $\tau \in \{0.12, 0.20\}$, $\delta \in \{0.01, 0.03\}$, $b \in \{1, 3\}$, and safety intervals of 2 and 8. All other settings remain fixed. We report the SR, progress, exact terminal-state agreement, and detector invocation fraction. Detector invocation represents the computational demand rather than desktop latency, which is not an embedded hardware measurement.

The motion blur test uses the same 601-observation trace, which contains three sequential target queries. We replay the RGB input without corruption and with deterministic directional motion kernels of 7, 15, and 31 pixels. These kernels represent mild, moderate, and severe blur. Their direction cycles through 0°, 45°, 90°, and 135° across observations. The depth, pose, query order, thresholds, burst length, and safety interval remain fixed. For every condition, the Jetson executes the same TensorRT SED encoder and records v_t, semantic threshold events, and the resulting detector schedule across all 601 observations. Clean-input decisions provide the matched reference. Added and missed invocations quantify schedule changes rather than YOLO classification errors. Recovery delay measures the observations from a missed clean-input invocation to the next detector call under blur.

The resolution scaling experiment isolates projection and map fusion on the Jetson in MAXN. It compares fused and dense projection with matched FP16 persistent map storage. We test $r \in \{0.10, 0.075, 0.05, 0.03, 0.02\}$ m while retaining a 60×60 m physical extent and $d = 768$. Both paths process the same 128 depth and pose samples from the fixed 601-step trace. They also receive identical $24 \times 24 \times 768$ FP16 feature inputs. We exclude the first ten updates. Every resolution uses a one-cell uncertainty kernel. This setting isolates interpolation, destination cell aggregation, and indexed map fusion without introducing resolution-dependent neighborhood expansion.

5. Results

The experiments proceed from local costs to complete-system behavior. We first locate OneMap's repeated costs, validate the modified components, and isolate their contributions to embedded execution. We then measure the navigation quality on the complete benchmarks, evaluate the Jetson performance and closed-loop episode behavior, and finally test robustness to the map resolution, scheduler parameters, and visual corruption.

5.1. Runtime Characterization

Figure 3 shows that the repeated workload is distributed across perception, semantic projection, and navigation processing. OneMap requires 111.01 ms per step on the desktop profiling episode. Dense perception is the largest block at 65.06 ms, but projection and frontier processing add another 35.91 ms. These two non-perception stages alone account for 32.3% of the complete step, while all non-perception stages account for 41.4%. This distribution establishes the need for pipeline-level rather than model-only acceleration.

The detector-enabled path in our implementation requires 43.62 ms, giving a 2.55× step-level speedup before adaptive detector scheduling. Concurrent TensorRT perception reduces the combined image encoder and detector time to 19.88 ms. Fused projection falls from 22.92 to 9.08 ms, and frontier processing falls from 12.99 to 5.56 ms. Map fusion also becomes shorter after persistent GPU storage removes the repeated processor transition.

Path computation is the only displayed block that is longer in the accelerated trace, increasing from 3.77 to 5.49 ms. The planner itself is unchanged; the two closed-loop runs present it with different navigability masks, frontier goals, and search states. These bars therefore describe different planning instances. In the test with fixed inputs, which is reported next, accelerated mask processing is faster and preserves every reference path.

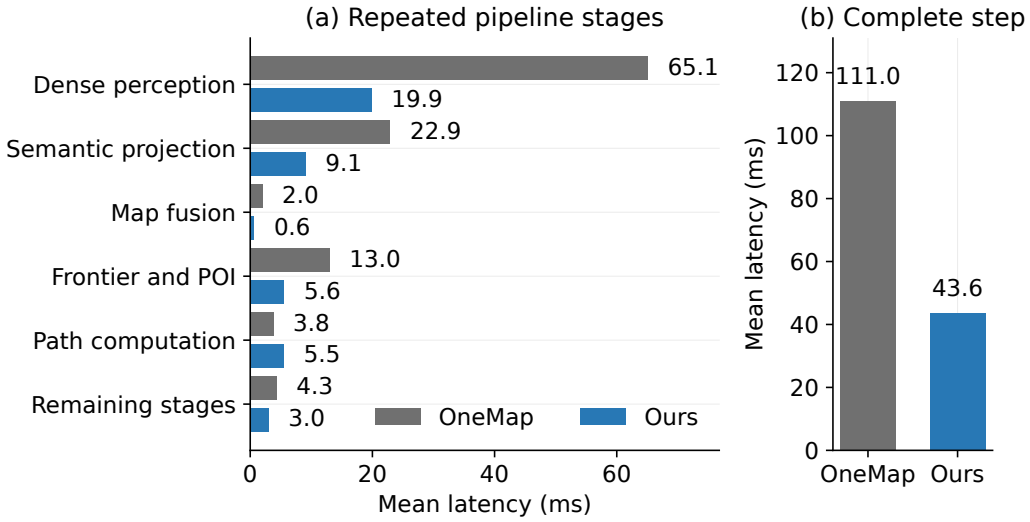


Figure 3. Desktop runtime profile. (**a**) Mean latency of repeated stages in OneMap and our detector-enabled path. Dense perception is sequential image encoding and YOLO for OneMap and concurrent TensorRT execution for ours. The planner row is trace-dependent rather than an identical-input comparison. (**b**) Complete synchronized step latency on the same profiling episode.

5.2. Component Validation

Table 4 reports the isolated component timing together with the corresponding correctness check. These values differ from the closed-loop stage means in Figure 3: the table controls component inputs, whereas the figure averages the states and invocation mix encountered along separate profiling trajectories. The dense encoder row measures SED alone rather than the grouped concurrent perception stage. Likewise, the frontier/POI row executes the complete routine on matched stored map states, while the figure reflects the trajectory-dependent number and spatial distribution of frontiers and semantic candidates. Validation uses neural similarity with fixed inputs, direct numerical equivalence for fused projection, synthetic point-of-interest equivalence, reference-path agreement for obstacle dilation, and a controlled terminal outcome for FP8 storage. Together, the tests show that the complete speedup is assembled from individually verified changes.

The original and fused projection paths update identical destination cell sets in all 601 observations, giving a cell-set intersection over union of 1.0, and their obstacle labels agree exactly. Across 647,482 updated cells and 497,266,176 semantic feature values, the mean cell feature cosine similarity is 1.000000 and the minimum is 0.99999976. The mean absolute semantic feature error is 1.14×10^{-8}, with a relative L_2 error of 3.94×10^{-6}. Confidence values have a mean absolute error of 3.19×10^{-7} and relative L_2 error of 7.31×10^{-6}. The residual differences are consistent with the floating-point reduction order.

The cosine values compare neural outputs under fixed inputs. The projection check directly compares the numerical outputs before fusion from the original and fused operators over 601 replayed observations. The point-of-interest check compares C++ and Python clustering on synthetic maps, and the obstacle dilation check compares planned paths under fixed inputs. The separate 50-trial test in Section 4.3 verifies that the desktop Triton and Jetson CUDA projection backends produce identical reductions. FP8 halves semantic map allocation but changes a controlled terminal outcome and is therefore excluded. The accepted system stores map features in FP16 and accumulates confidence in FP32.

This FP8 test is a post-training cast of the online semantic map state, not the quantization of network weights or activations. Conventional quantization-aware training inserts simulated quantization into a trainable model so that its parameters adapt to the resulting numerical error [41]. The persistent map has no trainable parameters: it is created after

deployment through geometric projection and recurrent confidence-weighted fusion. Compensating for its FP8 error through training would therefore require fine-tuning the SED encoder through projection, recurrent map updates, map text similarity, and a task-level objective. Such training would modify OneMap's pretrained checkpoint and potentially its image–text alignment, introducing a new navigation model rather than preserving the reference representation for a controlled hardware comparison. QAT can reduce quantization errors but does not guarantee unchanged closed-loop outcomes. Together with the absence of native FP8 execution on AGX Orin, these considerations motivated the evaluation of FP8 only as a storage candidate and the selection of hardware-native FP16 as the lowest accepted map precision.

5.3. Component Ablation

Table 5 compares the complete implementation with four leave-one-out variants using the fixed trace described in Section 4.5. Its three panels report the compute time, trace energy, and peak RAM separately so that the effects of each component can be compared at every Jetson power mode. The variants are ordered from largest to smallest overall compute and energy effect, followed by the full-system reference.

Table 5. Component ablation on the Jetson AGX Orin across four power modes. Each entry is measured from the same 601-step trace. Compute values are mean on-device processing times, energy is the summed rail energy accumulated over the trace, and memory is peak system RAM use.

Configuration	15 W	30 W	50 W	MAXN
Mean compute time (ms/step)				
Without fused projection	1579.4	970.9	477.8	331.8
Without TensorRT	959.4	512.6	268.7	184.1
Without adaptive detection	822.1	449.7	248.9	175.6
Without FP16 storage	740.3	402.7	221.4	152.5
Full system	743.2	401.7	223.7	154.9
Trace energy (kJ)				
Without fused projection	9.86	8.28	6.05	7.01
Without TensorRT	6.13	4.87	3.97	4.67
Without adaptive detection	5.61	4.51	3.79	4.57
Without FP16 storage	5.03	4.03	3.36	4.03
Full system	5.00	4.05	3.36	4.03
Peak system RAM (MiB)				
Without fused projection	10,836	10,952	11,021	11,819
Without TensorRT	7927	7971	8028	8913
Without adaptive detection	6976	7030	7005	7636
Without FP16 storage	7065	7095	7180	7917
Full system	6068	6084	6154	7050

The ablation separates the roles of the four optimizations. Fused projection provides the largest individual contribution: restoring the dense intermediate increases the total computation by 120.2%, energy by 88.5%, and peak RAM by approximately 4.70 GiB. Within these 601-step fixed-trace Jetson runs, the dense projection path is between 5.45× and 6.58× slower than the fused path across the four power modes. These stage timer ratios are distinct from the isolated fixed-input desktop measurement in Table 4. TensorRT provides the second-largest execution benefit: replacing it with PyTorch increases computation by 23.8%, energy by 19.2%, and peak RAM by approximately 1.83 GiB.

Adaptive detection reduces detector execution from 591 to 292 eligible observations. Disabling the scheduler increases computation by 11.8% and energy by 12.5%, demonstrating a consistent benefit beyond model compilation. FP16 semantic map storage primarily affects capacity: replacing it with FP32 adds approximately 975 MiB of peak RAM while

producing no systematic latency or energy change. Thus, fused projection principally controls projection and the temporary memory cost, TensorRT accelerates perception, adaptive detection removes unnecessary inference, and FP16 storage reduces the persistent memory demand.

All five configurations reproduce the reference action and target-found outputs for all 601 replayed steps in every power mode. The fixed-trace experiment therefore attributes computational changes without introducing differences in input sequence. The component effects should not be summed directly because disabling one optimization changes shared memory lifetimes and GPU contention.

5.4. Efficiency–Quality Trade-Off

Adaptive detection supplies the final reduction in repeated work. Across the complete 236-episode multi-object benchmark, the scheduler processes 228,403 observations. It invokes concurrent image encoding and YOLO on 116,526 observations and the image encoder alone on 111,877, avoiding detector execution on 48.98% of steps. The four-observation safety interval remains active throughout the run. On the desktop, the mean step latency is 39.01 ms, with a median of 36.54 ms and a 95th percentile of 56.04 ms.

Table 6 gives the corresponding navigation quality. The comparison uses the complete benchmark splits rather than the three episodes used later for paired hardware measurements. On 2000 single-object episodes, our method reaches a 51.65% SR and 33.36% SPL, retaining 92.6% and 89.2% of the reproduced OneMap scores. On all 236 multi-object episodes, it retains 96.1% of the SR and progress, 91.1% of the SPL, and 92.7% of the PPL.

Table 6. Navigation quality on the complete HM3D benchmarks. Both OneMap and our method were evaluated in this study using the same software environment, checkpoints, benchmark configurations, official splits, and evaluation code. The reproduced OneMap scores match those reported in [3] to the displayed precision.

Benchmark	Method	SR (%)	SPL (%)	Progress (%)	PPL (%)
Single-object	OneMap	55.80	37.40	-	-
Single-object	Ours	51.65	33.36	-	-
Multi-object	OneMap	54.24	27.77	65.54	32.68
Multi-object	Ours	52.12	25.31	62.99	30.30

The final operating point retains 96.1% of the multi-object SR and progress while delivering the computational gains evaluated below. Discovery metrics are retained more strongly than path efficiency metrics, and retention is highest on the sequential multi-object task for which the persistent map is intended. Because the baseline and accelerated implementation use the same evaluation environment, the retention ratios directly characterize the quality change at the proposed operating point. These complete-benchmark results quantify the quality side of the measured efficiency–quality trade-off.

5.5. Embedded Performance

Figure 4 separates the per-step compute speedup from the complete episode duration speedup on the Jetson AGX Orin. Each point is the geometric mean across nine paired runs, comprising three preselected episodes repeated three times at one power mode. The accelerated implementation is faster in all 36 paired comparisons, with overall geometric mean speedups of 4.64× for on-device computation and 3.55× for the complete episode duration.

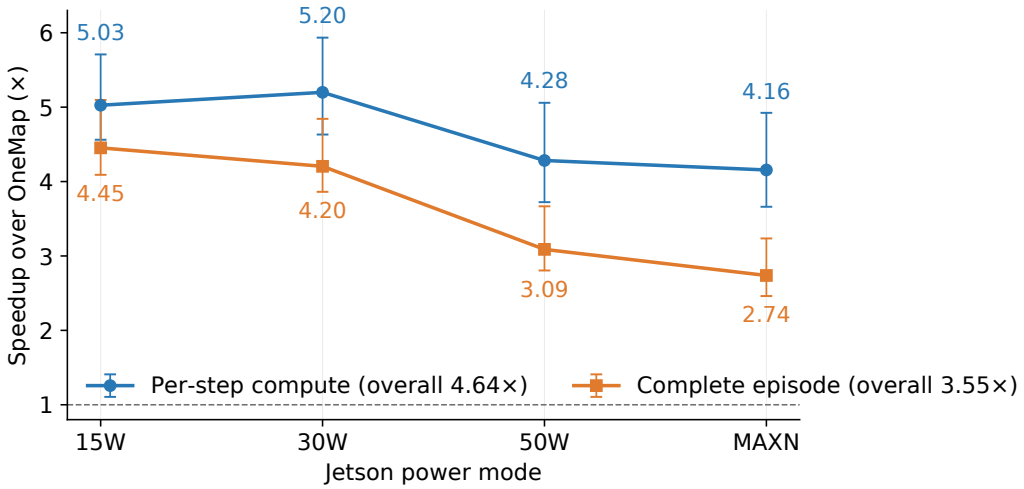


Figure 4. Jetson AGX Orin speedup across power modes. Per-step compute speedup uses the on-device request processing time. Complete episode duration speedup includes Habitat execution, host Jetson communication, Jetson computation, planning, and control. Points are geometric means over nine paired runs at each power mode, and whiskers show the paired minimum and maximum. The overall values in the legend are geometric means over all 36 paired comparisons. The dashed horizontal line denotes parity with OneMap.

The complete episode duration includes Habitat execution, host Jetson communication, Jetson computation, planning, and control computation. The reported 4.64× value isolates the on-device navigation computation accelerated by this study. Physical deployment places camera acquisition, sensor driver and direct memory access transfer, pose estimation, and actuator responses around both navigation actors; the integrated-platform evaluation described in Section 6.6 will quantify these shared system costs.

Table 7 provides the absolute measurements underlying these ratios. Accelerated 15 W computation is 1.24× faster than OneMap at 50 W, and accelerated 30 W is 1.57× faster than OneMap in MAXN. The gain therefore comes from the execution design rather than a higher board power setting. For both implementations, MAXN finishes sooner than 50 W but consumes more energy because its higher summed rail power exceeds the saving from the shorter duration.

Peak system RAM use falls by 51.21% as a mean paired reduction. The ratio remains close to one half in every power mode, consistent with the combined memory effects of TensorRT execution, FP16 map storage, persistent buffers, and the removal of the dense intermediate at depth resolution. The mean paired reduction in summed rail energy is 75.32%. The corresponding geometric mean energy-to-completion speedup is 4.08×. A shorter active duration combines with a lower computational demand to produce this energy reduction.

The per-step compute and peak memory remain comparatively concentrated within each power mode. The complete duration and energy span a wider range because the three cases were deliberately selected from different optimal-distance percentiles and therefore contain substantially different numbers of observations. Across repetitions of the same episode, method, and power mode, the median coefficients of variation are 0.20% for compute, 0.30% for duration, 2.07% for memory, and 0.43% for energy. The corresponding maxima are 1.19%, 1.63%, 8.23%, and 1.21%. These results distinguish low run-to-run variation from the broader variation caused by the episode workload.

Table 7. Absolute Jetson AGX Orin compute time, episode duration, peak system RAM, and summed rail energy across three episodes and four power modes.

Mode	Method	Compute (ms/Step)	Duration (s)	Memory (MiB)	Energy (kJ)
15 W	OneMap	4166.8; 1.01 [4121.4, 4228.9]	1045.7; 2.15 [446.5, 2610.3]	14,137; 1.03 [13,599, 14,797]	10.24; 2.14 [4.37, 25.37]
	Ours	829.0; 1.09 [740.8, 903.5]	234.8; 1.99 [105.0, 515.8]	6905; 1.05 [6113, 7441]	2.32; 2.00 [1.03, 5.10]
30 W	OneMap	2308.5; 1.02 [2272.1, 2361.4]	594.9; 2.15 [254.5, 1489.9]	13,990; 1.02 [13,615, 14,353]	8.04; 2.14 [3.44, 19.92]
	Ours	444.0; 1.09 [397.9, 492.5]	141.5; 1.97 [64.4, 309.2]	6737; 1.06 [6026, 6997]	1.86; 1.99 [0.82, 4.10]
50 W	OneMap	1032.1; 1.07 [981.9, 1123.7]	287.4; 2.20 [122.2, 746.6]	14,080; 1.02 [13,730, 14,628]	5.92; 2.16 [2.48, 14.90]
	Ours	241.0; 1.07 [222.1, 263.7]	93.0; 1.96 [43.0, 204.6]	6891; 1.05 [6132, 7041]	1.56; 2.00 [0.69, 3.46]
MAXN	OneMap	695.4; 1.07 [664.2, 758.7]	204.8; 2.17 [88.1, 527.0]	14,858; 1.02 [14,364, 15,337]	7.09; 2.16 [3.01, 17.77]
	Ours	167.3; 1.07 [153.8, 181.4]	74.8; 1.93 [35.0, 163.6]	7292; 1.04 [6956, 7762]	1.86; 2.00 [0.82, 4.10]

Note: Entries are GM; GSD [minimum, maximum] ($n = 9$), aggregating three episodes and three repetitions. GSD is the geometric standard deviation, calculated as the exponential of the sample standard deviation of the log-transformed measurements; it is a multiplicative, dimensionless dispersion factor. Dispersion includes episode workload and repeated-run variation. RAM is reported by `tegrastats`; energy uses Equation (8).

5.6. Closed-Loop Evaluation

The complete benchmark in Table 6 establishes the navigation quality, while the hardware study isolates the execution behavior on one clustered, one median-length, and one distributed multi-object episode selected from the 10th, 50th, and 90th percentiles of the annotated optimal distance. Figure 4 summarizes the complete episode duration speedup, and Table 7 reports the underlying absolute durations.

Both implementations succeed in the clustered and distributed episodes and fail in the median-length episode at every power mode. The two successful pairs use identical action counts and recorded path lengths, isolating the execution efficiency as the source of their duration differences. In the failed pair, OneMap terminates after 103 actions and ours after 104.

The distributed episode contains 601 observations and shows the largest speedup at every power mode, reaching 5.06× at 15 W and 3.22× in MAXN. Across all 36 paired comparisons, the geometric mean episode duration speedup is 3.55×. The result shows how repeated per-step savings accumulate during a sequential search: the benefit is present in short episodes and becomes larger when the robot must process more observations before termination.

5.7. Robustness Analysis

The following controlled tests examine whether the proposed execution mechanisms remain effective when map discretization, scheduler parameters, or visual input quality depart from the primary evaluation conditions. They vary the map resolution while holding the physical extent and feature dimension fixed, perturb each scheduler parameter around its selected value, and apply simulated directional motion blur to the scheduler input.

Table 8 reports a separate fixed-input Jetson MAXN experiment that separates the projection latency from the persistent map fusion cost as the map resolution increases. Across 118 measured updates at each of the five resolutions, the proposed projection remains between 30.88 and 35.42 ms, whereas OneMap's dense path remains between 183.11 and 185.86 ms. The proposed operator therefore retains a projection speedup between 5.17×

and 5.99× without a resolution-dependent latency increase. Both paths produce identical destination cell sets at every resolution.

Table 8. Resolution scaling on the Jetson AGX Orin in MAXN. OneMap denotes its dense projection path with FP16 persistent map storage matched to the proposed path. Values are means over 118 updates with fixed inputs after ten warm-up observations. Peak allocation is the maximum memory allocated by the measured CUDA process.

r (m)	Grid	Projection (ms)		Projection + Fusion (ms)		Peak Allocation (GiB)	
		OneMap	Ours	OneMap	Ours	OneMap	Ours
0.100	600^2	183.11	35.42	186.21	38.48	3.50	0.60
0.075	800^2	184.55	33.96	188.47	38.06	3.91	1.03
0.050	1200^2	184.91	30.91	191.21	36.75	5.10	2.30
0.030	2000^2	185.08	30.88	200.27	46.32	8.87	6.26
0.020	3000^2	185.86	32.46	227.84	76.86	16.20	13.85

Reducing r from 0.10 to 0.02 m increases the grid from 600^2 to 3000^2 cells. The FP16 semantic map allocation consequently rises from 0.515 to 12.875 GiB, as predicted by the shared $1/r^2$ capacity relation. The mean number of destination cells updated by one observation increases from 1109 to 18,625. Fusion therefore rises from 3.06 to 44.40 ms for the proposed path and from 3.11 to 41.98 ms for OneMap. This growth comes from shared indexed map fusion rather than the proposed interpolation operator. Even at 0.02 m, the proposed path remains 5.73× faster in projection and 2.96× faster for projection plus fusion. It also reduces peak allocation by 2.34 GiB.

Across the 25× increase in the map cell count, the aggregate useful traffic throughput rises modestly. The proposed path increases from 2.23 GB/s at 0.10 m to 2.58 GB/s at 0.02 m. The matched OneMap path increases from 2.19 to 2.73 GB/s. Each rate divides the total counted useful bytes by the total fusion time over 118 measured updates. Together with the flat projection latency, these values show no observed fine-resolution interpolation bandwidth cliff. The measured growth instead occurs in shared indexed fusion and persistent map capacity. The rates are conservative lower bounds that count one FP16 feature read, one FP32 update read, and one FP16 feature write per updated cell. They exclude cache transactions, indexing traffic, and other kernel accesses and therefore do not represent the achieved physical DRAM bandwidth.

Table 9 evaluates the scheduler around the calibrated operating point. The selected setting reaches a 62.50% SR and 68.75% progress on these episodes. Varying δ, the burst length, or the safety interval preserves both values and exactly matches the selected terminal states for all 16 episodes, although detector invocation changes from 44.14% to 68.04%. Lowering τ to 0.12 preserves the aggregate SR and progress but changes one terminal-state sequence and increases detector invocation to 65.49%. Raising τ to 0.20 reduces detector invocation to 44.10%, but the SR falls to 50.00% and progress to 60.42%. Thus, the scheduler is locally insensitive to the tested δ, burst, and safety settings, whereas an overly restrictive semantic threshold trades navigation quality for fewer detector calls.

The final robustness test examines the adaptive scheduler under matched visual corruption. Table 10 reports the fixed-trace analysis. Correlation with the clean evidence sequence falls from 0.959 under mild blur to 0.776 under severe blur. Under severe blur, the mean absolute score change is 0.0196. These perturbations move individual scheduled observations but have a limited effect on total detector use. The clean schedule invokes the detector on 295 of 601 observations (49.1%). Mild blur produces 292 calls, moderate blur 302, and severe blur 319. Even under the strongest corruption, the scheduler avoids 282 calls or 46.9% of the always-on workload.

Table 9. One-factor-at-a-time scheduler sensitivity on 16 path-length-stratified multi-object episodes. Each cell reports the low/high setting in the order shown. Detector use is the percentage of observations that invoke YOLO; it represents the computational demand rather than desktop latency. The calibrated values are $\tau = 0.16$, $\delta = 0.02$, $b = 2$, and safety interval 4.

Parameter	Low/High	Selected	SR (%)	Progress (%)	Detector Use (%)
τ	0.12/0.20	0.16	62.50/50.00	68.75/60.42	65.49/44.10
δ	0.01/0.03	0.02	62.50/62.50	68.75/68.75	67.06/46.39
Burst length b	1/3	2	62.50/62.50	68.75/68.75	44.55/58.30
Safety interval	2/8	4	62.50/62.50	68.75/68.75	68.04/44.14

Table 10. Fixed-trace stability of adaptive detection under simulated directional motion blur. Added and missed calls are measured relative to the clean-input schedule and denote detector invocation changes, not object classification false positives or false negatives. Recovery delay is measured from a missed clean-input call to the next blurred-input detector call.

Input	Kernel (Pixels)	Score Correlation	Added/Missed Calls	Detector Calls (%)	Calls Avoided vs. Always On	Recovery Mean/Max
Clean	0	1.000	0/0	295 (49.1)	306	-
Mild	7	0.959	43/46	292 (48.6)	309	1.85/3
Moderate	15	0.925	60/53	302 (50.2)	299	1.74/3
Severe	31	0.776	87/63	319 (53.1)	282	1.59/3

The added and missed calls show that blur produces schedule churn rather than invariance. Under severe blur, 87 calls are added and 63 clean-schedule calls are missed, yielding 24 net additional invocations; this is an 8.1% increase over the clean count. Every missed clean-schedule invocation is followed by a blurred-input detector call within three observations; the mean recovery remains below 1.85 observations at every blur level. This bound is enforced by the four-observation safety interval. Thus, changing visual evidence can modestly reduce the computational savings, but it cannot suppress detector execution indefinitely. The fixed trace deliberately isolates the scheduler response; motion blur can separately affect YOLO classification and the semantic map in both OneMap and the accelerated implementation.

6. Discussion

The results provide three findings about the hardware organization of persistent open-vocabulary navigation. First, performance depends on the complete perception–mapping–planning dataflow rather than neural inference alone. Second, changing the order in which semantic features are represented and moved can provide substantial gains without reducing their dimension or map resolution. Third, acceleration must be validated through closed-loop navigation because local numerical and scheduling changes can alter later observations. The following discussion develops these findings, examines the resulting efficiency–quality trade-off, quantifies the cost associated with persistent semantic capability, and considers their implications for edge deployment.

6.1. Cross-Layer Findings

The first finding is that persistent open-vocabulary navigation behaves as a heterogeneous dataflow workload. SED image encoding is the largest individual stage in OneMap, but semantic projection and frontier processing together consume almost one third of the complete step latency. Persistent map storage, processor transfers, and unconditional target detection add further repeated costs. The 4.64× geometric mean Jetson compute speedup consequently results from coordinated changes across dense perception, semantic projection, persistent mapping, and module scheduling. Accelerating the visual encoder alone would leave a substantial fraction of the repeated workload unchanged.

The second finding concerns the representation order. OneMap retains open-vocabulary evidence across viewpoints and object queries, which makes projection a high-dimensional and repeatedly executed operation. Expanding the low-resolution feature grid to depth resolution before map cell reduction creates an $H \times W \times d$ temporary whose cost is determined by the image area and feature dimension. Direct interpolation and accumulation into valid map cells removes this temporary while preserving the same feature dimension, map resolution, and confidence-weighted fusion rule. On the desktop, this dataflow provides a 2.34× speedup in the isolated fixed-input component benchmark and a 2.53× projection-stage speedup in the closed-loop profile. The separate fixed-trace Jetson ablation reports a 5.45–6.58× projection-stage speedup across power modes. Each result therefore quantifies the same operator under its stated execution context rather than a reduction in semantic representation capacity.

The resolution sweep sharpens this distinction. Increasing the grid density by 25× does not materially increase the projection latency for either path because the image and feature inputs remain fixed. The proposed operator preserves a projection advantage between 5.17× and 5.99× across the sweep. Finer discretization instead distributes each observation over more destination cells and enlarges the shared persistent state. The indexed fusion and map capacity therefore become dominant for both implementations. The proposed dataflow removes the intermediate at image resolution without adding a new interpolation bottleneck. At 0.02 m, the increased shared persistent map work reduces the complete projection-and-fusion advantage, but the proposed path remains faster and uses less memory.

The third finding is that embodied acceleration requires validation at the level at which observations and decisions interact. The integral image planner mask transformation preserves all 40 reference paths, whereas FP8 map storage changes the terminal outcome in the controlled test and is therefore excluded. Rejecting FP8 despite its 2× reduction in map storage demonstrates that the validation protocol prioritizes preserved navigation behavior over an isolated hardware gain. FP16 storage, TensorRT execution, and adaptive detection introduce smaller local changes that remain acceptable only after evaluation on the complete navigation benchmarks. These results establish a practical validation hierarchy: equivalence under fixed inputs is appropriate for isolated kernels, but the final operating point must be assessed through closed-loop task outcomes.

6.2. Efficiency and Navigation Quality

The reported operating point exchanges 4.15 percentage points of the single-object SR and 2.12 points of the multi-object SR for a 4.64× geometric mean Jetson compute speedup. FP16 feature storage changes the accumulated semantic similarities, TensorRT kernels introduce small numerical differences from PyTorch, and adaptive detection can defer target confirmation until a safety or semantic trigger occurs. In closed-loop navigation, these local changes can alter frontier ordering and generate a different sequence of future observations. The complete benchmark therefore provides the appropriate quality measure for the accelerated system.

Quality is retained most strongly on sequential multi-object search, the task for which persistent semantic memory is intended. The accelerated implementation retains 96.1% of the multi-object SR and progress, compared with 91.1% of the SPL and 92.7% of the PPL. This pattern indicates that semantic evidence remains effective for discovering successive targets, while altered frontier ordering can produce longer routes. The distinction matters in deployment: discovery metrics characterize whether the required objects are found, whereas path metrics expose the navigation cost of the changed execution path.

The precision and detector scheduling controls also define a family of operating points. A deployment that prioritizes quality can use the full detector schedule and FP32 semantic storage. A search robot with limited power can use mixed-precision storage and adaptive detection to increase the control frequency, release unified memory capacity, and reduce the energy to completion. The reported configuration represents the measured balance between these objectives rather than a fixed requirement of the acceleration design.

The sensitivity analysis identifies where this balance changes. The detector workload responds continuously to δ, the burst length, and the safety interval across the tested ranges without changing the terminal outcomes, while $\tau = 0.20$ crosses a quality boundary by suppressing target-relevant confirmation too aggressively. The calibrated $\tau = 0.16$ therefore retains semantic responsiveness while the remaining parameters regulate the detector workload within a locally stable quality region.

The motion blur stress test further separates semantic trigger stability from detector accuracy. Changing the blur direction perturbs the scalar evidence sequence and moves individual detector calls, yet the severe condition increases the total invocations by only 8.1% and still avoids 46.9% of the always-on workload. More importantly, the safety interval converts a suppressed semantic trigger into a bounded scheduling delay of at most three observations. The scheduler therefore degrades toward additional computation rather than unbounded detector omission as the feature stability decreases.

6.3. Cost of Persistent Semantics

The profile underlying Figure 3 provides a conservative decomposition of the cost associated with open-vocabulary capabilities. Dense perception, semantic projection, semantic map fusion, and map text similarity require 90.41 ms of OneMap's 111.01 ms step, or 81.5%. The corresponding stages require 29.88 ms of our 43.62 ms step, or 68.5%, representing a 3.03× reduction in the measured open-vocabulary portion. Frontier and POI processing is excluded from this subtotal because its combined timer contains both geometric exploration and semantic goal selection work.

This measured cost follows three nested capability layers. Geometric goal execution maintains navigability, computes a path to a known metric goal, and produces control actions. Frontier exploration adds goal selection for unknown space. Persistent open-vocabulary navigation then infers target locations from language and visual observations through dense encoding, target confirmation detection, semantic projection and fusion, map text retrieval, and semantic goal selection. OneMap and our implementation contain all three layers; this decomposition attributes the cost by added capability without treating a graph search kernel as functionally equivalent to language-conditioned navigation.

The persistent semantic state also explains the principal representation-level memory difference. At 600×600 cells and $d = 768$, OneMap's FP32 semantic tensor requires approximately 1055 MiB, while the proposed FP16 tensor requires approximately 527 MiB. One FP32 scalar grid of the same spatial size requires 1.37 MiB. The semantic tensor therefore requires 768× and 384× the storage of one scalar geometric layer, respectively. A complete geometric navigator can maintain several scalar layers, but their storage is independent of the language feature dimension d.

The component ablation locates the corresponding energy cost. Restoring dense projection increases trace energy by 88.5%, replacing TensorRT execution with PyTorch increases it by 19.2%, and executing detection continuously increases it by 12.5%. These results identify repeated semantic inference and feature movement as the main costs of persistent open-vocabulary capabilities and show how the proposed organization reduces them while retaining language-conditioned navigation.

6.4. Hardware Implications

The analysis in Section 3.7 separates two scaling constraints. Persistent semantic storage remains $\Theta(Nd)$, making the map area and feature dimension the principal capacity terms, whereas fused projection removes the $\Theta(Rd)$ per-observation intermediate. Adaptive detection changes the detector contribution from one invocation per observation to the observed invocation fraction ρ, while persistent buffers keep runtime memory independent of the episode length.

The power mode results show that software organization can be as important as the nominal power envelope. Accelerated execution at 15 W is faster than native execution at 50 W, and accelerated execution at 30 W is faster than native MAXN. Increasing clocks therefore does not compensate for an inefficient dataflow. The native system repeatedly pays for framework dispatch, dense temporary tensors, memory movement, and unconditional detector execution; these costs remain present even when more power is supplied. By reducing these costs, the accelerated implementation obtains higher performance while leaving thermal and electrical headroom for localization, communication, control, or additional perception workloads.

The 51.21% mean paired reduction in peak system RAM has comparable system-level significance. The Jetson AGX Orin uses a shared system-memory architecture for CPU and accelerator workloads [40]. Reducing measured use from approximately 14 GB to approximately 7 GB increases the capacity available to robot middleware, state estimation, control, sensor processing, and additional onboard models.

At the nominal throughput, D455 acquisition can overlap with the navigation actor with substantial timing and bandwidth headroom. The physical OneMap system uses an Intel RealSense D455, which supports 1280 × 720 depth and 1280 × 800 RGB at 30 Hz over USB 3.1 [3,42]. Assuming a 16-bit depth and 16-bit YUY2 RGB, these modes produce a 33.3 ms frame period and an approximately 117 MB/s uncompressed payload. The frame period is substantially shorter than the accelerated MAXN geometric mean compute time of 167.3 ms per step reported in Table 7, while the payload is well below the Jetson AGX Orin's 204.8 GB/s peak memory bandwidth [43]. Non-overlapped driver, localization, middleware, and actuation effects remain part of physical-system validation, as discussed in Section 6.6.

For long sequential searches, the energy to completion captures the interaction between power draw and the execution time more directly than instantaneous power alone. A navigation evaluation that considers the completion time similarly distinguishes the elapsed task time from the geometric path length [7]. At 15 W, the native and accelerated summed rail power are similar, yet the shorter accelerated episodes yield a mean paired reduction of 77.27% in summed rail energy. The per-step latency compounds over hundreds of observations, and the 601-step distributed episode produces the largest duration speedup and energy reduction.

These findings complement two established efficiency strategies. VL-Nav obtains a high operating frequency through a rolling representation and partial frontier processing [19]. Our results show that a cross-layer dataflow design can instead retain and accelerate a persistent dense representation. Adaptive systems such as ApproxDet, LiteReconfig, and RED regulate DNN workloads using content and resource conditions [22–24]. The proposed scheduler instead connects detector execution to the current navigation goal through current-view target-conditioned evidence. Representation design and navigation-aware execution control therefore offer complementary routes to efficient embodied inference.

6.5. Transferability

The evaluated visual representation is OneMap's SED encoder aligned with CLIP, but the mapping and scheduling mechanisms transfer through the paired image–text interface defined in Section 3.2. OpenCLIP provides open implementations and checkpoints spanning multiple contrastive image–text backbones, whereas SigLIP retains paired encoders but replaces CLIP's softmax contrastive objective with a pairwise sigmoid objective [44,45]. Either family can support fused projection, persistent storage, and target-conditioned scheduling after exposing dense spatial features. The feature dimension changes the map cost, similarity thresholds require recalibration, and each backbone requires a new TensorRT engine and numerical validation. The execution design is therefore interface-compatible with OpenCLIP and SigLIP, while the reported quality and speedup apply to the evaluated SED-CLIP configuration.

The OneMap pairing provides a controlled test of the proposed execution mechanisms by holding the semantic representation, navigation rules, benchmark inputs, and target hardware fixed. The component ablation consequently attributes the measured changes to TensorRT execution, fused projection, FP16 map storage, and adaptive detection, while Table 1 provides a published cross-system context rather than a direct ranking across unequal workloads. The numerical claims therefore describe within-architecture acceleration over the reproduced OneMap baseline. Applying the same matched protocol to a second complete edge-resident pipeline would test transfer to sparse, object-centric, or differently partitioned systems.

Precision transfer depends on the target hardware. On an FP8-native edge platform, quantization-aware or distillation-based co-design could train the dense encoder to preserve language-aligned features after recurrent FP8 map storage and fusion. Such an extension should validate the feature cosine similarity, accumulated map text similarity, and complete navigation outcomes rather than neural outputs alone. For AGX Orin, INT8 is the hardware-native precision below FP16 and is therefore the relevant target for future quantization-aware execution, although preserving signed language-aligned embeddings and confidence-weighted accumulation would require calibrated scaling and closed-loop validation.

6.6. Practical Applicability and Deployment Trade-Offs

6.6.1. Evidence and Deployment Boundary

Navigation quality is established on all 2236 HM3D benchmark episodes, whereas hardware efficiency is measured on one NVIDIA Jetson AGX Orin using three distance-stratified episodes repeated three times per implementation and power mode. Transfer to other edge GPUs will depend on compiler support and on the balance between arithmetic throughput and memory bandwidth [5,8].

The HIL protocol preserves the closed-loop interaction among observations, navigation decisions, and subsequent trajectories under repeatable conditions. The cadence and nominal transfer analysis in Section 6.4 shows that the reference D455 stream has sufficient throughput for the measured actor. The next integration level combines this actor with the RGB-D driver overhead, stream synchronization, pose estimation, depth artifacts, motion blur, actuator dynamics, and concurrent robot middleware. Accordingly, compute measurements quantify the embedded navigation actor, while the episode duration quantifies the complete evaluated HIL loop.

6.6.2. Environment and Sensing Envelope

The implementation retains OneMap's semantic representation, confidence-weighted map update, and navigation rules to isolate hardware acceleration. Its static-scene map ac-

cumulates evidence without temporal decay or explicit dynamic object tracking. Moving objects or displaced obstacles can therefore leave stale evidence after repeated high-confidence observations. In densely populated scenes, nearby objects with similar language-aligned features may contribute to the same or adjacent cells, increasing the ambiguity in similarity peaks, target association, and frontier scoring. The formulation uses calibrated RGB-D observations and sufficiently accurate poses; depth artifacts and pose drift can displace or blur projected semantic evidence. These properties define the operating envelope of OneMap's representation and remain unchanged by the numerically matched fused projection.

A changing-scene deployment could add fading without a full-map pass. The map would store the last update time ℓ_i for each cell and decay confidence only when that cell is revisited:

$$\bar{c}_i^{t-1} = \lambda^{\,t-\ell_i} c_i^{t-1}, \qquad 0 < \lambda < 1,$$

Equation (3) would then use $\bar{c}_i^{t-1}$ in place of c_i^{t-1}. This lazy cell-local operation touches only the U_t cells updated by the current observation. It preserves $\Theta(U_t d)$ update complexity and adds one scalar timestamp per map cell. A confidence cap could also guarantee a minimum gain for new evidence. Long-horizon retention and dynamic-scene responsiveness provide the validation criteria for either extension. The present study retains OneMap's original static-scene fusion to preserve controlled attribution of the acceleration results.

The inherited representation and sensing assumptions above are distinct from the deployment choices introduced by the accelerated implementation. Table 11 summarizes the practical role and design trade-off of each proposed mechanism.

Table 11. Practical deployment characteristics of the proposed acceleration mechanisms. The application contexts illustrate where each mechanism provides direct operational value; they are not additional experimental platforms.

Optimization Mechanism	Deployment Benefit	Representative Application Context	Design Trade-Off
Compiled mixed-precision perception	Reduces encoder and detector latency while retaining the pretrained task interface.	Battery-powered service and inspection robots with a fixed RGB-D perception stack.	Model, input shape, or platform changes require TensorRT engine rebuilding and numerical validation.
Fused direct-to-map projection	Removes the image-resolution feature intermediate.	Indoor inventory, object search, and facility inspection with semantic map updates at every observation.	Projection follows the camera geometry and map definition; at fine resolution, shared map fusion and capacity become dominant.
Persistent FP16 accelerator storage	Reduces map memory and repeated processor transfers.	Long multi-target searches in warehouses, hospitals, and offices that reuse mapped evidence.	The map occupies shared accelerator memory; encoder or fusion changes require renewed precision validation.
Compiled navigation operations	Reduces repeated interpreter, allocation, and data conversion overhead.	Large building exploration with frequent frontier extraction and path replanning.	Navigation logic or data layout changes can require recompilation and output equivalence testing.
Language-conditioned detector scheduling	Avoids confirmation inference when current-view target similarity does not justify it.	Search and retrieval where targets appear intermittently across rooms or warehouse aisles.	Encoder or domain changes require trigger calibration; brief visibility can delay confirmation within the safety bound.

At the system level, the complete framework is suited to edge-resident robots that perform prolonged sequential searches under memory, energy, and thermal constraints. Representative uses include service-robot search and retrieval, warehouse inventory, hospital and office assistance, and facility inspection. Its modular mechanisms also support

deployment-specific operating points. Resource-constrained missions can combine FP16 map storage with adaptive detection, whereas applications requiring frame-by-frame target confirmation can retain continuous detection while still using compiled perception, fused projection, persistent storage, and compiled navigation operations.

Deploying a different RGB-D camera entails engine rebuilding and projection reconfiguration from its intrinsics, depth registration, pose convention, and selected map resolution. Monocular RGB, LiDAR, fisheye, and asynchronous sensors require a corresponding feature-to-geometry interface. Temporal map decay, dynamic object tracking, instance-aware storage, batched mask processing, and cross-sensor validation extend the framework to operating conditions beyond the retained OneMap formulation.

6.6.3. Measurement Scope

The peak RAM and summed rail energy use `tegrastats` board telemetry for consistent system-level accounting [38]. External wall-power measurement and process-isolated memory profiling form the next attribution level for an integrated robot. The reproduced OneMap and accelerated quality evaluations share the same simulator, dependency versions, checkpoints, benchmark inputs, and metric implementation. Component ablations, fixed-trace tests, and Jetson measurements likewise execute the OneMap reference and proposed paths under matched inputs and profiling conditions.

A future evaluation will integrate the pipeline with onboard RGB-D sensing and localization. This platform will measure acquisition, transfer, pose estimation, actuation, task success, the elapsed duration, and the external wall power together. Additional Jetson generations and a second edge-resident architecture will test hardware and architectural transfer. A later runtime controller could coordinate precision, detector scheduling, and the board power mode from semantic evidence and the hardware state. Every operating point would retain navigation-level validation.

7. Conclusions

This study shows that persistent open-vocabulary navigation is fundamentally a heterogeneous perception–mapping–planning workload. Foundation-model inference is its largest individual cost, but semantic projection, persistent state, memory movement, frontier processing, and repeated detection collectively determine whether the complete system can operate efficiently on edge hardware. We address these costs through compiled mixed-precision perception, fused direct-to-map aggregation, persistent accelerator storage, compiled navigation operations, and language-conditioned detector scheduling.

On the complete HM3D benchmarks, comprising 2236 episodes, the accelerated implementation retains 92.6% and 96.1% of the reproduced OneMap single- and multi-object success rates, respectively. These complete-benchmark results establish navigation quality.

Hardware efficiency is evaluated separately using three distance-stratified multi-object episodes across four Jetson AGX Orin power modes. Each implementation runs three repetitions per condition. Across 36 paired comparisons, the implementation achieves a 4.64× compute speedup and a 3.55× episode duration speedup. It also reduces the peak system RAM by 51.21% and the summed rail energy by 75.32% on average while preserving every paired outcome. These magnitudes characterize the evaluated episodes, power modes, software configuration, and AGX Orin platform. Within these conditions, accelerated execution at 15 W exceeds native execution at 50 W. Accelerated 30 W also exceeds native MAXN. Dataflow organization can therefore provide a greater benefit than raising the power envelope of an inefficient execution path.

The broader design principle is that embodied AI acceleration must preserve and move task evidence efficiently across the complete closed loop. Optimizing neural models in

isolation leaves substantial mapping and control costs unresolved, while validating kernels in isolation misses the trajectories produced by their numerical and scheduling effects. Hardware-aware navigation should therefore optimize representation, data movement, and module execution jointly and evaluate these choices through the task behavior that they produce.

Author Contributions: Conceptualization, M.C.A. and H.L.; methodology, M.C.A.; software, M.C.A.; validation, M.C.A.; formal analysis, M.C.A.; investigation, M.C.A.; resources, H.L.; data curation, M.C.A.; writing (original draft preparation), M.C.A.; writing (review and editing), M.C.A. and H.L.; visualization, M.C.A.; supervision, H.L.; project administration, H.L.; funding acquisition, H.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2026-RS-2024-00437190, 50%) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation), and in part by the Regional Innovation System & Education (RISE), Regional Growth Innovation LAB program through the Gyeongbuk RISE Center, funded by the Ministry of Education (MOE) and Gyeongsangbuk-do, Republic of Korea (2026-rise-15-105).

Data Availability Statement: The Habitat-Matterport 3D datasets analyzed in this study are available through the Habitat platform, subject to their respective terms of use. The implementation and configuration files supporting the findings of this study are available from the corresponding author upon reasonable request and will be released publicly upon publication.

Acknowledgments: During the preparation of this work, the authors used ChatGPT (version 5.6, OpenAI) to support language refinement, clarity, and the organization of the text. All scientific content, analyses, interpretations, and conclusions were conceived, verified, and finalized by the authors. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of the data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial intelligence
CPU	Central processing unit
CUDA	Compute Unified Device Architecture
FP8/16/32	8-/16-/32-bit floating point
GPU	Graphics processing unit
HIL	Hardware-in-the-loop
HM3D	Habitat-Matterport 3D
INT8	8-bit integer
NMS	Non-maximum suppression
POI	Point of interest
PPL	Progress weighted by path length
RAM	Random access memory
RGB-D	Red–green–blue and depth
SAM	Segment Anything Model
SED	Simple Encoder–Decoder for Open-Vocabulary Semantic Segmentation
SPL	Success weighted by path length
SR	Success rate
VLM	Vision–language model
YOLO	You Only Look Once

References

1. Gadre, S.Y.; Wortsman, M.; Ilharco, G.; Schmidt, L.; Song, S. CoWs on Pasture: Baselines and Benchmarks for Language-Driven Zero-Shot Object Navigation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 17–24 June 2023; pp. 23171–23181. [CrossRef]
2. Yokoyama, N.; Ha, S.; Batra, D.; Wang, J.; Bucher, B. VLFM: Vision-Language Frontier Maps for Zero-Shot Semantic Navigation. In Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA), Yokohama, Japan, 13–17 May 2024; pp. 42–48. [CrossRef]
3. Busch, F.L.; Homberger, T.; Ortega-Peimbert, J.; Yang, Q.; Andersson, O. One Map to Find Them All: Real-Time Open-Vocabulary Mapping for Zero-Shot Multi-Object Navigation. In Proceedings of the 2025 IEEE International Conference on Robotics and Automation (ICRA), Atlanta, GA, USA, 19–23 May 2025; pp. 14835–14842. [CrossRef]
4. Mayoral-Vilches, V.; Jabbour, J.; Hsiao, Y.-S.; Wan, Z.; Crespo-Álvarez, M.; Stewart, M.; Reina-Muñoz, J.M.; Nagras, P.; Vikhe, G.; Bakhshalipour, M.; et al. RobotPerf: An Open-Source, Vendor-Agnostic, Benchmarking Suite for Evaluating Robotics Computing System Performance. In Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA), Yokohama, Japan, 13–17 May 2024; pp. 8288–8297. [CrossRef]
5. Williams, S.; Waterman, A.; Patterson, D. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* **2009**, *52*, 65–76. [CrossRef]
6. Chen, Y.-H.; Krishna, T.; Emer, J.S.; Sze, V. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE J. Solid-State Circuits* **2017**, *52*, 127–138. [CrossRef]
7. Yokoyama, N.; Ha, S.; Batra, D. Success Weighted by Completion Time: A Dynamics-Aware Evaluation Criteria for Embodied Navigation. In Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Prague, Czech Republic, 27 September–1 October 2021; pp. 1562–1569. [CrossRef]
8. Chen, T.; Moreau, T.; Jiang, Z.; Zheng, L.; Yan, E.; Shen, H.; Cowan, M.; Wang, L.; Hu, Y.; Ceze, L.; et al. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Carlsbad, CA, USA, 8–10 October 2018; pp. 578–594.
9. Micikevicius, P.; Narang, S.; Alben, J.; Diamos, G.; Elsen, E.; Garcia, D.; Ginsburg, B.; Houston, M.; Kuchaiev, O.; Venkatesh, G.; et al. Mixed Precision Training. In Proceedings of the International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, 30 April–3 May 2018.
10. Wang, K.; Liu, Z.; Lin, Y.; Lin, J.; Han, S. HAQ: Hardware-Aware Automated Quantization with Mixed Precision. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 16–20 June 2019; pp. 8604–8612. [CrossRef]
11. Yang, T.-J.; Howard, A.; Chen, B.; Zhang, X.; Go, A.; Sandler, M.; Sze, V.; Adam, H. NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications. In *Computer Vision-ECCV 2018*; Springer: Cham, Switzerland, 2018; pp. 289–304. [CrossRef]
12. Huang, C.; Mees, O.; Zeng, A.; Burgard, W. Visual Language Maps for Robot Navigation. In Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA), London, UK, 29 May–2 June 2023; pp. 10608–10615. [CrossRef]
13. Peng, S.; Genova, K.; Jiang, C.; Tagliasacchi, A.; Pollefeys, M.; Funkhouser, T. OpenScene: 3D Scene Understanding with Open Vocabularies. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 17–24 June 2023; pp. 815–824. [CrossRef]
14. Jatavallabhula, K.M.; Kuwajerwala, A.; Gu, Q.; Omama, M.; Iyer, G.; Saryazdi, S.; Chen, T.; Maalouf, A.; Li, S.; Keetha, N.; et al. ConceptFusion: Open-Set Multimodal 3D Mapping. In Proceedings of the Robotics: Science and Systems XIX, Daegu, Republic of Korea, 10–14 July 2023. [CrossRef]
15. Yamazaki, K.; Hanyu, T.; Vo, K.; Pham, T.; Tran, M.; Doretto, G.; Nguyen, A.; Le, N. Open-Fusion: Real-Time Open-Vocabulary 3D Mapping and Queryable Scene Representation. In Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA), Yokohama, Japan, 13–17 May 2024; pp. 9411–9417. [CrossRef]
16. Gu, Q.; Kuwajerwala, A.; Morin, S.; Jatavallabhula, K.M.; Sen, B.; Agarwal, A.; Rivera, C.; Paul, W.; Ellis, K.; Chellappa, R.; et al. ConceptGraphs: Open-Vocabulary 3D Scene Graphs for Perception and Planning. In Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA), Yokohama, Japan, 13–17 May 2024; pp. 5021–5028. [CrossRef]
17. Khanna, M.; Ramrakhya, R.; Chhablani, G.; Yenamandra, S.; Gervet, T.; Chang, M.; Kira, Z.; Chaplot, D.S.; Batra, D.; Mottaghi, R. GOAT-Bench: A Benchmark for Multi-Modal Lifelong Navigation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 16–22 June 2024; pp. 16373–16383.
18. Ortega-Peimbert, J.; Busch, F.L.; Homberger, T.; Yang, Q.; Andersson, O. DIV-Nav: Open-Vocabulary Spatial Relationships for Multi-Object Navigation. *arXiv* **2026**, arXiv:2510.16518.
19. Du, Y.; Fu, T.; Zhao, Z.; Su, S.; Zhan, Z.; Du, Q.; Chen, Z.; Li, B.; Wang, C. VL-Nav: Neuro-Symbolic Reasoning-Based Vision-Language Navigation. *arXiv* **2025**, arXiv:2502.00931.
20. Huang, J.; Huang, G. BEVPoolv2: A Cutting-Edge Implementation of BEVDet Toward Deployment. *arXiv* **2022**, arXiv:2211.17111.

21. Zhou, H.; Ge, Z.; Li, Z.; Zhang, X. MatrixVT: Efficient Multi-Camera to BEV Transformation for 3D Perception. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Paris, France, 1–6 October 2023; pp. 8548–8557. [CrossRef]
22. Xu, R.; Zhang, C.-L.; Wang, P.; Lee, J.; Mitra, S.; Chaterji, S.; Li, Y.; Bagchi, S. ApproxDet: Content and Contention-Aware Approximate Object Detection for Mobiles. In Proceedings of the 18th ACM Conference on Embedded Networked Sensor Systems (SenSys), Virtual Event, 16–19 November 2020; pp. 449–462. [CrossRef]
23. Xu, R.; Lee, J.; Wang, P.; Bagchi, S.; Li, Y.; Chaterji, S. LiteReconfig: Cost and content aware reconfiguration of video object detection systems for mobile GPUs. In Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys), Rennes, France, 5–8 April 2022; pp. 334–351. [CrossRef]
24. Li, Z.; Ren, T.; He, X.; Liu, C. RED: A Systematic Real-Time Scheduling Approach for Robotic Environmental Dynamics. In Proceedings of the 2023 IEEE Real-Time Systems Symposium (RTSS), Taipei, Taiwan, 5–8 December 2023; pp. 210–223. [CrossRef]
25. Yang, Z.; Zheng, S.; Xie, T.; Xu, T.; Yu, B.; Wang, F.; Tang, J.; Liu, S.; Li, M. EfficientNav: Towards On-Device Object-Goal Navigation with Navigation Map Caching and Retrieval. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2025; Volume 38.
26. Barbas Laina, S.; Boche, S.; Papatheodorou, S.; Schaefer, S.; Jung, J.; Oleynikova, H.; Leutenegger, S. FindAnything: Open-Vocabulary and Object-Centric Mapping for Robot Exploration in Any Environment. *arXiv* **2026**, arXiv:2504.08603.
27. Singh, R.; Ray, D.; Smith, C.; Adve, S. SemanticXR: Low Power and Real-Time Queryable Semantic Mapping with an Object-Level Device-Cloud Architecture. *arXiv* **2026**, arXiv:2606.12849.
28. Xie, B.; Cao, J.; Xie, J.; Khan, F.S.; Pang, Y. SED: A Simple Encoder-Decoder for Open-Vocabulary Semantic Segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 16–22 June 2024; pp. 3426–3436. [CrossRef]
29. Radford, A.; Kim, J.W.; Hallacy, C.; Ramesh, A.; Goh, G.; Agarwal, S.; Sastry, G.; Askell, A.; Mishkin, P.; Clark, J.; et al. Learning Transferable Visual Models from Natural Language Supervision. In Proceedings of the 38th International Conference on Machine Learning (ICML), Virtual Event, 18–24 July 2021; pp. 8748–8763.
30. Wang, C.-Y.; Bochkovskiy, A.; Liao, H.-Y.M. YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 17–24 June 2023; pp. 7464–7475. [CrossRef]
31. Zhang, C.; Han, D.; Qiao, Y.; Kim, J.U.; Bae, S.-H.; Lee, S.; Hong, C.S. Faster Segment Anything: Towards Lightweight SAM for Mobile Applications. *arXiv* **2023**, arXiv:2306.14289.
32. NVIDIA Corporation. NVIDIA TensorRT Documentation. Available online: https://docs.nvidia.com/deeplearning/tensorrt/latest/ (accessed on 25 July 2026).
33. Tillet, P.; Kung, H.T.; Cox, D. Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations. In Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL), Phoenix, AZ, USA, 22 June 2019; pp. 10–19. [CrossRef]
34. Savva, M.; Kadian, A.; Maksymets, O.; Zhao, Y.; Wijmans, E.; Jain, B.; Straub, J.; Liu, J.; Koltun, V.; Malik, J.; et al. Habitat: A Platform for Embodied AI Research. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019; pp. 9339–9347. [CrossRef]
35. Ramakrishnan, S.K.; Gokaslan, A.; Wijmans, E.; Maksymets, O.; Clegg, A.; Turner, J.; Undersander, E.; Galuba, W.; Westbury, A.; Chang, A.X.; et al. Habitat-Matterport 3D Dataset (HM3D): 1000 Large-Scale 3D Environments for Embodied AI. In Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks, Virtual Event, 6–14 December 2021; Volume 1.
36. Batra, D.; Gokaslan, A.; Kembhavi, A.; Maksymets, O.; Mottaghi, R.; Savva, M.; Toshev, A.; Wijmans, E. ObjectNav Revisited: On Evaluation of Embodied Agents Navigating to Objects. *arXiv* **2020**, arXiv:2006.13171.
37. Anderson, P.; Chang, A.; Chaplot, D.S.; Dosovitskiy, A.; Gupta, S.; Koltun, V.; Kosecka, J.; Malik, J.; Mottaghi, R.; Savva, M.; et al. On Evaluation of Embodied Navigation Agents. *arXiv* **2018**, arXiv:1807.06757.
38. NVIDIA Corporation. Tegrastats Utility. In *Jetson Linux Developer Guide*; Release 36.4.4; NVIDIA: Santa Clara, CA, USA, 2025. Available online: https://docs.nvidia.com/jetson/archives/r36.4.4/DeveloperGuide/AT/JetsonLinuxDevelopmentTools/TegrastatsUtility.html (accessed on 1 September 2026).
39. NVIDIA Corporation. NVIDIA TensorRT Support Matrix. Available online: https://docs.nvidia.com/deeplearning/tensorrt/latest/getting-started/support-matrix.html (accessed on 1 September 2026).
40. NVIDIA Corporation. Platform Power and Performance. In *Jetson Linux Developer Guide*; Release 36.4.4; NVIDIA: Santa Clara, CA, USA, 2025. Available online: https://docs.nvidia.com/jetson/archives/r36.4.4/DeveloperGuide/SD/PlatformPowerAndPerformance.html (accessed on 1 September 2026).
41. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018; pp. 2704–2713. [CrossRef]

42. RealSense, Inc. *RealSense Product Family D400 Series Datasheet*; Revision 023; Document No. 337029-017; RealSense, Inc.: Cupertino, CA, USA, 2026. Available online: https://www.realsenseai.com/wp-content/uploads/2026/03/RealSense-D400-Series-Datasheet-Mar-2026.pdf (accessed on 1 September 2026).
43. Karumbunathan, L.S. *NVIDIA Jetson AGX Orin Series: A Giant Leap Forward for Robotics and Edge AI Applications*; Technical Brief v1.2; Document No. TB_10749-001_v1.2; NVIDIA Corporation: Santa Clara, CA, USA, 2022. Available online: https://www.nvidia.com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf (accessed on 1 September 2026).
44. Cherti, M.; Beaumont, R.; Wightman, R.; Wortsman, M.; Ilharco, G.; Gordon, C.; Schuhmann, C.; Schmidt, L.; Jitsev, J. Reproducible Scaling Laws for Contrastive Language-Image Learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 17–24 June 2023; pp. 2818–2829. [CrossRef]
45. Zhai, X.; Mustafa, B.; Kolesnikov, A.; Beyer, L. Sigmoid Loss for Language Image Pre-Training. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Paris, France, 2–6 October 2023; pp. 11975–11986.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.