NeuSLAM: Dense Visual SLAM on Edge Devices

Aniket Gupta[1], Tianye Ding[1], Ajay Rajendra Kumar[1], Dennis Giaya[1], Pia Bideau[2], Charles Saunders[2], Qu Cao[2], Aruni RoyChowdhury[2], Hanumant Singh[1], and Huaizu Jiang[1]

[1]*Northeastern University* [2]*MathWorks*

***Abstract*— We present NeuSLAM, a hybrid architecture for dense visual SLAM designed specifically for stereo and RGB-D sensors on resource-constrained edge devices. While recent learning-based dense SLAM methods achieve strong trajectory accuracy and rich scene reconstruction, their learned back-ends typically maintain dense correlation volumes and feature maps on the GPU throughout optimization. This requires several gigabytes of memory, restricting them to desktop-grade hardware. To overcome this bottleneck, we introduce a lightweight neural network extending NeuFlow-V2, that jointly predicts dense optical flow and stereo disparity alongside per-pixel confidence maps via a shared feature encoder. For RGB-D setups, the disparity branch is simply bypassed in favor of sensor depth. Our front-end isolates high-confidence sparse correspondences for camera pose estimation, while a classical back-end maintains a lightweight pose graph over keyframes to ensure global consistency and keep GPU memory usage minimal. In zero-shot synthetic-to-real evaluations, our network achieves a state-of-the-art F1-all score of 9.9% for optical flow on KITTI 2015, alongside competitive stereo disparity on both KITTI 2015 and ETH3D. On TUM RGB-D and FLSea-VI datasets, NeuSLAM achieves trajectory accuracy competitive with DROID-SLAM while running 6.5× faster and consuming 87% less GPU memory. Our model and code will be open sourced.**

I. Introduction

Simultaneous Localization and Mapping (SLAM) remains the backbone of autonomous robotic systems, yet deploying dense SLAM on edge hardware presents a fundamental, unsolved and important problem. On one side of the methodology spectrum, classical feature-based systems [1]–[4] achieve real-time performance on resource-constrained platforms by operating on *sparse* keypoint representations on CPUs. Efforts like cuVSLAM [5] have further ported sparse feature tracking to embedded GPUs, achieving faster inference speed. However, the sparse maps produced by these systems offer limited utility for downstream tasks that require dense scene understanding, such as obstacle avoidance, path planning through cluttered spaces, or object-level reasoning. On the other end of the spectrum, learning-based *dense* SLAM methods [6]–[8] achieve high-fidelity reconstructions and strong trajectory accuracy, but their network architectures demand several gigabytes of VRAM and desktop-grade GPUs. This leaves dense, learning-based SLAM largely impractical for the embedded robotic platforms where it is most needed.

Dense, reliable correspondences are the foundation of dense SLAM, driving both accurate pose estimation and detailed scene reconstruction. Recent systems such as DROID-SLAM [6] and MASt3R-SLAM [7] achieve state-of-the-art

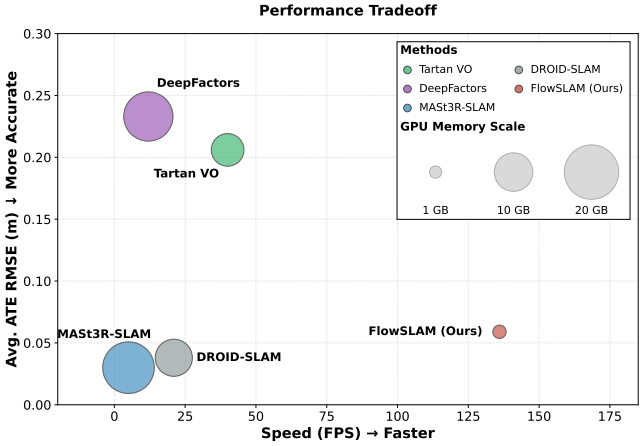

Fig. 1: **Speed, Accuracy and Memory Efficiency of NeuSLAM:** Compared to dense SLAM baselines on TUM-RGBD, our hybrid architecture provides a superior balance of trajectory accuracy and computation cost. When evaluated on a standard desktop GPU (RTX 3090), NeuSLAM demonstrates an exceptionally low memory footprint and high frame rate compared to existing methods. This fundamental efficiency allows our robust tracking capabilities to successfully scale down to resource-constrained edge hardware, achieving a near real-time 10 FPS on a Jetson Orin Nano. Note that we present results here on the desktop GPU because most dense SLAM architectures run out of memory on the Jetson Orin Nano.

results by coupling powerful learned correspondence front-ends with differentiable optimization back-ends that jointly refine poses and dense geometry. However, these back-ends must maintain and update a large number of map points in GPU memory, making them the primary source of the prohibitive memory and compute requirements discussed above. This motivates our design: we resort to a lightweight neural network that produces high-quality dense correspondences along with their confidence at edge-compatible speeds. A front-end then selects only high-confidence sparse correspondences for camera pose estimation. Unlike learned back-ends that maintain dense correlation volumes and feature maps on the GPU throughout optimization, our back-end operates on a lightweight pose graph over keyframes for global consistency, keeping GPU memory usage minimal.

Building on this insight, we present **NeuSLAM**, a *hybrid* architecture that pairs an efficient learned front-end for

dense optical flow and stereo disparity estimation with a classical geometric back-end for pose estimation and global consistency. For the front-end, we extend NeuFlow-V2 [9], a highly efficient optical flow architecture designed for edge devices, to jointly predict both optical flow and stereo disparity, along with per-pixel confidence maps. When an RGB-D sensor is available, the learned disparity branch can be replaced by sensor depth, further reducing inference time while retaining the flow-based tracking pipeline. We use the predicted confidence maps to select geometrically reliable correspondences for pose estimation, while the backend utilizes an iSAM2-based [10] pose graph optimization for global trajectory consistency, supplemented by DBoW3 [11]-based loop closure with geometric verification. By design, NeuSLAM treats deployment as the defining constraint. Every architectural decision, from the choice of shallow CNN backbones over Transformers, to scanline matching over global attention for stereo, to pose graph optimization over dense bundle adjustment, is guided by the edge-device memory and compute (latency) budget.

We evaluate NeuSLAM on standard benchmarks for both its learned front-end and the full SLAM pipeline. For optical flow, our front-end network achieves the best KITTI2015 [12] F1-all score (9.9%) among all methods tested in zero-shot evaluation. For stereo disparity, it achieves competitive zero-shot results on KITTI2015 and ETH3D [13], with over $10\times$ faster inference speed than all baselines. Both tasks run at around 85 FPS on the RTX 3090. With simple TensorRT [14] compilation, the inference speed can go up to 166 FPS. For SLAM, we evaluate on TUM RGB-D [15] (indoor) and FLSea [16] Dataset. On both datasets, NeuSLAM achieves comparable accuracy to DROID-SLAM, while running $6.5\times$ faster and consuming 87% less GPU memory. Critically, DROID-SLAM cannot run on the Orin Nano at all, while NeuSLAM operates at roughly 18 FPS for these datasets.

In summary, our contributions are:

- A hybrid SLAM method, NeuSLAM, that achieves dense mapping on edge devices by combining an efficient neural front-end with classical geometric estimation and optimization.
- A joint optical flow and stereo disparity module, NeuSLAM-Net, that extends NeuFlow-V2 with shared encoding and per-task confidence prediction, achieving state-of-the-art zero-shot accuracy at edge-compatible speeds.
- Extensive evaluation on optical flow, stereo disparity and SLAM benchmarks showing that both our frontend model and SLAM pipeline achieves accuracy competitive with state-of-the-art methods at a fraction of the computational cost.

II. Related Work

Dense Visual SLAM. Visual SLAM methods are commonly categorized by how they establish correspondences. Feature-based methods [1], [2], [17], [18] extract and match sparse keypoints across frames, then optimize reprojection error to recover camera poses. These methods are computationally efficient and run comfortably on CPUs, but their sparse maps provide limited information for downstream tasks such as path planning and collision avoidance [19]. Direct methods [20]–[22] bypass feature extraction and instead minimize photometric error over raw pixel intensities. While they can exploit more image information than keypoint-based approaches, their reconstructions remain sparse or semi-dense.

Recent learning-based methods push dense SLAM further. DROID-SLAM [6] builds RAFT-derived [23] correlation volumes and applies differentiable bundle adjustment to jointly refine poses and dense depth. DPVO [24] and DPV-SLAM [25] show that sparse patch-based correspondences can match dense flow accuracy at lower cost. MASt3R-SLAM [7] and GRS-SLAM3R [26] leverage learned 3D reconstruction priors for dense matching which are robust to textureless regions, while VGGT-SLAM [8] aligns feed-forward dense reconstructions via projective optimization on the SL(4) manifold. All of these methods require desktop-grade GPUs with multiple gigabytes of VRAM, leaving dense visual SLAM on edge hardware an open problem.

SLAM on Edge Devices. Traditional feature-based SLAM systems [1], [2], [17], [20] run on CPUs and have been deployed on embedded platforms, but cannot leverage the GPU compute available on modern edge devices. cuVSLAM and Jetson-SLAM [5], [27] port sparse feature tracking to embedded GPUs, achieving high throughput on Jetson hardware but producing only sparse maps. Among learning-based methods, DPVO [24] and DPV-SLAM [25] significantly reduce memory relative to DROID-SLAM, but their 7–9 GB VRAM footprint still exceeds the capacity of low-power GPUs such as the Jetson Orin Nano. To address resource constraints, EdgeSLAM and EdgeSLAM2 [28], [29] propose a decoupled architecture that offloads global optimization to an edge server, at the cost of network dependency and periodic map synchronization. To date, no existing dense visual SLAM system has been demonstrated on the Jetson Orin Nano (8 GB VRAM), the most resource-constrained member of the Orin family.

Learned Optical Flow and Stereo for Geometry. Dense correspondences are a core building block for visual SLAM, and learning-based methods have dramatically improved their quality. RAFT [23] introduced iterative refinement over 4D correlation volumes, achieving strong accuracy but at significant computational cost. Subsequent work on optical flow estimation has improved both accuracy [30]–[35] and efficiency [36], yet most methods remain too expensive for edge deployment. NeuFlow [37] and NeuFlow-V2 [9] achieve 10–70$\times$ speedups over state-of-the-art flow methods while maintaining competitive accuracy, running at over 20 FPS on the Jetson Orin Nano.

For stereo disparity, RAFT-Stereo [38] adapts RAFT's iterative refinement to epipolar-constrained matching, and numerous methods [39]–[43] have advanced accuracy on standard benchmarks. SENSE [44] demonstrated that a single shared encoder can serve both optical flow and stereo

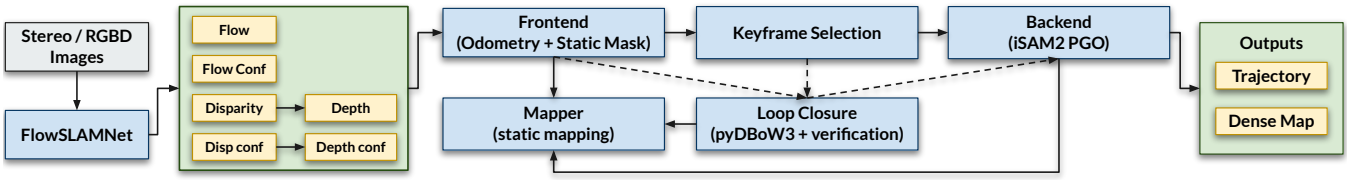

Fig. 2: **NeuSLAM Pipeline:** Solid arrows denote the per-frame tracking and optimization path; dashed arrows denote asynchronous loop-closure processing. The back-end performs pose-graph optimization (PGO) and provides optimized poses to mapping and to final exports; maps are exported/visualized in the optimized frame via pose-consistent correction.

disparity without sacrificing per-task accuracy, while reducing overall computation through multi-task feature sharing. Several SLAM systems use learned flow as a front-end component: DROID-SLAM [6] derives its correspondence module from RAFT, and FlowFusion [45] directly consumes optical flow obtained from PWC-Net [46] for dense RGB-D tracking. However, no existing system integrates an efficient shared-encoder flow-and-stereo network into a SLAM pipeline targeting edge deployment, which is the fundamental problem we pursue in this work.

III. Method

A. System Overview

The NeuSLAM pipeline is illustrated in Fig. 2. Given stereo or RGB-D input, a learned network, NeuSLAM-Net (§III-B), jointly predicts dense optical flow, depth, and per-pixel confidence maps. A tracking front-end (§III-C) uses these predictions to estimate per-frame camera poses via confidence-aware PnP-RANSAC and produces a static mask by checking reprojection consistency. Keyframes are selected based on accumulated motion and passed to a loop closure module that retrieves candidates via visual place recognition and verifies them geometrically using the static depth. The back-end (§III-D) jointly optimizes accepted loop constraints and odometry edges in an iSAM2-based [10] pose graph. A mapper incrementally builds a dense point cloud from static pixels, with per-keyframe rigid corrections applied to maintain consistency with the optimized trajectory.

B. Dense Correspondence Estimation

NeuSLAM-Net jointly estimates dense optical flow and stereo disparity with per-pixel confidence maps from three input images: the stereo pair I_L^t, I_R^t at time t and the left image I_L^{t+1} at time $t{+}1$. The architecture, illustrated in Fig. 3, follows a two-branch design built on top of NeuFlow-V2 [9]. Following the shared-encoder principle of [44], a single feature encoder extracts multi-scale features at 1/8 and 1/16 resolution from all three input images, serving both the optical flow and stereo disparity branches. A separate context encoder processes I_L^t alone to produce context features that guide the convex upsampling. When an RGB-D sensor is available, the stereo disparity branch is bypassed and sensory depth is used directly.

Optical flow branch. Features from I_L^t and I_L^{t+1} are paired for global matching at 1/16 resolution to capture large displacements, then iteratively refined at 1/16 and 1/8 resolution. This branch follows the same global-to-local design as NeuFlow-V2 [9]. At each iteration of the 1/8 refinement stage, an MLP head predicts a per-pixel confidence map from the iteratively updated context features.

Stereo disparity branch. Features from I_L^t and I_R^t follow the same refinement pipeline, with one key difference: the global matching module is replaced by scanline matching at 1/16 resolution. Since stereo correspondences lie along horizontal epipolar lines, restricting the search to scanlines reduces computation while providing a stronger geometric prior. The refinement stages and confidence head are identical in structure to the flow branch.

Training. NeuSLAM-Net is trained with synthetic datasets only, including Sceneflow [47], WMGStereo [48], and Virtual KITTI 2 [49]. For datasets that provide only disparity labels, we generate pseudo optical flow by negating the disparity values as horizontal displacement, setting vertical displacement to zero, and applying a small random homography to simulate vertical motion. This enables joint training of both branches from disparity-only datasets.

The training loss for each refinement iteration i is

$$\mathcal{L}_i = \mathcal{L}_{\text{disp}} + \mathcal{L}_{\text{flow}} + \lambda_{\text{conf}}\,\mathcal{L}_{\text{conf}} + \lambda_{\text{grad}}\,\mathcal{L}_{\text{grad}}\,, \qquad (1)$$

where $\mathcal{L}_{\text{disp}}$ and $\mathcal{L}_{\text{flow}}$ are ℓ_1 losses against ground-truth disparity and flow, $\mathcal{L}_{\text{conf}}$ is the sum of binary cross-entropy losses for the disparity and flow confidence maps (with targets set to 1 if the corresponding ℓ_1 error is below a threshold and 0 otherwise), and $\mathcal{L}_{\text{grad}}$ is a gradient loss [50] on predicted disparity that preserves sharp depth edges while encouraging smoothness in planar regions. We set $\lambda_{\text{conf}} = 0.5$ and $\lambda_{\text{grad}} = 0.5$. The total loss aggregates over N refinement iterations with exponentially increasing weights:

$$\mathcal{L} = \sum_{i=1}^{N} \gamma^{N-i}\,\mathcal{L}_i\,, \qquad (2)$$

with $\gamma = 0.9$.

C. Confidence-Aware Tracking

Given the predicted depth $\mathbf{D}_t$, optical flow $\mathbf{F}_{t\to t+1}$, and per-pixel confidence maps from FlowSLAMNet, the tracking module estimates the relative camera pose and produces a static mask for downstream mapping and loop closure. We represent camera poses as rigid transforms $T_{c_t\to w} \in SE(3)$ mapping points from camera frame c_t to the world frame w.

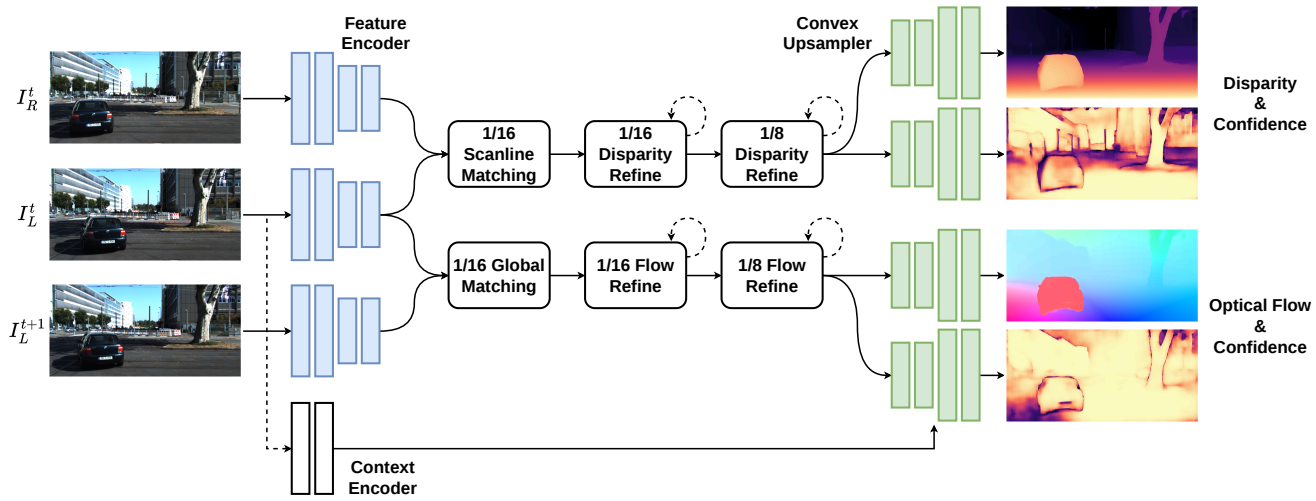

Fig. 3: NeuSLAM-Net takes in a stereo image pair at timestep t, and the left image from timestep $t+1$ to produce optical flow, flow confidence, stereo disparity and disparity confidence. We use a shared shallow backbone to extract multi-scale features from the images, and a separate context encoder to extract context image features from the reference left image. The disparity and optical flow fields are initialized with 1/16 resolution features using scanline matching and global matching respectively, they go through iterations of refinements at 1/16 resolution, then bilinearly upsampled and further refined at 1/8 resolution to produce coarse disparity and optical flow predictions with their confidence maps. The full resolution outputs are obtained using a shared learnable convex upsampler with the reference context features.

3D-2D correspondence construction. Rather than using all pixels, we sample on a regular grid with a fixed stride over the image to keep the number of correspondences bounded and the runtime predictable on edge hardware. For each sampled pixel $\mathbf{p} = (x, y)$ in frame t, we obtain a 3D point in camera frame c_t by back-projection $\mathbf{X}(\mathbf{p}) = \mathbf{D}_t(\mathbf{p}) \cdot K^{-1}\tilde{\mathbf{p}}$, where K is the camera intrinsics matrix and $\tilde{\mathbf{p}} = (x, y, 1)^\top$. The corresponding pixel in frame $t{+}1$ is given by $\mathbf{p}' = \mathbf{p} + \mathbf{F}_{t\to t+1}(\mathbf{p})$. Correspondences with invalid depth, depth outside a configured range, or flow endpoints outside the image bounds are discarded.

Confidence-aware pose estimation. For each remaining correspondence, we compute a scalar reliability score $s(\mathbf{p})$ by combining the depth and flow confidence values (we use their sum in practice) and retain only the top fraction of correspondences. This pruning is deliberately aggressive: it reduces computation and improves robustness by restricting pose estimation to the most geometrically reliable regions. The relative pose is then recovered by minimizing the reprojection error

$$\min_{T_{c_t\to c_{t+1}}} \sum_k \left\| \pi\big(K,\, T_{c_t\to c_{t+1}},\, \mathbf{X}(\mathbf{p}_j)\big) - \mathbf{p}'_j \right\|_2^2, \quad (3)$$

where the summation is over correspondences retained after confidence-based pruning. $\pi(K, T, \mathbf{X}(\mathbf{p}_j)) = KT\,\mathbf{X}(\mathbf{p}_j)$ denotes perspective projection followed by conversion to Cartesian coordinates. This is solved via PnP-RANSAC [51], [52], which jointly recovers the pose and rejects outlier correspondences.

Static mask. The tracking module produces a per-pixel static mask used by the mapper and loop closure module (§III-D). Pixels corresponding to PnP inliers are marked as static. We further refine this mask with a reprojection consistency check: using the estimated pose $T_{c_t\to c_{t+1}}$ and depth $\mathbf{D}_t$, we predict where each pixel $\mathbf{p}$ in frame t should appear in frame $t{+}1$ and compare with the flow-predicted location $\mathbf{p}'$. Pixels whose reprojection error falls below a threshold are classified as static; the remaining pixels are treated as potentially dynamic or unreliable.

D. Loop Closure and Global Optimization

Frame-to-frame pose estimation (§III-C) inevitably accumulates drift over long sequences. To correct this, NeuSLAM maintains a pose graph over selected keyframes, detects revisited locations via loop closure, and globally optimizes the trajectory. Keyframes are selected when accumulated translation or rotation exceeds a threshold, keeping the graph sparse while ensuring sufficient visual overlap for reliable loop detection. Loop closure operates asynchronously with respect to per-frame tracking.

Candidate retrieval. For each keyframe, we extract ORB descriptors [53] and query a DBoW3 [11] database built from a fixed vocabulary. Candidates within a temporal exclusion window of the current keyframe are discarded to suppress trivial matches. Only the top-K candidates are passed to geometric verification, and the number of accepted loop constraints per keyframe is capped to keep computation bounded.

Geometric verification. For a candidate reference keyframe k and the current query keyframe l, we match ORB descriptors using a ratio test. Matched reference keypoints are back-projected into 3D using the stored depth map $\mathbf{D}_k$, and correspondences are retained only if the depth is finite, within a configured range, and the reference keypoint lies on a pixel previously labeled static by the tracking front-end (§III-C). This explicitly suppresses dynamic objects during loop

verification. We then estimate the relative transform $T_{c_k \to c_l}$ via PnP-RANSAC and accept the loop only when the inlier count exceeds a minimum threshold.

Pose graph optimization and mapping. The back-end estimates a globally consistent trajectory using iSAM2-based [10] pose graph optimization, where nodes are keyframe poses and edges are relative-pose constraints from odometry and accepted loop closures. A mapper incrementally builds a dense point cloud by integrating depth from static pixels at each keyframe, storing the insertion pose alongside each map chunk. After pose graph updates, the map is kept consistent with the optimized trajectory by applying per-keyframe rigid corrections at export time, avoiding expensive online reintegration.

IV. Experiments

A. Setup

Implementation details. Our NeuSLAM-Net is trained for 300k steps with a batch size of 64, with 1 refinement iteration at 1/16 resolution and 8 iterations at 1/8 resolution. At each epoch, all samples from Sceneflow [47] and Virtual KITTI 2 [49] are used, and randomly drawn 40k samples from WMGStereo [48]. We use a one-cycle learning rate scheduler with a maximum learning rate of $2e^{-4}$. Our model is trained with a fixed resolution of 368×768 by randomly cropping training sample pairs. Training data augmentation is applied in the same manner as in [23], [38].

To ensure real-time performance, NeuSLAM employs an asynchronous multiprocessing architecture comprising four concurrent processes: neural model inference, front-end pose estimation, loop closure detection, and back-end optimization. Because model inference is the most computationally intensive component, it is exclusively offloaded to the GPU. Meanwhile, the remaining three tracking and mapping processes execute concurrently on the CPU, maximizing hardware utilization and maintaining tractable runtimes on edge devices.

Evaluation. First, we evaluate NeuSLAM-Net's zero-shot generalization capability on Sintel [54] and KITTI-2015 [12] for optical flow estimation, and benchmark disparity estimation with KITTI-2015 [12] and ETH3D [13]. For optical flow benchmarks, we report the end-point-error (EPE), and for KITTI-2015 we additionally report the percentage of pixels with EPE above the standard threshold of 3px. For disparity benchmarks, we report the percentage of pixels with EPE greater than of 3px for KITTI-2015 and 1px for ETH3D. The input images are downsampled to our training resolution, following which the predictions are interpolated back to original dataset resolutions.

Second, to benchmark SLAM performance, we evaluate NeuSLAM on TUM-RGBD [15] and FLSea-VI [16] datasets. We use Absolute Trajectory Error (ATE) to evaluate accuracy.

B. Zero-Shot Disparity and Flow Results

We evaluate NeuSLAM-Net's performance to generalize from synthetic training data to other unseen datasets. This ability is critical since no large-scale real-world datasets are available for training, and stronger generalization capability enables seamless deployment from training directly to practical usage.

TABLE I: Zero-shot optical flow benchmark, our method is run with the optical flow estimation branch only. We report inference FPS for both our PyTorch model and one compiled with TensorRT. Best results for each evaluation metric are **bolded**, second best are underlined.

Method	Sintel (train)		KITTI-15 (train)		FPS
	Clean	Final	EPE	F1-all (%)	
GMFlowNet [35]	<u>1.14</u>	2.71	4.24	15.4	6.6
FlowFormer [31]	**1.01**	<u>2.40</u>	4.09	14.7	1.5
SEA-RAFT [32]	1.19	4.11	<u>3.62</u>	<u>12.9</u>	15.6
GMA [30]	1.30	2.74	4.69	17.1	9.9
CRAFT [33]	1.27	2.79	4.88	17.5	4.3
RAPIDFlow [36]	1.58	2.94	5.87	17.7	39.5
GMFlow [34]	1.50	2.96	10.3	33.6	32.6
RAFT [23]	1.43	2.71	5.04	17.4	14.9
NeuFlow-V2 [9]	1.24	2.67	4.33	15.3	<u>85.4</u>
Ours	1.53	**2.39**	**2.68**	**9.9**	**89.8** (166.4)

TABLE II: Zero-shot stereo benchmark, our method is run with the disparity estimation branch only. We report inference FPS for both our PyTorch model and one compiled with TensorRT. Errors are the percent of pixels with EPE greater than the specified threshold. We use the standard evaluation thresholds: 3px for KITTI, 1px for ETH3D. Best results for each evaluation metric are **bolded**, second best are underlined.

Method	KITTI-15	ETH3D	FPS
GwcNet [39]	22.7	30.1	4.8
ACVNet [40]	11.7	9.4	5.0
GANet [41]	11.7	14.1	0.6
DSMNet [42]	6.5	6.2	0.8
CFNet [43]	<u>5.8</u>	<u>5.8</u>	<u>7.1</u>
RAFT-Stereo [38]	**5.7**	**3.3**	6.4
Ours	6.5	6.9	**84.7** (165.9)

In Tables I and II, we report NeuSLAM-Net's generalization ability by training on multiple synthetic datasets and evaluating on the Sintel and KITTI-2015 training sets for optical flow, and on KITTI-2015 and ETH3D for disparity, comparing against other methods under the same zero-shot setting. We also report inference speed in terms of FPS (frames per second) for running the optical flow and disparity estimation branches separately on an RTX A5000 GPU.

For optical flow, our model achieves the best results on Sintel Final and on both KITTI-2015 metrics (EPE and F1-all), while remaining competitive on Sintel Clean. Notably, it reduces KITTI-2015 F1-all error from 12.9% (SEA-RAFT) to 9.9% while running $5.8\times$ faster. For stereo disparity, our model reports comparable results with other stereo networks such as CFNet [43] and DSMNet [42], while running over $10\times$ faster than all baselines including RAFT-Stereo [38]. With TensorRT compilation, the network runs

TABLE III: Average translation error (m) on the TUM-RGBD benchmark. 'F' means that the method failed to run on that specific sequence. '-' indicates data not available or applicable. OOM means out of GPU memory. We report inference FPS for both our PyTorch model and one compiled with TensorRT. DROID-SLAM and earlier results are reproduced from Table 4 of [6].

Method	360	desk	desk2	floor	plant	room	rpy	teddy	xyz	avg	Orin Nano		RTX 3090	
											FPS	Memory	FPS	Memory
ORB-SLAM2 [2]	F	0.071	F	0.023	F	F	F	F	0.010	-	13	-	37	-
ORB-SLAM3 [1]	F	0.017	0.210	F	0.034	F	F	F	**0.009**	-	12	-	34	-
DeepTAM [55]	0.111	0.053	0.103	0.206	0.064	0.239	0.093	0.144	0.036	0.116	-	-	-	-
TartanVO [56]	0.178	0.125	0.122	0.349	0.297	0.333	0.049	0.339	0.062	0.206	≤ 2	6.4 GB	40	7.2 GB
DeepV2D [57]	0.243	0.166	0.379	1.653	0.203	0.246	0.105	0.316	0.064	0.375	-	-	-	-
DeepFactors [58]	0.159	0.170	0.253	0.169	0.305	0.364	0.043	0.601	0.035	0.233	OOM	OOM	12	16.4 GB
DROID-SLAM [6]	0.111	0.018	0.042	**0.021**	**0.016**	**0.049**	**0.026**	0.048	0.012	0.038	OOM	OOM	21	9.3 GB
MASt3R-SLAM [7]	**0.049**	**0.016**	**0.024**	0.025	0.020	0.061	0.027	**0.041**	**0.009**	**0.030**	0.8	7 GB	≤ 5	18.7 GB
Ours - Odom	0.148	0.099	0.078	0.069	0.210	0.354	0.051	0.254	0.024	0.143	**22**	0.8 GB	**147**	1.1 GB
Ours - SLAM	0.088	0.035	0.039	0.048	0.080	0.084	0.030	0.115	0.015	0.059	10 (18)	1.05 GB	84 (136)	1.2 GB

TABLE IV: Average translation error (m) on the FLSea-VI benchmark

Methods	Canyons				Red Sea								Avg	RTX 3090	
	UC	Fla	HC	TC	NP	LP	DP	PP	CTL	CPL	BDL	SP		FPS	Memory
ORB-SLAM3	0.084	0.076	0.078	0.028	0.049	0.054	0.096	**0.070**	0.039	0.047	**0.040**	0.041	0.060	35	-
DROID-SLAM	0.076	**0.053**	**0.024**	0.028	0.032	0.049	0.059	0.086	0.032	**0.029**	0.064	**0.040**	0.047	19	10.2 GB
DUV-SLAM	**0.067**	0.064	0.026	**0.024**	**0.031**	**0.045**	0.057	0.087	**0.021**	0.034	**0.040**	**0.040**	**0.045**	21	9.37 GB
NeuSLAM	0.080	0.067	0.044	0.029	0.039	0.052	**0.054**	0.082	0.037	0.039	0.051	0.042	0.051	78 (132)	1.3 GB

at over 165 FPS, making it suitable as a real-time front-end for edge-deployed SLAM.

C. Indoor and Outdoor SLAM Results

TUM-RGBD [15]. This RGBD dataset consists of indoor scenes captures with a handheld Kinect camera. The dataset is notoriously difficult for SLAM due to rolling shutter artifacts, motion and heavy rotation. As shown in Table. III, classical works like ORB-SLAM3 tend to fail on most of the sequences and while DROID-SLAM performs robustly, we can only run it on a desktop. NeuSLAM on the other hand is both robust and fast, achieving the second best performance while being 6.5 times faster and consuming 87% less GPU memory. Note that we turn off the disparity branch in NeuSLAM because we already have depth data available, which gives a further boost to inference speed.

FLSea-VI dataset features forward looking visual sequences captures across the Mediterranean Sea's Canyons and the Red Sea. Because the dataset provides reliable depth maps, we disable NeuSLAM's learned disparity branch and directly utilize the provided depth data. As demonstrated in Table IV, NeuSLAM performs robustly against heavy baselines like DROID-SLAM [6] and DUV-SLAM [59] while requiring much less compute. Since no underwater data was used for training the NeuSLAM-Net network. Table. IV also serves as another evidence of the generalizability of our approach.

V. Conclusion and Future Work

In this work, we presented NeuSLAM, a hybrid dense visual SLAM system designed for resource-constrained edge devices. By pairing an efficient shared-encoder network for joint optical flow and stereo disparity estimation with a classical geometric back-end for pose graph optimization, NeuSLAM achieves a practical balance between dense scene reconstruction and computational efficiency that has previously been out of reach on embedded hardware. Our learned front-end, NeuSLAM-Net, achieves state-of-the-art zero-shot optical flow accuracy on KITTI-2015 and competitive stereo disparity results while running at over 85 FPS on a desktop GPU. The full SLAM pipeline achieves trajectory accuracy competitive with DROID-SLAM on TUM RGB-D while running 6.5× faster and consuming 87% less GPU memory. On a Jetson Orin Nano, NeuSLAM operates at 18 and 10 FPS for RGB-D and stereo setups, respectively, demonstrating that dense SLAM with competitive accuracy can be achieved within the strict compute budgets of modern edge devices.

Several directions remain for future work. First, while our confidence-aware tracking is effective at suppressing dynamic objects, explicitly modeling scene dynamics, for instance through motion segmentation or object-level tracking, could further improve robustness in highly dynamic environments. Second, our system currently relies on ORB-based loop closure, which can struggle under significant viewpoint changes. Replacing or augmenting this with learned place recognition methods, while respecting the edge-device compute budget, is a promising avenue to pursue further. Finally, extending NeuSLAM to tightly coupled IMU measurements would broaden its applicability to agile robotic platforms where visual tracking alone may be insufficient during fast rotations or temporary occlusions.

REFERENCES

[1] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE T-RO*, vol. 37, no. 6, pp. 1874–1890, 2021.

[2] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE T-RO*, vol. 33, no. 5, pp. 1255–1262, 2017.

[3] T. Qin, S. Cao, J. Pan, and S. Shen, "A general optimisation-based framework for global pose estimation with multiple sensors," *IET Cyber-Systems and Robotics*, vol. 7, no. 1, p. e70023, 2025.

[4] V. Usenko, N. Demmel, D. Schubert, J. Stückler, and D. Cremers, "Visual-inertial mapping with non-linear factor recovery," *IEEE RA-L*, vol. 5, no. 2, pp. 422–429, 2019.

[5] A. Korovko, D. Slepichev, A. Efitorov, A. Dzhumamuratova, V. Kuznetsov, H. Rabeti, J. Biswas, and S. Pouya, "cuvslam: Cuda accelerated visual odometry and mapping," *arXiv preprint arXiv:2506.04359*, 2025.

[6] Z. Teed and J. Deng, "Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras," *NeurIPS*, vol. 34, 2021.

[7] R. Murai, E. Dexheimer, and A. J. Davison, "Mast3r-slam: Real-time dense slam with 3d reconstruction priors," in *CVPR*, 2025.

[8] D. Maggio, H. Lim, and L. Carlone, "Vggt-slam: Dense rgb slam optimized on the sl (4) manifold," *arXiv preprint arXiv:2505.12549*, 2025.

[9] Z. Zhang, A. Gupta, H. Jiang, and H. Singh, "Neuflow v2: High-efficiency optical flow estimation on edge devices," in *IROS*, 2025.

[10] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "isam2: Incremental smoothing and mapping using the bayes tree," *IJRR*, vol. 31, no. 2, pp. 216–235, 2012.

[11] "DBoW3 dbow3," 2017. [Online]. Available: https://github.com/rmsalinas/DBow3

[12] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *CVPR*, 2015.

[13] T. Schops, J. L. Schonberger, S. Galliani, T. Sattler, K. Schindler, M. Pollefeys, and A. Geiger, "A multi-view stereo benchmark with high-resolution images and multi-camera videos," in *CVPR*, 2017.

[14] NVIDIA Corporation, "NVIDIA TensorRT: Programmable inference accelerator," https://developer.nvidia.com/tensorrt, 2023, accessed: March 2, 2026.

[15] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *IROS*, 2012.

[16] Y. Randall and T. Treibitz, "Flsea: Underwater visual-inertial and stereo-vision forward-looking datasets," 2023. [Online]. Available: https://arxiv.org/abs/2302.12772

[17] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: A versatile and accurate monocular slam system," *IEEE T-RO*, vol. 31, no. 5, pp. 1147–1163, 2015.

[18] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an open-source library for real-time metric-semantic localization and mapping," in *ICRA*, 2020.

[19] J. Pestana, M. Maurer, D. Muschick, M. Hofer, and F. Fraundorfer, "Overview obstacle maps for obstacle-aware navigation of autonomous drones," *Journal of field robotics*, vol. 36, no. 4, pp. 734–762, 2019.

[20] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *ECCV*, 2014.

[21] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE TPAMI*, vol. 40, no. 3, pp. 611–625, 2017.

[22] J. Zubizarreta, I. Aguinaga, and J. M. M. Montiel, "Direct sparse mapping," *IEEE T-RO*, vol. 36, no. 4, pp. 1363–1370, 2020.

[23] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *ECCV*, 2020.

[24] Z. Teed, L. Lipson, and J. Deng, "Deep patch visual odometry," *NeurIPS*, vol. 36, 2023.

[25] L. Lipson, Z. Teed, and J. Deng, "Deep patch visual slam," in *ECCV*, 2024.

[26] G. Shen, T. Deng, Y. Wang, Y. Chen, Y. Shen, J. Liu, and J. Wang, "Grs-slam3r: Real-time dense slam with gated recurrent state," *arXiv preprint arXiv:2509.23737*, 2025.

[27] A. Kumar, J. Park, and L. Behera, "High-speed stereo visual slam for low-powered computing devices," *IEEE Robotics and Automation Letters*, vol. 9, no. 1, pp. 499–506, 2023.

[28] A. J. Ben Ali, M. Kouroshli, S. Semenova, Z. S. Hashemifar, S. Y. Ko, and K. Dantu, "Edge-slam: Edge-assisted visual simultaneous localization and mapping," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 1, pp. 1–31, 2022.

[29] D. Li, Y. Zhao, J. Xu, S. Zhang, L. Shangguan, and Z. Yang, "Edgeslam2: Rethinking edge-assisted visual slam with on-chip intelligence," in *IEEE INFOCOM*, 2024.

[30] S. Jiang, D. Campbell, Y. Lu, H. Li, and R. Hartley, "Learning to estimate hidden motions with global motion aggregation," in *ICCV*, 2021.

[31] Z. Huang, X. Shi, C. Zhang, Q. Wang, K. C. Cheung, H. Qin, J. Dai, and H. Li, "Flowformer: A transformer architecture for optical flow," in *ECCV*, 2022.

[32] Y. Wang, L. Lipson, and J. Deng, "Sea-raft: Simple, efficient, accurate raft for optical flow," in *ECCV*, 2024.

[33] X. Sui, S. Li, X. Geng, Y. Wu, X. Xu, Y. Liu, R. Goh, and H. Zhu, "Craft: Cross-attentional flow transformer for robust optical flow," in *CVPR*, 2022.

[34] H. Xu, J. Zhang, J. Cai, H. Rezatofighi, and D. Tao, "Gmflow: Learning optical flow via global matching," in *CVPR*, 2022.

[35] S. Zhao, L. Zhao, Z. Zhang, E. Zhou, and D. Metaxas, "Global matching with overlapping attention for optical flow estimation," in *CVPR*, 2022.

[36] H. Morimitsu, X. Zhu, R. M. Cesar, X. Ji, and X.-C. Yin, "Rapidflow: Recurrent adaptable pyramids with iterative decoding for efficient optical flow estimation," in *ICRA*, 2024.

[37] Z. Zhang, H. Jiang, and H. Singh, "Neuflow: Real-time, high-accuracy optical flow estimation on robots using edge devices," in *IROS*, 2024.

[38] L. Lipson, Z. Teed, and J. Deng, "Raft-stereo: Multilevel recurrent field transforms for stereo matching," in *3DV*, 2021.

[39] X. Guo, K. Yang, W. Yang, X. Wang, and H. Li, "Group-wise correlation stereo network," in *CVPR*, 2019.

[40] G. Xu, J. Cheng, P. Guo, and X. Yang, "Attention concatenation volume for accurate and efficient stereo matching," in *CVPR*, 2022.

[41] F. Zhang, V. Prisacariu, R. Yang, and P. H. Torr, "Ga-net: Guided aggregation net for end-to-end stereo matching," in *CVPR*, 2019.

[42] F. Zhang, X. Qi, R. Yang, V. Prisacariu, B. Wah, and P. Torr, "Domain-invariant stereo matching networks," in *ECCV*, 2020.

[43] Z. Shen, Y. Dai, and Z. Rao, "Cfnet: Cascade and fused cost volume for robust stereo matching," in *CVPR*, 2021.

[44] H. Jiang, D. Sun, V. Jampani, Z. Lv, E. Learned-Miller, and J. Kautz, "Sense: A shared encoder network for scene-flow estimation," in *ICCV*, 2019.

[45] T. Zhang, H. Zhang, Y. Li, Y. Nakamura, and L. Zhang, "Flowfusion: Dynamic dense rgb-d slam based on optical flow," in *ICRA*, 2020.

[46] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume," in *CVPR*, 2018.

[47] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *CVPR*, 2016.

[48] D. Yan, A. Raistrick, and J. Deng, "Procedural dataset generation for zero-shot stereo matching," *arXiv e-prints*, 2025.

[49] Y. Cabon, N. Murray, and M. Humenberger, "Virtual kitti 2," *arXiv preprint arXiv:2001.10773*, 2020.

[50] H. Lin, S. Chen, J. Liew, D. Y. Chen, Z. Li, G. Shi, J. Feng, and B. Kang, "Depth anything 3: Recovering the visual space from any views," *arXiv preprint arXiv:2511.10647*, 2025.

[51] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Ep n p: An accurate o (n) solution to the p n p problem," *International journal of computer vision*, vol. 81, no. 2, pp. 155–166, 2009.

[52] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[53] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *ICCV*. Ieee, 2011.

[54] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *ECCV*, 2012.

[55] H. Zhou, B. Ummenhofer, and T. Brox, "Deeptam: Deep tracking and mapping," in *ECCV*, 2018.

[56] W. Wang, Y. Hu, and S. Scherer, "Tartanvo: A generalizable learning-based vo," in *CoRL*. PMLR, 2021.

[57] Z. Teed and J. Deng, "Deepv2d: Video to depth with differentiable structure from motion," in *ICLR*, 2020.

[58] J. Czarnowski, T. Laidlow, R. Clark, and A. J. Davison, "Deepfactors: Real-time probabilistic dense monocular slam," *IEEE RA-L*, vol. 5, no. 2, pp. 721–728, 2020.

[59] R. Liu, S. Fan, W. Wang, and Y. Yang, "Underwater visual slam with depth uncertainty and medium modeling," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2025, pp. 970–980.