Security
of Infocommunication Systems
and Internet of Things

2026 Vol 4, No 1

https://doi.org/10.31861/sisiot2026.1.01016

Received 14 May 2026; revised 19 June 2026; accepted 23 June 2026; published 30 June 2026

Feature Restoration for Distributed Neural Network Inference in Edge Systems

Viacheslav Antsybor* and Oleksandr Verba

Department of Computer Engineering, Faculty of Informatics and Software Engineering, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine

*Corresponding author (E-mail: antsybor.v.ye.-io51f@edu.kpi.ua)

ABSTRACT The growing volume of data generated by IoT devices has made deep neural networks an important tool in IoT and edge systems. The considerable computational and memory costs of deep neural networks made them difficult to run on constrained devices, which are limited by memory, computational power, and energy. However, IoT devices are usually connected to a network of other devices, which can help reduce the load on a single device and enable complex models to operate closer to the data source via distributed inference. Local networks used in such cases are usually wireless, so packet loss, interference, unstable bandwidth, and node inaccessibility are inherent to this system type. Such problems can lead to loss of the intermediate data representation during transmission and reduced model classification accuracy. This study suggests a lightweight restoration layer to reconstruct lost intermediate features during distributed neural network inference. The method defines feature loss as an inpainting problem. An autoencoder-based module reconstructs missing values using a binary mask indicating which components were received and which were lost and passes the results to the classifier. The method was tested by using a pretrained ResNet-18 model with frozen weights. The module was trained with error rates ranging from 0.0 to 0.5, in steps of 0.1, on a preprocessed 1000-class subset of the ImageNet-1k dataset. The Top-1 accuracy at the 50% feature-loss point, compared to a clean baseline, decreased from 69.73% to 52.84% without restoration, whereas the proposed layer increased it to 62.89%. Under the same circumstances, the Top-5 metric increased from 77.82% to 85.46% at the 50% feature-loss point, demonstrating a clear improvement. As shown by the results, the proposed layer exhibits a moderate reduction in accuracy under feature-loss conditions, compared to the larger drop observed in the model without feature restoration. The proposed approach can be used to improve model correctness by reducing losses in unstable wireless environments, which can help with implementing distributed inference in real-time systems with limited resources.

KEYWORDS distributed inference, edge computing, feature restoration, inpainting autoencoder.

I. INTRODUCTION

In recent years, interest in using Deep Neural Networks (DNNs) in the Internet of Things (IoT) field has increased. The reason for the attention to this subject is the growing volume of data from IoT and handheld devices, as well as DNNs' ability to extract patterns, detect anomalies, and process large volumes of data. On the other hand, high decision-making accuracy comes at the cost of high computational complexity, which demands large memory and high energy consumption. These properties create direct conflicts with the typical operating conditions of IoT devices, which are often constrained by battery capacity, processor performance, memory size, and requirements for long self-sufficient operation.

To reduce this conflict, different approaches are used, including model compression, cloud-based inference, edge inference, and distributed inference. The model compression approach is designed to reduce model size and computational cost at the expense of accuracy [1]. Cloud or edge offloading assigns part of the computation from IoT devices to more powerful servers, reducing the local computational load [2]. At the same time, this solution has its own issues, including dependence on connectivity, additional latency, and privacy risks. Therefore, collaborative execution of neural network inference near the data source remains an important research direction for IoT and edge systems.

Distributed inference is an approach that uses multiple nearby devices, or edge nodes, to execute neural network inference collaboratively. It allows splitting the inference process across local computational nodes (IoT devices) rather than sending all data to a remote cloud server [1, 3]. In this way, distributed inference can reduce the load on individual devices, speed up inference, and enhance data privacy while leveraging the capabilities and accuracy of larger DNN models [1]. Existing distributed inference frameworks demonstrate that workload partitioning and local device collaboration can improve inference performance in edge clusters [4, 5]. However, such a computation organization also introduces reliability issues because IoT systems often communicate over wireless channels with limited bandwidth, interference, packet loss, temporary disconnections, and variable transmission delays [6].

This paper proposes a lightweight restoration component based on an autoencoder to mitigate the loss of classification accuracy caused by partial loss of intermediate representations during distributed neural network inference.

II. CURRENT STATE OF DISTRIBUTED INFERENCE ON IOT DEVICES

The rapid development of the Artificial Intelligence of Things field and the increasing volume of data generated by connected devices [2] have intensified research interest in deploying deep neural network inference closer to the data source. IoT devices are typically limited by on-board memory, processing capability, and energy supply. However, they often operate in a local network that includes gateways, smartphones, single-board computers, edge nodes, and other nearby devices. Such an environment enables offloading part of the computations to a group of locally connected devices. Distributed inference aims to overcome the limitations of a single IoT device by moving computation closer to the data source and leveraging the combined resources of nearby devices or edge nodes. Existing research in this area focuses mainly on partitioning strategies, workload assignment, communication reduction, and runtime scheduling.

A. Inference parallelism strategies. Model partitioning is a major challenge in distributed inference. The partitioning strategy determines how computing time, memory, and network communication are utilized. In general, a trained neural network can be parallelized using three main strategies, as illustrated in Fig. 1: pipeline parallelism, model parallelism, and data parallelism.

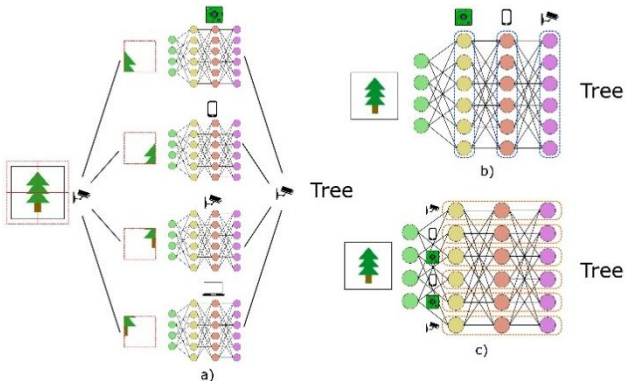

FIG. 1. DNN parallelism strategies.

Data parallelism (Fig. 1a) does not split the model itself. Input data is split into parts, which are handed to devices running the same model [3]. This approach has a simple structural composition and can improve the data throughput. At the same time, this approach has a crucial limitation on the model's characteristics due to the computational and memory capacity of a single device.

In the pipeline parallelism strategy (Fig. 1b), the pretrained model is split up into layers or groups of consecutive layers. These subnetworks are assigned to different devices, so the input is processed sequentially. Each device computes its part of the model and transfers the intermediate representation to the next device in the pipeline [7]. This way, memory and computational power usage for devices in a group can be reduced. However, the strategy requires a strict chain of devices in the pipeline, so if any device fails, the inference process will be interrupted.

Model parallelism (Fig. 1c) splits the computation inside a layer or fused group of layers, allowing devices to process different parts of the same inference stage in parallel. This distribution of computation can reduce inference latency and enable the execution of large layers that do not fit within a single IoT device's resource budget [8]. The efficiency of this parallelism model depends on the balance between communication overhead and parallel computation speedup, as the intermediate data representation must be synchronized and merged at every dependency point in the DNN during inference.

B. Practical distributed inference frameworks. EdgeLD demonstrates a pipeline-parallel architecture inspired by the MapReduce approach, with one group leader (GL) and follower nodes (FN) [5]. To make computation more effective, EdgeLD calculates the computational and communicational capabilities of each follower node and then distributes layers or fused layers based on the capabilities evaluation results. While balancing and layer fusion separately yielded small improvements, their combination reduced inference time and memory usage per device across clusters of different sizes.

Data parallelism itself does not improve the capability of the device group to use a larger model. EdgeCI extends this approach by combining auction-based workload assignment with fused-layer parallelization. Its auction mechanism assigns partitions of feature maps to suitable IoT devices, while the dynamic-programming-based fused-layer strategy reduces synchronization frequency by grouping layers into executable blocks [4]. Yuan and Li proposed a data-parallel strategy for optimizing the workload distribution algorithm [9]. They presented a new algorithm for data splitting and assigning in a heterogeneous edge system based on energy consumption and communication latency characteristics.

Model parallelism is a demanding strategy that, on the other hand, has a high level of model splitting and parallel computation. Du et al. note that CNN models are tightly coupled structures, so distributed DNNs require frequent data exchange to keep robust feature maps at every stage of inference [10]. DeCNN proposes three-scheme optimizations. At the structure-level, DeCNN uses group convolution and channel shuffle to decouple convolutional layers. At the partitioning level, DeCNN uses inter-channel partitioning for convolutional layers and input-based partitioning for fully connected layers. At the communication level, Du et al. propose an inter-sample parallelism to hide communication latency.

C. Distributed inference robustness. The previously mentioned works focus on distributing the computational payload across devices using different strategies. They assume a stable connection, which is rarely the case in real-world scenarios. The packet-loss-tolerant split inference method proposes to focus on model fault tolerance by additional training of the DNN on splitting points [6]. Itahara et al. suggest reducing the need for data retransmission in case of packet loss by training the DNN at the point of cut using the dropout technique, usually used to prevent model overfitting. The SI-NR framework demonstrated stable, predictable communication latency, with a slight degradation in accuracy compared to the base model with data retransmission.

RobustDiCE research paper [11] presents a model-parallelism framework that evaluates neuron importance

and additional neuron redundancy within groups. The proposed algorithm provides a robustness-based parameter, which indicates the weight of each neuron during inference. Based on this characteristic, neurons are grouped, and some are duplicated to ensure redundancy in case any combination of nodes fails. Guo et al. highlighted that this decentralized framework faces communication management challenges as the number of devices scales.

Many distributed inference frameworks are focused on workload partitioning, layer fusion, and local device collaboration to improve inference performance in edge clusters. However, these methods usually assume a stable connection and handle failures at synchronization points, delivering full feature maps to nodes. At the same time, the real-world environment introduces an unstable connection with unpredictable bandwidth, which leads to data loss. As a result, model accuracy can be reduced, or inference may even fail, weakening the practical reliability of distributed inference. Unlike dropout-based tolerance methods or redundancy-based partitioning methods, the proposed approach treats missing intermediate activations as a feature-restoration problem. Therefore, this paper investigates a lightweight autoencoder-based restoration module for reconstructing partially missing feature maps during distributed inference.

III. MODEL-BASED LOST FEATURE RESTORATION METHOD

Du et al. noted that convolutional DNNs like ResNet or the VGG family contain most of their memory usage and computation payload in their convolutional layers (CL), while VGG models also use a significant part of their computation time in fully connected (FC) layers [10]. Convolutional layers, compared with FC layers, have sparse spatial connections because each kernel operates only on a local receptive field. This property was used to group kernels from convolutional layers and distribute them between devices, making these groups independent [9]. On the other hand, FC layers require synchronization points between all layers.

A. Problem statement. Based on the previous statement, this work treats FC layers as part of the CNN, which won't be split and will be executed on a single device. We propose using a flattened data layer as a synchronization point, where devices receive feature map chunks from other devices. So, the initial part of DNN is executed on worker device(s) with convolutional layers, while the final part is executed on the finish node. In this way, the inpainting autoencoder will work with relatively small-dimensional data compared to the raw output of the convolutional layers. This approach leads to a smaller size of the autoencoder module.

First-stage node(s) execute the initial part of the neural network and produce an intermediate feature map. This feature map is transmitted through the communication channel. The final node receives a corrupted version of the feature map. A new module, placed between the input data and FC layer, restores lost values and then passes the restored representation to FC layer. So, the resulting inference pipeline will look as demonstrated in Fig. 2.

Let's denote input data as x, then the convolutional layer intermediate results can be expressed as:

$$z = Concat_C(f_1(x), f_2(x), \ldots, f_3(x)), \quad (1)$$

where f_1, f_2,..., f_n are results from device(s) with convolutional layers, $Concat_C$ is a merged result from all device(s) with kernel groups. The complete inference process can be expressed as:

$$y = f_{n+1}(z), \quad (2)$$

where f_{n+1} is the FC layer partition, and y is the final prediction.

Resulting CL feature maps are transmitted over a communication channel to the final inference node. If the connection is assumed to be ideal, the final node receives a complete feature map z and continues the inference process without deviation. However, in wireless IoT environments, network quality is unpredictable, so transmitted feature maps can be partially lost due to inference, temporary disconnections, or unstable bandwidth. Therefore, the final node obtains an incomplete representation:

$$\tilde{z} = M \odot z, \quad (3)$$

where M is a binary erasure mask and $\tilde{z}$ is the corrupted feature map. Elements corresponding to M=1 are successfully received, while elements corresponding to M=0 are lost. If the corrupted representation $\tilde{z}$ is passed directly to the remaining part of the neural network, the final prediction can become inaccurate because the downstream layers were trained to process complete feature maps.

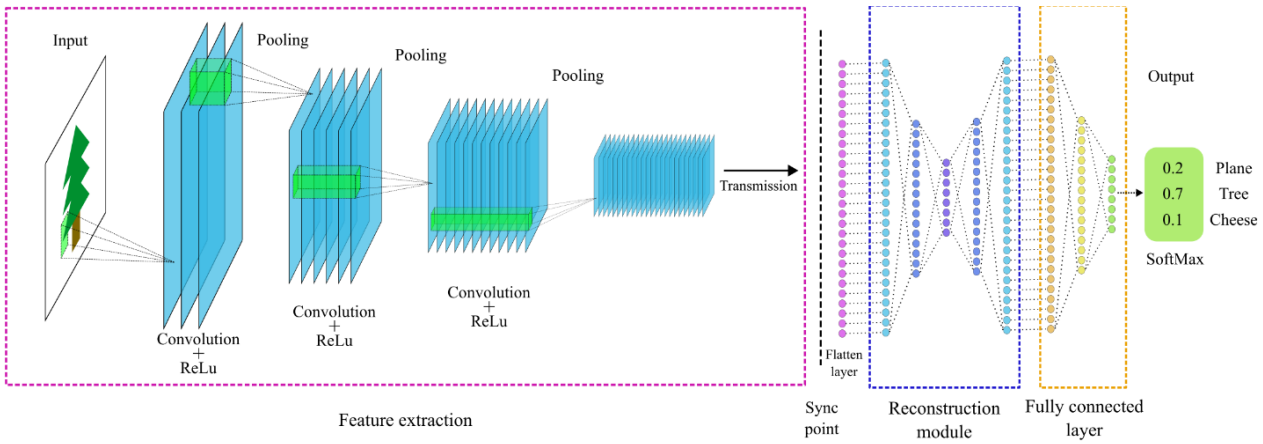

FIG. 2. Inference pipeline with a new restoration module.

B. Feature-loss model. In this study, feature loss is treated as the partial absence of an intermediate feature map transferred between distributed inference stages. Such a loss can appear when fragments of a feature map are transmitted as packets, and one or more packets fail to reach the next computational node. Depending on how the feature map is partitioned and transmitted, the missing data may correspond to individual feature values, spatial regions, entire channels, or continuous fragments of a flattened representation.

The proposed method in this work focuses on an element-wise loss approach, which can be treated as a simplified packet-like loss model at the synchronization point. The individual transmitted feature components may be unavailable at the receiving node. More structured losses, such as channel-wise or spatial block-wise erasures in earlier convolutional feature maps, are left as extensions of the same restoration framework.

For modeling purposes, the loss is described by an erasure mask applied to the feature map. This mask does not modify the neural network's parameters or its architecture. It only marks which parts of the intermediate representation are unavailable at the receiving node. This representation enables evaluation of the same restoration method across different loss rates and patterns.

C. Autoencoder-based feature reconstruction. Autoencoders are a frequent choice for reconstruction tasks due to their ability to learn a compact representation of object characteristics. This feature is used to restore corrupted input parameters, which is widely used for denoising [12] or even reconstructing lost parts of the data representation [13]. In this work, an autoencoder is used as a separate module to reconstruct missing feature data in cases of data loss due to transmission errors or node inaccessibility.

The restoration module is based on a two-part structure: encoder and decoder. The encoder transforms an incomplete feature map into its latent space representation. The decoder reconstructs the feature map from the latent representation of the corrupted data. Input data for module training purposes is derived from a clean feature map by applying a data-loss mask, as demonstrated in Eq. 4:

$$\hat{z} = R_\theta(\tilde{z}), \quad (4)$$

where R_θ is an autoencoder with trainable parameters θ and $\hat{z}$ is the reconstructed feature map. The objective for proper model training can be defined as a reconstruction loss L_{rec}:

$$L_{rec} = \|z - \hat{z}\|_2^2. \quad (5)$$

The loss metric for training, described in Eq. 5, is designed to train the autoencoder to restore the original feature map as precisely as possible. While this approach is a good starting point for autoencoder precision, it can be less effective for the full model inference pipeline. To make the training metric task-aware, a combined loss evaluation metric L_{total} based on the reconstruction and classification loss of the final neural network output is used:

$$L_{total} = \alpha L_{rec} + \beta L_{cls}, \quad (6)$$

where L_{cls} is the classification loss, and α and β are relative importance coefficients regulating the weight of feature reconstruction and final task accuracy for the autoencoder training regulation.

IV. EVALUATION OF FEATURE-LOSS RESTORATION LAYER

To test the proposed method for intermediate feature reconstruction in a distributed inference pipeline, an experiment was planned and conducted, with the autoencoder module trained and evaluated under varying error rates. The experiment used a pretrained ResNet-18 model with frozen weights, with 1000 output classes and input images of size 224×224. The reconstruction module was placed between the flattening layer and the fully connected layer.

For the module training, a dataset was used based on a preprocessed 1000-class subset of the ImageNet-1k dataset, consisting of 500 images per class from the training dataset, 25 images per class for validation, and 25 images per class for the final test from the validation dataset [14, 15]. For the module evaluation, a clean model baseline without errors, a split ResNet-18 model without feature reconstruction, and a split ResNet-18 model with feature reconstruction module were used.

The autoencoder's input and output are 512-dimensional vectors, the same as the fully connected layer feature map. The autoencoder has hidden dimensions of 256 and 128. The training and the evaluation have used an element-wise binary erasure mask $m \in \{0,1\}^{512}$, where each feature value was independently retained with probability $1-\rho$ and erased with probability ρ. During training, the erasure rate ρ was sampled per mini-batch from {0.0,0.1,0.2,0.3,0.4,0.5}, while the evaluation was performed at $\rho \in$ {0.0,0.1,0.2,0.3,0.4,0.5}.

The balancing hyperparameters for the joint loss function during training were set to the $\alpha = 1.0$ and $\beta = 0.5$. The reconstruction hyperparameter was set as the dominant weight because recovering missing values in the intermediate representation is the autoencoder's main goal. The classification term was included with a smaller weight to act as a semantic regularizer without biasing the optimization of the module toward classification at the expense of feature fidelity.

The final module has 328832 parameters, which is about 3% of the total number of parameters in the full classification model (11.7 million) without the reconstruction module [16].

A. Top-1 inference accuracy. The clean model inference result at 69.73% is used as a baseline. The accuracy of a model inference without restoration decreases as the feature-loss rate increases (Fig. 3).

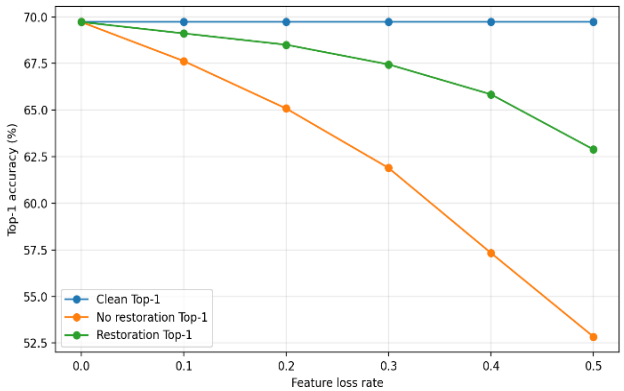

FIG. 3. Top-1 accuracy for tested models.

At a loss rate of 0.1, Top-1 accuracy dropped by 2 percentage points, at 0.3, it decreased to 61.9%, and at a 0.5 loss rate, it reached only 52.84%. These values demonstrate that feature loss directly influences the reliability of inference results.

The model with the restoration module demonstrates higher Top-1 accuracy across all loss rates. At a feature loss of 0.1, restored inference achieved 69.12%, which is close to the clean baseline. At 0.3, restored accuracy was 67.45%, compared with 61.9% without restoration. At 0.5, restored accuracy was 62.89%, 10.0 percentage points higher than in the non-restored case.

B. Top-5 inference accuracy. The Top-5 accuracy has the same pattern (Fig. 4), but the degradation is less steep.

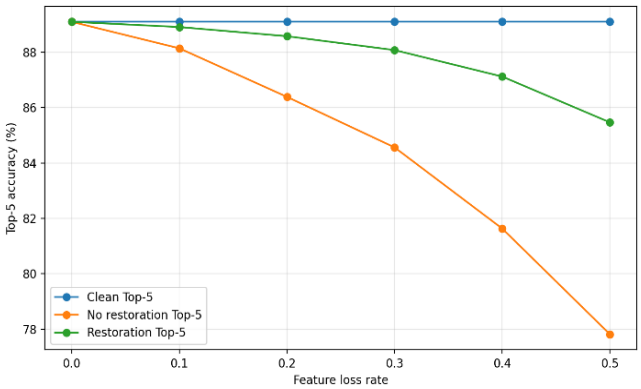

FIG. 4. Top-5 accuracy for tested models.

The clean model's Top-5 accuracy of 89.1% was set as the baseline. Without restoration, Top-5 accuracy decreased to 77.82% at a feature loss of 0.5. With restoration, Top-5 accuracy at the same loss rate remained 85.46%. At a feature loss of 0.5, the non-restored model lost 11.27 percentage points of Top-5 accuracy, while the restored model lost only 3.63 percentage points. This corresponds to a recovery of approximately 67.8% of the lost Top-5 accuracy.

C. Reconstruction quality. The reconstruction quality was measured using mean squared error (MSE) and cosine similarity between the clean and reconstructed feature maps. The results in Table 1 confirmed an increase in reconstruction error as the number of lost features increased. Even though the reconstruction module improved accuracy during feature loss, the increasing difficulty of reconstruction still influenced the inference results, as reflected in the cosine similarity between the reconstructed feature map and the original values.

TABLE 1. Module reconstruction metrics.

Feature loss	Module MSE	Cosine similarity
0.0	0.00	1.00
0.1	0.46	0.99
0.2	0.47	0.97
0.3	0.48	0.96
0.4	0.50	0.94
0.5	0.53	0.92

The cosine similarity remained above 0.92 even at a feature loss of 0.5. This result indicates that the module reconstructed most of the original vector's direction and structure, which explains the higher Top-1 and Top-5 accuracies compared to the non-restored model.

V. CONCLUSION

Distributed model inference is a method of DNN execution that can help overcome constraints in IoT and edge devices. However, it usually relies on a steady connection and device availability during this process. At the same time, wireless IoT systems are often deployed within environments having packet loss, bandwidth changes, and temporary disconnections, causing partial data loss during inference.

This paper proposed a light-weight autoencoder-based restoration method for distributed inference under unstable working conditions. The new layer reconstructs missing activations and passes the updated features to the remaining part of the neural network. The experiment showed that direct inference over corrupted intermediate features results in significant reductions in Top-1 and Top-5 accuracy as feature loss increases. The restoration layer reduced this degradation through tested loss rates ranging from 0% to 50% of the data and recovered most of the lost features. The results show that a small feature-restoration module can provide sufficient data to improve overall model correctness.

The next research should investigate different partition points, heterogeneous edge devices, real communication channels, and combined optimization of model partitioning and feature restoration. Further work should also focus on other error patterns, such as block-wise, channel-wise, and packet-like feature-loss patterns.

AUTHOR CONTRIBUTIONS

V.A. – conceptualization, methodology, software, validation, investigation, writing-original draft preparation, visualization; O.V. – validation, writing-review and editing, supervision.

COMPETING INTERESTS

The authors declare no conflict of interest.

REFERENCES

[1] I. Rodriguez-Conde, C. Campos, and F. Fdez-Riverola, “Horizontally Distributed Inference of Deep Neural Networks for AI-Enabled IoT,” *Sensors*, vol. 23, no. 4, Art. no. 1911, Feb. 2023, doi: 10.3390/s23041911.

[2] K. M. Hou, X. Diao, H. Shi, H. Ding, H. Zhou, and C. De Vaulx, “Trends and Challenges in AIoT/IIoT/IoT Implementation,” *Sensors*, vol. 23, no. 11, Art. no. 5074, May 2023, doi: 10.3390/s23115074.

[3] E. Baccour et al., “Pervasive AI for IoT Applications: A Survey on Resource-Efficient Distributed Artificial Intelligence,” *IEEE Commun. Surv. Tutor.*, vol. 24, no. 4, pp. 2366–2418, 2022, doi: 10.1109/COMST.2022.3200740.

[4] Y. Chen, T. Luo, W. Fang, and Neal. N. Xiong, “EdgeCI: Distributed Workload Assignment and Model Partitioning for CNN Inference on Edge Clusters,” *ACM Trans. Internet Technol.*, vol. 24, no. 2, pp. 1–24, May 2024, doi: 10.1145/3656041.

[5] F. Xue, W. Fang, W. Xu, Q. Wang, X. Ma, and Y. Ding, “EdgeLD: Locally Distributed Deep Learning Inference on Edge Device Clusters,” in *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Yanuca Island, Cuvu, Fiji: IEEE,

Dec. 2020, pp. 613–619, doi: 10.1109/HPCC-SmartCity-DSS50907.2020.00078.
[6] S. Itahara, T. Nishio, and K. Yamamoto, “Packet-Loss-Tolerant Split Inference for Delay-Sensitive Deep Learning in Lossy Wireless Networks,” in *2021 IEEE Global Communications Conference (GLOBECOM)*, Madrid, Spain: IEEE, Dec. 2021, pp. 1–6, doi: 10.1109/GLOBECOM46510.2021.9685179.
[7] Y. Huang et al., “GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism,” 2018, arXiv. doi: 10.48550/arXiv.1811.06965.
[8] F. Brakel, U. Odyurt, and A.-L. Varbanescu, “Model Parallelism on Distributed Infrastructure: A Literature Review from Theory to LLM Case-Studies,” 2024, arXiv. doi: 10.48550/arXiv.2403.03699.
[9] Q. Yuan and Z. Li, “Distributed Inference Models and Algorithms for Heterogeneous Edge Systems Using Deep Learning,” *Appl. Sci.*, vol. 15, no. 3, Art. no. 1097, Jan. 2025, doi: 10.3390/app15031097.
[10] J. Du et al., “Model Parallelism Optimization for Distributed Inference via Decoupled CNN Structure,” *IEEE Trans. Parallel Distrib. Syst.*, pp. 1–1, 2020, doi: 10.1109/TPDS.2020.3041474.
[11] X. Guo, Q. Jiang, A. D. Pimentel, and T. Stefanov, “Model and system robustness in distributed CNN inference at the edge,” *Integration*, vol. 100, Art. no. 102299, Jan. 2025, doi: 10.1016/j.vlsi.2024.102299.
[12] L. Gondara, “Medical Image Denoising Using Convolutional Denoising Autoencoders,” in *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, Barcelona, Spain: IEEE, Dec. 2016, pp. 241–246. doi: 10.1109/ICDMW.2016.0041.
[13] Y. Jiang, J. Xu, B. Yang, J. Xu, and J. Zhu, “Image Inpainting Based on Generative Adversarial Networks,” *IEEE Access*, vol. 8, pp. 22884–22892, 2020, doi: 10.1109/ACCESS.2020.2970169.
[14] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL: IEEE, Jun. 2009, pp. 248–255, doi: 10.1109/CVPR.2009.5206848.
[15] O. Russakovsky et al., “ImageNet Large Scale Visual Recognition Challenge,” 2014, arXiv. doi: 10.48550/arXiv.1409.0575.
[16] “resnet18 – Torchvision main documentation.” Accessed: May 5, 2026. [Online]. Available: https://docs.pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html

Viacheslav Antsybor

PhD student at the Department of Computer Engineering. In 2024, he graduated from the National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute” with a master's degree in Cybersecurity. Field of scientific interests: distributed inference, DNN, networking.

ORCID ID: 0009-0008-1838-6390

Oleksandr Verba

PhD, Associate Professor of the Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”. Field of scientific interests: Methods and tools for improving the efficiency of parallel computing in systems-on-chip.

ORCID ID: 0000-0001-5752-5121

Відновлення характеристик при розподіленому виконанні нейронних мереж в граничних системах

Вячеслав Анцибор*, Олександр Верба

Кафедра обчислювальної техніки, Факультет інформатики та обчислювальної техніки, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна

*Автор-кореспондент (Електронна адреса: antsybor.v.ye.-io51f@edu.kpi.ua)

АНОТАЦІЯ Обсяг даних, які генерують IoT-пристрої, постійно зростає, що робить глибокі нейронні мережі необхідними інструментами для обробки цих масивів. Глибокі нейронні мережі потребують значних обчислювальних ресурсів, що ускладнює їх розгортання на пристроях з обмеженою кількістю пам'яті та обчислювальними ресурсами. З іншого боку, IoT-модулі є частиною мережі інших пристроїв, які можна використати як додаткові ресурси для обчислень. Така конфігурація дає змогу застосувати розподілені обчислення для нейронних мереж та перемістити їх ближче до джерела даних. Локальні мережі, що використовуються в таких випадках, зазвичай є бездротовими, тому втрата пакетів, завади, нестабільна пропускна здатність та недоступність вузлів є звичними явищами для такого типу систем. Згадані проблеми бездротових мереж призводять до втрати проміжних значень характеристик в нейронній мережі, що в результаті відображається на точності роботи нейронної моделі. У цьому дослідженні запропоновано новий шар нейронної мережі для відновлення проміжних характеристик. Втрата проміжних значень у цій роботі визначена як проблема доповнення втрачених параметрів. Для вирішення цієї задачі використовується модуль на основі автоенкодера для реконструкції втрачених значень, який використовує бінарну маску отриманих та втрачених параметрів. Після відновлення дані передаються до класифікатора. Для перевірки гіпотези була використана попередньо навчена модель ResNet-18 із зафіксованими параметрами. Під час постановки експерименту було обрано коефіцієнти помилок від 0,0 до 0,5 з кроком 0,1. Набір даних був сформований на основі попередньо обробленого ImageNet-1k. Модель без модуля відновлення при 50% втрачених параметрів для Top-1 accuracy продемонструвала зниження точності з 69,73% до 52,84%, у той час як модуль відновлення досяг 62,89% точності. Для Top-5 accuracy при

втраті 50% параметрів модель без модуля відновлення продемонструвала зниження точності з 89,1% до 77,82%. Модель з модулем відновлення продемонструвала 85,46% точності. Згідно з отриманими результатами, використання запропонованого модуля при обчисленні демонструє помірне зниження точності при втратах у порівнянні з більш різким зниженням у моделі без відновлення. Таким чином, запропонований підхід може бути використаний для підвищення точності моделі завдяки зменшенню втрат точності при помилках, що може дозволити застосовувати розподілені обчислення в системах реального часу з обмеженими ресурсами.

КЛЮЧОВІ СЛОВА розподілені нейронні мережі, граничні обчислення, відновлення характеристик, доповнюючий автоенкодер.