EdgeLessPart: Distributed and Progressive Inference for the Edge-Cloud Continuum

1st Isaac David Núñez Araya
Chair for Computer Architecture and Parallel Systems
Technische Universität München
Garching (near München), Germany
isaac.nunez@tum.de

2nd Michael Gerndt
Chair for Computer Architecture and Parallel Systems
Technische Universität München
Garching (near München), Germany
gerndt@in.tum.de

3rd Mohak Chadha
Chair for Computer Architecture and Parallel Systems
Technische Universität München
Garching (near München), Germany
mohak.chadha@tum.de

***Abstract*—DNN inference at the edge is highly desirable, offering enhanced data privacy, cost benefits from specialized hardware, and reduced latency. Nonetheless, deploying DNNs on edge devices remains challenging due to resource limitations, often requiring cloud support for complex networks. Current research addresses this by splitting a DNN across the edge and cloud. It optimizes inference by inserting early classifiers between hidden layers, which reduces execution times by bypassing additional layer evaluations. Existing approaches focus on loss functions with system-defined classifiers and distribution with custom schedulers. Such an approach can split these classifiers across the edge and cloud, hindering performance as offloading remains necessary. To this end, we propose EdgeLessPart, a platform- and network-agnostic framework to evaluate DNN distribution and progressive inference on edge-cloud scenarios. EdgeLessPart uses code rendering to accept any exit classifier. Further, EdgeLessPart relies on device profile data and user constraints (latency or memory) to split a DNN, ensuring the early classifiers remain local to each part. EdgeLessPart thus effectively evaluates the performance and cost of distributed and progressive DNN inference on edge-cloud environments.**

***Index Terms*—Edge Deep Learning, Early Exits, Split Computing**

I. Introduction

Current technology developments have made computation ubiquitous, forcing the decentralization of services towards the edge of the network, near end-users and data sources [1], powered by resource-constrained devices such as Jetson Orin, Raspberry Pi or low-power computers. One notable field still struggling to adapt to the limited resources at the edge is *Deep Neural Networks* (DNN), since more complex networks demand increasingly computational power [2]. To this end, the Cloud offers a rich infrastructure that can run the most computationally-intensive applications. Nonetheless, it still suffers from the caveats of the cloud: centralization, reduced privacy as the provider controls the incoming data, and higher costs driven by the price of modern hardware required to handle many clients. In contrast, edge devices can afford their computational limitations as they typically serve fewer clients.

In an effort to bring inference of more complex networks towards the edge, current research explores progressive and distributed inference [3, 4, 5, 6]. Progressive inference is possible by inserting small classifier networks between the hidden layers, called *Early Exits* (EE) [7], while the place to insert them is an *exit point*. The EEs enable stopping execution at intermediate places within the network, given that some inputs are easily classifiable at earlier points [8]. For example, classifying a picture of a cat with a solid color background can be easier than a background from nature. For the former picture, a few layers can produce high confidence, which is not granted for the latter. Current works on EEs [9, 10, 8, 7, 11, 12, 13] focus on their training with custom loss functions and system-defined exit points and architecture.

In the case of distributed inference, active research explores the generation of DNN splits via *Split Computing* (SC) [14, 15, 16, 17, 18, 19]. These works utilize runtime measurements to determine the optimal *split point*, i.e., the connection between two hidden layers that will improve performance. Yet, their combination with EEs can distribute an exit between the edge and cloud, as EE are inserted before splitting, resulting in more data transmission. Nevertheless, our previous example can benefit from SC and EE. Whenever the edge device cannot yield high accuracy using a network part with EE, it can proceed in the cloud until an exit meets the necessary accuracy or evaluates the complete network.

Toward this, we propose EdgeLessPart to evaluate the effect of SC and EE on performance and cost. EdgeLessPart selects the candidate exit and split points based on user-defined constraints and target device profiling, which considers inference time and memory consumption. Unlike previous works, EdgeLessPart supports application-specific EEs via code rendering, and the placement of EEs is constrained within each network part. EdgeLessPart remains platform-agnostic by relying on standard training methods. Hence, EdgeLessPart provides a novel understanding of dynamic DNN inference for edge-cloud scenarios.

II. EdgeLessPart

Here, we introduce *EdgeLessPart*: a framework to explore distributed and progressive inference for edge-cloud scenarios. Figure 1 presents the workflow EdgeLessPart employs to generate the progressive network parts. Since EdgeLessPart can be deployed anywhere, i.e., platform-agnostic, it can leverage local resources, such as accelerators on edge devices.

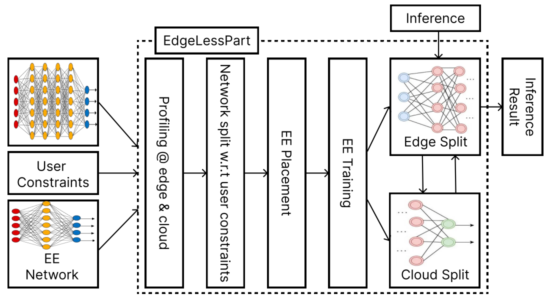

Fig. 1. EdgeLessPart workflow to optimize a given DNN for the edge-cloud continuum. Input network[20]. Output network[21].

To begin the optimization, EdgeLessPart requires a DNN, user constraints, and an exit network. Unlike previous works, EdgeLessPart translates the EE network, stemming from the user input, into executable code. This approach enables users to employ their DNN's insights to determine an appropriate EE network instead of relying on default ones. The constraints determine the exit and split points once the profiling is done. EdgeLessPart can select the split point based on **memory** or **latency** constraints. For latency, EdgeLessPart will split wherever a set of layers on the edge runs as long as the complete network in the cloud. We limit to **one** split per network to avoid network traffic overhead. For memory, the user decides the percentage of layers to run on the edge. Then, the cloud will run the remaining ones. Considering the split point, EEs are placed **linearly** or **pareto** [22] distributed within each network split. We limit to **three** EEs per part to mitigate the effect on memory usage.

Once the points have been generated, EdgeLessPart inserts the EEs, trains them, and generates the respective network parts. After this step, EdgeLessPart evaluates the resulting DNNs across the target clusters. It explores a range of confidence thresholds (c_{thr}), based on the validation accuracy. Unlike some previous works, EdgeLessPart infers in the edge first. In case the local EEs do not yield accuracy greater than c_{thr}, it will offload to the cloud, where the next network part will continue as it happens in the edge.

III. Results

Evaluation Setup: The evaluation environment for EdgeLessPart combines two clusters, edge and cloud respectively. The edge is a pair of Jetson Orin Nano (the 40 TOPS model) with a Virtual Machine (VM) as the control node. The cloud is hosted by our university's provider and it features three nodes with 45 GB of RAM and 10 vCPUS each. We developed EdgeLessPart on top of TensorFlow [23] v2.16.1 with CUDA and CPU support for the edge and cloud, respectively. We train our ResNet50 [24], MobileNetV2 [25], and InceptionV3 [26] using their respective parameters and the CIFAR-10 [27] dataset. For the exit network, we replicate the classifier of each original network. Further, we evaluate a range of c_{thr} between 0.3 and 0.9 to determine the impact on inference time, memory usage, and cloud cost. We use Google Cloud Functions' cost model [28], as users must only pay for used resources.

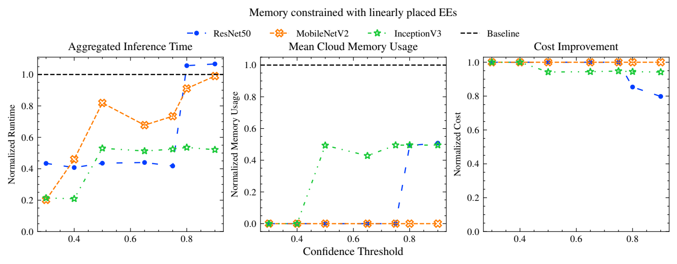

Fig. 2. Distributed and Progressive DNN performance for linearly distributed EEs.

Outcomes: To evaluate EdgeLessPart, we partition each DNN wherever a set of layers require 70 % of memory from the complete network. Here, Figures 2 and 3 present the performance differences once the EEs are placed linearly or Pareto within each network part. In the case of ResNet50, placing EEs linearly results in 20 % more costs than Pareto for $c_{\text{thr}} \geq 0.8$. Additionally, Pareto distribution of EE shows lower inference times with considerable speedups for ResNet50 and InceptionV3. Nonetheless, our results show the impact of SC and EE on the performance of MobileNetV2, an optimized DNN for mobile devices. For higher c_{thr}, the overhead of intermediate classifiers and network traffic outweighs the original model. Yet, linear EEs allow for a smaller MobileNetV2 with desirable accuracy, depicted by the cost improvements and memory usage in Figure 2, which reflects edge-only inference.

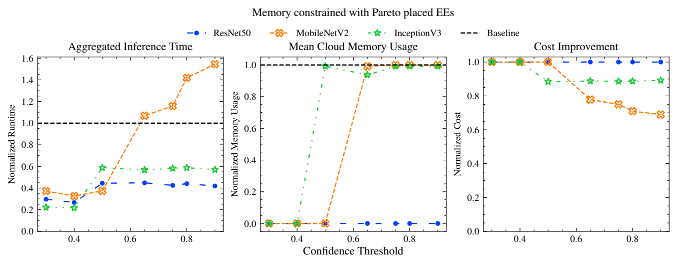

Fig. 3. Distributed and Progressive DNN performance for Pareto distributed EEs.

In contrast, InceptionV3 also demonstrates the importance of cloud-offloading, considering DNN-specific EEs, with 40 % and 50 % faster inference for Pareto and linearly distributed EEs, respectively. Given its complex architecture, such improvements result in 5 % and 10 % cost (compared to cloud-only inference) for linearly and Pareto EEs, respectively. We argue that its cost is negligible since it enables higher accuracy.

IV. Conclusion and Future Work

The results shown in § III demonstrates the importance of further DNN optimizations targeting edge devices, considering the network architecture and its placement. The results show better performance for complex DNNs when coupled with progressive and distributed inference capabilities. EdgeLessPart enables this through progressive (via EE) and distributed (with SC) inference. By collecting runtime information of a target DNN, EdgeLessPart generates distributed and progressive DNNs to investigate the best selection of EEs and split points that improve inference time, cloud memory usage, and cost. It leverages edge resources while offloading to the cloud if the accuracy is insufficient.

Further, our results present another optimization avenue: optimize each network part to leverage local resources for performance gains. Given the realm of optimizations within EdgeLessPart, we intend to integrate a Neural Architecture Search (NAS) module. It can search for smaller and more capable network parts at the edge and automatically search for the exit classifier per exit point. To further limit the search space, NAS can rely on Multi-Objective Optimizations (MOO) that model the target clusters to ensure each split does not exhaust local resources while improving performance. Moreover, such a module allows for better exit and split points, which EdgeLessPart can leverage to bridge on-device DL with the cloud, where the MOO can guide the search to meet on-device constraints.

References

[1] Xiaofei Wang et al. "Convergence of Edge Computing and Deep Learning: A Comprehensive Survey". In: *IEEE Communications Surveys & Tutorials* 22.2 (2020). Conference Name: IEEE Communications Surveys & Tutorials, pp. 869–904. ISSN: 1553-877X. DOI: 10.1109/COMST.2020.2970550.

[2] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. "Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning". en. In: *Sustainable Computing: Informatics and Systems* 38 (Apr. 2023), p. 100857. ISSN: 22105379. DOI: 10.1016/j.suscom.2023.100857. URL: https://linkinghub.elsevier.com/retrieve/pii/S2210537923000124 (visited on 01/10/2024).

[3] Liekang Zeng et al. "Boomerang: On-Demand Cooperative Deep Neural Network Inference for Edge Intelligence on the Industrial Internet of Things". In: *IEEE Network* 33.5 (Sept. 2019). Conference Name: IEEE Network, pp. 96–103. ISSN: 1558-156X. DOI: 10.1109/MNET.001.1800506.

[4] En Li et al. "Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing". In: *IEEE Transactions on Wireless Communications* 19.1 (Jan. 2020). Conference Name: IEEE Transactions on Wireless Communications, pp. 447–457. ISSN: 1558-2248. DOI: 10.1109/TWC.2019.2946140.

[5] Stefanos Laskaridis et al. "SPINN: synergistic progressive inference of neural networks over device and cloud". en. In: *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. London United Kingdom: ACM, Sept. 2020, pp. 1–15. ISBN: 978-1-4503-7085-1. DOI: 10.1145/3372224.3419194. URL: https://dl.acm.org/doi/10.1145/3372224.3419194 (visited on 02/12/2023).

[6] Jin Huang, Hui Guan, and Deepak Ganesan. "Rethinking computation offload for efficient inference on IoT devices with duty-cycled radios". en. In: *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. Madrid Spain: ACM, July 2023, pp. 1–15. ISBN: 978-1-4503-9990-6. DOI: 10.1145/3570361.3592514. URL: https://dl.acm.org/doi/10.1145/3570361.3592514 (visited on 08/13/2023).

[7] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. "BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks". In: *arXiv: Neural and Evolutionary Computing* (Sept. 2017). DOI: 10.1109/ICPR.2016.7900006.

[8] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. "Shallow-Deep Networks: Understanding and Mitigating Network Overthinking". en. In: *Proceedings of the 36th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, May 2019, pp. 3301–3310. URL: https://proceedings.mlr.press/v97/kaya19a.html (visited on 08/03/2023).

[9] Meiqi Wang et al. "DynExit: A Dynamic Early-Exit Strategy for Deep Residual Networks". In: *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*. ISSN: 2374-7390. Oct. 2019, pp. 178–183. DOI: 10.1109/SiPS47522.2019.9020551.

[10] Stefanos Laskaridis et al. "HAPI: hardware-aware progressive inference". en. In: *Proceedings of the 39th International Conference on Computer-Aided Design*. Virtual Event USA: ACM, Nov. 2020, pp. 1–9. ISBN: 978-1-4503-8026-3. DOI: 10.1145/3400302.3415698. URL: https://dl.acm.org/doi/10.1145/3400302.3415698 (visited on 08/01/2023).

[11] Ershad Banijamali et al. "Pyramid Dynamic Inference: Encouraging Faster Inference Via Early Exit Boosting". en. In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Rhodes Island, Greece: IEEE, June 2023, pp. 1–5. ISBN: 978-1-72816-327-7. DOI: 10.1109/ICASSP49357.2023.10096928. URL: https://ieeexplore.ieee.org/document/10096928/ (visited on 08/04/2023).

[12] Wangchunshu Zhou et al. "BERT Loses Patience: Fast and Robust Inference with Early Exit". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 18330–18341. URL: https://proceedings.neurips.cc/paper/2020/hash/d4dd111a4fd973394238aca5c05bebe3-Abstract.html (visited on 08/03/2023).

[13] Ji Xin et al. *DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference*. arXiv:2004.12993 [cs]. Apr. 2020. URL: http://arxiv.org/abs/2004.12993 (visited on 08/04/2023).

[14] Minchen Yu et al. "Gillis: Serving Large Neural Networks in Serverless Functions with Automatic Model Partitioning". In: *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. ISSN: 2575-8411. July 2021, pp. 138–148. DOI: 10.1109/ICDCS51616.2021.00022.

[15] Liekang Zeng et al. "CoEdge: Cooperative DNN Inference With Adaptive Workload Partitioning Over Heterogeneous Edge Devices". In: *IEEE/ACM Transactions on Networking* 29.2 (Apr. 2021). Conference Name: IEEE/ACM Transactions on Networking, pp. 595–608. ISSN: 1558-2566. DOI: 10.1109/TNET.2020.3042320.

[16] Beibei Zhang et al. "Dynamic DNN Decomposition for Lossless Synergistic Inference". In: *2021 IEEE 41st International Conference on Distributed Computing Systems Workshops (ICDCSW)*. ISSN: 2332-5666. July 2021, pp. 13–20. DOI: 10.1109/ICDCSW53096.2021.00010.

[17] Jananie Jarachanthan et al. "AMPS-Inf: Automatic Model Partitioning for Serverless Inference with Cost Efficiency". In: *Proceedings of the 50th International Conference on Parallel Processing*. ICPP '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1–12. ISBN: 978-1-4503-9068-2. DOI: 10.1145/3472456.3472501. URL: https://doi.org/10.1145/3472456.3472501 (visited on 08/08/2023).

[18] Hongzhou Liu et al. "LoADPart: Load-Aware Dynamic Partition of Deep Neural Networks for Edge Offloading". In: *IEEE International Conference on Distributed Computing Systems* (July 2022). DOI: 10.1109/ICDCS54860.2022.00053.

[19] Min Xue et al. "EosDNN: An Efficient Offloading Scheme for DNN Inference Acceleration in Local-Edge-Cloud Collaborative Environments". In: *IEEE Transactions on Green Communications and Networking* 6.1 (Mar. 2022). Conference Name: IEEE Transactions on Green Communications and Networking, pp. 248–264. ISSN: 2473-2400. DOI: 10.1109/TGCN.2021.3111731.

[20] *Architecture of DNN*. URL: https://cdn-images-1.medium.com/v2/resize:fit:800/1*5egrX--WuyrLA7gBEXdg5A.png (visited on 11/02/2024).

[21] Valdivino Santiago Júnior. *Deep neural networks: How to define?* en. Oct. 2021. URL: https://towardsdatascience.com/deep-neural-networks-how-to-define-73d87bf36421 (visited on 11/02/2024).

[22] *Explaining the 80-20 Rule with the Pareto Distribution — D-Lab*. URL: https://dlab.berkeley.edu/news/explaining-80-20-rule-pareto-distribution (visited on 10/10/2023).

[23] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[24] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: (Dec. 2015). arXiv:1512.03385 [cs]. URL: http://arxiv.org/abs/1512.03385 (visited on 08/30/2023).

[25] Mark Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: (Mar. 2019). arXiv:1801.04381 [cs]. URL: http://arxiv.org/abs/1801.04381 (visited on 08/30/2023).

[26] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. arXiv:1512.00567 [cs]. Dec. 2015. URL: http://arxiv.org/abs/1512.00567 (visited on 08/30/2023).

[27] *CIFAR-10 and CIFAR-100 datasets*. URL: https://www.cs.toronto.edu/~kriz/cifar.html (visited on 08/30/2023).

[28] *Pricing — Cloud Functions*. en. URL: https://cloud.google.com/functions/pricing (visited on 08/29/2023).