Future Generation Computing Systems
(Author Responses to Minor Revision Feedback for FGCS-D-24-00804)

PowerTrain: Fast, Generalizable Time and Power Prediction Models to Optimize DNN Training on Accelerated Edges

Prashanthi S. K., Saisamarth Taluri, Beautlin S, Lakshya Karwa and Yogesh Simmhan
Department of Computational and Data Sciences,
Indian Institute of Science, Bangalore 560012, India
Email: prashanthis@iisc.ac.in, simmhan@iisc.ac.in

We thank the reviewers for their detailed review and insightful suggestions on how to improve our article as part of this minor revision. We have incorporated feedback and corrections from the reviews into our paper. Below is our detailed responses and summary of changes made to the article in order to address the reviewers' comments.

I. Reviewer 1

1) *PowerTrain uses transfer learning but does not specify the scope of application. It is experimentally proved that a model pre-trained on Orin AGX can be transferred to Xavier AGX. What if the difference between the user device and Orin AGX is even greater. For example, can a pre-trained model on Orin AGX also be applied to a more powerful GPU (e.g. A100) or a less powerful device (e.g. Raspberry Pi)? If not, to what extent can the model be transferred?*

Response: This is a natural question, to examine how far we can generalize this approach. The focus of our work is on accelerated hardware such as NVidia Jetson that offer a multitude of time and power trade-offs by offering the ability to choose from 1000s of power modes, and operate in a constrained environment where such power and time trade-offs are important. But such a challenge is not encountered in server GPUs that come with pre-defined power limit knobs that can internally trigger frequency scaling automatically to stay within a power limit [1], and do not need methods proposed by us. Other non-accelerated edge devices, such as Raspberry Pi, which have only CPU cores for compute are unlikely to be used for training heavy models and also have a much smaller power footprint (1-10W), making power optimizations less relevant. Therefore, we limit the scope of our article to evaluating Jetson edge devices of different generations and transferring the time and power prediction models among them.

That said, the reviewer's comments spurred us to investigate two aspects. First, we extend our evaluation of PowerTrain to yet another less-powerful Jetson device in the latest Jetson generation, Jetson Orin Nano [2], compared to Jetson AGX Orin and Jetson Xavier AGX we evaluated in the initial submission. The Nano is $6.9\times$ less powerful than the Orin and consumes one-third the power. We present the results for this additional device in Section §4.3.4 to demonstrate the generalizability of PowerTrain to less powerful accelerated edge devices, but with a similar trade-off space. We observe that PowerTrain is able to generalize well to the Nano, and its power and time predictions have less than 10% MAPE for both MobileNet and ResNet.

Second, as a point of reference to compare the compute power of these Nvidia Jetson devices, we ran the 5 training workloads on 3 additional device types and contrast their compute performance against the Nvidia Jetson Orin device used in our experiments [1]: (1) A GPU server with NVIDIA RTX A5000 [3], (2) A GPU workstation with NVIDIA RTX 3090 [4], and (3) The latest generation (non-accelerated) Raspberry Pi5 edge device [5] and report the average epoch times in Figure 1 We observe that the 3090 is significantly faster than the A5000, which can be explained by the 3090 having more CUDA cores of the same Ampere generation ($10,496$ vs. $8,192$). The Orin AGX is slower than the A5000 and the 3090, as expected, due to fewer Ampere CUDA cores (2048). The Raspberry Pi 5 trains using just the ARM CPU cores, and is two orders of magnitude slower than the Orin. BERT was unable to run on the RPi5 (`DNR` in Figure 1) as it ran out of memory in

[1]We use the same library versions for these new devices (LTS versions; PyTorch 2.3, Ultralytics 8.2.25, CUDA 12.1). But these are newer than the versions in Orin AGX due to Nvidia JetPack's slower update cycle.

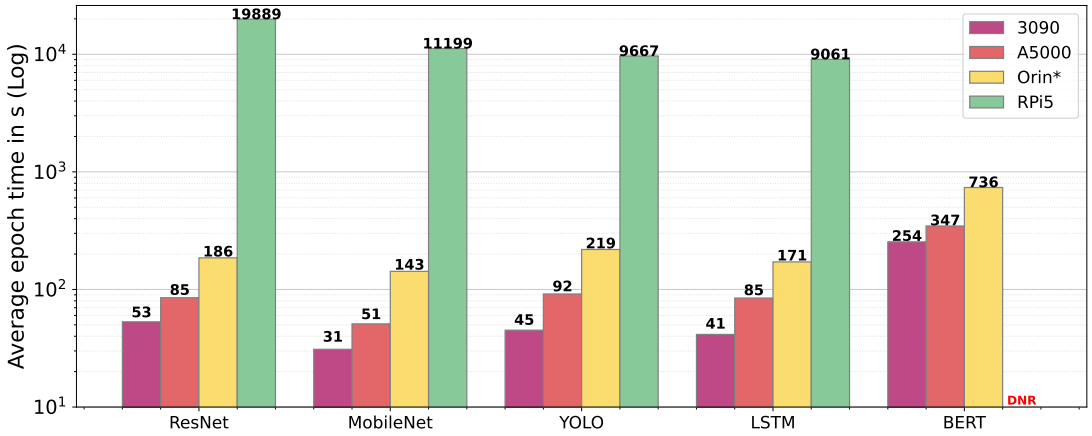

Fig. 1: Average epoch times across various edge and server devices. The device specifications are: **3090** *(12th Gen Intel i9 CPU with 16 cores@ 5.2GHz, 128GB RAM, 3090 Ampere GPU with 10496 cores, 24GB GPU RAM)*; **A5000** *(AMD EPYC 753 CPU with 32 core@ 3.4GHz, 512GB RAM, A5000 Ampere GPU with 8192 cores, 24GB GPU RAM)*; **Orin** *(ARM A78AE CPU, 12 cores@ 2.2GHz, Ampere GPU with 2048 cores, 32 GB shared RAM)*; **RPi5** *(ARM A76 CPU, 4 cores@ 2.4GHz, 8GB RAM)*

8GB of memory available in the RPi. We have added these results to the Appendix and clarified our choice of hardware in this context.

Revision: Since the Orin Nano results directly support our generalizability claims, we have included the results of PowerTrain predictions on the Jetson Orin Nano to Section §4.3.4 (Unseen Device from the same generation) in the revised manuscript as follows.

We also perform a limited set of experiments to examine the generalizability of PowerTrain to a less powerful accelerated edge device, *Jetson Orin Nano* [2], but from the same Jetson generation as the Orin AGX. We randomly sample 180 of the available 1800 power modes for Orin Nano and collect profiling data for ResNet and MobileNet workloads. Using the ResNet workload trained on Orin AGX as the reference model for prediction, we transfer-learn to Orin Nano by retraining using 50 random power modes for ResNet (or MobileNet). We validate the predictions of the transferred model using all 180 power modes we profile for the workload. As shown in Figure 9d, we observe low median time and power errors, with MAPEs of 7.85% and 5.96% for ResNet, and 8.98% and 4.72% for MobileNet. This confirms the strong generalizability of PowerTrain to even Jetson accelerators that are 6.9× less powerful than the Orin AGX. However, while transferring to such very different device types, hyperparameter tuning is needed during retraining. E.g., when transfer learning from Orin AGX to Orin Nano, The loss metric was changed from MSE to MAPE to achieve this good accuracy.

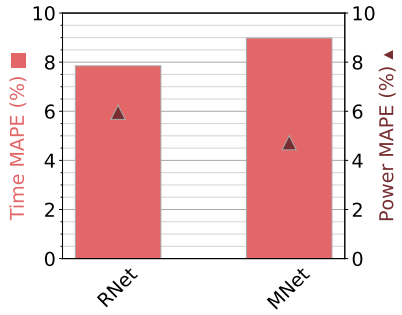

Fig. 2: PowerTrain generalization when reference model is transferred to new workloads, devices and batch sizes: Inter-device, Orin to Nano

We have also added the specification of the Orin Nano device to Table 2 and made changes in Section §2.2 to reflect this.

> ...We also use the Nvidia Jetson Xavier AGX [6] developer kit, the predecessor of Orin AGX, and the Jetson Orin Nano [2], a less powerful device in the same generation of Orin AGX for generalizability experiments in a later section. ~~We refer to the two devices as Orin and Xavier respectively.~~ We refer to the three devices as Orin, Xavier and Nano, respectively.

We have added the results of our training runs on additional device types to the Appendix in Figure A.14 and clarified our choice of Jetson device in Section §2.1.

> Nvidia Jetsons are arguably the most popular high-end accelerated edge devices [7] and are leading the MLPerf benchmarks [8]. Hence, we develop our models and validate them on Jetson devices. As a point of reference to compare the compute power of these Nvidia Jetson devices, we run our 5 training workloads on 3 additional device types (server GPU, workstation GPU, Raspberry Pi 5) and report their training times against the Nvidia Jetson Orin used in our experiments. These are reported in the Appendix in Figure A.14. The Raspberry Pi 5 is two orders of magnitude slower than the Orin, has only CPU cores for compute and is unlikely to be used for training heavy models. It also has a much smaller power footprint (1-10W), making power optimizations less relevant. On the other hand, server GPUs come with pre-defined power limit knobs that can internally trigger frequency scaling automatically to stay within a power limit [1], and do not need methods proposed by us. The Orin offers an interesting combination of near sever-grade compute power with a lot of power modes to tradeoff power and performance.

2) *The baselines of the experiments are quite simple and easy to beat. There are many existing studies on predicting training time/energy on GPU servers. The authors think these prediction methods cannot be used on edge device because of different architectures. However, some existing methods, e.g. literature [20] in the article, do not require the specific architecture. The authors should further explain how the architecture of edge devices prevent these prediction methods from being used. If they can also used on edge device, they should be included in the baselines.*

Response: This is a fair comment. However, existing literature are not directly usable to the problem we solve and hence cannot be used as a comparative baseline. Specifically, *NeuralPower* [9] uses polynomial regression techniques at the granularity of layers to predict the time and power consumed for Neural Network inferencing. The authors use several DNN parameters in their regression, some of which are related to the DNN architecture (E.g., kernel size, stride size, batch size) and others that are related to the physical operations of each layer (E.g., number of memory accesses, FLOPs). While their method can be adapted and fitted to other hardware for inference predictions, it is not designed to be a function of the CPU, GPU or memory frequencies of that hardware. The power modes of Jetson accelerators uniquely allow us to change these hardware parameters and our primary aim is to predict the time and power for training using these hardware configurations. In summary, NeuralPower predicts power and runtime for inferencing using different DNN architectures on a given hardware (since configuration) while we predict power and runtime for training for using different DNN architectures on different configurations (power modes) of a given hardware, and also generalize to other hardware. In fact, a brief part of our earlier work [7] that used linear regression to fit the training time and power as a function of the CPU, GPU and memory frequencies and core-count did not give good accuracy compared to PowerTrain. As a result, we do not consider NeuralPower as a baseline.
Other works have similar shortcomings. Paleo [10] analytically models the training time of CNNs on servers. However, they do not take into account the CPU/GPU/memory frequencies and cannot be used directly to solve the problem we propose. Others [11] use fine-grained CUDA kernel-level modeling based on performance counters to predict power. But this cannot be applied to DNNs because of framework optimizations such as kernel fusion. Thus, there is no other comparable work on server or edge platforms that considers power modes that affect multiple hardware dimensions and predict power and time performance, and which we can use as a baseline. This is a gap we address in this paper.

Revision: We have added the following text to the related work in Section §6.2 of the revised article to clarify this lack of alternative baselines, and specifically why NeuralPower and these other works targeted for servers cannot be used for prediction for edge power modes.

> NeuralPower [9] uses polynomial regression to predict power, runtime and energy consumption of CNNs for inference on server-grade GPUs as a function of the DNN architecture and their resource usage on a hardware. While it can generalize to other hardware, it needs to profile it initially. Our primary focus in this article is to predict training time and power for DNN training on Jetsons when these dimensions are modulated by the power modes. Each power mode effectively becomes a new hardware, causing each to be profiled by NeuralPower rather than predicted by us. Some [11] use linear regression to predict the GPU power consumption of CUDA kernels based on hardware performance counters, but this cannot be applied to DNNs because of framework optimizations such as kernel fusion. Our prior work [7] shows that linear regression on these parameters is inadequate to make high quality predictions. Paleo [10] builds an analytical model of training time for DNNs based on the computational requirements of the DNN architecture, and maps them to the design space of software, hardware and communication strategies on server-grade GPUs. However, they too do not account for the hardware configurations such as frequencies, and are not applicable. Others [12] build a power, performance and energy model for application kernels on desktop or server-grade GPUs. Recent work [13] builds regression models that predict the training speed in a distributed setup with cloud GPUs. All of these focus on server-grade hardware and they do not look into fine-grained power modes or frequencies along many dimensions, which we do. As a result, there are no other works that directly solve the problem that we address, and this also limits our choices for state-of-the-art baselines to empririically evaluate PowerTrain against.

We have also added another related work Oppertune [14] in Section §6.3.

> Zeus [1] studies the trade-off between training time and energy...Oppertune [14] addresses post-deployment optimization of applications by figuring out which parameters to tune and using reinforcement learning to tune both numerical and categorical variables. However, they often sample 100s of configurations, which is much costlier in our case and not usable.

3) *Using PowerTrain requires many user settings, but the author does not clarify how to select these settings. The setting includes: 1) NN achitecture, 2) profiling epoch number for pre-training/fine-tuning, 3) the number of sampled power modes for pre-training/fine-tuning, 4) the type of reference DNN, etc. The setting chosen by the author work well in the experiments. However, how does users choose these settings if their devices for pre-training/fine-tuning and DNN workloads are different?*

Response: We regret this oversight in justifying our choice of various parameter configurations, and listing all hyperparameters used. These include the following:

1 We have chosen the NN architecture based on a hyperparameter search (Section §3.1). For clarity, we have now represented the NN architecture pictorially in Figure 4

2 We performed experiments with $10, 20, 30$ and 40 minibatches when profiling the pre-training and fine-tuning. The time and power MAPEs were similar across minibatch sizes used during training. E.g., when training using $10, 20, 30$ and 40 minibatches, the time MAPEs for the for pre-training models were 10.1%, 10.0%, 14.9% and 10.2% and the power MAPEs were 4.4%, 4.1%, 4.1% and 3.2% respectively. We consciously select 40 minibatches during training because having fewer minibatches for faster power modes resulted in zero samples to be collected during power profiling (available only at 1 second granularity). We have included a discussion on this in the revised article.

3 The number of sampled power modes for pre-training has been selected as a 90:10 split (as is common) over the full profiled set of power modes, i.e., 4368 power modes used for training. We performed a study on the impact of the number of power modes used during pre-training and do not observe any noticeable difference in MAPEs when using ≥ 546 power modes. It did not have any impact on Transfer Learning accuracies either. Since pre-training is a one-time profiling activity, we use a larger 90% subset. We have

discussed this in the revision.

4 The number of sampled power modes for fine-tuning and its effect on prediction accuracies are discussed earlier in Section §4.2 and Figures 6 and 7.

5 The choice of reference DNN was made based on the Transfer Learning Matrix shown in Figure 5 and discussed in Section §4.1. We notice that ResNet performs the best to transfer from because it has the highest variation in power values and is able to generalize well.

For other devices and workloads, the same NN architecture, sampled power modes and profiling minibatches can be used, as we show in our generalizability experiments.

Revision: In the revised manuscript, we have added this NN architecture in Figure 2.

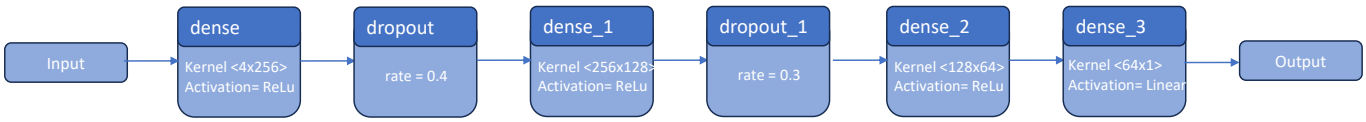

Fig. 3: NN Architecture

We have also included a new Table 1 listing all the hyperparameters we use, as given below. Further, we have

Feature	**Value**
Layers	4 (Dense)
Dropouts	After layers 1 and 2
No of neurons	256, 128, 64, 1
Activation	ReLu x 3, Linear
Optimizer	Adam
Loss function	MSE
Learning rate	0.001
Training epochs	100
Profiling minibatches	40
Power modes (Ref)	4,368
Power modes (TL)	50

TABLE I: NN hyperparameters

also added the following details justifying our choice of the number of power modes used during training and the number of minibatches used for profiling. Specifically, we expand Section §2.5 as follows:

> Our prediction models use as input the profiling information collected during DNN workload training for a specific power mode of the device. When profiling each power mode, we train the candidate DNN workload for ≈ 40 minibatches, and record the training time per minibatch and the sampled power usage during the training period. We conducted a sensitivity study by varying the number of minibatches used when profiling each power mode and evaluating its impact on the accuracy of the trained time and power prediction models. There is minimal impact on the pre-training and fine-tuning accuracies when varying from 10–40 minibatches (e.g., MAPE range of 10.0–14.9% for time prediction, 3.2–4.4% for power prediction during pre-training) since the training time and power are remain stable across minibatch samples. However, we notice that with fewer minibatches and at faster power modes, the power profiling is unable to collect any telemetry since the sampling interval is $1s$ and the training of all the minibatches completes within this interval. As a result, we conservatively pick 40 minibatches to train over during profiling.

We also include this in Section §3.2:

> As discussed later (§4), we train the reference NN for one of the three default DNN workloads using the full corpus of $4.4k$ power modes as training data. Further, we test the impact of the number of power modes used in training the reference model, increasing it from 500 to 4368. We do no observe find any

significant difference in MAPEs when predicting time or power for other power modes of the reference model, or for the predictions returned by the transfer-learned models. Since training the reference model is a one-time profiling activity, we use the full set of 4368 power modes.

II. Reviewer 2

1) *Expanding Hardware Compatibility Testing: While PowerTrain is shown to perform effectively on Orin AGX with potential transferability to Xavier AGX, expanding its testing to include lower-end models like the Jetson Nano could provide valuable insights into its broader applicability. It would also be beneficial to explore whether the approach can be adapted beyond the Jetson series to other edge computing hardware, enhancing its versatility.*

Response: We appreciate this valuable comment. A similar suggestion was also offered by *Reviewer 1*.

We have extended our evaluation of PowerTrain to include Jetson Orin Nano [2], a less powerful edge accelerator than Jetson AGX Orin and Jetson Xavier AGX but from the latest generation of Jetsons. We include these results in Section §4.3.3 of the revision to demonstrate the generalizability of PowerTrain to less powerful edge devices.

Regarding the applicability of PowerTrain to other hardware, we provide a detailed response above in Response 1.1. In short, the benefits of PowerTrain are applicable to edge accelerators like Jetson where there are 1000s of power modes to explicitly choose from, and such a diversity is not seen in other server and workstation platforms.

Revision: We have added the results of PowerTrain predictions on the Jetson Orin Nano to Section §4.3.3 and highlighted the scope of this work to Jetson-like devices, as described in Response 1.1.

2) *Impact of Batch Size on Training Metrics: The manuscript presents an exploration of predicting power consumption and training time for DNN training on edge devices, yet it could be enhanced by addressing how batch sizes influence key performance metrics, specifically training time and power consumption. As highlighted in related works such as Zeus [18], which studies the trade-offs between training time and energy, discussing batch size variations could significantly enrich this manuscript. Choosing the right batch size can reduce energy consumption by 23.8%-74.7% for diverse workloads [18], emphasizing the critical impact of these factors. Batch size has a notable effect on GPU utilization, where larger batch sizes generally enhance GPU throughput, potentially reducing training time. These changes may also affect power consumption due to increased GPU utilization, aligning directly with the manuscript's focus on predicting these two crucial metrics.*
Specifically, it would be insightful to explore whether PowerTrain can maintain accurate predictions of training time and power consumption when training the same workload (DNN and dataset) with varying batch sizes, and identifying an optimal batch size that maximizes training efficiency within a given power budget would offer a more comprehensive view of the balance between training efficiency and power use. Integrating this discussion would provide a comprehensive view of the balance between training efficiency and power use, thereby improving the relevance and applicability of PowerTrain across diverse computational scenarios.

Response: Finding the optimal batch size for a given workload is indeed an important problem during training. However, it is not the primary focus of our work. While automatically selecting the minibatch size to tune training time, power and training accuracy is beneficial, we scope our work to the less examined problem of tuning the power modes (CPU/GPU/memory frequencies, CPU core count) from among 1000s by estimating the training time and power taken for a given power mode, which is complex enough. We do see the potential for extending our work to also include training minibatch size as a tuning factor. We have included this in our future work in Section §7. As a sanity check, we also train to stable accuracy for the DNN workloads

and report the number of epochs taken – this impacts the duration of training and the energy consumed, and motivates the need for the PT optimizations.
We do agree that it is important to verify if the time and power predictions given by PowerTrain are accurate for different minibatch sizes configured for the DNN training. Following the reviewer's suggestion, we have evaluated PowerTrain for 3 different minibatch sizes, $8, 16, 32$ (one higher and lower than our default 16 minibatches) when training two DNN models (ResNet and MobileNet) and present results on training time and power prediction. PowerTrain adapts well to different batch sizes for both models, with time MAPEs under 12% and power MAPEs under 7.3%.

Revision: We have made several edits to the revised article to address this feedback.

We have conducted experiments to evaluate minibatch sizes and reported the results in Section §4.3.4 as follows:

> We extend the evaluation of PowerTrain to predict the training time and power consumption when using 3 different minibatch sizes during training: $8, 16$ and 32, which are commonly used. We test this for 2 DNN workloads, ResNet and MobileNet. Each new minibatch size is seen as a new DNN training workload.
> In the first experiment, we use our reference NN, trained on ResNet with a minibatch size of 16 (ResNet/16), and transfer learn onto the same ResNet DNN, but using minibatch sizes of 8 and 32 (ResNet/8 and ResNet/32). We profile and use 50 power modes for transfer learning. As shown in Figure 9e, for ResNet/8, our time predictions have a median MAPE of 10.84% and power prediction a median MAPE of 6.86%. For ResNet/32, the time and power MAPEs are comparable at 11.2% and 7.28%, respectively. These are similar to the MAPE for the default ResNet/16, which was 8.83% and 7.39% for time and power respectively.
> In the second experiment, we transfer the same reference NN (ResNet/16) to a different DNN workload and with different minibatch sizes, MobileNet/8, /16 and /32. PT's predictions generalize well here too. The median MAPE for time predictions range narrowly from 7–9.4% and for power between 5.5–5.7%. This shows that PowerTrain is robust to changes in training minibatch sizes.

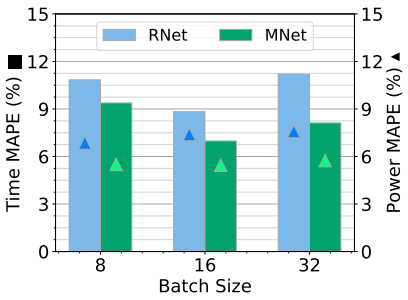

Fig. 4: Varying batch sizes

We have changed our Contributions in Section §1.6 to reflect consideration of batch sizes.

> We comprehensively validate NN and PT for accuracy and generalizability across 7 different DNN workloads and ~~a~~ two different Jetson edge devices. We also validate PT predictions on 3 different training batch sizes across 2 workloads.

We have also performed training to target accuracy runs and clarified our time benefit in that context. This is added to Section 1.4.

> ...PT has the lowest time penalty of 1% in excess of the optimal while also having the lowest percentage (26.5%) that exceeds the power limit by over $1W$ above the threshold. This low time penalty is especially beneficial when doing long training runs over several epochs. For instance, we see that a typical training of YOLO takes 200 epochs to converge, lasting almost 49 hours on the Orin. A 12% time benefit achieved

by PT accrues over all epochs and the training can complete 5.88 hours faster. Similarly, MobileNet takes 148 epochs (50 hours) to converge, with time reduction of 6.5 hours when optimized using PT.

Finally, we include optimization of training minibatch size as a future work in Section §7.

...We plan to explore variants of this problem, such as time and power prediction for concurrent training and inference workloads and power-aware minibatch size optimization for training workloads.

3) *Visual Clarity and Layout of Figures: Consider refining Figures 2 and 3 to optimize the use of space and improve the logical flow. Tightening the layout in Figure 2 to reduce whitespace and introducing clearer, sequential labeling in Figure 3 can greatly enhance clarity.*

Revision: We have improved Figures 2 and 3 in accordance with the reviewer's suggestion. In Figure 2, we have increased the width of the bars for better readability and used both columns for the 3 subfigures for better space utilization.
In Figure 3, we have introduced sequential labeling for each of the steps and modified the text to reflect this. Correspondingly, we have updated the text to reflect this labeling. Section §3.2 has been updated as follows:

Adopting this, ~~PowerTrain first bootstraps a reference NN model each for time and power using the architecture described above for any one DNN workload on a large set of profiling data (see Figure 3)~~ PowerTrain uses a reference DNN workload (1(a) in Figure 3), performs detailed profiling (1(b)) to collect a large set of profiling data (1(c)) which is then used to bootstrap a reference NN model each for time and power (1(e)) using the architecture described above.
Subsequently, when a new DNN workload arrives, we modify this reference NN slightly by removing the last dense layer and adding a fresh layer. ~~We then fine-tune the reference NN model by training it on a limited set of profiling data for the new DNN workload.~~ We then take the new DNN workload (2(a) in Figure 3), perform limited profiling (2(b)), and use this time and power data (2(c)) to fine-tune the reference NN model by performing Transfer Learning (2(d)). This again generates NN models, one each for time and power (2(e)). The intuition is to retain and utilize the representations learned in the internal layers of the neural network for the reference DNN workload and only change the final output layer in accordance with the new DNN workload's profiling behavior. We do this for both time and power prediction models.

We have also updated Section §5 with these labels:

Using the prediction models, we follow a similar approach, with the key difference that the models are used to estimate the training time and power for all possible ($4.4k$) power modes (3(b) in Figure 3). We build a similar Pareto from PT predictions and this predicted Pareto (*PT Pred Pareto* in Figure 11 and 3(c) in Figure 3) is used for the optimization decisions. Specifically, for a user-specified power limit (3(a)), we perform a lookup on the Predicted Pareto to find the optimal power mode (2(d)).

4) *There are a few typographical errors present in the text that need addressing*

Response and Revision: We regret these. We have done a thorough check for typographical and grammatical errors and fixed them.

REFERENCES

[1] J. You, J.-W. Chung, and M. Chowdhury, "Zeus: Understanding and optimizing {GPU} energy consumption of {DNN} training," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 119–139.

[2] NVIDIA., "Jetson orin nano developer kit," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/, 2023.

[3] NVIDIA, "Rtx a5000," https://www.nvidia.com/en-us/design-visualization/rtx-a5000/, 2021.

[4] ——, "Rtx 3090," https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/, 2020.

[5] R. Pi, "Raspberry pi 5," https://www.raspberrypi.com/products/raspberry-pi-5/, 2024.

[6] NVIDIA, "Jetson xavier series," https://www.nvidia.com/en-in/autonomous-machines/embedded-systems/jetson-agx-xavier/, 2018.

[7] Prashanthi S.K, S. A. Kesanapalli, and Y. Simmhan, "Characterizing the performance of accelerated jetson edge devices for training deep learning models," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 3, December 2022.

[8] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "Mlperf inference benchmark," in *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, 2020.

[9] E. Cai, D.-C. Juan, D. Stamoulis, and D. Marculescu, "Neuralpower: Predict and deploy energy-efficient convolutional neural networks," in *Asian Conference on Machine Learning*, 2017.

[10] H. Qi, E. R. Sparks, and A. Talwalkar, "Paleo: A performance model for deep neural networks," in *5th International Conference on Learning Representations, ICLR*, 2017.

[11] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of gpu kernels using performance counters," in *International Conference on Green Computing*, 2010.

[12] S. Song, C. Su, B. Rountree, and K. W. Cameron, "A simplified and accurate model of power-performance efficiency on emergent gpu architectures," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, 2013.

[13] S. Li, R. J. Walls, and T. Guo, "Characterizing and modeling distributed training with transient cloud gpu servers," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 2020.

[14] G. Somashekar, K. Tandon, A. Kini, C.-C. Chang, P. Husak, R. Bhagwan, M. Das, A. Gandhi, and N. Natarajan, "OPPerTune: Post-Deployment configuration tuning of services made easy," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024.