A Performance Prediction-based DNN Partitioner for Edge TPU Pipelining

Bohua Zou, Binqi Sun, Yigong Hu, Tomasz Kloda, Marco Caccamo, Tarek Abdelzaher

A Performance Prediction-based DNN Partitioner for Edge TPU Pipelining

Bohua Zou*, Binqi Sun*†, Yigong Hu†, Tomasz Kloda‡, Marco Caccamo*, Tarek Abdelzaher†
*Technical University of Munich, Germany
†University of Illinois Urbana-Champaign, USA
‡LAAS-CNRS, Université de Toulouse, INSA, Toulouse, France
Emails: {bohua.zou, binqi.sun, mcaccamo}@tum.de, {yigongh2,zaher}@illinois.edu, tkloda@laas.fr

***Abstract*—Intelligent IoT applications deployed in adversarial environments often operate without reliable cloud connections, requiring local execution of AI pipelines on resource-constrained edge devices. Edge Tensor Processing Unit (TPU) is a specialized AI hardware accelerator known for its low power consumption and high computational efficiency. To optimize DNN performance across multiple Edge TPUs, DNN models are often pipelined by partitioning them into segments. However, uneven workload distribution across these segments can lead to latency bottlenecks, reducing overall throughput, and increasing memory access due to the limited on-chip memory. This issue is especially concerning in mission-critical applications, where minimizing memory contention and ensuring robust performance are critical. To overcome these challenges, we develop a novel performance prediction-based partitioning tool for DNN models on Edge TPU pipelines. This tool uses a Transformer-based model to accurately predict the inference time of individual DNN segments, enabling more efficient partitioning. We introduce two methods: one relying solely on the prediction model and another combining prediction with profiling. Tested on 120 models from the NASBench-101 dataset, both methods significantly improved partitioning robustness and efficiency, reducing solving time by up to 98.86% and 97.21%, respectively, compared to traditional profiling-based approaches, while maintaining comparable bottleneck latencies.**

***Index Terms*—DNN Partition, Edge TPU, Pipelining, Transformer.**

I. Introduction

In adversarial environments, intelligent IoT applications often cannot rely on connections to cloud infrastructures, as such environments frequently disrupt or block connectivity. As a result, these applications need to run their AI models locally on resource-constrained edge devices. Given the limited resources available at the edge, it is crucial to optimize deep neural network (DNN) inference performance to ensure robustness, predictability, and meet real-time requirements [1]–[3]. This need is particularly pronounced in mission-critical applications such as disaster response [4], extreme environmental monitoring [5], and security systems [6].

Edge Tensor Processing Units (TPUs) [7] are a key technology developed to meet these demands. As specialized AI accelerators, Edge TPUs are optimized for running deep neural network (DNN) models with low energy consumption and high inference performance, making them particularly suitable for IoT applications in challenging environments. However, reliance on a single Edge TPU for inference poses several challenges, such as limited on-chip memory (approximately 8 MB) [8], underutilization of processing elements [9], and inefficiencies in memory systems leading to excessive energy consumption [9]. These limitations are particularly concerning in environments where reliability and resilience are crucial.

To address these challenges, a DNN pipelining technique has been developed to distribute DNN workloads across multiple Edge TPUs, effectively expanding the on-chip memory available for inference and enhancing overall system efficiency. Pipelining allows the entire DNN to reside within the Edge TPU's SRAM, thereby reducing dependency on DRAM, which is a shared system resource prone to contention and latency issues. Minimizing DRAM access at runtime is critical for improving the resilience and robustness of AI systems in mission-critical environments. Additionally, by balancing the workload across multiple Edge TPUs, more DNN parameters can be stored within SRAM, further decreasing reliance on DRAM and enhancing system stability. Moreover, optimizing the pipeline by reducing the latency of the largest segment plays a key role in increasing overall throughput and improving system efficiency.

However, non-optimal partitioning of the DNN models can create significant bottlenecks, leading to underutilized resources and diminished system resilience, which is a critical concern for mission-critical applications. To tackle this issue, a profiling-based partitioner [10] has been developed, which employs a binary search algorithm for design space exploration and evaluates each candidate solution by profiling on real hardware. The approach provides considerably better partitioning results than the default Edge TPU compiler, which partitions DNN models solely based on the number of DNN weights. However, the profiling-based partitioner exhibits several instability issues, *e.g.*, runtime errors occur when partitioning some specific DNNs, failing to generate feasible partitioning outputs. Besides, the partitioning process is time-consuming, especially for large DNNs that are partitioned on a large number of Edge TPU segments. These inefficiencies are especially problematic in mission-critical IoT applications, where rapid and reliable deployment of AI models is essential. To enhance the efficiency and robustness of the partitioning process, this

The work was partially completed while Binqi Sun was a visiting scholar at the University of Illinois Urbana-Champaign.

paper investigates the limitations of an existing profiling-based partitioning tool [10] and proposes a novel performance prediction-based DNN partitioner. The developed partitioner integrates a Transformer-based prediction model to estimate the DNN segment inference times, facilitating more effective partitioning and ensuring resilient and efficient operation of AI systems deployed on Edge TPUs in adversarial environments.

We summarize our contributions as follows.

1) We formulate a Directed Acyclic Graph (DAG) partition problem to optimize the DNN pipelining on multiple Edge TPUs.
2) We identify the limitations of the existing profiling-based partitioner designed for Edge TPU pipelining.
3) We develop a graph-transformer-based model to predict the DNN model segment latency.
4) We utilize the prediction model to address the main drawbacks of the profiling-based partitioner. Experiments demonstrate that the proposed approaches significantly improve the robustness and efficiency of the traditional profiling-based partitioner.

The remainder of the paper is organized as follows. Section II reviews relevant literature on DNN partitioning and Edge TPU deployment. Section III discusses the limitations of the existing Edge TPU partitioner and proposes our performance prediction-based partitioner. The experimental results are reported in Section IV, and Section V concludes the paper.

II. Related Work

A. DNN Partitioning

Significant research has been focused on developing DNN partitioning algorithms and tools for various edge AI applications. Parthasarathy *et al.* [11] developed a partitioning and placement algorithm of DNNs to maximize the inference throughput of such pipelines across clusters of edge devices by solving an optimization problem with DAG. With edge device communication as the primary bottleneck, optimization only focuses on segment data transfers, treating edge device computational resources as constraints. Zheng *et al.* [12] proposed an efficient partitioning approach based on dynamic programming (DP) and node clustering. Given that DP time complexity grows with network complexity and node count, they developed a clustering algorithm that fuses contiguous elementwise layers and groups nodes by depth. This approach enhances interlayer data reuse and reduces time complexity. However, defining and acquiring the DP cost function is challenging, and the approach is unsuitable for Edge TPUs due to their limited computing resources and on-chip memory. Guo *et al.* [13] presented a dynamic computation offloading strategy for DNN partitioning based on the Louvain algorithm and Kuhn-Munkres algorithm, which can dynamically find the partition point of DNN models to minimize the overall delay within the D2D (Device-to-Device) network. Yang *et al.* [14] devised a DNN distributing strategy together with a collaborative Edge Computing system called CoopAI. CoopAI can perform both inter-layer and intra-layer partitioning, but it can only be applied to convolutional and fully connected layers.

B. DNN Inferences on Edge TPU Pipelines

In the domain of Edge TPU pipelining, [15], [16] studied the real-time scheduling of multiple DNN inference tasks on Edge TPU pipelines by formulating each DNN inference as a non-preemptive gang task. [17] designed a framework to satisfy the timing requirement of multiple DNN inference tasks. Although these works consider pipelined Edge TPU inferences, they focus on the runtime behavior and use the default Edge TPU compiler for model partitioning. Yin *et al.* [18] introduced an Edge TPU workload balancing framework based on integer linear programming (ILP). More recently, [19] developed a reinforcement learning (RL) method called RESPECT to imitate the solutions obtained by the ILP model in their previous work [18]. One limitation of their partitioning strategy is that it balances the workloads in terms of static model characteristics rather than real system dynamics. Unfortunately, since the full implementation of the proposed algorithms and the workload dataset in these two papers have not been released, we cannot compare our method with theirs in the experimental evaluation.

III. Performance Prediction-Based Partitioner

A. Problem Description and Formulation

We consider a DNN model to be partitioned across a pipeline of N Edge TPU segments. A DNN model can be modeled as a DAG, where the nodes represent DNN operations (*e.g.*, 2-D convolution and pooling), and the edges correspond to the data dependencies between operations. A partition of a DNN model refers to a mapping of DAG nodes to the N Edge TPU segments. A valid DNN partition on Edge TPUs must fulfill the dependency constraints, meaning that any preceding model segments should not contain any nodes that are successors of any nodes assigned to the succeeding segment. A candidate partition can be encoded by a topological sort of the DAG together with a list indicating the number of operations assigned to each segment (*i.e.*, a certain number of nodes are assigned to each Edge TPU segment in the pipeline following the order of the topological sort). The objective is to optimize the pipeline throughput by minimizing the bottleneck latency defined as the largest segment latency in the pipeline:

$$\min_{S=\{S_1,\ldots,S_N\}} \max\{t(S_i) \mid \forall S_i \in S\}, S \in X, \tag{1}$$

where $S = \{S_1, ..., S_N\}$ denotes a valid candidate solution, including the set of nodes S_i allocated to the ith segment ($i \in [1, N]$); $t(S_i)$ represents the resulting latency of the ith model segment; and X represents the feasible decision space. We note that, even with a given topological sort, the number of possible partitions is combinatorially large. For instance, for a model with 200 operations partitioned across 8 Edge TPUs, there are approximately 2.2×10^{12} potential partitioning configurations. The vast search space makes an exhaustive search approach highly impractical and inefficient. Therefore, it is reasonable

to develop heuristic search algorithms to find solutions of good quality but not necessarily optimal.

B. Limitations of Existing Partitioning Tools

There have been two publicly available software tools for Edge TPU DNN partitioning: (i) the default Edge TPU compiler (*Default*) [8] and (ii) the profiling-based partitioner (*ProfPar*) [10]. The former is a closed-sourced tool that partitions the model using an implicit heuristic that aims to distribute DNN weights evenly among different segments. However, the evenly distributed weights do not necessarily translate to an even distribution of computational time.

ProfPar utilizes a search-based algorithm to explore the decision space and evaluates each candidate solution by profiling it on real Edge TPU hardware. It consists of three critical components. The first is a binary search algorithm, which iteratively updates the target bottleneck latency (*i.e.*, the upper bound of all segment latencies) for the next iteration based on the current partitioning results. The initial target bottleneck latency is set as the median value of the maximum and minimum segment latencies. The second component is a recursive algorithm that aims to allocate as many operations as possible to each segment without violating the current target latency constraint. The third component is a latency profiler, which evaluates each candidate solution by executing the partitioned DNN on a real Edge TPU pipeline. Preliminary experiments have revealed that *ProfPar* exhibits low efficiency and robustness. For example, *ProfPar* required approximately 2 hours to partition a DenseNet-201 [20] model across 5 Edge TPUs. The running time is expected to increase exponentially with respect to the number of Edge TPUs due to the nature of the recursive search used in the algorithm. Besides, we have the following observations that provide insights for improving the partitioner.

Observation 1: The compiling process of the latency profiler is the most time-consuming procedure in ProfPar. Our experiments show that the model compilation takes at least $10\times$ longer than the model execution, as illustrated in Figure 1. We note that multiprocessing is used to accelerate the compilation process.

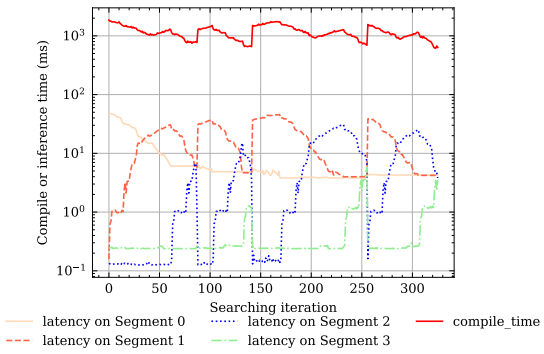

Fig 1. Partitioning process of *ProfPar* on Inception V3 [21] with 4 segments.

Observation 2: The inference time of DNN models on Edge TPU pipelines is predictable. Our experiments show that the inference time of a DNN model does not fluctuate for different inputs. For example, running the Inception V1 [21] model on 3 Edge TPUs with 200 different inputs resulted in an average bottleneck segment latency of 2.87 milliseconds with a standard deviation of only 0.015 milliseconds. This benefits from the capability of caching more DNN weights on the SRAM of the pipelined Edge TPUs (so that DRAM contention can be reduced).

Observation 3: A learning-based model is more suitable than a mathematical model for predicting the Edge TPU inference time. Given the closed-source nature of the Edge TPU compiler and hardware architecture, a mathematical model-based performance prediction is not accessible. However, the DAG nature of a DNN model provides opportunities to estimate the inference time using advanced machine learning techniques such as graph neural networks (GNNs) [22] and Transformers [23]. We will demonstrate the applicability of such a learning-based model in Section III-C.

Observation 4: The size of the intermediate tensors transferred between two Edge TPUs significantly impacts the inference latency. Due to data dependencies within the model, especially residual connections, improper partitioning can lead to excessive data transfers between segments. Intermediate tensors cannot be directly passed between different Edge TPUs and must be transferred through external DRAM. If these tensors are large, it can result in significant overhead.

Observation 5: The off-chip memory significantly impacts the segment latency. Each Edge TPU has 8 MB on-chip SRAM, where around 6 MB is reserved for caching the DNN weights. Suppose the model cannot be fully cached to the on-chip memory. In that case, the remaining weights must be fetched from the external DRAM, which could result in unpredictable interference with other tasks running in the system and thus affect inference time. A similar phenomenon has also been observed in [24].

C. Transformer-based Prediction Model

Based on the above observations, we develop a transformer-based graph neural network model that can predict DNN latency, taking into account the size of both DNN weights and intermediate tensors. The goal is to replace the latency profiler used in *ProfPar*, thereby saving time in compiling and profiling. The prediction model is derived from a neural architecture called Graphormer [25]. The overall structure of the developed prediction model is illustrated in Figure 2.

The model follows an encoder-decoder structure. The input of the encoder is the embedding of input data, including node features and the topology information of the DNN workload. The node features include three important properties of the node operation: (i) operation type, (ii) node in-degree (*i.e.*, the number of predecessors), and (iii) output tensor shape. The DAG topology information includes (i) the adjacency matrix of the original graph and (ii) the topological sort represented by an adjacency matrix where each node has only one predecessor

and successor (despite the input and output nodes which have only one successor and predecessor, respectively).

The embeddings of the adjacency matrices and their corresponding transposes are added as a bias to the multi-head attention matrix. Additionally, we introduce a virtual node to the graph to encode the graph-level features including the DNN's input and output sizes and the number of DNN weights. The graph-level features can effectively help the prediction model understand the size of the DNN workload and take into account the amount of data required to be loaded from the external DRAM at runtime.

The encoder consists of multiple layers, each of which contains a multi-head attention followed by a shared multi-layer perceptron (MLP). The output of the last encoder layer is served as input to the decoder. The decoder is an MLP shared across different nodes. The input dimension is equal to the node embedding, and the output is a single tensor representing the estimated latency in milliseconds.

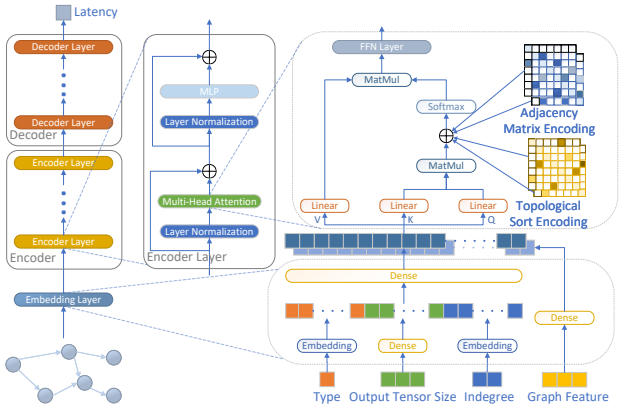

Fig 2. Graphormer-based prediction model architecture.

D. Performance Prediction-based Partitioner

We propose a performance prediction-based partitioning method based on the developed transformer-based prediction model. The overall strategy of the partitioner is derived from *ProfPar*. We modify the partitioner by improving the time-consuming latency profiler using the prediction model. Due to the statistical nature of the prediction model, there is no theoretical guarantee of the prediction accuracy. Therefore, we consider two alternative approaches. The first one relies solely on the prediction results (dubbed *PredPar*), while the second one is a hybrid approach that combines the latency profiler with the prediction model (dubbed *Pred-ProfPar*):

- *PredPar*: evaluates the latency of a candidate solution solely based on the prediction model.
- *Pred-ProfPar*: uses the profiler only if a better solution is claimed to be found by the prediction model (*i.e.*, all estimated segment latencies are less than the bottleneck latency of the last search iteration).

The detailed flowchart of the proposed prediction-based partitioner is illustrated in Figure 3.

Furthermore, we observed that the binary search algorithm in the original *ProfPar* can potentially miss optimal solutions due to the relaxation of the lower bound. Specifically, when the current target latency is set too low to be achievable, *ProfPar* adjusts the lower bound for the next iteration to match the current target latency. However, this adjustment can lead to an overshoot, increasing the target latency more than necessary. To address this issue, we introduce a new parameter called the *relaxing factor*, which provides finer control over the lower bound update:

$$LB' = LB + \frac{UB - LB}{\gamma}, \tag{2}$$

where LB and UB represent the current lower and upper bounds, respectively; LB' is the updated lower bound; and γ is the relaxing factor. Notably, when the *relaxing factor* is set to 2, the update behaves identically to the *ProfPar* algorithm.

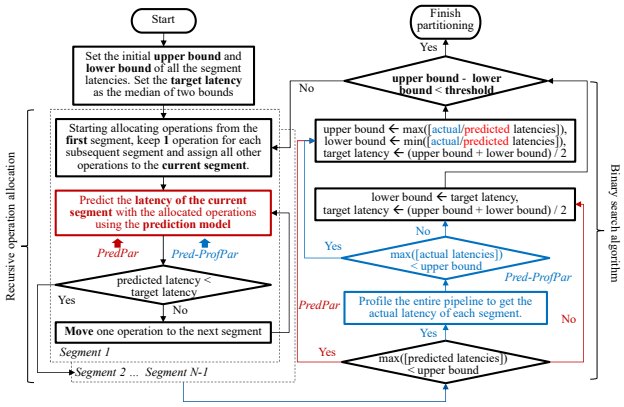

Fig 3. Workflow of performance prediction-based DNN partitioner.

IV. Experiments

A. Training of the Prediction Model

We generate our training and evaluation data using a diverse set of DNN workloads provided by NASBench-101 [26]. NASBench-101 offers a comprehensive dataset of 423,624 distinct convolutional neural network architectures, encompassing features from various classic models such as Inception [21] and ResNet [27]. The DNN models in NASBench-101 are composed of multiple stacks, with each stack containing several cells. Each cell is defined by a specific configuration that includes both the topological structure and the types of operations within it. By varying the number of stacks, the number of cells, and the cell configurations, we generate a total of 600 DNN models for our training and evaluation dataset, which can be used to evaluate model pipelining across different quantities of Edge TPUs. The number of segments is calculated by dividing the model size by 6 MB (*i.e.* $\lceil model\_size/6\,MB \rceil$). For each DNN, we applied 3 partition methods to these 600 models to create a comprehensive training set, *i.e.* random partitioning (32 partitions for each model), default partitioning from the Edge TPU compiler, and partitioning at specific operations (*e.g.* MaxPool2d). We note that the dataset includes

model segments with off-chip memory, as it is unavoidable for large DNN workloads. As a result, we generate a total of 416,000 different DNN segments.

We measured the latency of the generated DNN segments using the profiler provided by *ProfPar*. Our experiments were conducted on an ASUS AI Accelerator card CRL-G18U-P3D[1], integrated with 8 Edge TPUs connected via PCIe to a workstation equipped with Intel Xeon Silver 4216 CPUs running GNU/Linux. Although our dataset was generated on this specific hardware setup, our pre-trained prediction model can be easily migrated to other PCIe-based Edge TPU platforms. Additionally, transfer learning can be employed to fine-tune the model for different platform specifications, such as USB-interfaced Edge TPUs. To ensure clean and reliable training data, we disabled the Linux kernel's real-time throttling mechanism and the dynamic frequency scaling on the Edge TPU. We also set the CPU dynamic frequency scaling to the `performance` mode. Each model segment was executed 10 times, with the average inference time used as the label. The dataset was randomly split into training, validation, and test sets with a 6:2:2 ratio across different DNN models.

We trained the transformer-based prediction model using *Adam* [28] optimizer on a single NVIDIA A100 GPU. The model was trained for 100 epochs, with the total training time being under one hour. After training, we evaluated the prediction model on the test set. The evaluation results are presented in Figure 4, where we used an absolute error range of $\pm 1\,ms$ to assess the accuracy of the model's predictions. While predictions are relatively accurate in regions with smaller latencies, underestimation frequently occurs with longer inference times, leading to an invalid solution being incorrectly identified as valid under the current target bottleneck latency. However, the prediction model is unlikely to overlook any valid solutions. To mitigate this, in *Pred-ProfPar*, the profiler is employed only for solutions deemed valid by the prediction model, thereby minimizing the risk of false negatives.

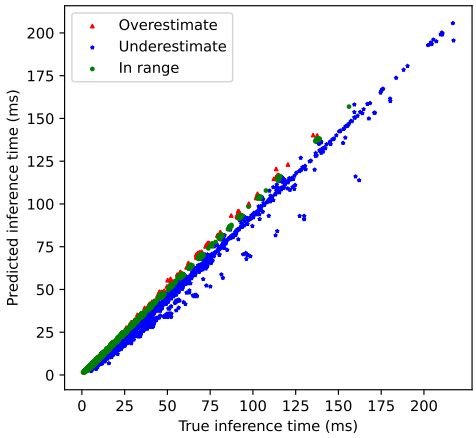

Fig 4. Evaluation results on the test dataset. Green, blue, and red dots represent instances with $|Deviation| < 1\,ms$, $Deviation < -1\,ms$, and $Deviation > 1\,ms$, respectively.

[1]https://www.asus.com/networking-iot-servers/aiot-industrial-solutions/gpu-edge-ai-accelerators/ai-accelerator-pcie-card/

B. DNN Partitioning Results

We evaluate the proposed *PredPar* and *Pred-ProfPar* partitioners by comparing them with the default Edge TPU compiler *Default* and *ProfPar*. We run each partitioner on the same set of DNN models (*i.e.*, the 120 NASBench-101 models in the test set). Our partitioner is implemented in Python 3.11 using *TFLite* library. We used the same rule (*i.e.* $[model\_size/6\,MB]$) to determine the number of segments for each DNN model. We compare the performance of different partitioners in terms of their bottleneck latency, success ratio (fraction of DNN models successfully partitioned by a given partitioner), and partitioning time.

Table I and Table II present the average bottleneck latency and the success ratio achieved for each group of models with the same number of segments. While *ProfPar* achieves the best performance due to its use of an exact latency profiler, it struggles with partitioning most large models (95%) across 8 Edge TPUs, leading to unstable and non-robust performance—an issue that is particularly problematic in mission-critical scenarios. In contrast, *Pred-ProfPar* results in slightly higher bottleneck latency than *ProfPar* but successfully partitions any given DNN model, regardless of its size. *PredPar* does not match the bottleneck latency performance of *ProfPar*, but it still outperforms the default partitioner in terms of both bottleneck latency and success ratio.

N	*Default*	*ProfPar*	*PredPar*	*Pred-ProfPar*
2	7.74	6.05	6.85 (+13%)	**6.12 (+1%)**
3	6.12	5.11	6.16 (+21%)	**5.17 (+1%)**
4	7.05	5.7	6.87 (+21%)	**5.73 (+0.5%)**
5	9.47	4.71	6.37 (+35%)	**4.78 (+1%)**
6	11.98	6.73	11.94 (+77%)	**6.5 (-3%)**
7	11.88	4.76	5.4 (+13%)	**5.32 (+12%)**
8	60.78	Mostly failed	33.17 *(-45%)*	**30.58** ***(-50%)***
Avg	39.07	-	22.92	**20.99**

Table I. Average bottleneck latency of each group of DNNs with the same number of segments (in ms). Values in the brackets present the relative bottleneck latency increase compared to *ProfPar*. For $N = 8$, since *ProfPar* fails to partition most of the models, the relative average bottleneck latency is derived by comparing with *Default* instead.

N	*Default*	*ProfPar*	*PredPar*	*Pred-ProfPar*
2	88.9%	100%	100%	100%
3	100%	100%	100%	100%
4	100%	100%	100%	100%
5	90%	100%	100%	100%
6	100%	100%	100%	100%
7	100%	100%	100%	100%
8	86.3%	**5.5%**	100%	100%

Table II. Success ratio of each DNN group with the same number of segments. A partitioning is deemed successful if a valid partition is generated.

The partitioning times are summarized in Table III. Both *PredPar* and *Pred-ProfPar* substantially reduce the partitioning time, achieving at least an 84% reduction compared to *ProfPar*. This significant decrease in partitioning time is crucial, especially in mission-critical applications, where rapid deployment and adaptation are essential for maintaining system performance and resilience.

N	*Default*	*ProfPar*	*PredPar*	*Pred-ProfPar*
2	3.0	393.97	**8.15 (-98%)**	63.27 (-84%)
3	4.07	965.41	**41.88 (-96%)**	144.82 (-85%)
4	4.78	1188.27	**68.54 (-94%)**	176.89 (-85%)
5	5.3	1396.31	**57.53 (-96%)**	132.6 (-91%)
6	6.7	2386.74	**98.52 (-96%)**	259.99 (-89%)
7	7.39	3153.41	**123.6 (-96%)**	254.04 (-92%)
8	16.81	Mostly failed	**168.9**	385.55
Avg	11.89	-	**118.75**	286.74

Table III. Average partitioning time of each DNN group with the same number of segments (in seconds) and speed-up compared to *ProfPar*.

V. Conclusion and Future Work

In this paper, we presented a novel approach to optimizing the partitioning of deep neural networks (DNNs) across multiple Edge TPUs. Our work addresses the limitations of existing partitioning methods, particularly the profiling-based partitioner (*ProfPar*), which, while effective, often struggles with large models and incurs significant partitioning time. We introduced a transformer-based prediction model that accurately estimates the inference latency of DNN segments, which we integrated into the existing search algorithm to enhance the partitioning process. Our proposed methods, *PredPar* and *Pred-ProfPar*, achieve not only substantial reductions in partitioning time—up to 84%—but also maintain high success rates in generating valid partitions, even for large DNN models. This is particularly important for mission-critical IoT applications, where rapid and reliable deployment is of utmost importance.

Future work may explore further enhancements to the prediction model and its application across a broader range of edge AI platforms and scenarios.

Acknowledgement

Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research. Research reported in this paper was sponsored in part by DEVCOM ARL under Cooperative Agreement W911NF-17-2-0196 (ARL IoBT CRA), and in part by NSF CNS 20-38817, and the Boeing Company. It was also supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. The views and conclusions contained in this document are those of the authors, not the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] J. Li, R. Ma, V. S. Mailthody, C. Samplawski, B. Marlin, S. Chen, S. Yao, and T. Abdelzaher, "Towards an accurate latency model for convolutional neural network layers on gpus," in *IEEE Military Communications Conference (MILCOM)*, 2021, pp. 904–909.

[2] Y. Hu, I. Gokarn, S. Liu, A. Misra, and T. Abdelzaher, "Algorithms for canvas-based attention scheduling with resizing," in *IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2024, pp. 348–359.

[3] W. A. Hanafy, L. Wu, T. Abdelzaher, S. Diggavi, and P. Shenoy, "Failure-Resilient ML Inference at the Edge through Graceful Service Degradation," in *IEEE Military Communications Conference (MILCOM)*, 2023, pp. 144–149.

[4] M.-F. R. Lee and T.-W. Chien, "Artificial intelligence and Internet of Things for robotic disaster response," in *International Conference on Advanced Robotics and Intelligent Systems (ARIS)*, 2020, pp. 1–6.

[5] K. Kant, A. Jolfaei, and K. Moessner, "IoT systems for extreme environments," *IEEE Internet of Things Journal*, vol. 11, no. 3, pp. 3671–3675, 2024.

[6] T. Chen, J. Liu, Y. Xiang, W. Niu, E. Tong, and Z. Han, "Adversarial attack and defense in reinforcement learning-from AI security view," *Cybersecurity*, vol. 2, pp. 1–22, 2019.

[7] "Edge TPU," https://cloud.google.com/edge-tpu, accessed: 2024-08-25.

[8] "Edge TPU Compiler," https://coral.ai/docs/edgetpu/compiler/, accessed: 2024-08-25.

[9] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, "Mitigating edge machine learning inference bottlenecks: An empirical study on accelerating google edge models," *arXiv preprint arXiv:2103.00768*, 2021.

[10] "Profiling-based partitioner for the Edge TPU Compiler," https://coral.ai/docs/edgetpu/compiler/#profiling-partitioner, accessed: 2024-08-25.

[11] A. Parthasarathy and B. Krishnamachari, "Partitioning and Placement of Deep Neural Networks on Distributed Edge Devices to Maximize Inference Throughput," in *International Telecommunication Networks and Applications Conference (ITNAC)*, 2022, pp. 239–246.

[12] S. Zheng, X. Zhang, D. Ou, S. Tang, L. Liu, S. Wei, and S. Yin, "Efficient Scheduling of Irregular Network Structures on CNN Accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3408–3419, 2020.

[13] X. Guo, C. Dong, and W. Wen, "Dynamic Computation Offloading Strategy with DNN Partitioning in D2D Multi-Hop Networks," in *International Conference on Communications and Broadband Networking*, 2021, p. 172–178.

[14] C.-Y. Yang, J.-J. Kuo, J.-P. Sheu, and K.-J. Zheng, "Cooperative Distributed Deep Neural Network Deployment with Edge Computing," in *IEEE International Conference on Communications*, 2021, pp. 1–6.

[15] B. Sun, T. Kloda, J. Chen, C. Lu, and M. Caccamo, "Schedulability Analysis of Non-preemptive Sporadic Gang Tasks on Hardware Accelerators," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2023, pp. 147–160.

[16] B. Sun, T. Kloda, and M. Caccamo, "Strict Partitioning for Sporadic Rigid Gang Tasks," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2024, pp. 252–264.

[17] C. Han, H. S. Chwa, K. Lee, and S. Oh, "SPET: Transparent SRAM allocation and model partitioning for real-time dnn tasks on Edge TPU," in *ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[18] J. Yin, Z. Zhang, and C. Yu, "Exact Memory- and Communication-aware Scheduling of DNNs on Pipelined Edge TPUs," in *IEEE/ACM 7th Symposium on Edge Computing (SEC)*, 2022, pp. 203–215.

[19] J. Yin, Y. Li, D. Robinson, and C. Yu, "RESPECT: Reinforcement Learning based Edge Scheduling on Pipelined Coral Edge TPUs," in *ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[20] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *IEEE conference on computer vision and pattern recognition (CVPR)*, 2017, pp. 4700–4708.

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015, pp. 1–9.

[22] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[23] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[24] S. Hosseininoorbin, S. Layeghy, M. Sarhan, R. Jurdak, and M. Portmann, "Exploring edge TPU for network intrusion detection in IoT," *Journal of Parallel and Distributed Computing*, vol. 179, p. 104712, 2023.

[25] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" *NeurIPS*, vol. 34, pp. 28 877–28 888, 2021.

[26] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-bench-101: Towards reproducible neural architecture search," in *International Conference on Machine Learning*, 2019, pp. 7105–7114.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[28] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.