CLOSER 2024
14TH INTERNATIONAL CONFERENCE ON CLOUD COMPUTING AND SERVICES SCIENCE
ANGERS, FRANCE 2 - 4 MAY, 2024

UNCERTAINTY ESTIMATION IN MULTI-AGENT DISTRIBUTED LEARNING FOR AI-ENABLED EDGE DEVICES

Gleb Radchenko, Victoria Fill
Silicon Austria Labs

01 Research Goals

02 Federated and Decentralized Learning

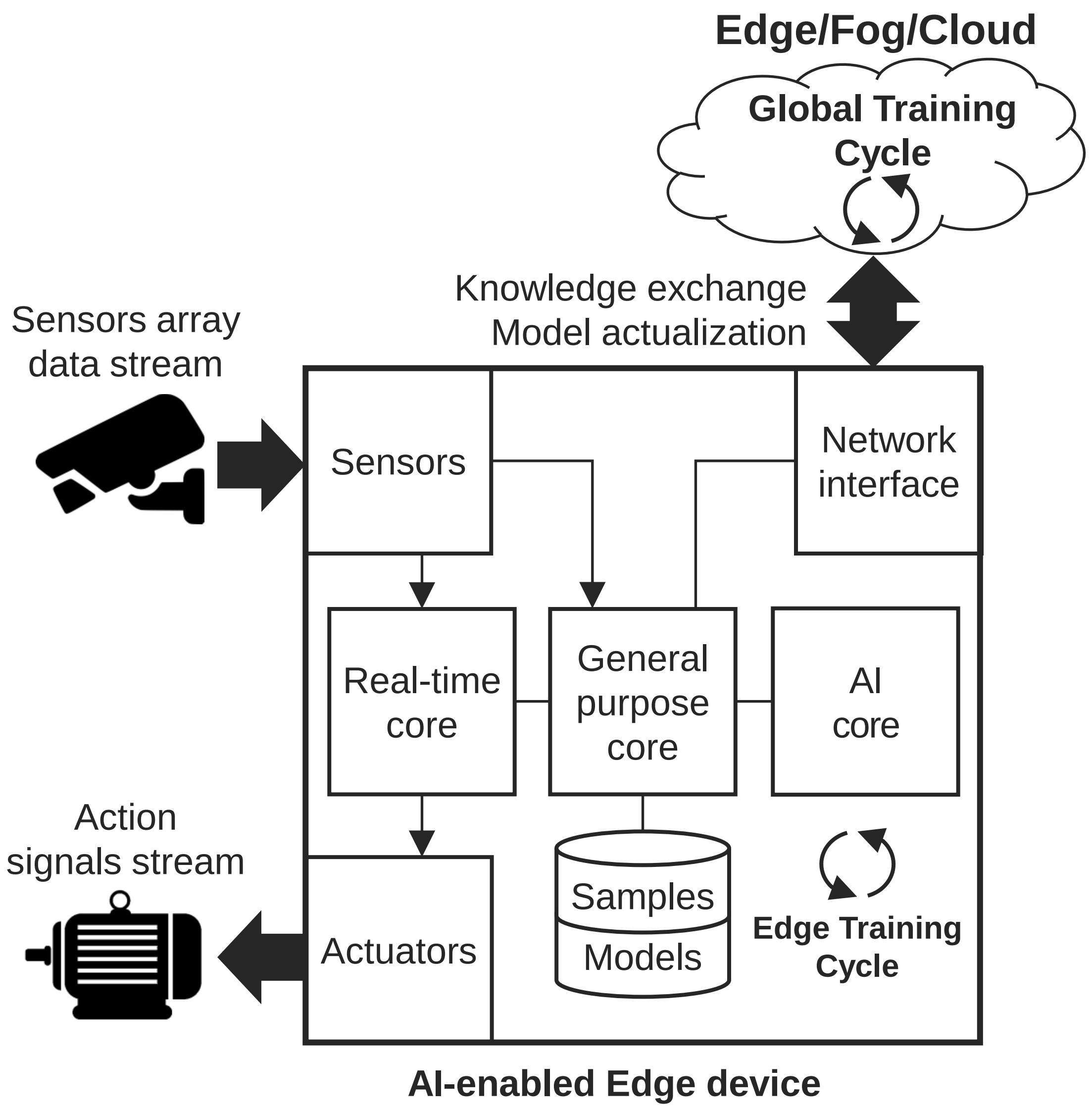
03 Uncertainty Estimation in ML

04 Test Case Outline and Proposed Solution

05 Results Evaluation and Future Work

AI-ENABLED EDGE DEVICES

- We explore the potential for enabling decentralized learning and knowledge sharing among AI-Enabled Edge Devices (AEEDs).
- An AEED is an agent device situated at the network edge, directly interfacing with data streams from various sensors.
- It may also control actuators to interact with its environment.
- Beyond standard computational capabilities, these devices feature an AI Core capable of conducting both inference and model training directly on the device.



RESEARCH QUESTIONS

- **Knowledge Exchange:** What are the most efficient methods to implement seamless knowledge sharing between AI-enabled edge devices to enable machine learning algorithms while maintaining data privacy?
  - We aim to avoid sharing raw training data between nodes to minimize network load and enhance data privacy

- **Spatiotemporal Locality and Non-IID Data:** What strategies most effectively incorporate localized and non-IID data to improve accuracy in distributed machine learning models?
  - Evaluating the uncertainty at the level of both the individual agent and the aggregated model is essential

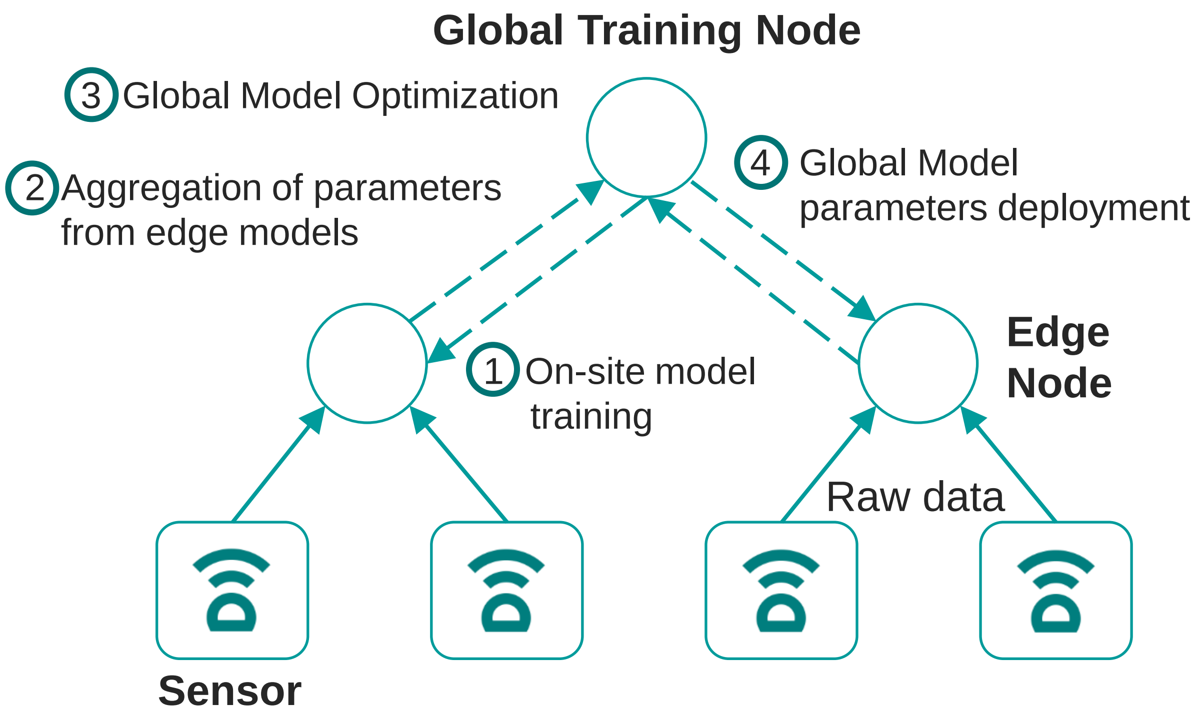
THE GOALS OF THIS REPORT

- Investigate the algorithms and methods for deploying distributed machine learning within the framework of autonomous, network-capable, sensor-equipped, AI-enabled edge devices.
- Within the framework of this study specifically, we focus on determining confidence levels in learning outcomes, considering the spatial and temporal variability of data sets encountered by independent agents.
- To achieve this, we address the following tasks:
  - Decouple the Distributed Neural Network Optimization (DiNNO) algorithm implementation into independent processes, enabling asynchronous network communication for distributed learning
  - Integrate distributed uncertainty estimation into the resulting models by applying Bayesian neural networks (BNN)
  - Implement and evaluate the proposed approaches within a case: simulation of robots navigating a 3D environment using the Webots platform, augmented with advanced LiDAR sensors for environmental mapping

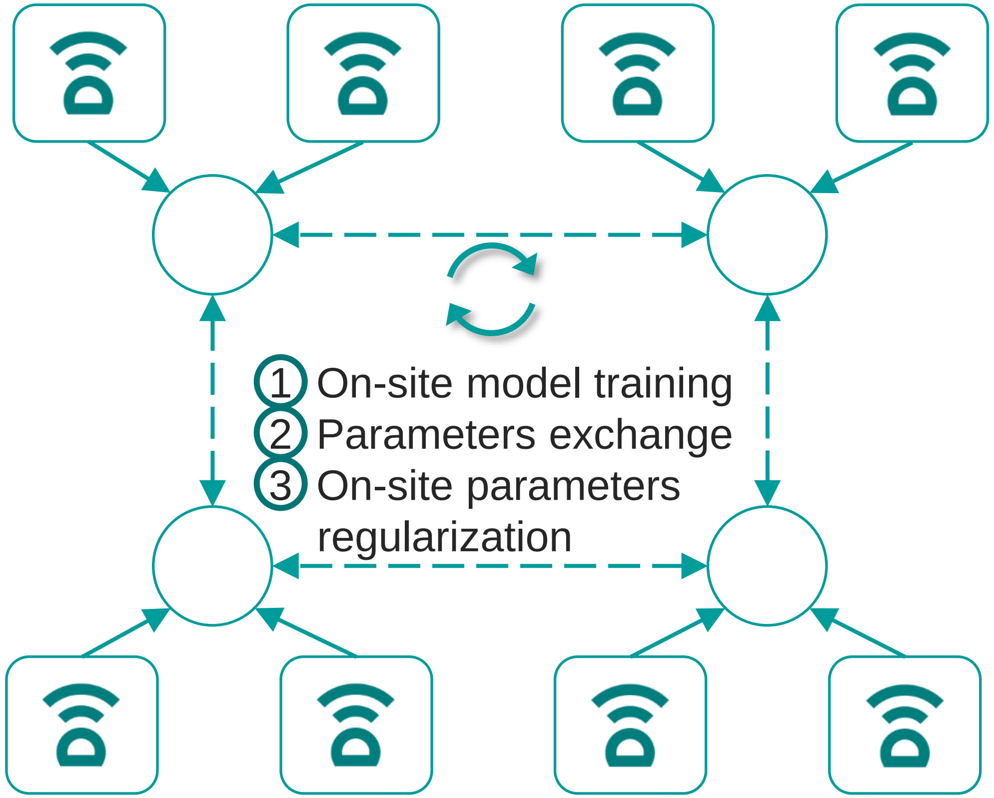
FEDERATED AND DECENTRALIZED (P2P) LEARNING

SAL SILICON AUSTRIA LABS

Federated Learning



Decentralized (P2P) Learning



ALTERNATING DIRECTION METHOD OF MULTIPLIERS (ADMM)

SAL SILICON AUSTRIA LABS

- ADMM is an algorithm that solves convex optimization problems by breaking them into smaller pieces, each of which are then easier to handle.
- We take the Distributed Neural Network Optimization (DiNNO) algorithm as a basis for our research

Algorithm 1 Distributed Neural Network Optimization (DiNNO)

Require: $\ell(\cdot)$, $\theta_{initial}$, $\mathcal{G}$, $\mathcal{D}$, ρ
for $i \in \mathcal{V}$ **do** ▷Initialize the iterates
  $p_i^0 = 0$ ▷Dual variable
  $\theta_i^0 = \theta_{initial}$ ▷Primal variable
end for
for $k \leftarrow 0$ to K **do** ▷Main optimization loop
  Communicate: send θ_i^k to neighbors $\mathcal{G}$
  for $i \in \mathcal{V}$ **do** ▷In parallel
    $p_i^{k+1} = p_i^k + \rho \sum_{j \in \mathcal{N}_i} (\theta_i^k - \theta_j^k)$
    $\psi^0 = \theta_i^k$
    for $\tau \leftarrow 0$ to B **do** ▷Approximate primal
      $\psi^{\tau+1} = \psi^\tau + G(\psi^\tau; \rho, p_i^{k+1}, \theta_i^k, \{\theta_j^k\}_{j \in \mathcal{N}_i}, \mathcal{D}_i)$
    end for
    $\theta_i^{k+1} = \psi^B$ ▷Update primal
  end for
end for
return $\{\theta_i^K\}_{i \in \mathcal{V}}$

J. Yu, J. A. Vincent and M. Schwager, "DiNNO: Distributed Neural Network Optimization for Multi-Robot Collaborative Learning," in IEEE Robotics and Automation Letters, vol. 7, no. 2, pp. 1896-1903, April 2022, doi: 10.1109/LRA.2022.3142402.

UNCERTAINTY ESTIMATION IN NN

- In a conventional neural network architecture, a linear neuron is characterized by a weight (w), a bias (b), and an activation function (f_{act}). Given an input x, a single linear neuron performs the following operation:

$$y = f_{act}\,(w \cdot\ x\ +\ b)$$

- Bayesian Neural Networks (BNNs) employ a Bayesian approach to train stochastic neural networks
- Instead of deterministic weights and biases, they utilize probability distributions, denoted $P(w)$ for weights and $P(b)$ for biases.
- Typically, these distributions are approximated as Gaussian, with mean and standard deviation derived from the training data. So, the operation of a Bayesian Linear neuron can be described as:

$$P(y|x) = f_{act}\left(\sum P(w)\ \times\ x + P(b)\right)$$

- For inference, BNNs might conduct multiple forward passes. The standard deviation of the inference values distribution indicate the model's uncertainty for each point in the input data space.

COLLABORATIVE MAPPING CASE

- We test our approach based on collaborative mapping task. This task involves deploying a network of independent, robotic edge devices (robots) at various starting points.
- Each device is tasked with building a coherent map of the environment, utilizing installed LiDAR, and exchanging knowledge about the environment with other devices.
- These devices are designed to update a local ML model with newly acquired data samples and implement inter-device communication via a network interface
- The CubiCasa5K data set was used as a reference for the floor plans generation

DECENTRALIZED STATE EXCHANGE ALGORITHM

- Original DiNNO implementation is a centralized learning framework that relies on sequential learning processes based on shared agents' memory.
- We have introduced an epoch-based algorithm to support the decentralized peer-to-peer exchange of NN parameters among agents. Generally, the following steps are implemented:
  1. Edge NN training using local data set
  2. P2P exchange of NN parameters
  3. Regularization of local NN parameters based on the parameters, received from the peers
- This version of the algorithm operates under the assumption that each message sent will eventually be received by its intended recipient.
- In that conditions, all the peers would eventually reach the NodeUpdate state and proceed to the next round of communication

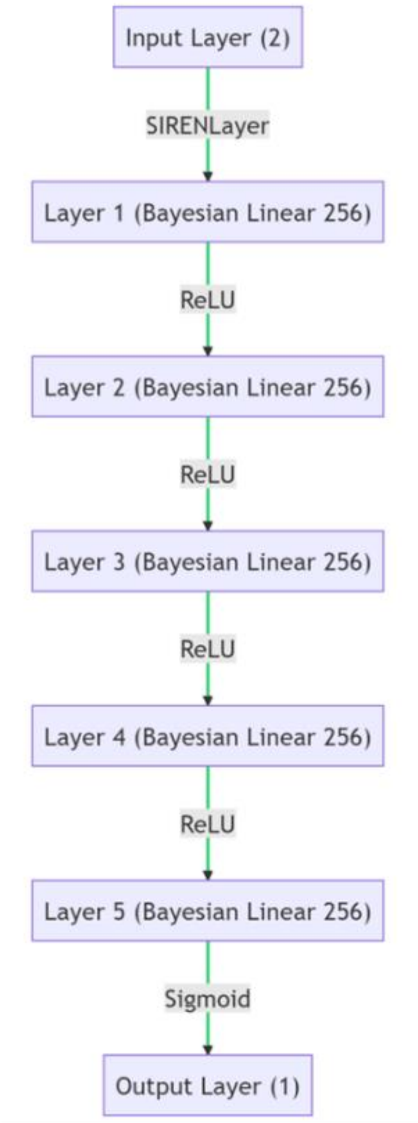
Algorithm 1. Peers State Exchange

```
Require: MaxRound, Socket, Id, State
Initialize: Round, PeerComplete[ ], PeerState[ ]
Message ← (State, 0)
SEND(Socket, Message, Id)
while Round < MaxRound do
    (Message, PeerId) ← RECEIVE(Socket)
    if Message is RoundComplete then
        PeerComplete[PeerId] ← TRUE
    else
        if Round < Message.Round then
            FINISHROUND
        end if
        PeerState[PeerId] ← Message.State
    end if
    if ∀s ∈ PeerState, s ≠ Ø then
        State ← NODEUPDATE(State, PeerState)
        ∀s ∈ PeerState, s ← Ø
        PeerCompleted[Id] ← TRUE
        PeerState[Id] ← State
        Message ← RoundComplete
        SEND (Socket, Message, Id)
    end if
    if ∀p ∈ PeerComplete, p = TRUE then
        FINISHROUND
    end if
end while
function FINISHROUND
    ∀p ∈ PeerComplete, p ← FALSE
    Round ← Round + 1
    Message.State ← State
    Message.Round ← Round
    SEND (Socket, Message, Id)
end function
```

IMPLEMENTATION OF BNN MODEL

SAL SILICON AUSTRIA LABS

- To address uncertainty estimation in the distributed mapping problem, we implement BNN model, introducing Bayesian Linear Layers in the NN architecture.
- The architecture of the BNN is detailed as follows
  - *Input Layer (2):* x, y – an input coordinate representing the global position on the environment map.
  - *SIRENLayer (256):* a layer with a sinusoidal activation function suitable for Neural Implicit Mapping.
  - *4 x Bayesian Linear Layers (256):* four Bayesian Linear layers with 256 nodes each, activated by the ReLU function. These layers are probabilistic and support uncertainty estimation.
  - *Output Layer (1):* a linear layer with one node activated by the Sigmoid function.
- This approach introduces probabilistic inference to the model, allowing for estimating uncertainty in the network's predictions.



BNN PARAMETERS REGULARIZATION

SAL SILICON AUSTRIA LABS

- To ensure correct regularization of the BNN parameters during the distributed learning regularization phase, Algorithm 2 has been developed to consider the semantics of median (μ) and standard deviation (ρ) parameters of BNN neurons.
- We utilize Kullback-Leibler Divergence (KL Divergence) for the regularization of BNN ρ-parameters between the models of individual actors.
- KL Divergence is employed to account for the difference between the Gaussian distributions that represent the parameters of the BNN. KL Divergence serves as a measure to quantify the dissimilarity between two probability distributions and can be generally computed as:

$$D_{KL}(g \parallel h) = \int g(x) \log \frac{g(x)}{h(x)} \mathrm{d}x$$

- Within the BNNs, applying KL Divergence helps quantify the deviation of the neural network's parameter distribution from a specified prior distribution

---

Algorithm 2. Optimization of BNN Parameters

Require: *Model*, $Optimizer_\mu$, $Optimizer_\rho$, W_μ, W_ρ, *Iter*, θ^{μ}_{reg}, θ^{ρ}_{reg}, $Duals_\mu$, $Duals_\rho$

for $i \leftarrow 1$ to *Iter* **do**

  Reset gradients of $Optimizer_\mu$ and $Optimizer_\rho$

  $PredLoss \leftarrow \text{ComputeLoss}(Model)$

  $\theta^{\mu}, \theta^{\rho} \leftarrow \text{ExtractParameters}(Model)$

  $Reg_\mu \leftarrow \text{L2Regularization}(\theta^{\mu}, \theta^{\mu}_{reg})$

  $Reg_\rho \leftarrow \text{D\_KL}(\theta^{\rho}, \theta^{\rho}_{reg})$

  $Loss_\mu \leftarrow PredLoss + \langle \theta^{\mu}, Duals_\mu \rangle + W_\mu \times Reg_\mu$

  $Loss_\rho \leftarrow \langle \theta^{\rho}, Duals_\rho \rangle + W_\rho \times Reg_\rho$

  $\text{UpdateParameters}(Optimizer_\mu, Loss_\mu)$

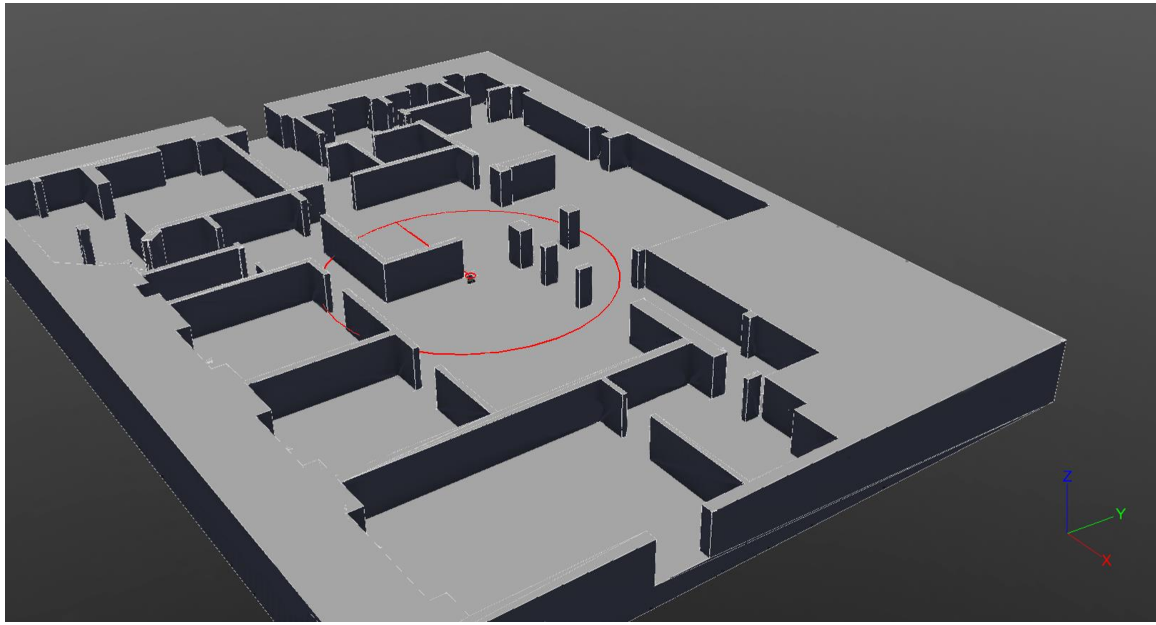
  $\text{UpdateParameters}(Optimizer_\rho, Loss_\rho)$

end for

SIMULATION ENVIRONMENT IMPLEMENTATION

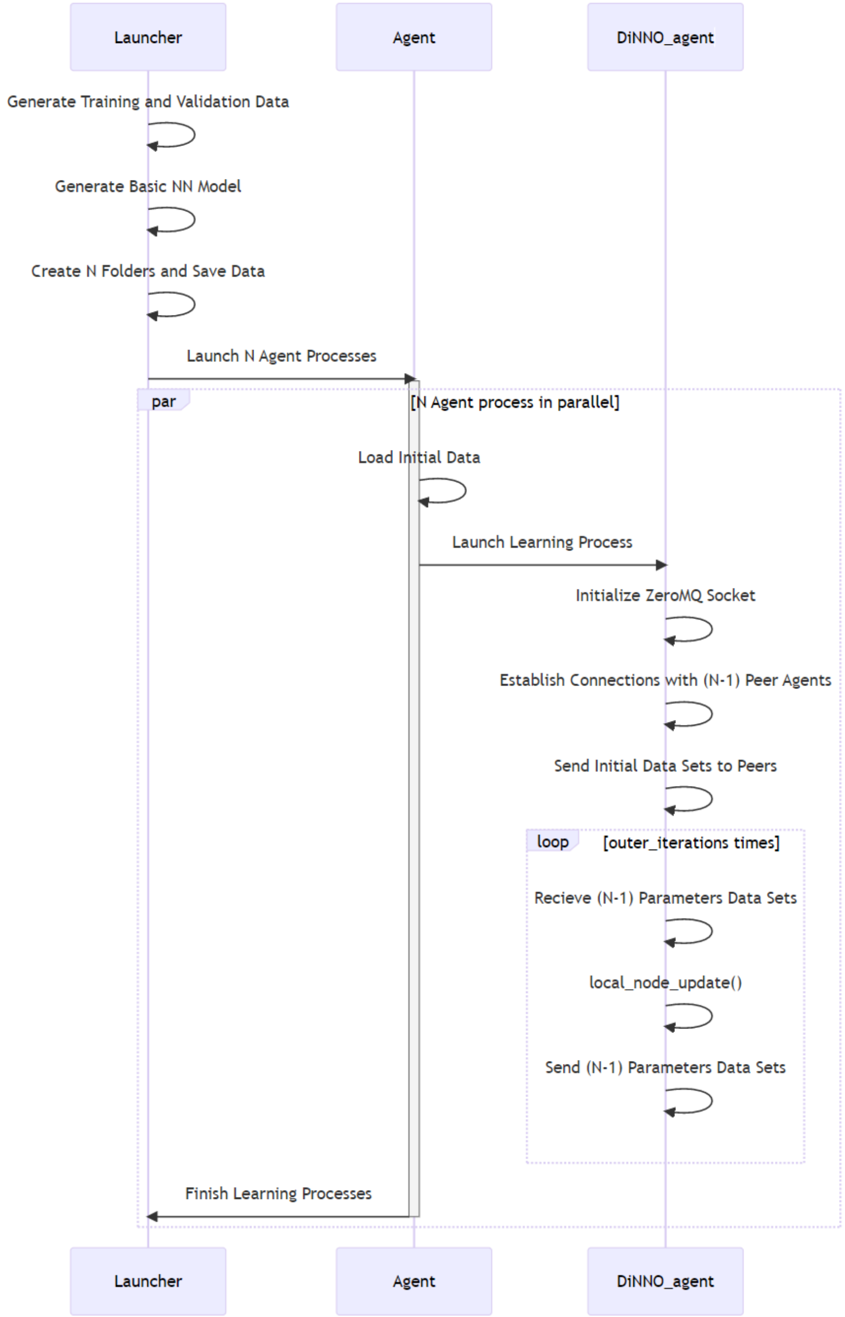
SAL SILICON AUSTRIA LABS

- Based on floor plans sourced from the CubiCasa5K dataset, we generated 3D interior models in STL format for robotic exploration
- To simulate the behavior of autonomous agents, these 3D interior models were imported into the Webots simulation platform where we deployed models of TurtleBot robots for navigation within these environments
- This methodology enabled us to use advanced LiDAR sensor models, incorporating realistic noise and measurement uncertainties into our experiments
- In this study, it is assumed that all robots can access global positioning information. Movement paths for the agents were pre-determined, enabling the generation of simulation programs for their traversal through the interiors



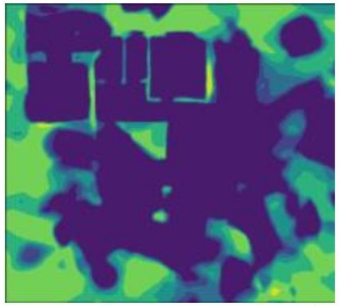
EXPERIMENT SETUP

- The experiment involves launching seven independent agents that gradually collect information from LiDAR sensors while exploring a virtual interior space
- Each agent runs as a separate Python process
- Agent communication is handled through direct TCP connections among the processes within the same virtual local network
- The ZeroMQ framework is used for asynchronous data exchange
- Containerization of agent processes is achieved using Singularity containers equipped with GPU access
- In the experiments outlined, we initiate all processes on GPU-enabled computing nodes managed by the SLURM workload manager

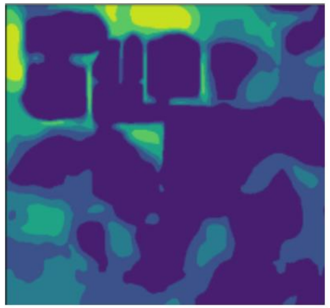


SINGLE-AGENT UNCERTAINTY ESTIMATION

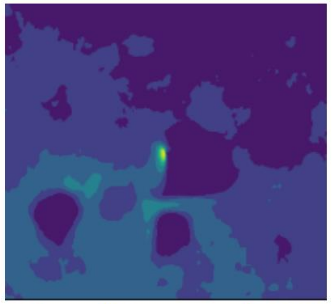
SAL SILICON AUSTRIA LABS



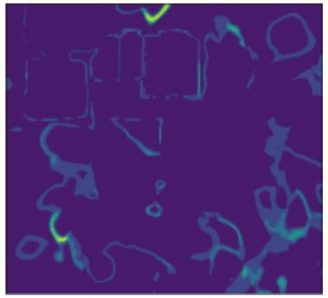
(a) Mean, $kl_{weight} = 10^{-4}$



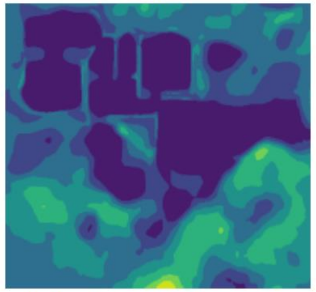
(b) Mean, $kl_{weight} = 5 \times 10^{-3}$



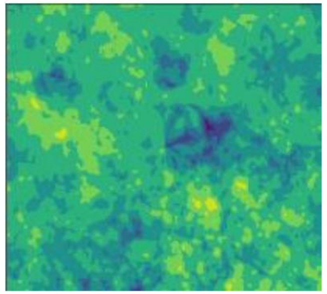
(c) Mean, $kl_{weight} = 5 \times 10^{-1}$



(d) Standard deviation, $kl_{weight} = 10^{-4}$



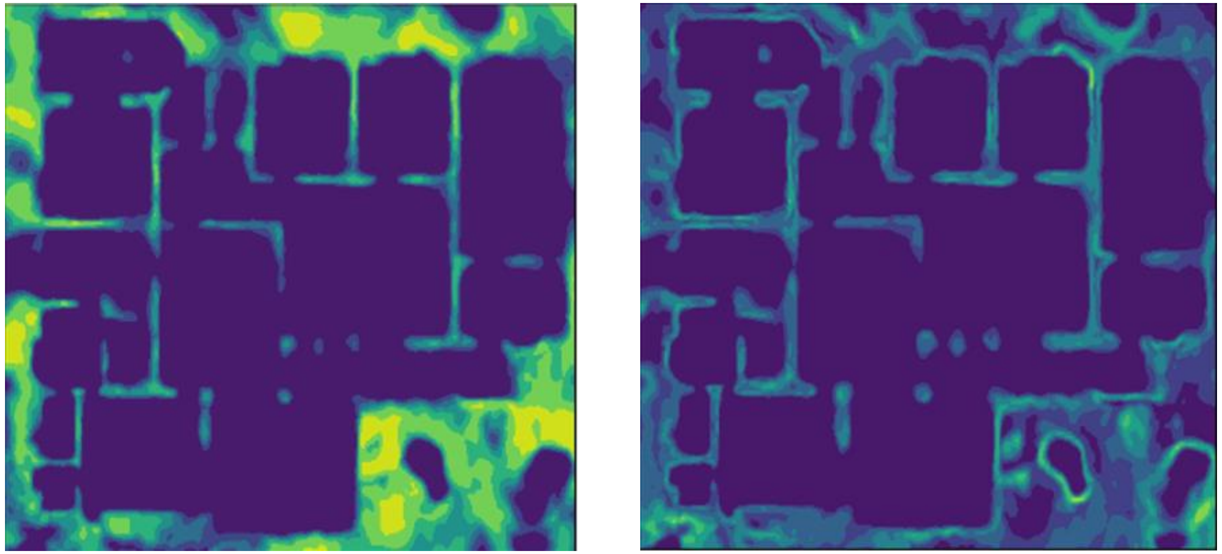
(e) Standard deviation, $kl_{weight} = 5 \times 10^{-3}$



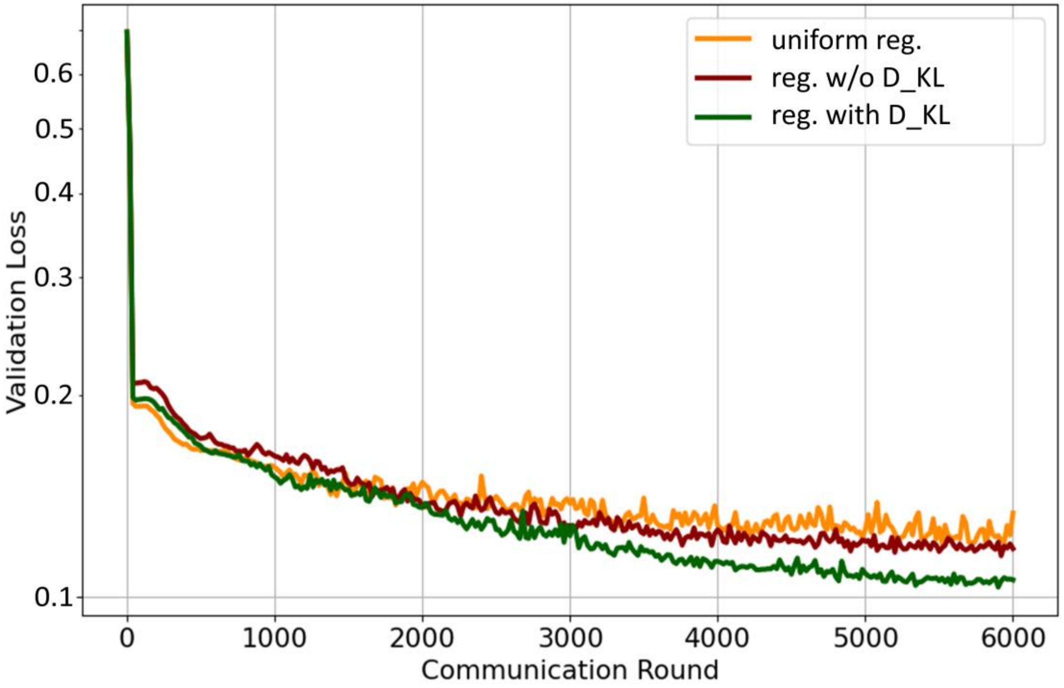
(f) Standard deviation, $kl_{weight} = 5 \times 10^{-1}$

- To generate outputs from the Bayesian neural network, 50 queries were made for each pair of input coordinates (x,y). Subsequently, a visualization was created to illustrate the mean values and standard deviations of the neural network responses.
- The kl_{weight} learning hyperparameter should be correctly “fine-tuned” if we want to distinguish the "hallucinations" of the neural network from areas with sufficient data

MULTI-AGENT BNN TRAINING



(a) Mean (b) Standard Deviation



- We evaluated the impact of different regularization approaches on the training quality of Bayesian neural networks within the decentralized environment
- We observe that applying Kullback–Leibler divergence for parameter regularization leads to a 12-30% decrease in the validation loss of the distributed BNN training compared to other regularization strategies

CONCLUSIONS

SAL SILICON AUSTRIA LABS

- We addressed a problem of uncertainty estimation within distributed machine learning based on AI-enabled edge devices:
  - We set up a simulation of a collaborative mapping problem using the Webots platform;
  - introduced an epoch-based algorithm to support the decentralized peer-to-peer exchange of NN parameters among agents;
  - and integrated distributed uncertainty estimation into our models by applying Bayesian neural networks.
- BNNs can effectively support uncertainty estimation in a distributed learning context.
- Applying Kullback–Leibler divergence for parameter regularization resulted in a 12-30% reduction in validation loss during the training of distributed BNNs compared to other regularization strategies.

FUTURE WORK

- We are currently exploring how distributed learning with BNNs can be tailored for embedded AI hardware.
- This would involve refining the NN architecture to suit the resource constraints of AI-enabled edge devices (such as Nvidia Jetson).
- We plan to compare the efficiently of distributed and decentralized NN training using Federated Learning, ADMM-based and Federated Distillation approaches, in cases of centralized and decentralized environments
- We also explore task management and offloading strategies within the multi-layered fog and hybrid edge-fog-cloud environments to improve computational efficiency and resource utilization

SAL
SILICON AUSTRIA LABS

THANK YOU!

GLEB.RADCHENKO@SILICON-AUSTRIA.COM

The research reported has been partly funded by the European Union's Horizon 2020 research and innovation program within the framework of Chips Joint Undertaking (Grant No. 101112268). This work has been supported by Silicon Austria Labs (SAL), owned by the Republic of Austria, the Styrian Business Promotion Agency (SFG), the federal state of Carinthia, the Upper Austrian Research (UAR), and the Austrian Association for the Electric and Electronics Industry (FEEI)