NeuroMesh: A Unified Neural Inference Framework for Decentralized Multi-Robot Collaboration

Yang Zhou[1], Yash Shetye[1], Long Quang[1,2], Devon Super[3], Jesse Milzman[2], Manohari Goarin[1], Aditya Azad[1], Devang Sunil Dhake[1], Jeffrey Mao[1], Carlos Nieto-Granda[2], and Giuseppe Loianno[4]

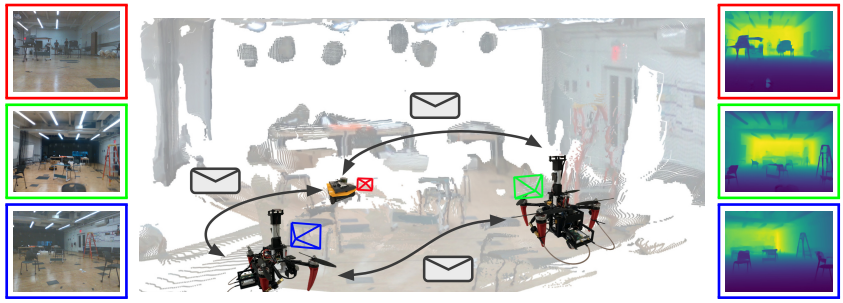


Fig. 1: A multi-robot collaborative depth perception experiment empowered by NeuroMesh framework with two aerial robots (green and blue) and one ground robot (red). The left column shows the input RGB images, the middle column shows the pointcloud represented in the consistent coordinate system. The right column shows the output depth image of each view.

***Abstract*— Deploying learned multi-robot models on heterogeneous robots remains challenging due to hardware heterogeneity, communication constraints, and the lack of a unified execution stack. We present NeuroMesh, a multi-domain, cross-platform, and modular decentralized neural inference framework that standardizes observation encoding, message passing, aggregation, and task decoding in a single pipeline. NeuroMesh combines a dual-aggregation paradigm for reduction- and broadcast-based fusion with a parallel architecture that decouples cycle time from end-to-end latency, and its C++/ROS 2 implementation uses Zenoh for inter-robot transport with hybrid GPU/CPU inference. We validate NeuroMesh on a heterogeneous aerial-ground team across collaborative perception, decentralized control, and task assignment, and characterize the transport and inference trade-offs that govern real-time deployment. We have released NeuroMesh as an open-source framework to the community.**

Supplementary Material

Project / Code: https://arplaboratory.github.io/NeuroMesh

I. Introduction

Learning-based multi-robot collaboration lets robots share neural representations and computational resources to achieve objectives beyond the capability of individuals [1], improving robustness to single-point failures, efficiency through parallelization, and performance through collective intelligence. Its potential has been shown in collaborative perception [2], cooperative control [3], [4], and task assignment [5], where Graph Neural Networks (GNNs) are popular because they are decentralized by construction and naturally express the team as a graph.

Transitioning these algorithms to practice, however, remains difficult. A deployable system must integrate perception, control, and decision-making in one execution stack, support heterogeneous compute and inference engines, stay robust under decentralized communication, and expose interfaces flexible enough for task-dependent models. Existing frameworks cover only part of this gap: ModGNN [6] targets architectural modularity for multi-agent control but provides no deployment stack with inter-robot communication or hardware-accelerated inference, while the ROS 2 framework of [7] demonstrates real-robot decentralized control but is Python-based and tailored to a single task family and platform-specific payloads, leaving it unsuited to perception-heavy or mixed-domain workloads.

We therefore present NeuroMesh, a unified decentralized neural inference framework for real-world multi-robot systems. Rather than a new training method, it provides a four-stage pipeline—observation encoding, message passing, aggregation, and task output decoding—with task-dependent encoder/decoder interfaces over a common execution stack, optimized for decentralized mesh networks while remaining compatible with centralized Wi-Fi. Our contributions are: **(i) multi-domain** operation through a dual-aggregation paradigm that unifies dense perception and compact-message inference in one formulation; **(ii) cross-platform** deployment, with modules that run inference on CPU or GPU to match hardware constraints; **(iii) modularity** via a microservice architecture whose encoders, aggregators, and decoders range from identity mappings to deep networks over the same parallel pipeline; and **(iv) efficiency** from a C++ implementation with an optimized network protocol and accelerated inference engine.

[1]New York University, New York, USA.
[2]U.S. Army Combat Capabilities Development Command, Army Research Laboratory, Adelphi, MD 20783, USA.
[3]Vanderbilt University, Tennessee, USA. [4]University of California Berkeley, Berkeley, CA 94720, USA.

This work was supported by the DEVCOM ARL grant SARA W911NF-24-2-0057, the DARPA YFA Grant D22AP00156-00, and the NSF CPS Grant CNS-2603416.

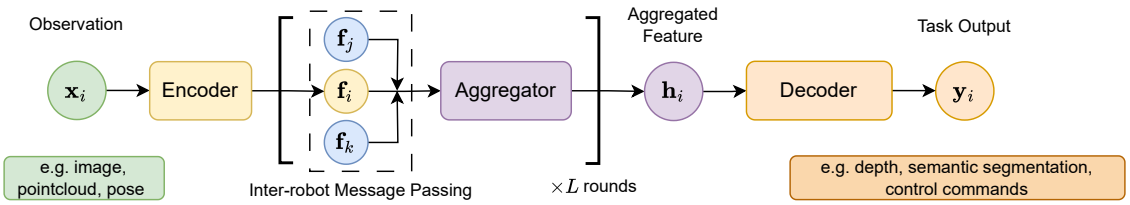
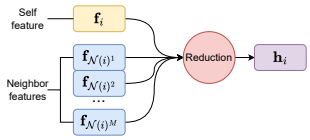
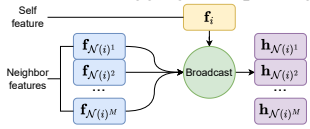


Fig. 2: NeuroMesh four-stage process modeling of decentralized multi-robot neural inference for a generic robot i. The encoded feature $\mathbf{f}_i$ and received encoded features ($\mathbf{f}_j, \mathbf{f}_k$, etc.) from connected robots are passed into the message passing and aggregation stage. The message passing and aggregation stage can be executed for L rounds. In the decoder stage, the aggregated feature $\mathbf{h}_i$ is transformed into a tensor and produces the task output $\mathbf{y}_i$.



(a) Reduction aggregation paradigm



(b) Broadcast aggregation paradigm

Fig. 3: The two aggregation paradigms. In Fig. 3a, neighbor features (blue) are reduced together with the self feature $\mathbf{f}_i$ (yellow) into a single aggregated feature $\mathbf{h}_i$ (purple). In Fig. 3b, $\mathbf{f}_i$ is broadcast against each neighbor feature to obtain a per-neighbor $\mathbf{h}_i^m$, which are assembled into $\mathbf{h}_i$.

II. NeuroMesh Framework

A. Problem Formulation

The decentralized multi-robot neural inference problem, shown in Fig. 2, is a four-stage process: in **observation encoding**, robot i encodes its local observation $\mathbf{x}_i$ (image, point cloud, poses, etc.) into a feature tensor $\mathbf{f}_i$; during **message passing**, robots exchange features with the neighbors $\mathcal{N}(i)$ in communication range; in **aggregation**, each robot fuses the received information into $\mathbf{h}_i$; and in **task output decoding**, each robot decodes $\mathbf{h}_i$ into the task output $\mathbf{y}_i$ (depth, semantics, control commands, etc.). For L communication rounds,

$$\begin{aligned} \mathbf{f}_i &= \text{Encode}(\mathbf{x}_i), \quad \mathbf{h}_i^{(0)} = \mathbf{f}_i, \\ \mathbf{h}_i^{(l+1)} &= \text{Aggregate}(\mathbf{h}_i^{(l)}, \underbrace{\{\mathbf{h}_j^{(l)} : j \in \mathcal{N}(i)\}}_{\text{Message Passing}}), \\ l &= 0, \cdots, L-1, \quad \mathbf{y}_i = \text{Decode}(\mathbf{h}_i^{(L)}). \end{aligned} \tag{1}$$

All four modules are task-dependent: the encoder/decoder pair may be a deep vision stack for perception, or a lightweight learned mapping for compact state or cost vectors in control and task assignment.

B. Dual Aggregation

A key contribution of NeuroMesh is unifying multiple collaboration domains through a dual-aggregation formulation. Existing formulations typically rely on a *reduction* paradigm (e.g., $\sum$ or max-pooling) that suffices for consensus-like control but destroys the pairwise relational information dense perception requires. Our dual-aggregation paradigm, shown in Fig. 3, spans algorithms from state synchronization to multi-view geometric reasoning without architectural redesign. Reduction collapses the self-feature $\mathbf{f}_i$ and neighbor features $\{\mathbf{f}_j : j \in \mathcal{N}(i)\}$ into one aggregated feature $\mathbf{h}_i$, for instance by averaging. *Broadcast* instead computes pairwise interactions: for M neighbors, $\mathbf{f}_i$ is replicated and concatenated with each $\mathbf{f}_j$, and a network processes these pairs in parallel to produce an aggregated tensor $\mathbf{h}_i$ with M channels that preserves relational information. Under message drops, the aggregation step can pause until the required messages arrive or fall back to partial-neighborhood or single-robot operation when the task still yields a valid output.

C. Parallel Execution Pipeline

Deployment must contend with latency accumulating across perception and control stacks. To decouple cycle time from total processing time, we split execution into three concurrent components—encoder, message passing and aggregation, and decoder—driven by the encoder callback. With computation times T_e, T_m, and T_d, the output cycle time is $T_{\text{pipeline}} = \max(T_e, T_m, T_d)$ while the end-to-end delay is $T_{\text{delay}} = T_e + T_m + T_d$, so the pipeline raises throughput over sequential computation without reducing end-to-end delay.

D. Implementation and Communication

We implement NeuroMesh in ROS 2 [8], instantiating each stage of Fig. 2 as a ROS 2 wrapper node over an optimized C++ backend, using TensorRT (TRT) [9] on NVIDIA platforms and ONNX Runtime elsewhere. The framework is agnostic to the network design, so users insert task-dependent networks in a plug-and-play manner: local observations enter a task-defined encoder callback, encoded tensors are published locally and over the network, the aggregator consumes the current self-feature together with buffered neighbor features, and the decoder emits the task output.

Communication cost easily exceeds the bandwidth of centralized infrastructure as the team grows, favoring a decentralized topology. Because native ROS 2 middleware such as Fast RTPS and Cyclone DDS did not meet our applications' requirements, we incorporate Zenoh [10]: in peer mode each robot exchanges traffic directly with peers or uses link-state routing, while in router mode routers forward traffic between nodes. Inter-robot messages carry a

timestamp and sequence index, and every robot keeps a keep-latest buffer per neighbor, discarding out-of-order and stale packets. Aggregation runs either in blocking mode, waiting for all required messages, or in best-effort mode on the most recent subset of neighbors.

III. Hardware Case Studies

We demonstrate the same runtime on three collaboration tasks—high-bandwidth perception, high-frequency control, and discrete assignment—swapping only task-specific modules. Robots carry Doodlelabs mini Mesh Rider Radios forming a decentralized mesh; aerial robots use an NVIDIA Jetson Orin NX and ground robots Mini-ITX computers with 4th-generation Intel i7 CPUs and NVIDIA 1050 Ti GPUs.

A. Multi-Robot Collaborative Perception

We run two perception examples on two aerial robots and one ground robot (Fig. 5). First, we build a decentralized version of DUSt3R [11] for collaborative depth estimation: a Vision Transformer encodes RGB images, encoded features are exchanged, broadcast aggregation pairs self and neighbor features through the transformer decoder, and the head regresses depth from the resulting point maps. Using the original weights, centralized and decentralized execution obtain the same absolute relative error of 5.29 and inlier ratio of 61.32 (threshold 1.03) on our indoor/outdoor dataset, so NeuroMesh distributes the model without altering its output, while the single-robot baseline yields less accurate depth. Second, we deploy decentralized GNN-based collaborative semantic segmentation [2] with CNN encoders/decoders and reduction aggregation, where collaboration helps each robot recognize objects seen from other viewpoints.

B. Multi-Robot Collaborative Control

We deploy control policies on three non-holonomic ground robots that must reach their targets without colliding, adapting the GNN approach of [7] from holonomic platforms to a unicycle model and training it with reinforcement learning under a reward that encourages fast goal reaching and penalizes collisions. Each robot obtains pose and heading from onboard localization (e.g., SLAM/VIO), and NeuroMesh exchanges only compact state features, so no motion-capture infrastructure is required. The observation is $\mathbf{o}_i = (\mathbf{g}_i - \mathbf{x}_i, \mathbf{x}_i, \mathbf{u}(\theta_i), \mathbf{x}_i + v_f \mathbf{u}(\theta_i)) \in \mathbb{R}^8$, where $\mathbf{x}_i$ is the 2D position, $\mathbf{g}_i$ the goal, $\mathbf{u}(\theta_i) = (\cos\theta_i, \sin\theta_i)$ the planar heading, and v_f the forward velocity. A four-layer MLP encodes $\mathbf{f}_i$; message passing applies a 3-layer MLP $\boldsymbol{g}$ to each neighbor difference and accumulates $\mathbf{h}_i = \sum_j \boldsymbol{g}(\mathbf{f}_j - \mathbf{f}_i)$; and a 4-layer decoder yields $\mathbf{y}_i \in \mathbb{R}^4$, to which an entrywise shifted ReLU $\tilde{\mathbf{y}}_i = \log(e^{\mathbf{y}_i} + 1) + 1$ is applied to parameterize the Beta distributions from which forward and angular velocities are sampled.

The policy achieved an 80% success rate in 2D simulation matching the training environment of [7], but real-world deployment revealed significant sim-to-real transfer challenges, particularly in angular velocity control. NeuroMesh enabled real-time decentralized execution on hardware, and with real-time correction of errant angular velocities we reached a 60% success rate in the three-robot navigation task (Fig. 6).

C. Multi-Robot Collaborative Task Assignment

Finally, 5 robots each select a goal while avoiding conflicts, minimizing $\sum_{i,j} c_{ij} s_{ij}$ subject to one-to-one matching, where c_{ij} is the cost of assigning robot i to goal j and $s_{ij} \in \{0, 1\}$ is the assignment label—the linear sum assignment problem solved centrally by the Hungarian algorithm [12]. Unlike the perception setting there is no raw sensor input: the observation is a compact cost vector $\mathbf{c}_i = [c_{i0}, \cdots, c_{i5}]^\top$ derived from the robot's state estimate and the candidate goals. Following [5], we train a GNN with the Hungarian algorithm as expert, framing the task as classification over assignment labels: a 2-layer MLP encodes the cost vector, a graph-attention aggregator with 3 heads and 2 layers fuses the exchanged embeddings, and a lightweight decoder outputs the assignment. Encoders and decoders thus stay lightweight while the framework still contributes decentralized transport, buffering, and graph-based aggregation. We validate this outdoors with a 5-robot team, our largest hardware experiment.

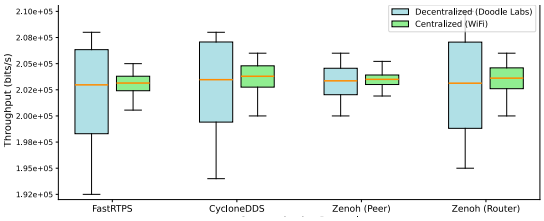
Over $N_{\text{tests}} = 20$ trials we measure the Success Rate (SR), the percentage of goals covered once robots reach their destinations, and the Total Cost Performance (TCP), the average cost increase over the optimum for fully covered runs. Sweeping the message size over 40, 64, 128, and 256 B gives $SR = 87.12$, 90.22, 93.92, and 94.01% with $TCP = 0.05$, 0.03, 0.03, and 0.02%: successful runs are always near-optimal, but message size significantly affects the success rate, and 128 B already suffices for near-optimal results while 256 B adds no further gain.

IV. Deployment Trade-offs

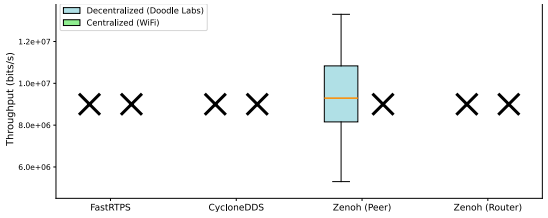
A. Communication

NeuroMesh supports decentralized mesh radios and centralized Wi-Fi with the four transports of Section II-D, evaluated indoors with 3 robots in motion at 1–7 m separation and moderate RF interference, where live pipelines generate a 128 B control payload and a 3.15 MB perception payload (Fig. 4).

On the mesh, Fast RTPS, CycloneDDS, and Zenoh router mode carry control messages but fluctuate between roughly 50 and 200 Hz on the 128 B payload, whereas Zenoh peer mode holds a stable 200 Hz with the lowest latency (4.8 ms vs. 6.7 and 8.6 ms), jitter (0.6 ms), and packet loss (0.3%). For the 3.15 MB perception payload the other transports fail and only Zenoh peer mode succeeds, at roughly 0.5 Hz; its latency is far higher (2712.2 ms, jitter 540.5 ms) because of the large serialized tensor, but it is the only transport that makes decentralized collaborative perception feasible at all. On centralized 802.11 Wi-Fi, all four transports reach comparable control-payload throughput, yet the perception payload could not be sustained because routing large tensors through the central node saturates the bandwidth. Large perception features thus stay communication limited even when a common framework makes the model executable. Emulating larger teams in Zenoh peer mode under a 6 MB/s per-node bandwidth limit, full 200 Hz throughput on 128 B messages holds up to 10 robots and degrades to 164.51 and



(a) 128-byte control payload.



(b) 3.15-MB perception payload.

Fig. 4: Transport throughput in decentralized mesh and centralized Wi-Fi deployments. Zenoh peer mode is the only tested transport that delivers the large perception tensor over the mesh.



Aerial robot Aerial robot Ground robot

Fig. 5: Multi-robot collaborative perception: input RGB (top) and collaborative depth estimation (bottom) with the decentralized DUSt3R [11] deployed through NeuroMesh on two aerial robots and one ground robot.

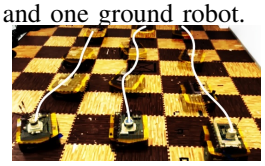


Fig. 6: Multi-robot collaborative control. Three ground robots perform point-to-point navigation without collision.

95.32 msgs/s at 30 and 50 robots, bounded by message size, per-node bandwidth, and aggregation cost.

B. Computation

Table I benchmarks per-module computation on the GPU-based TensorRT and CPU-based ONNX engines. The resource-intensive perception transformers benefit substantially from GPU parallelization, while their concatenation-based aggregation is nearly free. The small control MLPs (up to 64 channels) are instead faster on CPU, since CPU-GPU transfer overhead outweighs GPU acceleration; for assignment the MLP encoder likewise favors the CPU, but TensorRT wins on attention-based aggregation. The table also validates the parallel pipeline: its cycle time stays well below the sequential total (24.012 vs. 14.821 ms for perception on CPU and 5.441 vs. 2.957 ms on GPU), so the system accepts new data at a rate limited only by its slowest stage.

V. Conclusion

We presented NeuroMesh, a multi-domain, cross-platform, and modular decentralized framework for multi-robot neural inference whose hardware experiments across perception, control, and decision-making demonstrate its generality, efficiency, and modularity for real-world deployment. It can also host classical coordination algorithms such as consensus or flocking by reusing the same modules. Future work will target larger physical teams, broader RF environments, and heterogeneous sensing modalities.

TABLE I: Per-module runtime (ms) for the CPU (ONNX) and GPU (TRT) back ends; total parallel is the achievable output cycle time.

	Perception		Control		Assignment	
Module	ONNX	TRT	ONNX	TRT	ONNX	TRT
Encoder	9.191	2.484	16.1	23.4	0.069	0.294
Aggregation	0.000	0.000	16.1	18.1	0.385	0.184
Decoder	14.821	2.957	18.1	22.4	0.000	0.000
Total parallel	14.821	**2.957**	**18.1**	23.4	0.385	**0.294**

References

[1] B. Wu and C. S. Suh, "State-of-the-art in robot learning for multi-robot collaboration: A comprehensive survey," *arXiv preprint arXiv:2408.11822*, 2024.

[2] Y. Zhou, J. Xiao, Y. Zhou, and G. Loianno, "Multi-Robot Collaborative Perception With Graph Neural Networks," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2289–2296, 2022.

[3] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro, "Learning Decentralized Controllers for Robot Swarms with Graph Neural Networks," in *Proceedings of the Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, L. P. Kaelbling, D. Kragic, and K. Sugiura, Eds., vol. 100. PMLR, Oct 2020, pp. 671–682.

[4] S. Chen, Y. Sun, P. Li, L. Zhou, and C.-T. Lu, "Learning Decentralized Flocking Controllers with Spatio-Temporal Graph Neural Network," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 2596–2602.

[5] M. Goarin and G. Loianno, "Graph Neural Network for Decentralized Multi-Robot Goal Assignment," *IEEE Robotics and Automation Letters*, vol. 9, no. 5, pp. 4051–4058, 2024.

[6] R. Kortvelesy and A. Prorok, "ModGNN: Expert Policy Approximation in Multi-Agent Systems with a Modular Graph Neural Network Architecture," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 9161–9167.

[7] J. Blumenkamp, S. Morad, J. Gielis, Q. Li, and A. Prorok, "A Framework for Real-World Multi-Robot Systems Running Decentralized GNN-Based Policies," in *International Conference on Robotics and Automation (ICRA)*, 2022, pp. 8772–8778.

[8] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[9] NVIDIA, "TensorRT," *https://developer.nvidia.com/tensorrt*, 2022.

[10] A. Corsaro, L. Cominardi, O. Hecart, G. Baldoni, J. Avital, J. Loudet, C. Guimares, M. Ilyin, and D. Bannov, "Zenoh: Unifying Communication, Storage and Computation from the Cloud to the Microcontroller," in *26th Euromicro Conference on Digital System Design (DSD)*. Los Alamitos, CA, USA: IEEE Computer Society, Sep 2023, pp. 422–428.

[11] S. Wang, V. Leroy, Y. Cabon, B. Chidlovskii, and J. Revaud, "DUSt3R: Geometric 3D Vision Made Easy," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2024, pp. 20 697–20 709.

[12] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.