Uncertainty Estimation in Multi-Agent Distributed Learning for AI-Enabled Edge Devices

Gleb Radchenko[1][a] and Victoria Andrea Fill[2][b]
[1]*Silicon Austria Labs, Sandgasse 34, 8010 Graz, Austria*
[2]*FH Joanneum, Alte Poststraße 149, 8020 Graz, Austria*

Keywords: Edge Computing, Fog Computing, Machine Learning, IoT.

Abstract: Edge IoT devices, once seen as low-power units with limited processing, have evolved with introduction of FPGAs and AI accelerators, significantly boosting their computational power for edge AI. This leads to new challenges in optimizing AI for energy and network resource constraints in edge computing. Our study examines methods for distributed data processing with AI-enabled edge devices to improve collaborative learning. We focus on the challenge of assessing confidence in learning outcomes amid data variability faced by agents. To address this issue, we investigate the application of Bayesian neural networks, proposing a novel approach to manage uncertainty in distributed learning environments.

1 INTRODUCTION

Traditionally, Internet of Things (IoT) Edge devices have been perceived primarily as low-power components with limited capabilities for autonomous operations (Samie et al., 2016). However, in recent years, the focus of IoT research has shifted towards optimizing knowledge exchange and implementing AI and neural networks (NN) on edge devices. These advancements are largely due to the innovation of FPGAs and AI accelerators, which have exponentially increased the computational capabilities of Edge devices (Liang et al., 2023; Parmar et al., 2023; Wang et al., 2022).

This evolution raises critical questions that system developers should address:

- **Knowledge Exchange:** How can we implement seamless knowledge sharing between edge devices to refine machine learning algorithms while maintaining data privacy?
- **Resource Management:** What strategies can effectively manage the computational power of these increasingly autonomous, high-performance devices?
- **Spatiotemporal Locality:** How can we address the localized nature of data to ensure real-time or near-real-time task execution?

The challenges presented by limited resources on edge devices and the spatiotemporal locality of data are particularly significant. These issues require new approaches to manage computational capabilities and efficiently perform tasks in real-time or near-real-time modes.

The goal of this research is to investigate the algorithms and methods for deploying distributed machine learning within the framework of autonomous, network-capable, sensor-equipped, AI-enabled edge devices. Specifically, we focus on determining confidence levels in learning outcomes, considering the spatial and temporal variability of data sets encountered by independent agents. To address this issue, we investigate the potential of the Distributed Neural Network Optimization (DiNNO) algorithm (Yu et al., 2022), aiming to extend it for organizing distributed data processing and uncertainty estimation using Bayesian neural networks (BNN).

Within the scope of this paper, we explore the interaction of AI-enabled edge devices using the case of robotic platforms engaged in the task of

[a] https://orcid.org/0000-0002-7145-5630
[b] https://orcid.org/0009-0006-6289-0098

Radchenko, G. and Fill, V.
Uncertainty Estimation in Multi-Agent Distributed Learning for AI-Enabled Edge Devices.
DOI: 10.5220/0012728500003711

In *Proceedings of the 14th International Conference on Cloud Computing and Services Science (CLOSER 2024)*, pages 311-318
ISBN: 978-989-758-701-6; ISSN: 2184-5042

collaborative mapping. To achieve this, we need to address the following tasks:

- Implement a simulation of robots navigating a 3D environment using the Webots platform [1], augmented with advanced LiDAR sensors for detailed environmental mapping.
- Decouple the DiNNO algorithm implementation into independent processes, enabling asynchronous network communication for distributed learning.
- Integrate distributed uncertainty estimation into the resulting models by applying BNNs.

The rest of the paper is structured as follows. Section 2 is devoted to analyzing the state-of-the-art research, providing an overview of distributed machine learning methods, and implementing BNNs for uncertainty estimation. In Section 3, we present a collaborative mapping case. Section 4 introduces a distributed implementation of the DiNNO framework. Section 5 focuses on distributed uncertainty estimation, exploring techniques for integrating BNN into DiNNO. Implementation details and the evaluation of our approaches are detailed in Section 6, followed by Section 7, which offers our conclusions and discussion of future work directions.

2 RELATED WORK

2.1 Distributed Machine Learning Methods

Distributed machine learning (ML) algorithms, distinguished by their communication mechanisms, primarily support the exchange of model parameters, model outputs, or hidden activations. These exchanges can be enabled through peer-to-peer or client-server architectures (Park et al., 2021). The primary approaches utilized within these algorithms may be categorized as follows.

Federated Learning (FL) orchestrates the periodic transmission of local training parameters (e.g., weights and gradients of a NN) from workers to a central parameter server. This server then performs model averaging and disseminates the updated global model to the workers. Such a strategy not only may preserve data privacy by avoiding the need for raw data exchange but also may enhance communication efficiency through adjustable transmission intervals (Lim et al., 2020). The authors of (Abreha et al., 2022) identify FL as a solution to challenges in edge computing environments, such as unwanted bandwidth loss, data privacy issues, and legalization concerns. They highlight that FL allows for co-training models across distributed clients, such as mobile phones, automobiles, and hospitals, via a centralized server while maintaining data localization.

The authors of (Nguyen et al., 2022) propose an extension of the FL model, called FedFog, designed to enable FL over a wireless fog-cloud system. The authors address key challenges such as non-identically distributed data and user heterogeneity. The FedFog algorithm performs local aggregation of gradient parameters at fog servers and a global training update in the cloud.

Alternating Direction Method of Multipliers (ADMM)-derived Methods (Boyd, 2010) (such as DiNNO (Yu et al., 2022), GADMM, and CADMM) aim to implement the distributed learning in the absence of a central coordinating entity by enabling communication directly between the neighboring worker nodes in a peer-to-peer (P2P) manner. One critical issue of FL and such P2P learning methods is that the communication overhead is proportional to the number of model parameters, limiting their efficacy in supporting deep NNs (Elgabli et al., 2020).

Federated Distillation utilizes the exchange of model outputs, which are significantly lower in dimensionality compared to the full model sizes for distributed learning. In this approach, each worker performs local iterations based on its individual loss function. This process is enhanced with a regularization component that measures the discrepancy between the worker's predicted output for a specific training sample and the aggregated global output for the same class. A widely known application of knowledge distillation is model compression, through which the knowledge of a large pretrained model may be transferred to a smaller one (Ahn et al., 2019).

Split Learning (SL) partitions multi-layer NN into segments, thus making it possible to train large-sized deep NN that exceed the memory capacities of a single edge device. This approach divides the NN into *lower NN segments* on the workers' devices, each containing raw data and a shared *upper NN segment* hosted on a parameter server. The *NN cut layer* is a boundary between the lower and upper NN segments. Workers compute the activations at the NN cut layer and send these activations to the parameter server. The parameter server uses these activations as inputs for the upper NN segment to continue the forward

[1] https://www.cyberbotics.com

pass, compute the loss, and initiate the backward pass. Gradients calculated at the cut layer are then transmitted back to the workers, allowing them to update the weights of the lower NN segments. However, the efficacy of SL in terms of communication is subject to ongoing discussion (Koda et al., 2020).

2.2 Uncertainty Estimation and Bayesian Neural Networks

2.2.1 Bayesian Neural Networks

In a conventional NN 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:

$$y = f_{act}(w \cdot x + b) \tag{1}$$

where y is the output of the neuron.

Bayesian Neural Networks (BNNs) employ a Bayesian approach to train stochastic NNs (Jospin et al., 2022). 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. Hence, a Bayesian neuron outputs a range of possible values, not just one. So, the operation of a Bayesian Linear neuron can be described as:

$$P(y|x) = f_{act}\left(\sum P(w) \times x + P(b)\right) \tag{2}$$

In a BNN, the Gaussian distributions for weights and biases may be defined by the mean μ and the standard deviation σ. For weights, the distribution $P(w)$ is modeled as a Gaussian distribution with a mean w_μ and a standard deviation w_σ, where:

$$w_\sigma = \log(1 + e^{w_\rho}) \tag{3}$$

The parameter w_ρ ensures that the standard deviation is always positive. Similarly, the distribution $P(b)$ for biases is represented as a Gaussian distribution with a mean b_μ and a standard deviation b_σ , where:

$$b_\sigma = \log(1 + e^{b_\rho}) \tag{4}$$

During the forward pass of a Bayesian neuron, these distributions are sampled to obtain a weight and bias for each neuron. The sampled weights and biases are then used to compute the neuron's output. The parameters w_μ , w_ρ , b_μ , b_ρ are learned during NN training to optimize the network's performance.

BNNs, unlike traditional NNs that use a single forward pass, can perform multiple forward passes and calculate the mean and standard deviation of the outputs. These statistics can reveal the model's uncertainty for each input data point, varying with the problem the NN tackles.

2.2.2 Kullback-Leibler Divergence

Kullback-Leibler Divergence (KL Divergence) (Claici et al., 2020; Kullback & Leibler, 1951) 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:

$$D_{KL}(g \parallel h) = \int g(x) \log \frac{g(x)}{h(x)} \mathrm{d}x \tag{5}$$

where $g(x)$ and $h(x)$ are two probability density functions defined over the same support. The concept of "expected excess surprise" captures the core idea behind KL Divergence, reflecting the expected degree of "surprise" encountered when another "model" distribution approximates an actual distribution. As outlined by (Belov & Armstrong, 2011), if $N_0(\mu_0\,, \sigma_0)$ and $N_1(\mu_1\,, \sigma_1)$ are two normal probability density functions, equation (5) may be reduced to:

$$D_{KL}(N_0 \parallel N_1) = \frac{1}{2}\left(\log\frac{\sigma_1^2}{\sigma_0^2} + \frac{\sigma_0^2 + (\mu_0 - \mu_1)^2}{\sigma_1^2} - 1\right) \tag{6}$$

Within the BNNs, applying KL Divergence helps quantify the deviation of the neural network's parameter distribution from a specified prior distribution. The overall loss in a BNN model is generally expressed as:

$$total_{loss} = base_{loss} + kl_{weight} \times kl_{loss} \tag{7}$$

where $base_{loss}$ refers to the conventional loss function, such as Binary Cross-Entropy or Mean Squared Error; kl_{weight} is a hyperparameter that enables adjusting of the level of uncertainty in the model's outcomes; and kl_{loss} is the sum of the KL Divergence between the distribution of BNN modules $N_0(\mu_0, \sigma_0)$ and a predefined normal distribution $N_1(\mu_1, \sigma_1)$.

3 COLLABORATIVE MAPPING CASE

As a case study for a distributed AI application operating within the Edge Cloud, we have chosen a

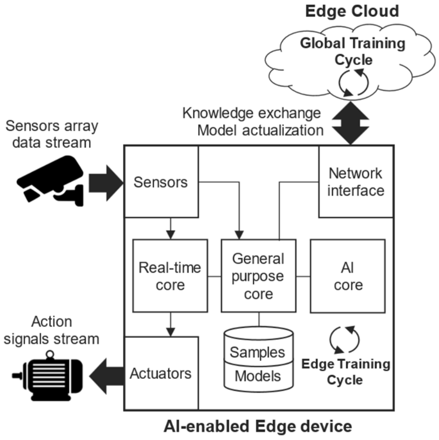


Figure 1: Components of AI-Enabled edge device.

collaborative environment mapping problem. 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 sensors, 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 facilitate inter-device communication via a network interface (see Figure 1). Equipped with computational cores dedicated to specific responsibilities, the devices feature:

- **Real-time core** for immediate data processing and direct control of actuators.
- **General-purpose core** for overall device control.
- **AI core** to support an edge training cycle.

The Distributed Neural Network Optimization (DiNNO) (Yu et al., 2022) algorithm is employed as the principal method for addressing the distributed machine learning problem in our study. DiNNO enables the implementation of distributed learning within a network of independent agents. These agents are identical robotic platforms regarding computational capabilities and sensor equipment. Each robot possesses its own sensor data set and maintains an individual version of the NN. These robots refine their NN models throughout the learning phase using fresh sensory inputs and then exchange NN parameters. This iterative process ensures that, over time, all agents align on a harmonized NN representation. The CubiCasa5K data set (Kalervo et al., 2019) was used as a reference for the floor plans generation (see Figure 2).

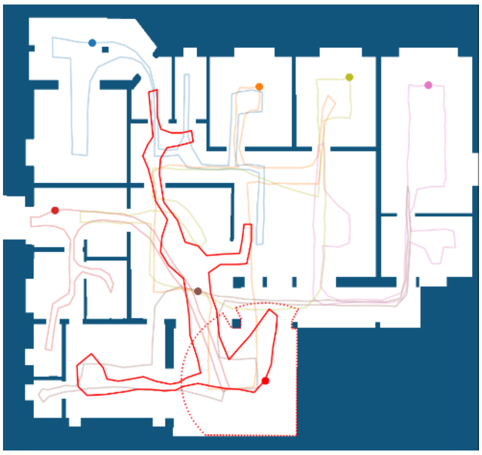


Figure 2: Visualization of the environment map, including starting points, exploration pathways and lidar range for the robotic agents, as described by (Yu et al., 2022).

4 DISTRIBUTED EDGE LEARNING APPROACH

To enhance the DiNNO algorithm for edge computing, we moved from a centralized learning framework that uses sequential processes and shared memory to a distributed version. In this setup, each agent operates independently, processing local LiDAR data, optimizing NN parameters locally, and exchanging updated NN parameters with peers through messages.

We have introduced an epoch-based decentralized consensus algorithm to support the decentralized peer-to-peer exchange of NN parameters among agents (see Algorithm 1). The maximum amount of synchronization epochs (*MaxRound*), network socket (*Socket*), unique peer identifier (*Id*), and the initial state of the NN parameters (*State*) are given as inputs for the algorithm. *PeerComplete[ ]* and *PeerState[ ]* structures are utilized to track the completion status and states of peers, respectively. The core of the algorithm lies in the exchange of two types of messages:

- The *RoundComplete* message indicates the completion of a round by a peer.
- The State message contains the peer's state for the current round.

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
```

The introduction of the *RoundComplete* message alongside the round finalization logic addresses the issues introduced by the latency in the message delivery. These challenges include out-of-order messages, delayed status updates, and desynchronization between rounds. A *RoundComplete* message is sent by a peer only after it has received all *State* messages from the other peers. It ensures that a peer only advances to the next round once all peers have completed the current round, indirectly handling message delays by waiting for all messages to be received before proceeding.

As for the out-of-order messages, the agent checks if the received *State* message is from the future round. If so, it triggers the FINISHROUND function to ensure the peer catches up to the correct round. This mechanism helps in managing out-of-order deliveries due to latency.

This algorithm version assumes all messages will eventually reach their intended recipients, not considering agent malfunctions, halts, or permanent network equipment failures that could cause message loss or communication breakdowns.

5 DISTRIBUTED UNCERTAINTY ESTIMATION

To address uncertainty estimation in the distributed mapping problem, we incorporate a BNN by replacing the conventional linear layers in the NN with Bayesian Linear Layers. 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 modification introduces probabilistic inference to the model, allowing for estimating uncertainty in the network's predictions.

BNNs stand out for their ability to measure prediction uncertainty, unlike deterministic networks that give a single output. By performing multiple forward passes to compute outputs' mean and standard deviation, BNNs offer insight into the model's confidence for each mesh grid point. 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 KL Divergence, as detailed in Equation (6), for the regularization of BNN ρ-parameters between the models of individual actors.

Algorithm 2: Optimization of BNN Parameters.

```
Require: Model, Optimizer_μ, Optimizer_ρ, W_μ, W_ρ, Iter,
$\theta^{\mu}_{reg}$, $\theta^{\rho}_{reg}$, Duals_μ, Duals_ρ
for i ← 1 to Iter do
    Reset gradients of Optimizer_μ and Optimizer_ρ
    PredLoss ← COMPUTELOSS(Model)
    $\theta^{\mu}$,$\theta^{\rho}$ ← EXTRACTPARAMETERS(Model)
    $Reg_{\mu}$ ← L2REGULARIZATION($\theta^{\mu}$,$\theta^{\mu}_{reg}$)
    $Reg_{\rho}$ ← D_KL($\theta^{\rho}$,$\theta^{\rho}_{reg}$)
    $Loss_{\mu} \leftarrow PredLoss + \langle \theta^{\mu}, Duals_{\mu} \rangle + W_{\mu} \times Reg_{\mu}$
    $Loss_{\rho} \leftarrow \langle \theta^{\rho}, Duals_{\rho} \rangle + W_{\rho} \times Reg_{\rho}$
    UPDATEPARAMETERS($Optimizer_{\mu}$,$Loss_{\mu}$)
    UPDATEPARAMETERS ($Optimizer_{\rho}$,$Loss_{\rho}$)
end for
```

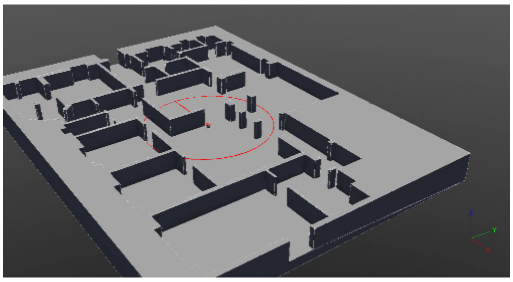


Figure 3: Visualization of the 3D model of the environment, generated from the floor plan, showcasing a LiDAR-equipped TurtleBot navigating the space in a Webots simulation.

6 IMPLEMENTATION AND EVALUATION

Using floor plans from the CubiCasa5K dataset, we created STL 3D interior models for robotic exploration. These models were imported into Webots to simulate TurtleBot robots navigating these spaces, as illustrated in Figure 3. Webots allowed us to implement 3D robotic systems and environments with typical sensors like cameras and LiDAR, enabling realistic sensor noise and measurement uncertainties in our experiments.

This study assumes that all robots have access to global positioning information. Agents' movement paths were predefined to create simulation programs for their interior navigation. During navigation, LiDAR sensor data collection is simulated as a Webots data stream.

The experiment involves launching seven independent agents that gradually collect information from LiDAR sensors while exploring a virtual interior space. This paper examines the analysis of data sets collectively collected and processed after the agents completed their traversal. This method is necessary because NN training requires significant energy, which might not be accessible to autonomous edge devices in mobile investigation mode.

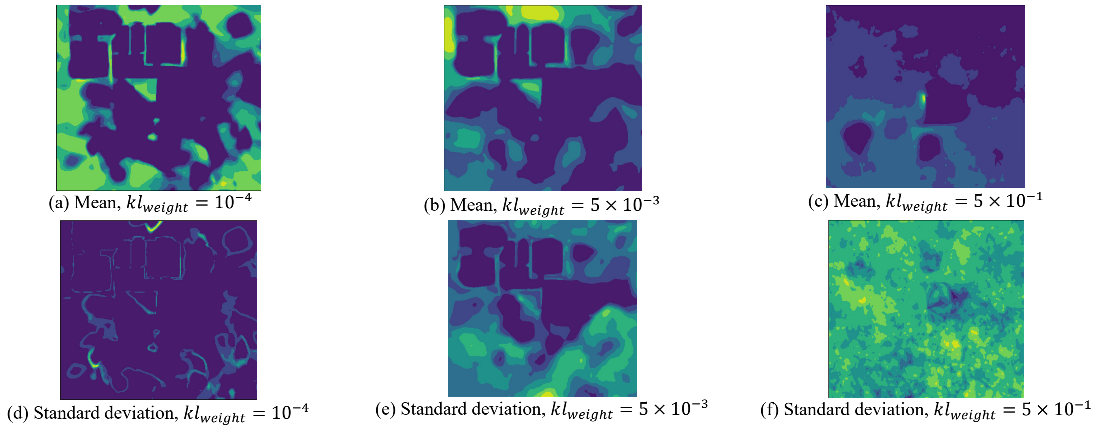
Each agent runs as a separate Python process. Agent communication occurs via direct TCP connections between processes on the same virtual local network. The ZeroMQ library is used for asynchronous data exchange. Containerization of agent processes is achieved using Singularity containers equipped with GPU access. In the experiments outlined, we initiate all processes on GPU-enabled computing nodes managed by the SLURM workload manager.

6.1 Single-Agent Uncertainty Estimation

We conducted a series of experiments with a single isolated agent to evaluate the effectiveness of the BNN architecture proposed in Section 5 for estimating uncertainty in NN outcomes and the impact of the kl_{weight} parameter from Equation (7). The agent was trained exclusively on local data during the experiment without exchanging information with other agents. The visualization of the training results is presented in Figure 4.

To generate outputs from the BNN, 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 NN responses.

It was observed that a low value of the kl_{weight} parameter leads to a low variance in the NN's results, which does not allow for distinguishing the "hallucinations" of the NN from areas with sufficient data to form a general understanding of the environment. Conversely, a high kl_{weight} parameter



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

Figure 4: Comparative visualization of single-agent uncertainty estimation.

value results in excessive noise and a high degree of uncertainty in the NN's results. Therefore, to ensure that the BNN provides an effective assessment of uncertainty, fine-tuning of the kl_{weight} parameter during the training process is required.

6.2 Multi-Agent Uncertainty Estimation

To assess the effectiveness of BNNs in estimating uncertainty within distributed, decentralized learning environments, a series of experiments were conducted. These experiments aimed to evaluate the impact of different regularization approaches on the training quality of BNNs. The validation loss was evaluated in the context of the following regularization strategies:

- Uniform L2 regularization of NN parameters without making distinctions between parameter types;
- Separate regularization of conventional and BNN parameters, applying L2 regularization for both;
- Separate regularization of conventional and BNN parameters, utilizing L2 regularization for conventional parameters and Kullback–Leibler divergence for Bayesian parameters (see Algorithm 2).

The results of the evaluation are presented in Figure 5. We observe that applying Kullback–Leibler divergence for parameter regularization (Algorithm 2) leads to a 12-30% decrease in the validation loss of the distributed BNN training compared to other regularization strategies. Additionally, this approach enhances the stability of the training process. The visualization of the outcome of decentralized BNN training according to Algorithm 2 is presented in Figure 6.

7 CONCLUSIONS

Within the scope of this paper, 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 decentralized consensus 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.

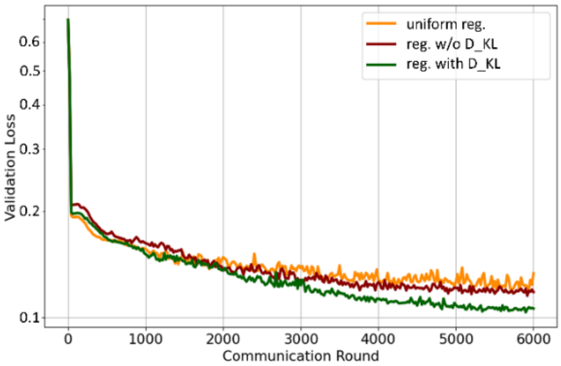


Figure 5: Comparison of validation loss during distributed BNN training 1) with uniform L2 regularization (*uniform reg.*); 2) separate L2 regularization (*reg. w/o D_KL*); 3) Kullback-Leibler divergence for regularization of BNN ρ-parameters (*reg. with D_KL*).

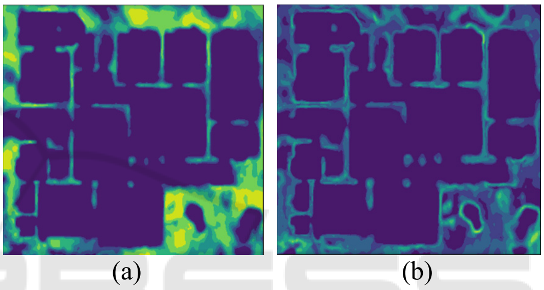


Figure 6: Visualization of the decentralized BNN training results according to Algorithm 2: a) mean; b) standard deviation.

Our experiments indicate that BNNs can effectively support uncertainty estimation in a distributed learning context. However, to ensure an effective assessment of uncertainty by the BNN, we highlighted the need for precise tuning of the learning hyperparameters during training. We also determined that 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.

For future work, we suggest 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. We also plan to explore task management and offloading strategies within the multi-layered fog and hybrid edge-fog-cloud environments to improve computational efficiency and resource utilization.

ACKNOWLEDGMENTS

The research reported in this paper 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).

REFERENCES

Abreha, H. G., Hayajneh, M., & Serhani, M. A. (2022). Federated Learning in Edge Computing: A Systematic Survey. In *Sensors* (Vol. 22, Issue 2).

Ahn, J. H., Simeone, O., & Kang, J. (2019). Wireless Federated Distillation for Distributed Edge Learning with Heterogeneous Data. *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, *2019-September*.

Belov, D. I., & Armstrong, R. D. (2011). Distributions of the Kullback–Leibler divergence with applications. *British Journal of Mathematical and Statistical Psychology*, *64*(2), 291–309.

Boyd, S. (2010). *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers* (Vol. 3).

Claici, S., Yurochkin, M., Ghosh, S., & Solomon, J. (2020). Model fusion with kullback-leibler divergence. *37th International Conference on Machine Learning, ICML 2020*, *PartF168147-3*.

Elgabli, A., Park, J., Bedi, A. S., Bennis, M., & Aggarwal, V. (2020). GADMM: Fast and Communication Efficient Framework for Distributed Machine Learning. *Journal of Machine Learning Research*, *21*.

Jospin, L. V., Laga, H., Boussaid, F., Buntine, W., & Bennamoun, M. (2022). Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users. *IEEE Computational Intelligence Magazine*, *17*(2), 29–48.

Kalervo, A., Ylioinas, J., Häikiö, M., Karhu, A., & Kannala, J. (2019). CubiCasa5K: A Dataset and an Improved Multi-task Model for Floorplan Image Analysis. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *11482 LNCS*, 28–40.

Koda, Y., Park, J., Bennis, M., Yamamoto, K., Nishio, T., Morikura, M., & Nakashima, K. (2020). Communication-efficient multimodal split learning for mmWave received power prediction. *IEEE Communications Letters*, *24*(6).

Kullback, S., & Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, *22*(1), 79–86.

Liang, Q., Hanafy, W. A., Ali-Eldin, A., & Shenoy, P. (2023). Model-driven Cluster Resource Management for AI Workloads in Edge Clouds. *ACM Transactions on Autonomous and Adaptive Systems*, *18*(1), 1–26.

Lim, W. Y. B., Luong, N. C., Hoang, D. T., Jiao, Y., Liang, Y.-C., Yang, Q., Niyato, D., & Miao, C. (2020). Federated Learning in Mobile Edge Networks: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials*, *22*(3), 2031–2063.

Nguyen, V. D., Chatzinotas, S., Ottersten, B., & Duong, T. Q. (2022). FedFog: Network-Aware Optimization of Federated Learning Over Wireless Fog-Cloud Systems. *IEEE Transactions on Wireless Communications*, *21*(10).

Park, J., Samarakoon, S., Elgabli, A., Kim, J., Bennis, M., Kim, S.-L., & Debbah, M. (2021). Communication-Efficient and Distributed Learning Over Wireless Networks: Principles and Applications. *Proceedings of the IEEE*, *109*(5), 796–819.

Parmar, V., Sarwar, S. S., Li, Z., Lee, H.-H. S., Salvo, B. De, & Suri, M. (2023). Exploring Memory-Oriented Design Optimization of Edge AI Hardware for Extended Reality Applications. *IEEE Micro*, *43*(6), 40–49.

Samie, F., Tsoutsouras, V., Bauer, L., Xydis, S., Soudris, D., & Henkel, J. (2016). Computation offloading and resource allocation for low-power IoT edge devices. *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, 7–12.

Wang, B., Dong, K., Zakaria, N. A. B., Upadhyay, M., Wong, W.-F., & Peh, L.-S. (2022). Network-on-Chip-Centric Accelerator Architectures for Edge AI Computing. *2022 19th International SoC Design Conference (ISOCC)*, 243–244.

Yu, J., Vincent, J. A., & Schwager, M. (2022). DiNNO: Distributed Neural Network Optimization for Multi-Robot Collaborative Learning. *IEEE Robotics and Automation Letters*, *7*(2), 1896–1903.