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

Prashanthi S. K.[a,*], Saisamarth Taluri[a], Beautlin S[a], Lakshya Karwa[a], Yogesh Simmhan[a]

[a]*Department of Computational and Data Sciences, Indian Institute of Science, Bengaluru, 560012, India*

Abstract

Accelerated edge devices, like Nvidia's Jetson with 1000+ CUDA cores, are increasingly used for DNN training and federated learning, rather than just for inferencing workloads. A unique feature of these compact devices is their fine-grained control over CPU, GPU, memory frequencies, and active CPU cores, which can limit their power envelope in a constrained setting while throttling the compute performance. Given this vast 10k+ parameter space, selecting a power mode for dynamically arriving training workloads to exploit power–performance trade-offs requires costly profiling for each new workload, or is done *ad hoc*. We propose *PowerTrain*, a transfer-learning approach to accurately predict the power and time that will be consumed when we train a given DNN workload (model + dataset) using any specified power mode (CPU/GPU/memory frequencies, core-count). It requires a one-time offline profiling of 1000s of power modes for a reference DNN workload on a single Jetson device (Orin AGX) to build Neural Network (NN) based prediction models for time and power. These NN models are subsequently transferred (retrained) for a new DNN workload, or even a different Jetson device, with minimal additional profiling of just 50 power modes to make accurate time and power predictions. These are then used to rapidly construct the Pareto front and select the optimal power mode for the new workload, e.g., to minimize training time while meeting a power limit. PowerTrain's predictions are robust to new workloads, exhibiting a low MAPE of $< 6\%$ for power and $< 15\%$ for time on six new training workloads (MobileNet, YOLO, BERT, LSTM, etc.) for up to 4400 power modes, when transferred from a ResNet reference workload on Orin AGX. It is also resilient when transferred to ~~an~~ two entirely new Jetson devices (Xavier AGX and Jetson Orin Nano) with prediction errors of $< 14.5\%$ and $< 11\%$. These outperform baseline predictions by more than 10% and baseline optimizations by up to 45% on time and 88% on power.

Keywords: Edge computing, Edge accelerators, DNN training, Performance modeling, Performance optimization

1. Introduction

The growth in domains like autonomous vehicles [1] and Internet of Things (IoT) for smart cities [2] are leading to edge computing devices emerging as a first-class computing layer. Edge devices are diverse in their compute capacity and power, ranging from Raspberry Pis to Nvidia Jetsons. Unlike low-end accelerated edges like Google's Coral and Intel's Movidius are designed for Deep Neural Network (DNN) inferencing, Nvidia's Jetson series of GPU accelerated edge devices offer performance that approaches GPU workstations [3] with a compact form-factor and low power envelope. E.g., the latest generation Jetson Orin AGX has 2048 Ampere CUDA cores, with performance comparable to an RTX 3080Ti workstation, but is as small as a paperback novel and has a peak power of under 60 W. So accelerated edges are competitive candidates for DNN training with power-constraints [4].

Edge devices deployed as part of private networks and edge clouds are particularly amenable for use in DNN training. Training can be done on private edge devices deployed within a single region or over a wide area network, e.g., over data from sensors deployed in factory to detect faults [5], from field cameras deployed by scientific collaborations to detect forest fires [6], and over video streams from a network of city safety cameras [7]. These complement training on pay-as-you-go shared private edge cloud infrastructure that is colocated with content distribution networks (CDNs) and 5G towers [8, 9]. Besides opportunistically making use of captive compute in private edge cloud systems, training on the edge can avoid the high bandwidth needed to move data to public clouds for training, e.g., from 1000s of traffic cameras [10]. Lastly, a key motivator for edge training is driven by privacy reasons, e.g., in healthcare and fintech, where data collected on the edge cannot be moved out, even within secure networks due to regulatory requirements. Federated learning [11, 12] leverages fleets of heterogeneous edge devices in edge cloud systems to train models locally over local data for subsequent aggregation on the cloud across multiple epochs and rounds. Such training can happen *recurrently* as part of continuous learning to adapt to data drift [13, 14], and involve *diverse DNN workloads* trained on captive edge devices to meet evolving application needs, e.g., using the same forest cameras to detect forest fires or to classify and count wild animals.

*Corresponding author

Email addresses: prashanthis@iisc.ac.in (Prashanthi S. K.), simmhan@iisc.ac.in (Yogesh Simmhan)

Edge training can also operate under environmental or user constraints. There may be *energy constraints* imposed by power banks on drones or solar-charged batteries in a forest. There can be *power limits* to avoid overheating of poorly ventilated IP-67 enclosures in an industrial site [15]. There can also be *time constraints*, e.g., to meet a training deadline for an application or limit the billed cost. In federated learning, such time estimates may also guide device selection [16, 17]. So, it is important to *configure* the edge accelerator to adapt to these constraints optimally and offer *predictable power, energy and time estimates* for DNN training.

1.1. Challenges

A unique feature of Nvidia's Jetson edge devices is their fine-grained control over CPU, GPU and memory frequencies and active CPU cores, enabled by dynamically setting their *power modes*. E.g., the Orin AGX offers 29 CPU frequencies up to 2.2GHz, 13 GPU frequencies up to 1.3GHz, 4 memory frequencies and 12 CPU core-count settings for a total of $\approx 18,000$ possible power modes (Table 2). These can help limit the power envelope in a constrained setting while throttling the performance. However, given this vast parameter space, selecting an *optimal power mode* to make such trade-offs for training workloads that arrive dynamically is non-trivial. Some frequencies span an order of magnitude, and our experiments show that they can have up to 36× impact on DNN training time and 4.3× impact on power usage. For instance, using a low power mode to train the ResNet model on ImageNet data (see § 2 for details) on Orin AGX takes 112 mins per epoch and consumes about 11.8 W of power, while increasing it to a higher MAXN power mode reduces the training time to 3.1 mins but increases the power load to 51.1 W.

These also vary a lot across *DNN workloads* that may arrive continuously over time, e.g., the BERT model on SQuAD dataset using the same MAXN power mode for Orin AGX takes a much longer 68.7 mins with a power of 57 W compared to ResNet which only took 3.1 mins above. As expected, these also vary substantially across *Jetson device generations*, which may co-exist in a deployment as new devices are deployed or old ones upgraded in a rolling manner. E.g., the AGX Xavier takes 8.47 mins per epoch and 36.4 W to train the ResNet model on ImageNet dataset, as against 3.1 mins and 51.1 W for AGX Orin, on the MAXN power mode on both devices.

Hence, a wrong power mode choice can cause high power and performance penalties, violating time and power load constraints or, in the worst case, destroying the device due to overheating. So, accurately estimating the power and time taken for training a given DNN workload on the edge accelerator using a specific power mode is critical [1].

A naïve solution of benchmarking all power modes for the device for each new DNN model is intractable. E.g., it takes 16.3 *h* to profile even 25% of power modes for Orin AGX for the ResNet model on ImageNet data, and this will not scale for every workload, especially if it arrives dynamically and has time constraints. Random or even targeted sampling of some power modes is inaccurate for optimizations, as we show later. Practical deployments may have multiple generations of accelerated devices, amplifying these costs. So, a *principled approach with low overheads and generalizability* is required to build prediction models to estimate training time and power for diverse DNN workloads on edge computing systems.

1.2. Gaps

There is substantial work on optimizing the energy consumption of DNN training on GPU servers [18] and predicting their runtime [19], and power and energy consumption [20]. But these do not translate directly to the edge due to architectural differences, such as the shared memory across CPU and GPU for edge accelerators, and the use of ARM-based rather than x86 CPUs. Similarly, several studies have examined energy and performance profiling for edge inferencing rather than training [21, 22, 23]. Some have also explored power modes and frequency scaling for micro-benchmarks on older Jetson architectures [3]. However, inferencing workloads have low compute demand compared to training, which can stress all resources of the edge device.

Our previous work focused on characterizing the performance of the previous generation Jetson Xavier AGX and Nano devices for DNN training, but offered only limited preliminary results on actual performance predictions or tuning of power modes [4]. Nvidia offers a limited set of pre-defined power mode choices (3 for Orin AGX apart from the default MAXN) with varying power budgets, but these are too few to allow targeted optimizations. As we show later, Nvidia's own tool highly overestimates the power load for Orin AGX, causing excess training time when meeting a power budget. In summary, there is a clear need for modeling, prediction and optimization studies for DNN training workloads on accelerated edges, but none exist. *We address this gap, and offer the first principled modeling and prediction approach of its kind in this work.*

1.3. Proposed Approach

We propose two data-driven approaches to address this challenge. First, as an intelligent baseline, we train two Neural Network (**NN**) models – one each for time and power predictions – over a subset of power modes for the given DNN workload and use this to make predictions for unseen power modes for that same DNN workload. Our second, more flexible approach, **PowerTrain**, trains such NN models over a large corpus of power modes measured for some reference DNN workload (Figure 1). When a new DNN workload arrives, it uses transfer learning over the time and power telemetry collected from profiling a small sample of $\approx$ 50 power modes for the new workload to retrain updated NN prediction models for it [2].

[1] Since *energy* (mWh) = *power* $(mW) \times$ *time* (h), we focus on predicting power and time for training, and can derive energy estimates from these.

[2] To avoid confusion, we use the term DNN to refer to training workloads, e.g., MobileNet, ResNet and YOLO and their datasets. We use the term NN to refer to neural network models we train for predicting the time and power for a DNN workload, and PowerTrain (PT) to refer to our specific transfer-learning approach.

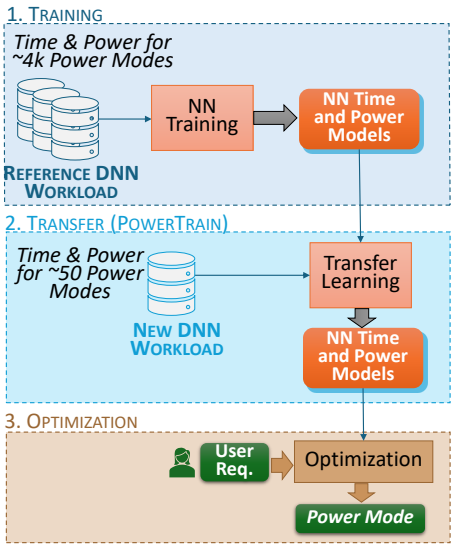


Figure 1: Time and Power Prediction Models using PowerTrain: (1) Initial training, (2) Transfer, and (3) Use for Optimization

The PowerTrain prediction models are then used to estimate the expected training time and power for all possible power modes. This is fast and can be used to find the Pareto front [3] across time and power, and solve a user's optimization goal, e.g. the power mode with the fastest training time for a given power limit, or the lowest power for a given time budget.

As is intuitive, PowerTrain has lower profiling overheads than NN for new workloads and, as we show, offers competitive prediction accuracies for subsequent optimizations It has a one-time profiling overhead on the reference workload and to train the initial NN prediction models. But the incremental overhead for a new workload is limited to profiling 10s of power modes. Further, PowerTrain generalizes to new DNN workloads, new datasets and, interestingly, to even a new device type, e.g., from a reference model trained on Orin AGX to a new workload running on Xavier AGX. Using just the NN based prediction models trained from scratch on 10s of power modes for a new workload instead of transfer learning from a pre-trained model is sub-optimal, taking at least twice as much profiling data to achieve similar prediction accuracies as PowerTrain.

1.4. Representative Results against Baselines

While a detailed evaluation is presented in § 4 and § 5, we offer some representative results here to illustrate the comparative benefits of our approach.

Predictions. Nvidia offers a *PowerEstimator* tool (NPE) [4] to estimate the power usage by Orin AGX for a specific power mode. We use this to estimate the power for two diverse power modes (PMi) each when training three DNN workloads. We also use our PowerTrain (PT) approach, using ResNet as the reference workload on Orin AGX, to predict the power usage for these workloads, and report the Mean Average Percentage Error (MAPE %) relative to the actual observed power usage in Figure 2a. Other than for PM1 for ResNet, where PT has a slightly larger error (5%) than NPE (4%), we give much better predictions in all other cases while NPE consistently overestimates.

Optimization. Having predictions for the power and training time for a DNN workload for any given power mode can help users optimize the system to meet different goals. E.g., in § 5, we find the power mode for a DNN training workload such that the power remains within a budget while the training time is minimized. Nvidia recommends 3 default power modes [5] along with their power budgets for users to choose from: 15 W, 30 W and 50 W. We use the best of these three to meet an optimization goal, defined as falling within a power limit while minimizing training time. We also solve this using our PowerTrain prediction models to discover a Pareto front (described in § 5) to select a custom power mode. We compare the observed training time for these two solutions against the ground-truth optimal found through brute force for these 3 power limits. Figure 2c shows that PowerTrain (PT) has the fewest % of solutions that exceed the optimal for 5 out of 6 cases (except ResNet 15W) compared to Nvidia's suggestions (NV), while falling within the power limits in most cases.

Lastly, in Figure 2b, we see that the PT-based optimization also outperforms simpler baselines like choosing the default MAXN power mode and doing random (RND) profiling of 50 power modes to build a partial Pareto and selecting the best, and our better baseline using a Neural Network (NN) trained on 50 power profile samples. For a set of optimization problems with power limits varying from 17–50W, 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.

These exemplars motivate the need for and illustrate the success of our robust PowerTrain prediction model to support power mode tuning for training DNN workloads on Jetson edge devices.

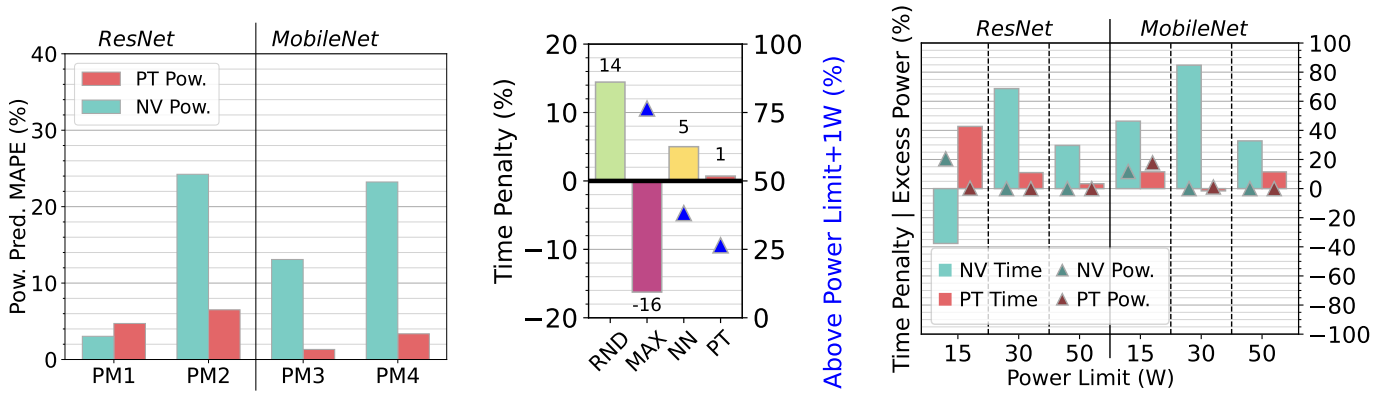
1.5. Discussion

It is useful to understand the scenarios under which these different approaches, including baselines and our proposed ones,

[3] The Pareto front is defined as a subset $\widehat{\mathbb{C}}$ from a set of choices $\mathbb{C} = \{c_1, c_2, ...\}$ that affect two optimization (say, minimization) variables X and Y (e.g., power modes that affect time and power). The front contains the set of trade-off choices such that for any given $\widehat{c_i} \rightarrow (x_i, y_i)$ present on the front, there does not exist any other $c_j \rightarrow (x_j, y_j)$ such that $x_j < x_i$ and $y_j < y_i$.

[4] https://jetson-tools.nvidia.com/powerestimator/

[5] https://docs.nvidia.com/jetson/archives/r35.1/DeveloperGuide/text/SD/PlatformPowerAndPerformance/JetsonOrinNxSeriesAndJetsonAgxOrinSeries.html



(a) Prediction Accuracy, PowerTrain (PT) vs. Nvidia Tool (NV)

(b) Optimization Performance, PowerTrain vs. Baseline Models

(c) Optimization Performance, PowerTrain (PT) vs. Nvidia Tool (NV)

Figure 2: Representative and comparative results on prediction and optimization. The power modes used in (a) are: **PM1**:12c/1.65C/0.62G/3.19M, **PM2**:12c/2.20C/1.23G/3.19M, **PM3**:10c/1.65C/0.82G/3.19M, **PM4**:12c/2.20C/1.03G/3.19M

Table 1: Summary of scenarios and solution approaches

Scenario	Frequency of training	DNN Workload Changes?	DNN Training Time	Suitable Solution	Data Collection Overhead for Solution
Training once on a large dataset	One time	Never	Few days	Brute force; Profile all power modes and pick the best	1200–1800 min
Fine-tuning a Model	Occasional	Rare	Few hrs	Neural Network; Profile at least 100 power modes	20–50 min
Continuous learning	Periodic	Rare	< 1 hr	PowerTrain; profile 50 power modes and transfer	10–20 min
Federated learning on edge cloud	Often	Often	Unknown	PowerTrain; profile 50 power modes and transfer	10–20 min

are best suited for. In Table 1, we outline four potential application and deployment scenarios, their characteristics and the solution approach is most applicable. These estimates are based on our detailed experiments later reported in Section 4.

For instance, there may be a one-time training workload submitted to an edge service that runs new and large DNN model on a large corpus of user data. Since the training is time-consuming, running into days, having the perfect power mode is helpful even if the profiling time is costly. Here, a brute force approach may be recommended since the data collection time is approximately a day, which is well within the runtime of the workload. This kind of training workload is uncommon on the edge, and this approach is therefore impractical.

Fine-tuning a pre-trained DNN on a smaller dataset is a more common edge usecase (e.g., model personalization on a private edge device) where the DNN training time is typically a few hours. In this scenario, the DNN workload seldom changes and our NN approach using profiling over 100s of power models is feasible because of the longer training time.

Continuous learning is another common scenario on edge devices where the same DNN is regularly retrained with small batches of new incoming data to address data drift. Since the training workload runs for a shorter time here, PowerTrain is the best approach since it has a smaller data collection time.

Further, when the edge device is part of a federated learning setup or a private edge-cloud supporting multiple applications, the device could be assigned any DNN training workload, submitted often and with an unknown training duration. In such a case, it is best to use PT to minimize data collection time for the new workload and ensure that the optimization criteria are met before the workload changes.

1.6. Contributions

We make the following specific contributions in this article:

1. We develop a Neural Network (NN) based prediction approach to *estimate the time and the power* for DNN training workloads on the Nvidia Jetson Orin AGX for any given power mode (§ 3) using a large profiling corpus. This is extended into PowerTrain (PT), a *Transfer Learning* based approach that greatly reduces the online profiling overheads for any new DNN workload on Orin or another Jetson device.

2. 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. (§ 4).

3. Finally, we apply PT predictions to optimize the training time and power limit trade-offs for several DNN workloads (§ 5), and demonstrate that it out-performs other baselines, including NN.

This modeling work leverages a prior in-depth study that we conducted on the effect of various factors, including power modes, on the Jetson devices for DNN training [4]. There, the focus was on the characterization and as a potential use-case of that study, we had proposed a simple linear regression technique (Section 5.7 in [4]) for predicting the DNN training time and energy on the Jetson devices, validated on 10 power models for one Jetson device, with mixed results. The present article focuses entirely on predicting the time and power for DNN workloads on Jetsons, proposing a wholly different, novel NN-based prediction technique along with the PowerTrain transfer learning approach for fast retraining and robust estimates. We further present optimization results as a case study of using our prediction models. All these contributions are novel, with no more than 20% conceptual overlap with the prior work.

2. Experiment Setup

We first offer a detailed description of the DNN workload and training configurations to highlight the rigor of our profiling setup, the diversity of our workloads, and to allow reproducibility.

2.1. Choice of hardware platform, applications, etc.

Nvidia Jetsons are arguably the most popular high-end accelerated edge devices [4] and are leading the MLPerf benchmarks [24]. 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 [18], 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. That said, these techniques can later be extended to other hardware platforms which offer a choice of power mode trade-offs. Since *energy* can be derived as a function of *power* and *time*, we focus on these two component metrics. Further, we focus on time taken for a minibatch of training as this is the practical unit of training, and the time taken for an epoch can be derived by multiplying the minibatch time with the number of minibatches in one epoch of the training data. We focus on DNN training workloads given their need for high compute and the growing prevalence of DNN training on the edge, e.g., federated learning. Other non-DNN workloads can be considered as future work.

2.2. Hardware and Settings

Nvidia Jetson Orin AGX [35] is the latest generation of the Jetson edge accelerators, released in March 2023. The specifications of the development kit used in this study is given in Table 2. The Orin supports several custom power modes, as described earlier and also shown in the table. The Orin devkit features INA3221 power sensors, from which we read the present power consumption of the device during our experiments. We set the fan to maximum speed to avoid any thermal throttling effects during our experiments. We disable Dynamic Voltage and Frequency Scaling (DVFS) so that the frequencies remain static at the value we have configured them to. Beside the GPU CUDA cores, the Orin also has two special purpose accelerators, DLA and PVA, which we do not use and leave in their default states and turned off. We also use the Nvidia Jetson Xavier AGX [36] developer kit, the predecessor of Orin AGX, and the Jetson Orin Nano [37], 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.

2.3. Training Framework and DNN Workloads

Orin runs Ubuntu 20.04 LTS and L4T kernel version 5.10.65. We configure it with Nvidia JetPack version 5.0.1, CUDA v11.4, PyTorch 1.12 and torchvision v0.13.

We select as our default workloads three popular computer vision DNN architectures and datasets that can train within the available resources of the Orin (Table 3): *ResNet-18* and *MobileNet v3* that perform image classification, and *YOLO v8n* that does object detection task, coupled with training datasets to form the workloads. MobileNet is a lightweight vision model optimized for smartphone applications. Our workload trains MobileNet using images from the *Google Landmarks Dataset v2 (GLD-23k)*, which consists of 23,080 photos of both human-made and natural landmarks, categorized into 203 different classes. ResNet is a family of Convolutional Neural Networks (CNNs) used for image classification, featuring residual blocks and skip connections. We train ResNet-18 on the validation subset of *ImageNet*, which consists of 50,000 images and occupies 6.7GB on disk.

YOLO v8 is the latest iteration in the popular "You Only Look Once" series of real-time object detectors. We use YOLO v8n, the smallest of its family, to train on a subset of the MS COCO dataset, *COCO minitrain*, which has 25,000 images which take up 3.9GB on disk. In addition, we also use *BERT* and *LSTM* DNN architectures with the SQuAD and Wikitext training datasets, used for query-response and next-word prediction, respectively, to explore the generalizability of our methods to other DNNs. All of these are representative of typical edge and federated learning workloads [16, 38, 39], and provide a wide diversity in DNN architectures (CNN, LSTM, Transformer), dataset sizes (17.8MB–6.7GB) and computational requirements (3.2M–110M parameters, 225M–11.5T FLOPS).

Table 2: Specifications of NVIDIA Jetson Orin AGX, Xavier AGX and Orin Nano devkits used in evaluations

Feature	Orin AGX	Xavier AGX	Orin Nano
CPU Architecture	ARM A78AE	ARM Carmel	ARM A78AE
# CPU Cores	12	8	6
GPU Architecture	Ampere	Volta	Ampere
# CUDA/Tensor Cores	2048/64	512/64	1024/32
RAM (GB)/Type	32/LPDDR5	32/LPDDR4	8/LPDDR5
Peak Power (W)	60	65	15
Form factor (mm)	$110 \times 110 \times 72$	$105 \times 105 \times 65$	$100 \times 79 \times 21$
Price (USD)	$1999	$999	$499

Features that vary across Power Modes	Orin AGX	Xavier AGX	Orin Nano
CPU core counts	1..12	1..8	1..6
# CPU freqs.	29	29	20
Max. CPU Freq. (MHz)	2200	2265	1500
# GPU freqs.	13	14	5
Max. GPU Freq. (MHz)	1300	1377	625
# Mem freqs	4	9	3
Max. Mem. Freq. (MHz)	3200	2133	2133
# Power modes	**18,096**	**29,232**	**1,800**

Table 3: DNN workloads and Datasets used in Experiments. All models are trained with batch size of 16. Estimated epoch training time (mins) for MAXN fastest power mode on Orin AGX is given as reference.

Task	Model	# Layers	# Params	FLOPs†	Dataset	# Samples	Size	Epoch Time (min)
Image classification	**MobileNetv3 [25]**	20	5.5M	225.4M	**GLD23k [26]**	23k	2.8GB	2.3
Image classification	**ResNet-18 [27]**	18	11.7M	1.8G	**ImageNet Val. [28]**	50k	6.7GB	3
Object detection	**YOLO-v8n [29]**	53	3.2M	8.7G	**COCO minitrain [30]**	25k	3.9GB	4.9
Question answering	**BERT base [31]**	12	110M	11.5T	**SQuAD [32]**	70k	40.2MB	68.6
Next word prediction	**LSTM [33]**	2	8.6M	3.9G	**Wikitext [34]**	36k	17.8MB	0.4

† As per the typical practice, FLOPs reported correspond to a forward pass with minibatch size 1.

PyTorch is used for training the DNNs and its Dataloader module is used to fetch and pre-process the data samples. A minibatch of the training data forms the smallest unit of data fetch and training. The *num_workers* flag is used to set the number of fetch and pre-process workers. When *num_workers* = 0, a single process handles all data transfer and GPU compute operations, with no pipelining. When the *num_workers* flag is set to $n \geq 1$, n processes are initiated for fetching and pre-processing, with a separate process carrying out GPU computation on each pre-processed minibatch sequentially. By default, we use a minibatch size of 16 samples during training and *num_workers* = 4 [6]. We run an iteration of the DataLoader to pre-fetch all data to memory before starting the workload to avoid any disk fetch overheads during our data collection.

2.4. Profiling Setup and Metrics

During profiling of the DNN workload training, we sample the current power of the board in milliWatt (mW) every 1s using the *jtop* library, which is a wrapper around *tegrastats*. This gives the Jetson module's power as reported by the power sensors and not the overall power drawn by the carrier board. This scopes the power to the key resource components used in the DNN training, and power drawn by other peripherals like USB ports, etc. are avoided, which in any case have been reported to be negligible [4].

We add instrumentation to the PyTorch code to measure the execution time for every minibatch in milliseconds (ms). This is done using `torch.cuda.event` with the `synchronize` option to accurately capture time spent on the GPU. We experimentally verify that our profiling overhead is minimal and does not affect the runtime of the workload. Before we start the workload, we pre-cache the dataset into the page cache to avoid part of the data being fetched from memory and the rest from disk. This gives consistent minibatch training performance.

2.5. Data Collection from Profiling Power Modes

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 $\approx$ 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. There is negligible variation in the time or power measured across the minibatches during DNN training. However, we note that the very first minibatch's training takes a long time to train, possibly due to PyTorch performing an internal profiling to select the best possible kernel implementation in this phase. Therefore, we discard the first minibatch profiling entry. Additionally, we notice that the power measurements when training using a new power mode take 2–3s to stabilize. So, we use a sliding window logic to detect when

[6] The YOLO model version has a bug for *num_workers* = 4 setting, which is fixed in a later version of PyTorch but not available for our Jetson JetPack version (`https://github.com/pytorch/pytorch/issues/48709`). Hence, we use *num_workers* = 0. This is valid, but causes YOLO training to have GPU stalls as the main process is responsible both for the data loading and the compute.

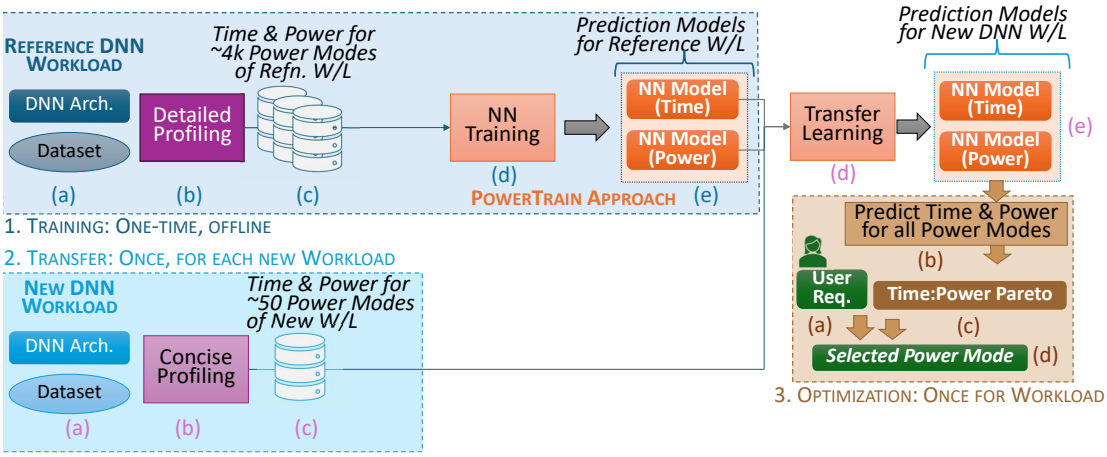


Figure 3: Time and Power Prediction Models: (1) Initial training, (2) Transfer, and (3) Use for Optimization

the power stabilizes, and use the profiling entries subsequent to that while ensuring 40 "clean" minibatch training samples are collected per power mode.

For the three default workloads, ResNet, MobileNet and YOLO, we collect and assemble a large corpus of ground truth power mode profiling data to help us with prediction model training and validation of their performance. The Orin power modes span 29 CPU frequencies, 13 GPU frequencies, 4 memory frequencies and 12 CPU cores for a total of 18,096 possible power modes [7]. From these, we choose 4,368 power modes for profiling that are uniformly distributed through the solution space; these include all combinations of GPU (13) and memory (4) frequencies, even number of CPU cores (6), and every alternate CPU frequency that is available, excluding the two slowest ones (14) [8]. The data collection time depends on the power mode chosen, e.g., a lower configuration will take longer to profile $\approx$ 40 minibatches of training, and we report these times as part of our results.

This large corpus of profiling data for different power modes for diverse DNN training workloads helps us rigorously evaluate our prediction models. This dataset collected is available at `Anonymized` and in itself forms a rich community asset.

3. ML-driven Modeling and Prediction

Here, we propose two data-driven approaches based on ML-based modeling to predict the power and training time per minibatch for a DNN training workload.

Our proposed methods were arrived at after exploring and eliminating other simpler prediction methods such as linear regression, random forest and decision trees, which did not perform well: DNN workload performance appeared inherently non-linear and linear regression was inaccurate; and random forest and decision trees suffered from overfitting. We also explored using a Multi-Layer Perceptron (MLP) to predict time and power. While this performed well with very few samples, our eventual solution based on Neural Networks (PT) outperforms it with a slightly larger number of samples but with the key added benefit of being able to transfer the learning to a new workload.

A key aspect that we consider here is the *overhead* for training the prediction models for a new DNN workload and, potentially, new hardware as well. This overhead is manifest primarily in the form of *profiling time* since the actual time to perform prediction model training itself is negligible, taking 15 minutes at most. NN models tend to be more accurate with more training samples, yet collecting profiling data for a growing number of power modes can take a linearly longer time. While the effort spent profiling can be reused as part of the actual training activity of the DNN workload (i.e., it is not wasted even from the user's perspective), the benefits of constructing the prediction model and performing optimizations on it are subdued if this data collection time starts being a large part of the multi-epoch DNN workload training time.

3.1. Neural Network (NN) Prediction Model per Workload

Neural Networks (NN) [40] consist of multiple layers of interconnected neurons, which allow it to learn complex, non-linear relationships in the data. Neural networks usually contain one input layer, one or more hidden layers, and an output layer. We first develop simple NN architectures to predict the training time and the power for a given power mode on an edge accelerator, for a specific DNN workload.

Our *NN architecture* (Figure 4) has 4 dense layers with 256, 128, 64 and 1 neurons, respectively. We use the ReLu activation function for the first 3 dense layers, and a linear activation function for the final layer. This was arrived at based on a grid search for hyper-parameters using *sklearn* library's GridSearchCV. The input feature vector consists of the configuration

[7] Interestingly, the number and actual frequencies available change with the Board Support Package (BSP) version. We also noticed that the same Orin device, but with a different JetPack version, had a different set of frequencies available. Nvidia's usually active forums do not have any comments related to this.

[8] During our experiments, we found that the Jetson device only supports changing from higher to lower CPU and GPU frequencies. Other changes require reboots, and cause an error otherwise. We came up with an ordering of the power modes that satisfies this requirement.

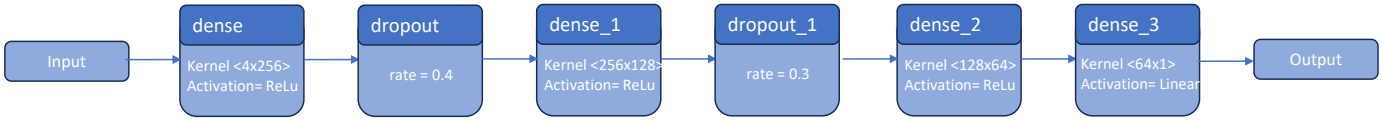


Figure 4: NN Architecture

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 4: NN hyperparameters

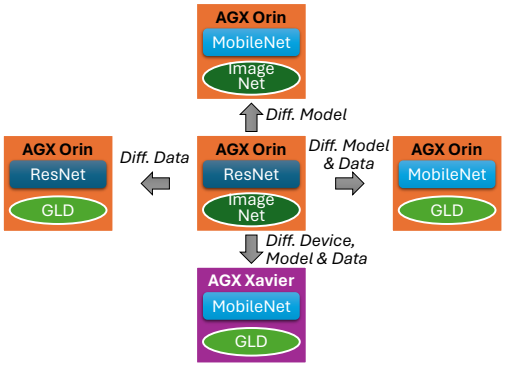


Figure 5: Generalizability of PowerTrain.

for a power mode: CPU cores, CPU frequency, GPU frequency and memory frequency, while the output layer returns the predicted training time per minibatch or power. Each input feature is normalized to a value between 0.0 to 1.0 using the *sklearn* library's StandardScaler to ensure better model generalization and faster convergence by providing consistent scales for all inputs. Adam is used as the optimizer, with a learning rate of 0.001 and the Mean Squared Error (MSE) serves as the loss function. We also introduce two dropout layers after the first and second dense layers to avoid overfitting. By default, we train the NN for 100 epochs and verify convergence. We use model checkpointing to save the best weights (i.e., model with the least validation loss) seen during training and use it as the final model. The hyperparameters are listed in Table 4.

We take two alternatives for providing the training data to the NN: (1) *All* gives it the full profiling data for a DNN workload using a 90:10 split of training and test, which is about 4.4k samples for our Orin AGX for the default DNN workloads. (2) The second provides a much smaller number of training data, from 10 to 100 uniformly sampled from the 4.4*k*, again using a 90:10 split. Clearly, the first approach is time-consuming to collect the profiling data, e.g., taking over 14*h* for ResNet, but can give better prediction accuracy. In contrast, a single epoch of training for ResNet using the MAXN faster power mode takes 3 mins to train, with a typical training lasting for 120 epochs, or 6 h. The latter sampling approach takes a much smaller time, running into 10s of minutes, but can suffer from lower accuracy. We explore these trade-offs in the evaluation.

3.2. *PowerTrain: Generalizable Transfer Learning Model*

While the NN approach is simple, it needs to be retrained for every new DNN workload we wish to run on the edge device. This requires fresh data collection overheads for different power modes of the new DNN workload. To overcome this limitation, we propose *PowerTrain*, which leverages Transfer Learning (TL) from one NN to another to offer a lower overhead generalizable approach.

Transfer learning [41, 42] is a popular technique in deep learning where a model developed for one task is applied to a different but related task. The core idea behind this is that a model trained on a large and generalized dataset can perform well on a related problem, but with significantly less specialized data and training time.

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. This is done once, offline, and serves as the reference NN to transfer from. 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.

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. We then transfer this to other DNN workloads, but using only a small number power modes for profiling, which serve as training data for transfer learning. We later discuss the choice of this reference DNN workload. We also use different counts of randomly sampled power modes, from 10 till 100, for transfer learning and evaluate the trade-off between data collection overheads and the accuracy of the retrained models for time and power prediction. The actual time for retraining itself remains in the order of < 10 mins.

PowerTrain can be generalized across DNN architectures and datasets (which together form a new workload), on the same device the reference model was trained on. We also attempt to transfer it to a DNN workload running on a different Jetson device, Xavier AGX. Specifically, we evaluate PowerTrain for three generalization scenarios (Figure 5): (1) Same DNN architecture or dataset as the reference DNN workload, (2) Unseen DNN architecture and dataset, ~~and~~ (3) Unseen device from a different generation, (4) Unseen device from the same generation, and (5) Unseen training batch sizes. Results from these are presented next.

4. Prediction Results

In this section, we present a detailed evaluation of PowerTrain (PT) for time and power predictions, and compare it with the Neural Network (NN) baseline using large and small sampling. These are evaluated on their Mean Absolute Percentage Error (MAPE%) relative to the ground truth time taken and power for a power mode based on actual measurement, and also on the overheads for data collection. We also evaluate the generalizability of PowerTrain to new DNN workloads and devices.

The PowerTrain reference model and NN large sampling (*All*) are both trained on 90% of the profiling data from all 4386 power modes, with 40 minibatches of telemetry entries available for each power mode. We also use *smaller samples* for NN training and for transfer learning in PowerTrain. Here, we randomly select between 10–100 power modes from 4386, with 40 minibatch entries each. Since power data is recorded every 1s, each power mode has a different number of power samples depending on the duration for profiling 40 minibatches. We use the maximum length of power samples as our entry count, and replicate power mode minibatch entries in case fewer are available to ensure the same number of training entries. By default, the *validation* of these models is done on all 4386 power modes. We perform 10 training and validation runs for every configuration and report the median results for these along with $Q1$–$Q3$ quartile range values as whiskers.

Training the full NN for the reference DNN takes around 15mins on a RTX 3090 GPU, but this can be done offline and the model weights saved. The training time for the NN model on 10–100 of samples takes a few seconds. Similarly, PowerTrain takes under 30s for transfer learning.

4.1. Choice of Base DNN Workload for PowerTrain

The choice of the *reference DNN workload* to train PowerTrain on can influence the quality of transfer learning. We have three candidate DNN workloads, ResNet, MobileNet and YOLO for which we have 4.4k training samples to allow reference model training. In Figure 6, we compare the validation MAPE when using each of these three as a reference and transferring to the other two, against the ground truth observed time and power values. We also report the validation MAPE for the reference model itself without any transfer, i.e., equivalent to NN model with all samples; this is likely to be the best possible prediction model. This is seen in the diagonal elements (green), which have MAPE between 8.1–9.7% for time predictions and 3.6–4.8% for power.

Among the off-diagonal elements using PowerTrain transfer learning, ResNet proves to be the best candidate as reference model. It results in time and power MAPE of 14.5% and 5.6% for MobileNet and 11.5% and 4.9% for YOLO (highlighted in yellow). In contrast, picking MobileNet as the reference results in time and power MAPE of 15.0% and 7.9% for ResNet and 11.8% and 4.9% for YOLO. YOLO performs even worse as a base model. Intuitively, we reason that ResNet has the highest variation in observed power values across the different power modes, and is able to generalize better to other DNNs which operate over a limited power range during training. Unless stated otherwise, in subsequent experiments, PT uses ResNet as the reference DNN.

4.2. Time and Power Prediction Accuracy vs. Profiling Overheads

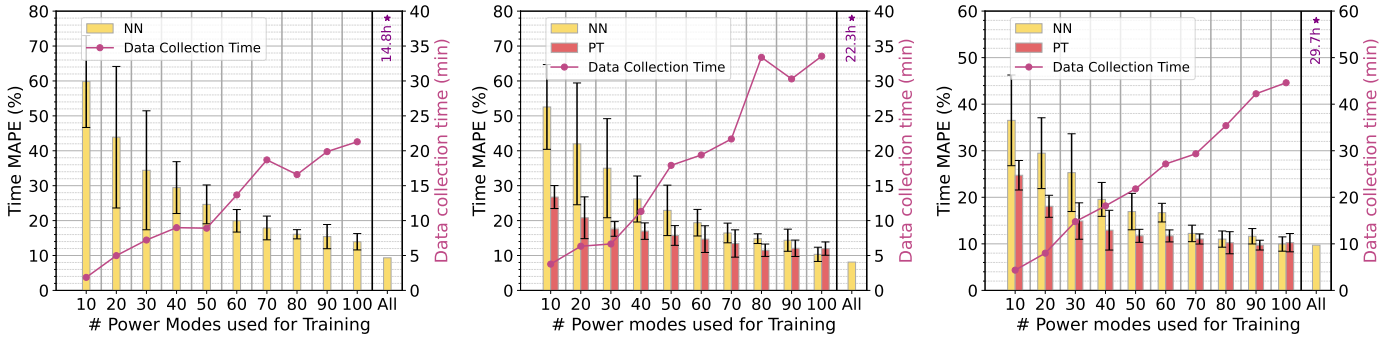
Here, we compare the performance of NN model trained using *All* profiling samples and with different smaller samples for the three default DNN workloads, against PowerTrain (PT) trained on ResNet as reference and transferred to the other two, with a different number of training samples used for transfer learning. Figures. 7 and 8 report the median MAPE for validation of the prediction models on all 4386 power modes across 10 training and validation runs. Here, the X axis indicates the size of the training data used to train the model (NN), or update the transferred model (PT) from the base ResNet NN mode. *All* indidates NN training on all 4.4k samples. The median MAPE is shown as bars on the left Y axis with whiskers indicating $Q1$–$Q3$ error ranges, while the profiling time taken for these many training is shown on the right Y axis (line plot) as overheads. In both cases, smaller is better. Since ResNet is the reference model, we do not report PT results when ResNet is the target DNN workload.

PowerTrain performs better than NN on time predictions for a new DNN with fewer number of power modes profiled.

In Figure 7, the MAPE for PT with 10 power modes as training input for MobileNet is 26.7% while the error for NN using 10 power modes is 52.6%. With just 30 power modes, PT

Transferred To	MobileNet		ResNet		YOLO	
From Base	Time Val %	Power Val %	Time Val %	Power Val %	Time Val %	Power Val %
MobileNet	8.12	3.62	15.03	7.98	11.77	4.98
ResNet	14.53	5.62	9.34	4.06	11.50	4.95
YOLO	17.03	9.71	19.76	12.88	9.72	4.81

Figure 6: Effectiveness of PowerTrain using different *reference DNN workloads* to transfer from. MAPE for time and power predictions validated against ground truth is reported.



(a) ResNet* (b) MobileNet (c) YOLO

Figure 7: Prediction error and profiling overheads for *time prediction* for different DNN workloads

improves to a MAPE of < 20% for MobileNet while NN has a much higher 35% error. Similarly, for YOLO, at 30 power modes, PT's MAPE is < 15%, while at 10 samples it is 24.7%. Additionally, the $Q1$–$Q3$ whisker is much tighter for PT than NN, showing less variability on the accuracy even with different random subsets of power modes picked for training. This shows the potential of PowerTrain to generalize from a model trained fully on one reference DNN to another with few samples. The data collection time (right Y axis, line) for 30 power modes is under 15 mins for all 3 DNNs, and under 25 mins for 50 power modes. A typical training to convergence runs for 100s of epochs and many hours. Hence, the data collection overhead is a smaller fraction.

PowerTrain for power predictions achieves a higher accuracy for a new workload with fewer profiling samples than NN.

In Figure 8b, 20 power mode samples for MobileNet allows PowerTrain to have a MAPE of just 8.5%, as compared to 12% for NN trained on a similar number of samples. Similarly, Figure 8c for YOLO shows PT achieve a power MAPE of 6.8% with 10 samples compared to 21% for NN. At larger number of samples of 50 or beyond, they have similar accuracies for YOLO though PT maintains a consistent 5% improvement over NN for MobileNet.

PT power prediction has lower MAPEs (5–10%) as compared to time predictions (10–20%) across all DNNs.

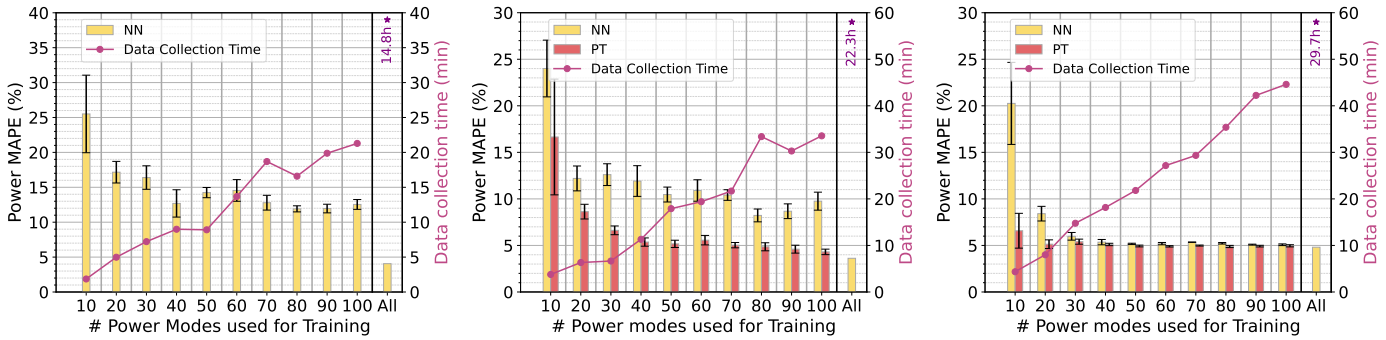
From Figures 8b and 7b, at 10 power mode samples for MobileNet, we observe a 2× lower MAPE for power as compared to time. This holds across DNN workloads, and for both PT and NN. At 50 power modes, which will be our default configuration for PT going forward, the power MAPEs for MobileNet and YOLO are 5.2% and 4.9% in contrast to time MAPEs of 15.7% and 11.7% for MobileNet and YOLO, respectively. This can be explained by the fact that we see little variation in power values for a given power mode, enabling the ML-based techniques to predict more accurately.

By 100 power modes, PowerTrain reaches close to the optimum accuracy for time and power prediction, and is comparable to NN training on all 4.4k power modes profiled.

In Figure 7b, at 100 power modes, PT has a MAPE of just 11.9% while NN trained on all samples has a MAPE of 8.1%. For power predictions on MobileNet, PT at 100 samples has a MAPE of 4.3% while the NN on all samples has a MAPE of 3.6%. For YOLO, the MAPE for time predictions is 10.18% for PT at 100 samples while the NN on all is a close 9.7%, and a similar trend is seen for power prediction as well. This again demonstrates the ability of PT to give higher accuracy with fewer samples.

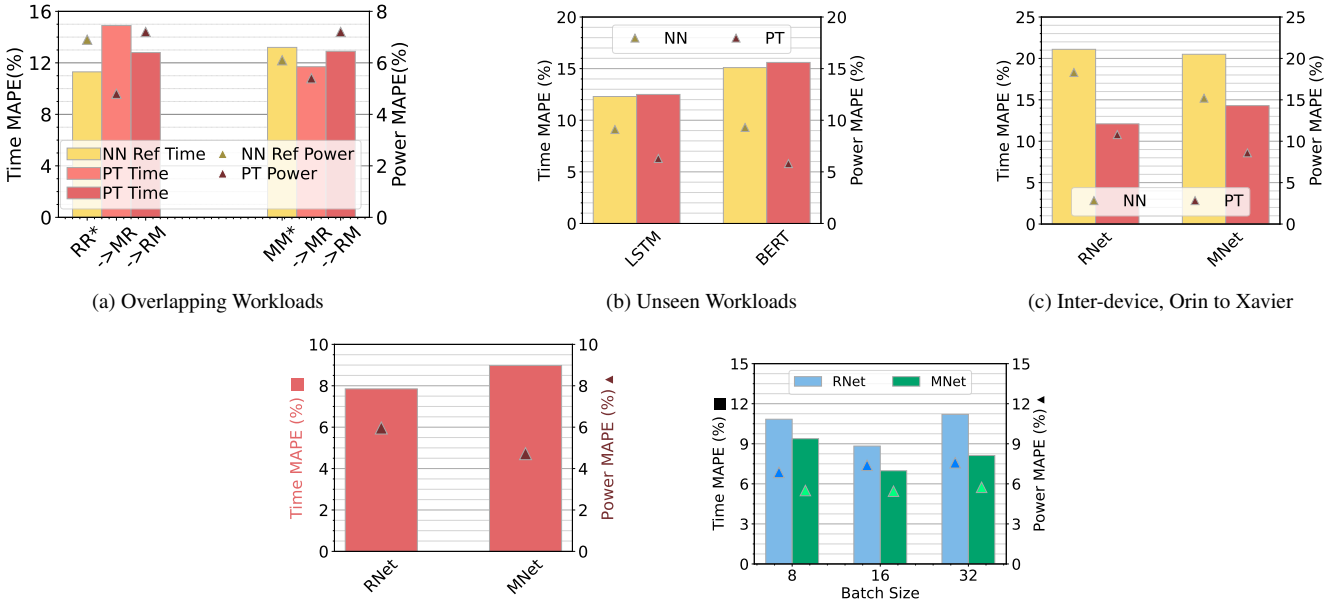
4.3. Generalization of PowerTrain

A vital benefit of PowerTrain is its ability to generalize to heterogeneous DNN workloads using a few profiling samples to achieve a higher accuracy. We next evaluate this generalizability along three dimensions (Figure 5), each being more diverse compared to the reference DNN workload trained: (1) *Either* the DNN architecture or the dataset being different from the reference DNN workload, (2) *Both* the DNN architecture and dataset being different, and (3) The *edge device* and the *DNN workloads* being different. The latter is particularly beneficial since it can encompass a wide range of DNN workloads and devices, e.g., federated learning over heterogeneous edge accelerators.



(a) ResNet* (b) MobileNet (c) YOLO

Figure 8: Prediction error and profiling overheads for *power prediction* for different DNN workloads



(a) Overlapping Workloads (b) Unseen Workloads (c) Inter-device, Orin to Xavier

(d) Inter-device, Orin to Nano (e) Varying Batch sizes

Figure 9: PowerTrain generalization when reference model is transferred to new workloads, devices and batch sizes.

4.3.1. Different DNN Architecture or Dataset from Reference

Here, PowerTrain uses two different reference NN models: one trained on ResNet with its native dataset ImageNet (Table 3), which we term as $RR*$, and another trained on MobileNet with its native GLD dataset ($MM*$). Now, PT tries to use each of these reference models and transfer to a new workload where either the DNN architecture or the dataset is different [9]. These combinations are listed as $RR* \rightarrow RM$, when transferring to ResNet architecture with MobileNet's native dataset GLD, $RR* \rightarrow MR$ when transferring to MobileNet architecture with ResNet's native dataset ImageNet, and similarly $MM* \rightarrow MR$ and $MM* \rightarrow RM$. So, either the architecture or dataset overlaps with the reference. We use 50 random power mode samples for transferring using PT. We report the time and power MAPE for these scenarios when validated against 100 samples of power modes, and also offer the MAPE for the reference DNN $RR*$ and $MM*$, which serves as the best case.

PowerTrain generalizes equally well if either DNN or the dataset changes.

We report the time MAPEs on the left Y axis as bars and the Power MAPEs on the right Y axis as markers. As seen in Figure 9a, having overlapping DNN architectures, $RR* \rightarrow RM$ and $MM* \rightarrow MR$, exhibit a MAPE for time that is not much worse than the reference models, $RR*$ and $MM*$, increasing in error from 11.3 to 12.8% for the former and in fact improving from 13.2 to 11.7% for the latter. For the power predictions, there is less than 1% change in MAPE. Similarly, when the datasets overlap but the DNN architecture changes, i.e., $RR* \rightarrow MR$ and $MM* \rightarrow RM$, we see the time MAPE increases from 11.3% to 14.9% and 13.2% to 12.9%, respectively.

[9] Both ImageNet and GLD are image datasets that are suitable for training classifiers, and can be used with both ResNet and MobileNet architectures.

Here too, the power MAPE does not deteriorate by more than 1%. So PowerTrain generalizes well when either the DNN architecture or the dataset overlaps with the reference.

4.3.2. Unseen and Diverse DNNs Workloads

The earlier results in Figures 7 and 8 showed PT transfer from ResNet reference to MobileNet and YOLO workloads that did not have any workload overlaps. Here, we expand that pool to include two workloads with very different DNN architectures compared to ResNet's CNN base: a BERT transformer architecture and an LSTM RNN-based architecture (Table 3), and text-based training datasets rather than images. BERT (Bidirectional Encoder Representations from Transformers) Base Uncased [31] is a type of transformer-based DNN designed for natural language processing. It captures contextual information and bidirectional dependencies in text. In our workload, BERT is used for Question Answering, using the SQuAD (Stanford Question Answering Dataset) V2.0 Train Set [32]. LSTM (Long Short-Term Memory) is a Recurrent Neural Network (RNN) that is designed for learning temporal patterns over lengthy sequences using memory cells and gating mechanisms. Our workload trains an LSTM model for next-word prediction using training data from WikiText [34].

PT uses 50 randomly profiled power mode samples from BERT and LSTM to transfer from ResNet reference, and we validate the transferred models on 50 other randomly sampled power modes to report the MAPE for time and power relative to the ground truth. This transfer and validation is repeated 20 times, and the results reported in Figure 9b in comparison with an NN model as baseline trained using the same 50 power mode samples.

PowerTrain generalizes well to diverse DNN workloads with few samples, and outperforms NN on power predictions.

As seen from Figure 9b, for LSTM, the time MAPE is 12.5% for PT and 12.3% for NN, which are comparable. Similarly, for BERT, the time MAPEs of 15.6% and 15.1% are close. The benefits of PT is visible for power predictions, where PT has a MAPE of 6.3% while NN is 9.1% for LSTM, and likewise, PT has a 3.5% better MAPE for BERT.

4.3.3. Unseen Device from a Different Generation

In this experiment, we use PowerTrain to transfer a reference workload trained on Orin AGX to workloads that run on the Xavier AGX developer kit [36], which is the comparable previous generation top-end Jetson device. As shown in Table 2, compared to the Orin, the Xavier edge accelerator has an 8 core (vs. 12 for Orin) custom Carmel ARM CPU, a previous generation CUDA architecture (Volta rather than Ampere), and with $\frac{1}{4}^{th}$ the number of CUDA cores (512 vs. 2048), and a previous generation RAM (LPDDR4 vs. LPDDR5). We randomly profile 1000 power modes out of the available 29,000 and collect power and time data for ResNet and MobileNet workloads in a similar manner as before. We use the ResNet workload trained on Orin AGX as our reference model for time and power predictions. Like before, PowerTrain then uses 50 random power modes for ResNet (or MobileNet) DNN workload profiled on Xavier to transfer-learn onto and validates the re-trained time and power prediction models using the remaining 950 power mode samples for that workload.

PT generalizes well to a device from a different generation and outperforms NN-based training.

Figure 9c, we see that when PT transfers from the reference DNN trained using ResNet workload on Orin to the same workload on Xavier (device changes), we see a time prediction MAPE of 12% and power prediction MAPE of 11%. This is much better than training an NN on just 50 power mode samples of the DNN workload on Xavier, which reports a MAPE of 21% for time and 18% on power. Similar benefits hold when both the device and the workloads are different for PT, where transferring the prediction models to MobileNet workload on Xavier has time MAPE of 14% and power MAPE of 9%, which are once again much better than NN.

4.3.4. Unseen device from the same generation

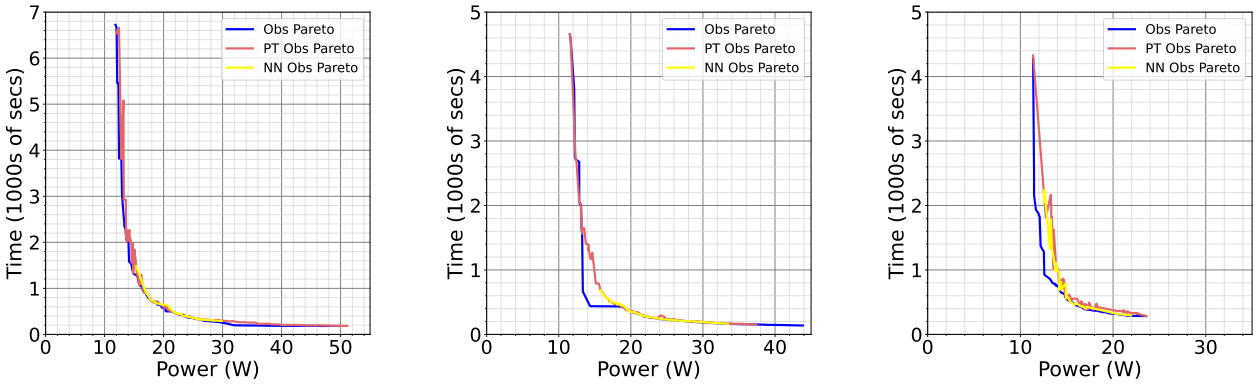
We also perform a limited set of experiments to examine the generalizability of PowerTrain to a less powerful accelerated edge device, *Jetson Orin Nano* [37], but from the same Jetson generation as the Orin AGX. We randomly sample 180 of the available 1800 power modes for Orin Nano and collect profiling 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.

This is a powerful result that suggests that there is sufficient similarity between the Jetson generations and DNN workload behaviors that can be learned during reference workload training, and adapted to a much different execution setting. This opens up opportunities to try even more diverse Jetsons like NX and older Nano series ~~, or even different edge models such as Raspberry Pi~~ . This is left to future work.

4.3.5. Unseen training batch sizes

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



(a) ResNet Ref. Workload on All (b) PT onto MobileNet (c) PT onto YOLO

Figure 10: Power–Time Pareto front over scatter plot of predicted training time and predicted power using all power modes, using Transfer Learned model from ResNet NN

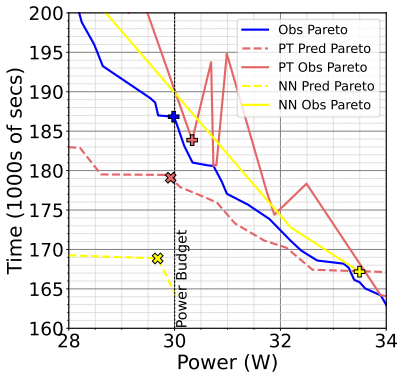


Figure 11: Zoomed in Power–Time Pareto front for MobileNet for a power target of 30W illustrating the performance of NN and PT wrt ground truth

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.

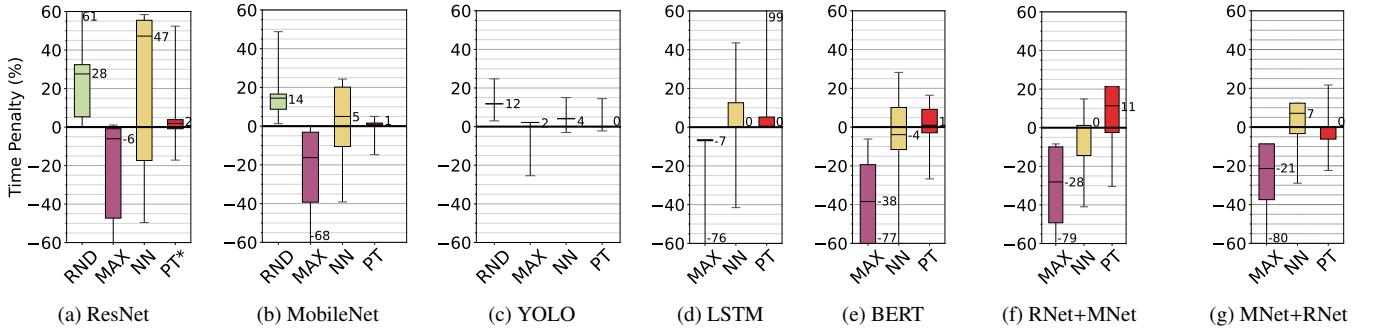
5. Optimizing DNN Workloads using our Prediction Models

One of the applications of our time and power prediction models is to optimize the power mode configuration of DNN workloads. We formulate an optimization problem as follows. Given a DNN workload m_{tr}, choose a power mode pm from a set $\mathbb{PM}$ that minimizes training time per epoch t_{tr} while ensuring that the power load P_{tr} falls within a user-specified budget P_b, i.e.,

$$\begin{aligned} &\textit{Given } m_{tr}, \\ &\text{select } pm \in \mathbb{PM} \\ &\quad \min \ t_{tr} \\ &\text{s.t.} \quad P_{tr} \leq P_b \end{aligned}$$

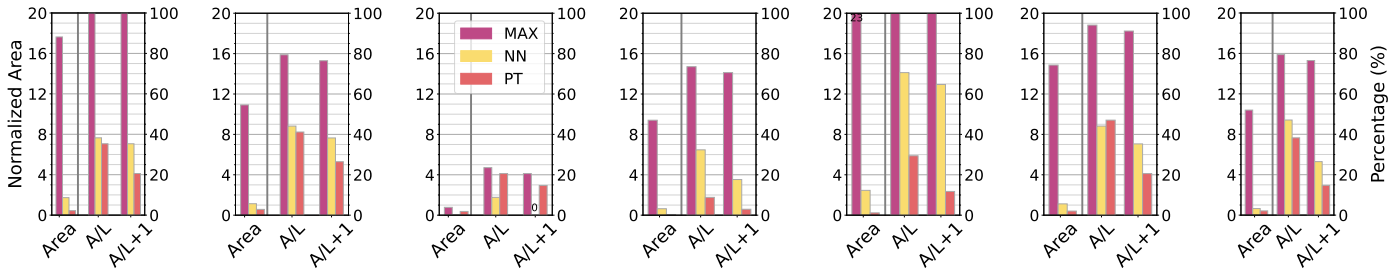
As illustrated in the workflow in Figure 3, we use our prediction models to solve the optimization problem. For PowerTrain, we use the reference model trained using the ResNet workload on 4368 power modes, and transfer to other target workloads using 50 random power mode profiling samples. Since the time predictions are per minibatch, we scale them to per epoch times. As a ground truth optimization solution, we draw the observed Pareto front (*Obs Pareto* in Figure 10) using the profiling information for 4.4k power modes to minimize time and power. These set of points offer the least time value for a given power, and vice versa. For various power limits specified by the user, we can trivially search Pareto points to find the *optimal power mode* with a power value that is closest to but less the power budget and report the training time for this.

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. 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)).. We also report the matching ground-truth values for these predicted power modes on the Pareto, shown as *PT Obs Pareto* in Figures 10 and 11. The NN Prediction model using 50 samples serves as a baseline to compare against, and is shown as *NN Obs Pareto* and *NN Pred Pareto*.



(a) ResNet (b) MobileNet (c) YOLO (d) LSTM (e) BERT (f) RNet+MNet (g) MNet+RNet

Figure 12: *Time Penalty %* for model-based optimization relative to ideal Pareto time. Lower is better. *PT for ResNet indicates training of base model on full data.



(a) ResNet (b) MobileNet (c) YOLO (d) LSTM (e) BERT (f) RNet+MNet (g) MNet+RNet

Figure 13: Pareto Power Errors for different DNN workloads

5.1. Baselines

In addition, we also offer a few baseline approaches for solving the optimization problem to contrast against PowerTrain. **MAXN** is the default power mode setting on the Jetson Orin AGX, and offers the best time performance (albeit with a high power load). **Random Sampling Pareto (RND)** is a simple sampling-based baseline where we randomly profile 50 power modes and build a Pareto front from these, which is then used for the optimization. We continue to use the **NN model** custom trained for each DNN workload using 50 random power modes, and use this as a baseline to predict the Pareto front over all power modes and perform optimization.

5.2. Results

We solve the optimization problem for a parameter sweep of power limits, from 17W to 50W in increments of 1W, to get a set of optimal solutions for the set of problems, each having a different power limit. We compare the results of the optimization solutions using our approaches and the baselines against the optimal solution given by the ground-truth Pareto. We report several metrics: the *time penalty%*, which is the excess training time spent for the power mode selected by a strategy compared to the optimal power mode; the *normalized area under the curve (AUC)* of power in excess of the given budget for the given solution by a strategy (W per solution); the *% of solutions that exceeds the power limit* for a strategy (A/L); and the *% of solutions that exceed the limit by more than a 1-Watt threshold* (A/L+1). There are no power errors for the observation-based (rather than prediction based) random sampling.

PT observed Pareto is close to the ground truth Pareto while it deviates for NN, leading to better optimization results for PT.

In Figure 10, we draw the *Observed Pareto* using our full profiling information (blue line). Further, we come up with the prediction Paretos for NN and PT and report the matching ground-truth values for these predicted power modes on the Pareto, shown as *NN Obs Pareto* and *PT Obs Pareto*, respectively. As can be seen, PT closely follows the Observed Pareto while NN is limited to a small region.

Further, in Figure 11, we present a specific instance of the optimization problem for MobileNet at a power budget of 30 W by zooming into the Pareto plot. In this figure, we report the predicted Paretos along with their observed counterparts. The ground truth optimal solution takes 186 s per epoch and consumes 29.9 W. The NN optimization chooses a power mode predicted to take 168.9 s and consume 29.7 W. However, in reality, this power mode takes a comparable 167.2 s but consumes 33.5 W, which overshoots the power budget. The PT optimization chooses a power mode predicted to take 179.1 s and consume 29.9 W. In reality, this power mode takes 183.9 s and consumes 30.3 W (marginal overshoot by 0.3 W), which is very close to the optimal.

PT optimization results in a lower time penalty as compared to NN in most cases.

As seen from Figure 12, for MobileNet, the median time penalty for PT is just 0.7% over the optimal as compared to 5% for NN. Similarly, for YOLO, PT is as good as optimal with 0% penalty while NN has a median time penalty of 4%. For LSTM and BERT, PT has the same median time penalty as NN, however the distribution is tighter for PT. For MobileNet using

ResNet training dataset (Figure 12f), PT has a lower median time penalty. PT does worse than NN only for ResNet with MobileNet data (Figure 12g).

PT optimization has the lowest normalized excess power AUC for most models and exceeds the power budget within a 1W threshold under 25% of the time.

In Figure 13, we report three metrics: Area, A/L and A/L + 1. Area stands for the *normalized area under the curve (AUC)* of power in excess of the given budget for the given solution by a strategy (W per solution); A/L stands for the *% of solutions that exceeds the power limit* for a strategy; and A/L + 1 stands for the *% of solutions that exceed the limit by more than a 1-Watt threshold*.
As seen from Figure 13, the *Area* is the lowest for PT across all DNN workloads except YOLO (6 out of 7 cases). Additionally, this budget + 1W buffer (A/L + 1) is breached in $< 20\%$ of the solutions for 6 out of 7 DNNs and is 25% for MobileNet. So PT's predictions and solutions help stay within the power budget most of the time.

MAXN almost always exceeds the power budget while offering the best time, while Random sampling has higher time penalties with no power violations

As seen from Figure 12, MAXN has negative time penalties for ResNet, MobileNet and YOLO, i.e., is much faster than the optimal solution. But this is because of setting all frequencies to the maximum value, thus violating the power limit constraints often (Figure 13). In contrast, a Pareto from Random sampling is 12–28% slower than the optimal. So these simpler baselines are sub-optimal and potentially unusable.

6. Related Work

6.1. Energy, Power and Performance Modeling on Edge devices

Earlier works [3] use roofline modeling on a much older Jetson TK1 and TX1 to understand and characterize the performance of CPU and GPU micro-benchmarks for matrix multiplication. MIRAGE [43] uses gradient tree boosting to predict runtime and power consumption for DNN inference on a Jetson Xavier AGX. Others [21, 44] have investigated the impact of frequencies and cores with the latency, power and energy for inferencing on the Jetson Nano. Abdelhafez and Ripeanu [22] examine the effect of frequencies on power consumption for stream processing workloads. Some [45] have developed methods to measure fine-grained layer-wise energy for inference workloads on the Jetson TX1. All of these have either looked at inferencing or micro-benchmarks, and not DNN training.

Our previous work characterized training of DNNs on Jetson AGX, NX Xavier and Nano [4]. There, we offered initial insights on the opportunities for deep learning training on Jetson and some of the performance and power behavior. We also proposed a simple time and energy prediction model based on linear regression. But it is only evaluated on $\approx$ 10 power modes, and its errors are much worse than what we observe in this work. In contrast, our goal is to model and predict the power and time for DNN training.

6.2. Energy, Power and Performance Modeling on GPU servers

NeuralPower [20] 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 [46] 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 [4] shows that linear regression on these parameters is inadequate to make high quality predictions. Paleo [19] 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 [47] build a power, performance and energy model for application kernels on desktop or server-grade GPUs. Recent work [48] 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.

6.3. Optimization studies on the edge and server

Zeus [18] studies the trade-off between training time and energy, and uses this to minimize the cost of a training job by selecting the best batch size and GPU power limit. Others [49] select optimal GPU frequency that lowers power while minimizing the performance impact on server-grade GPUs. D3 [50] looks at optimizing the runtime accuracy trade-offs for DNN inference workloads in autonomous vehicle pipelines that have dynamically varying deadlines. ALERT [51] selects a DNN and a power limit to meet accuracy, latency and energy constraints for inference tasks on laptop and desktop CPUs and GPUs. AxoNN [52] distributes layers of a DNN inference workload between a performance-efficient GPU and a power-efficient DLA on the Jetson Xavier AGX to minimize time and stay within an energy budget. Others [53] find the Pareto optimal scheduling of multiple deep learning applications in terms of the response time and energy consumption on smartphones. Mephesto [54] models memory contention for kernel placement on heterogeneous SoCs to achieve the desired energy performance tradeoff. Oppertune [55] 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. We focus on modeling the training time and power for a DNN workload on an *accelerated edge device* and solving an optimization problem using these models, which is a gap in the existing literature.

7. Discussions and Conclusion

In this work, we design two modeling techniques to predict the training time and power for DNN training workloads on Jetson edge devices. PowerTrain uses hours of offline data collection and rigorous training for a reference DNN workload to bootstrap the prediction model for a whole new DNN workload within a few minutes of profiling for $\approx$ 50 power modes and retraining. PowerTrain's power and time predictions generalize well across overlapping DNNs and datasets, unseen DNN workloads and even new devices. We leverage PowerTrain to come up with a predicted Pareto to trade-off power limits against training time for a given DNN workload training. This helps identify the optimal power mode to stay within a given power limit while minimizing the training time for the DNN workload. Our results confirm the robustness of PowerTrain to be reused in diverse workload and device conditions with low overheads, and its out-performance on predictions and in solving the optimization project compared to simple and NN baselines. This makes it a valuable methodology for efficient configuration of dynamic DNN training workloads in edge deployments.

In future, PowerTrain can be used in an online fashion for practical deployment, allowing us to collect sampling data for ML training while the DNN workload runs productively. We also plan to explore reinforcement learning based methods to solve this problem. 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. This work can also be investigated for use in ~~other edge devices like Rapberry Pi and~~ GPU servers, and also non-DNN workloads.

Acknowledgments

The authors would like to thank students in the DREAM:Lab including Pranjal Naman and Kedar Dhule for their assistance with the article. The first author was supported by a Prime Minister's Research Fellowship (PMRF) from the Ministry of Education, India.

References

[1] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, W. Shi, Edge computing for autonomous driving: Opportunities and challenges, Proceedings of the IEEE 107 (8) (2019).

[2] L. U. Khan, I. Yaqoob, N. H. Tran, S. M. A. Kazmi, T. N. Dang, C. S. Hong, Edge-computing-enabled smart cities: A comprehensive survey, IEEE Internet of Things Journal 7 (2020).

[3] H. Halawa, H. A. Abdelhafez, A. Boktor, M. Ripeanu, Nvidia jetson platform characterization, in: Euro-Par 2017: Parallel Processing, 2017.

[4] Prashanthi S.K, S. A. Kesanapalli, Y. Simmhan, Characterizing the performance of accelerated jetson edge devices for training deep learning models, Proc. ACM Meas. Anal. Comput. Syst. 6 (3) (December 2022).

[5] T. P. Carvalho, F. A. A. M. N. Soares, R. Vita, R. da P. Francisco, J. P. Basto, S. G. S. Alcalá, A systematic literature review of machine learning methods applied to predictive maintenance, Computers & Industrial Engineering 137 (2019).

[6] S. Yang, M. Lupascu, K. S. Meel, Predicting forest fire using remote sensing data and machine learning, Proceedings of the AAAI Conference on Artificial Intelligence 35 (17) (2021).

[7] S. Chun, N. Hamidi Ghalehjegh, J. Choi, C. Schwarz, J. Gaspar, D. McGehee, S. Baek, Nads-net: A nimble architecture for driver and seat belt detection via convolutional neural networks, in: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops, 2019.

[8] G. Silvestre, S. Monnet, R. Krishnaswamy, P. Sens, Caju: A content distribution system for edge networks, in: Euro-Par Workshops, 2012.

[9] C. W. Zaw, S. R. Pandey, K. Kim, C. S. Hong, Energy-aware resource management for federated learning in multi-access edge computing systems, IEEE Access 9 (2021).

[10] Y. Kim, S. Park, S. Shahkarami, R. Sankaran, N. Ferrier, P. Beckman, Goal-driven scheduling model in edge computing for smart city applications, Journal of Parallel and Distributed Computing 167 (2022).

[11] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y. Arcas, Communication-Efficient Learning of Deep Networks from Decentralized Data, in: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, 2017.

[12] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage, J. Roselander, Towards federated learning at scale: System design, in: Proceedings of Machine Learning and Systems, 2019, pp. 374–388.

[13] K. Shmelkov, C. Schmid, K. Alahari, Incremental learning of object detectors without catastrophic forgetting, in: Proceedings of the IEEE international conference on computer vision, 2017, pp. 3400–3409.

[14] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, Y. Shu, N. Karianakis, K. Hsieh, P. Bahl, I. Stoica, Ekya: Continuous learning of video analytics models on edge compute servers, in: 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), 2022.

[15] M. Zhang, C. Krintz, R. Wolski, Sparta: Heat-budget-based scheduling framework on iot edge systems, in: Edge Computing – EDGE 2021, 2022.

[16] A. M. Abdelmoniem, A. N. Sahu, M. Canini, S. A. Fahmy, Refl: Resource-efficient federated learning, in: Proceedings of the Eighteenth European Conference on Computer Systems, 2023.

[17] Z. Chai, A. Ali, S. Zawad, S. Truex, A. Anwar, N. Baracaldo, Y. Zhou, H. Ludwig, F. Yan, Y. Cheng, Tifl: A tier-based federated learning system, in: Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing, 2020.

[18] J. You, J.-W. Chung, 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.

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

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

[21] S. Holly, A. Wendt, M. Lechner, Profiling energy consumption of deep neural networks on nvidia jetson nano, in: 2020 11th International Green and Sustainable Computing Workshops (IGSC), 2020.

[22] H. A. Abdelhafez, M. Ripeanu, Studying the impact of CPU and memory controller frequencies on power consumption of the Jetson TX1, in: IEEE Intl. Conf. on Fog and Mobile Edge Comp. (FMEC), 2019.

[23] Y. G. Kim, C.-J. Wu, Autoscale: Energy efficiency optimization for stochastic edge inference using reinforcement learning, in: 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2020, pp. 1082–1096. doi:10.1109/MICRO50266.2020.00090.

[24] 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, Y. Zhou, Mlperf inference benchmark, in: Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture, 2020.

[25] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, H. Adam, Searching for mobilenetv3, in: IEEE/CVF International Conference on Computer Vision (ICCV), IEEE, 2019.

[26] TensorFlow, Tff gldv2, https://www.tensorflow.org/federated/api_docs/python/tff/simulation/datasets/gldv2/load_data (2022).

[27] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database, in: CVPR09, 2009.

[29] G. Jocher, A. Chaurasia, J. Qiu, Ultralytics yolov8 (2023).
URL https://github.com/ultralytics/ultralytics

[30] N. Samet, S. Hicsonmez, E. Akbas, Houghnet: Integrating near and long-range evidence for bottom-up object detection, in: European Conference on Computer Vision (ECCV), 2020.

[31] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in: Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, 2019.

[32] P. Rajpurkar, R. Jia, P. Liang, Know what you don't know: Unanswerable questions for squad, arXiv preprint arXiv:1806.03822 (2018).

[33] E. Wisam, Language modeling with LSTMs in pytorch, https://towardsdatascience.com/language-modeling-with-lstms-in-pytorch-381a26badcbf/ (2022).

[34] Hugging Face, Datasets : wikitext, https://huggingface.co/datasets/wikitext (2021).

[35] NVIDIA., Jetson agx orin developer kit, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/ (2022).

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

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

[38] D. Jallepalli, N. C. Ravikumar, P. V. Badarinath, S. Uchil, M. A. Suresh, Federated learning for object detection in autonomous vehicles, in: 2021 IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService), 2021.

[39] Y. Tian, Y. Wan, L. Lyu, D. Yao, H. Jin, L. Sun, Fedbert: When federated learning meets pre-training, ACM Trans. Intell. Syst. Technol. 13 (4) (2022).

[40] W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, The bulletin of mathematical biophysics 5 (1943) 115–133.

[41] S. Bozinovski, A. Fulgosi, The influence of pattern similarity and transfer learning upon training of a base perceptron b2, in: Proceedings of Symposium Informatica, Vol. 3, 1976, pp. 121–126.

[42] S. Bozinovski, Reminder of the first paper on transfer learning in neural networks, 1976, Informatica 44 (3) (2020).

[43] H. A. Abdelhafez, H. Halawa, M. O. Ahmed, K. Pattabiraman, M. Ripeanu, Mirage: Machine learning-based modeling of identical replicas of the jetson agx embedded platform, in: 2021 IEEE/ACM Symposium on Edge Computing (SEC), 2021.

[44] A. Dutt, S. P. Rachuri, A. Lobo, N. Shaik, A. Gandhi, Z. Liu, Evaluating the energy impact of device parameters for dnn inference on edge, in: International Green and Sustainable Computing, 2023.

[45] C. F. Rodrigues, G. Riley, M. Luján, Fine-grained energy profiling for deep convolutional neural networks on the jetson tx1, in: 2017 IEEE International Symposium on Workload Characterization (IISWC), 2017.

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

[47] S. Song, C. Su, B. Rountree, 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.

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

[49] G. Ali, M. Side, S. Bhalachandra, N. J. Wright, Y. Chen, Performance-aware energy-efficient gpu frequency selection using dnn-based models, in: Proceedings of the 52nd International Conference on Parallel Processing, 2023.

[50] I. Gog, S. Kalra, P. Schafhalter, J. E. Gonzalez, I. Stoica, D3: A dynamic deadline-driven approach for building autonomous vehicles, in: Proceedings of the Seventeenth European Conference on Computer Systems, 2022.

[51] C. Wan, M. Santriaji, E. Rogers, H. Hoffmann, M. Maire, S. Lu, ALERT: Accurate learning for energy and timeliness, in: 2020 USENIX Annual Technical Conference (USENIX ATC 20), 2020.

[52] I. Dagli, A. Cieslewicz, J. McClurg, M. E. Belviranli, Axonn: Energy-aware execution of neural network inference on multi-accelerator heterogeneous socs, in: Proceedings of the 59th ACM/IEEE Design Automation Conference, 2022.

[53] D. Kang, J. Oh, J. Choi, Y. Yi, S. Ha, Scheduling of deep learning applications onto heterogeneous processors in an embedded device, IEEE Access 8 (2020) 43980–43991.

[54] M. A. H. Monil, M. E. Belviranli, S. Lee, J. S. Vetter, A. D. Malony, Mephesto: Modeling energy-performance in heterogeneous socs and their trade-offs, in: Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques, 2020, pp. 413–425.

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

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

[57] NVIDIA., Rtx 3090, https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/ (2020).

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

Table A.5: Specifications of Server GPUs and Edge Devices

Feature	3090	A5000	Orin AGX	Raspberry Pi5
CPU	12th Gen Intel i9-12900K	AMD EPYC 7532	ARM Cortex A78AE	ARM Cortex-A76
# CPU Cores	16	32	12	4
Max CPU frequency (MHz)	5200	3400	2200	2400
GPU (arch/CUDA cores)	Ampere, 10496	Ampere, 8192	Ampere, 2048	VideoCore VII (graphics only)
Max GPU frequency (MHz)	1695	2505	1300	800
RAM (GB)	128 (CPU) + 24 (GPU)	512 (CPU) + 24 (GPU)	64 (shared)	8
Peak Power (W)	350	230	60	27

Appendix A.

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 [10]: (1) A GPU server with NVIDIA RTX A5000 [56], (2) A GPU workstation with NVIDIA RTX 3090 [57], and (3) The latest generation (non-accelerated) Raspberry Pi5 edge device [58] and report the average epoch times in Figure A.14. The specifications of the devices are in Table A.5. 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 A.14) as it ran out of memory in 8GB of memory available in the RPi.

[10] 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.

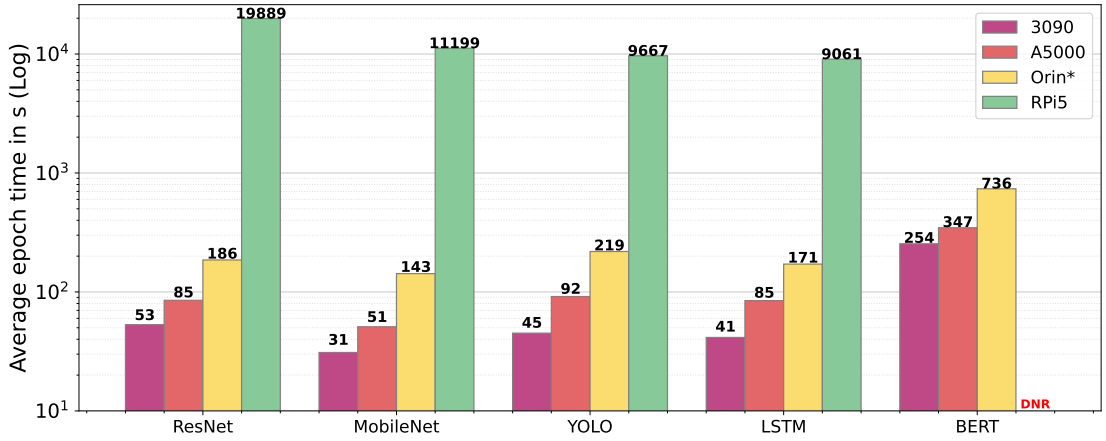


Figure A.14: Average epoch times across various edge and server devices. The device specifications are: **3090** *(12th Gen Intel i9 CPU with 16 cores@ 5.2GHz, 128GB RAM, 3090 Ampere GPU with 10496 cores, 24GB GPU RAM)*; **A5000** *(AMD EPYC 753 CPU with 32 core@ 3.4GHz, 512GB RAM, A5000 Ampere GPU with 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)*