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

ANONYMOUS AUTHOR(S)

Accelerated edge devices, like Nvidia's Jetson with 1000+ CUDA cores, are increasingly used for training DNN workloads and federated learning, rather than just 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 power–performance trade-offs for dynamically arriving training workloads 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 consumed to train a given DNN workload (model+dataset) using any specified power mode. It requires a one-time offline profiling of 1000s of power modes for a reference DNN workload on a Jetson Orin AGX to build Neural Network (NN) based prediction models for time and power. These NN models are subsequently retrained (transferred) for a new DNN workload, or even a different Jetson Xavier AGX 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 make optimal time–power trade-offs for a new workload, e.g., to select a power mode that minimizes training time while meeting a power threshold set by the user. Our predictions exhibit 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, and errors of $< 14.5\%$ and $< 11\%$ even when transferred to an entirely new Jetson device, Xavier AGX. These outperform baseline predictions by more than 10% and baseline optimizations by up to 45% on time and 88% on power. Further, the Pareto front that is identified overlaps with a balanced-ratio between the CPU and GPU frequencies for the device, which we analytically determine, and indicates that certain ideal power modes fall in the sweet-spot of training efficiency.

1 INTRODUCTION

With the growth in domains like autonomous vehicles [34] and IoT for smart cities [30], edge computing devices have emerged 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 approach GPU workstations [22] with 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 [41].

Motivation. Training on edge devices can be done to opportunistically make use of captive compute, e.g., on edge devices and sensors deployed in factory to detect faults [10] or forest fires from field cameras [56]. Such training can also happen on pay-as-you-go shared edge infrastructure that are colocated with content distribution networks (CDNs) and 5G towers [49, 58]. This can avoid the high bandwidth needed to move data to public clouds for training, e.g., from 100s of traffic cameras [12, 31]. 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. Federated learning [6, 36] leverages fleets of heterogeneous edge devices to train models locally over local data for subsequent aggreation on the cloud across multiple epochs and rounds.

In summary, training can happen *recurrently* on edge devices as part of continuous learning to adapt to data drift [5, 48], involve *diverse DNN workloads* trained on captive or shared edge devices, and take long to train (e.g., it takes 27.7 hours for YOLOV8n model to convergence on

COCO-minitrain [47] on a Jetson Orin AGX). Edge training can also operate under environmental or user constraints. There may be *energy constraints* imposed by power bank on drones or solar-charged batteries in a forest. There can be *power limits* to avoid overheating of IP-67 enclosures in an industrial site [59]. 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 [3, 11]. So, it is important to *configure* the edge accelerator to adapt to these constraints and offer *predictable power, energy and time estimates* for DNN training.

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 18k possible power modes (Table 1). These can help limit their power envelope in a constrained setting while throttling the performance. But given this vast parameter space, selecting a power mode for such trade-offs with dynamic training workloads is non-trivial. Some frequencies span an order of magnitude, and can have 36× impact on DNN training time and 4.3× on power usage, e.g., 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 models that are being trained, 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. As expected, these also vary substantially across Jetson generations. Hence, a wrong power mode choice can cause high power and performance penalties, violating constraints. So, accurately estimating the power and time taken for training a given DNN model 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, and this will not scale for every new workload (online) or every possible workload (offline). Random or even targeted sampling of some power modes is inaccurate for optimizations. Practical deployments may have multiple generations of accelerators, 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 the edge.

Gaps. There is substantial work on optimizing the energy consumption of DNN training on GPU servers [57] and predicting their runtime [42], and power and energy consumption [9]. 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 on energy and performance profiling for edge inferencing rather than training [2, 24, 32]. Some have also explored power modes and frequency scaling for micro-benchmarks on older Jetson architectures [22]. However, inferencing workloads have low compute demand compared to training, which can stress all resources of the edge device. A recent work characterized the performance of the previous generation Jetson Xavier AGX and Nano devices for DNN training, but offered limited insights on performance prediction or tuning their power modes [41]. 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, Nvidia's own tool highly overestimates the power load for

[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.

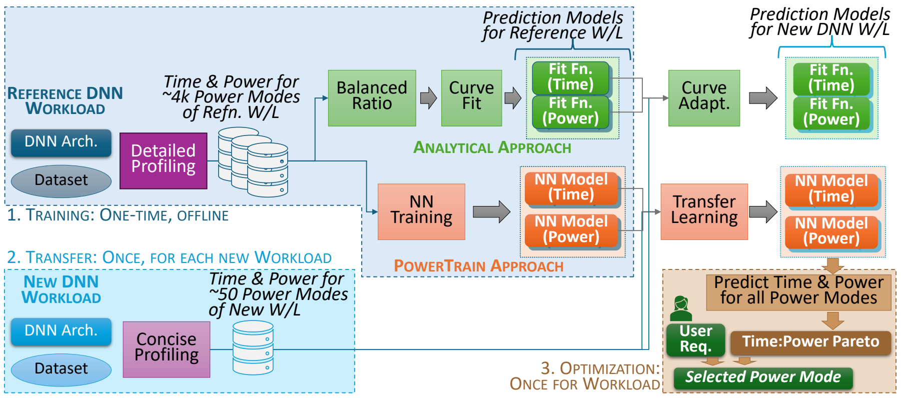


Fig. 1. Time and Power Prediction Models: (1) Initial training, (2) Transfer, and (3) Use for Optimization

Orins, causing excess training time when meeting a power budget. In summary, there is a clear need for modeling and optimization studies for DNN training on accelerated edges, but none exist. *We address this gap, and offer the first principled modeling approach of its kind.*

Proposed Approach. We take two alternative methodologies to solving this problem, summarized in Fig. 1 and described in § 3 and § 4. First, we analytically study the effect of the power modes on the training time and power load by profiling a large subset of the power modes for a reference DNN workload. We identify *balancing ratios between the CPU and GPU frequencies* such that any deviation from this ratio increases the power without any reduction in training time or *vice versa*. This ratio is used to *fit a reference curve* for the time and power prediction for a workload, and adapted to new DNN workloads with minimal additional profiling. The intuition is that the Pareto front [2] between time and power is likely to lie on the power modes with the balanced ratio.

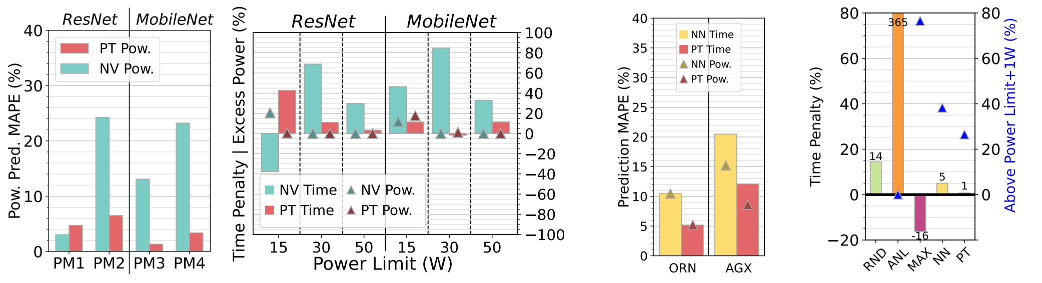
A second approach is data-driven, and we refer to this as **PowerTrain**. Here, we train two NN models – for time and power predictions – over the large subset of power modes measured for some reference DNN workload. For a new DNN workload, we use transfer learning over time and power telemetry collected from profiling a small sample of ≈ 50 power modes for the workload to get updated NN prediction models.

Both the analytical and 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 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 lower power for a given time budget. In both these approaches, there is a one-time profiling overhead on the reference workload and fitting/training the initial curves/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.

Representative Results. While a detailed evaluation is presented in § 5 and § 6, we offer some results here to illustrate the comparative benefits of our approach.

Predictions. Nvidia offers a *PowerEstimator* tool (NPE) [3] 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

[2]The Pareto front is defined as a subset $\widehat{\mathbb{C}}$ from a set of choices $\mathbb{C} = \{c_1, c_2, \ldots\}$ 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 $y_j < y_i$.
[3]https://jetson-tools.nvidia.com/powerestimator/



(a) Prediction vs. Nvidia (b) Optimization vs. Nvidia (c) Prediction vs. NN (d) Optimz. vs. Baselines

Fig. 2. Representative results: Prediction and optimization. In (a) the power modes 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

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 Fig. 2a. Other than for PM1 for ResNet, where PT is has a slightly larger error (5%) than NPE (4%), we give much better predictions in all other cases while NPE consistently over-estimates.

We also compare the prediction time and power using PowerTrain's reference ResNet model on Orin AGX transferred to the MobileNet workload on Xavier AGX. We see that the MAPE% for time and power predictions are just $\approx$ 5% when transferring to a different workload (ResNet to MobileNet) on same the device type and 7–12% for a different workload on an entirely different device, compared to the ground-truth. The data collection time for 50 power modes used by PT for transfer learning is around 20min, which is much smaller than the training time for 100s of epochs that runs into hours. Further, the data collection is also a valid DNN mini-batch training that contributes to the workload progress. This highlights the generalizability of the PowerTrain approach, and its high accuracy with low overheads.

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 § 6, we find a 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 [4] 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 prediction models to discover a Pareto front (described in § 6) 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 2b shows that PowerTrain (PT) has the fewest % of solutions that exceed the optimal 5 out of 6 cases (except ResNet 15W) compared to Nvidia's suggestions, while falling within the power limits in most cases.

In Fig. 2d, PT based optimization also outperforms weaker baselines like choosing the default MAXN power mode, doing random (RND0 profiling of 50 power modes to build a partial Pareto and select the best, and a Neural Network (NN) based one that is trained on 50 power profile samples. For a set of problems with power limits varying from 17 to 50W, Fig. 2d shows 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.

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

These exemplars motivate the need for a robust prediction model to support the application tuning of power models for custom DNN training on the Jetson edge devices.

Contributions. We make the following specific contributions in this paper.

(1) We develop a Neural Network (NN) based prediction model to *estimate the time and power* for DNN training workloads on the Nvidia Jetson Orin AGX for any given power mode (§ 4) using a large profiling corpus. This is extended into PowerTrain, a *Transfer Learning (PT)* that greatly reduces the online profiling overheads for any new DNN workload on Orin or another Jetson device. These are comprehensively validated for accuracy and genralizability (§ 5).
(2) We complement this data-driven approach with an analytical study of the Orin's power modes to identify a *balancing ratio between the CPU and GPU frequencies*, which indicate a balanced configuration and are Pareto adjacent to optimal solutions (§ 3) . We develop a prediction model using these ratios but find them to be inadequate for predictions and optimization.
(3) Finally, apply PowerTrain predictions to optimize training time and power limit trade-offs for several DNN workloads (§ 6), which out-perform other baselines.

Choice of hardware platform, applications, etc. Nvidia Jetsons are arguably the most popular high-end accelerated edge devices [41] and are lead the MLPerf benchmarks [44]. That said, these techniques can be extended to other hardware platforms. Since energy can be derived as a function of power and time, we focus on these 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 dividing the training data size by the batch size to get the number of minibatches in the epoch, and hence the epoch time. We focus on DNN training workloads given their need for high compute and the growing push towards federated learning.

2 EXPERIMENT SETUP

2.1 Hardware and Settings

Nvidia Jetson Orin AGX [40] 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 1. 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, we we do not use and leave them in their default states and turned off. We also use the Nvidia Jetson Xavier AGX [39] developer kit, the predecessor of Orin AGX, for generalizability experiments in a later section. We refer to the two devices as Orin and Xavier respectively.

2.2 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 2): *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 smart phone 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)

Table 1. Specifications of NVIDIA Jetson Orin AGX and Xavier AGX devkits used in evaluations

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

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

Table 2. 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**	**#Samp.**	**Size**	**E.Time (m)**
Image classif.	**MobileNetv3[25]**	20	$5.5M$	$225.4M$	**GLD23k [52]**	$23k$	$2.8GB$	2.3
Image classif.	**ResNet-18 [23]**	18	$11.7M$	$1.8G$	**ImageNet Valid. [15]**	$50k$	$6.7GB$	3
Object det.	**YOLO-v8n [27]**	53	$3.2M$	$8.7G$	**COCO minitrain[47]**	$25k$	$3.9GB$	4.9
Question ans.	**BERT base [16]**	12	$110M$	$11.5T$	**Squad [43]**	$70k$	$40.2MB$	68.6
Next word pred.	**LSTM [55]**	2	$8.6M$	$3.9G$	**Wikitext [18]**	$36k$	$17.8MB$	0.4

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

used for vision modeling, featuring residual blocks or 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. All of these are representative of typical edge and federated learning workloads[3, 26, 53] , 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).

PyTorch is used for training the DNNs and the Dataloader module is used to fetch and pre-process the image 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 $num\_workers = 4$. However, this YOLO version has a bug for this worker setting, which is fixed in a later version of PyTorch but not available for our JetPack version [5]. Hence, we use $num\_workers = 0$, which causes YOLO training to have GPU stalls as the main process is responsible both for the data loading and the compute. 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.

[5]https://github.com/pytorch/pytorch/issues/48709

2.3 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 [41].

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.

2.4 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. 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 the power stabilize, 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 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 [6]. 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) [7]. 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. It also helps us derive a sound analytical model. This dataset collected is available at `Anonymized` and in itself forms a rich community asset.

3 ANALYTICAL MODELING AND PREDICTION

We first propose an analytical modeling approach to understand the power and time characteristics of the DNN workloads on Orin AGX using detailed profiling data. This helps us identify a sweet

[6]Interestingly, the number and actual frequencies available change with the Board Support Package (BSP) version. We also notice 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.

[7]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.

spot at which the CPU and GPU are balanced, and also appear proximate to the Pareto front of time and power. We then use this intuition to develop a prediction model based on a curve-fit over these balance points for a reference DNN workload, which is adapted to other workloads with limited incremental profiling. We report performance results for this analytical modeling in § 5 which, however, are not promising.

3.1 Balanced Systems

DNN training workloads can stress all resources of an edge device. Pre-processing is CPU intensive, while forward and backward pass training are GPU intensive, and all stages use the shared memory. A bottleneck in any one of these resources can cause the training pipeline to slow down. At the same time, these resource settings also influence the power load on the system. Having a resource over-provisioned while another is the bottleneck can cause the power to increase without any increase in the performance. As a result, our hypothesis is that having a balanced system configuration by setting the right power mode can provide the best possible training time with the least power load.

Such a hypothesis is common in computer systems, most famously captured by Amdahl's Second Law, which proposed a rule of thumb for a balanced system and was subsequently revised by Jim Gray [14, 21]. These involve ratios between the disk I/O speed and CPU speed (IOPS vs. MIPS), and memory speed and CPU speed (MBps vs. MIPS) based on observations of applications and data platforms. These have helped design computer systems and clusters with adequate hardware provisioning of CPU cores and speeds, memory capacity and speeds, and disk I/O speeds [20, 51]. Similar trends have been observed in HPC systems, documenting the FLOPS performance of CPUs to (GPU) accelerators in the Top 500 supercomputers [29] where GPUs have come to dominate.

However, the contrast is not as stark in edge accelerators given the workloads and the CPU and GPU capabilities . Depending on the DNN workload, CPU pre-processing can be the bottleneck [41], while in other cases, the GPU can clearly be the limiting factor where the forward and backward passes dominate. We posit that similar balanced ratios exist between CPU and GPU clockspeeds for accelerated edge devices for DNN training workloads, and in the context of Jetson devices with fine-grained control over power modes, these can be exploited by setting the right power modes rather than *a priori* selection of the hardware configuration, and attempt to discover them here. In particular, we are interested in the time and power trade-offs offered by such configurations.

3.2 Identifying Balanced Power Modes for DNN Workloads

We analyze the detailed profiling data collected for the three default DNN workloads on Orin AGX to examine the variation of training time and power, across different CPU and GPU frequencies for a fixed memory frequency. We focus of the two of the four highest memory frequencies, 2133MHz and 3199MHz, since the memory speed is the limiting factor at lower frequencies of 204MHz and 665MHz. These come to 4368/2/6=364 power modes out of $\approx 4.4k$ profiled.

In Figs. 3a and 3b, we report the training time and power as contour plots for the MobileNet workload for different GPU (X axis) and CPU (Y axis) speeds. All points on the contour have the same value of time or power, as applicable, while the color of the contour lines indicates the value. The plots for the other workloads are given in the Appendix under Fig. 13.

There exists a time balanced ratio between CPU and GPU frequencies for a DNN workload. We define the *time balance ratio* $\beta_t = \frac{f_c}{f_g}$ such that the time does not improve if the frequency of just CPU or GPU is increased without correspondingly increasing the other. In the contour plots in Fig. 3a, we observe that there exists an inflection point or region in the frequencies beyond which time saturates and does not decrease. E.g., for memory speed 2133MHz, the third contour line has a training time of 300 s with a inflection point at CPU=800MHz and GPU=525MHz ($ratio \approx 1.5$).

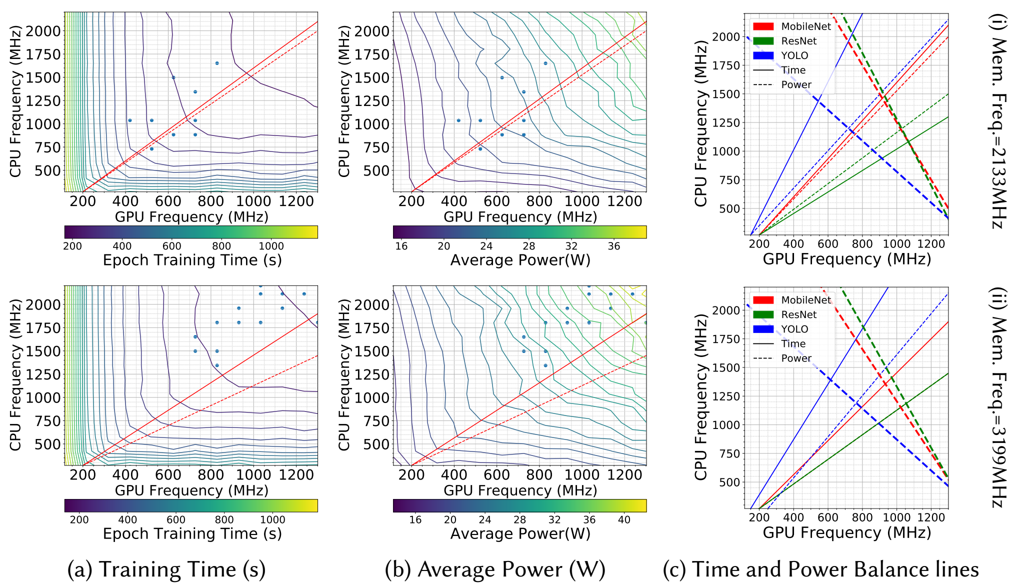


Fig. 3. (a) and (b) show contour plots of observed *epoch training time* and *average power* per epoch for MobileNet across diverse CPU and GPU frequencies and 12 CPU cores, for 2 memory frequencies (top and bottom rows). Red solid (time) and dashed (power) lines are drawn through the observed balance ratios between CPU and GPU frequencies. The ground truth Pareto points are shown as blue dots on the contour plots. (c) shows the balance lines for MobileNet, ResNet and YOLO together.

Increasing the GPU beyond 525MHz without increasing the CPU speed does not improve the time (horizontal section of contour line); likewise, increasing the CPU speed without increasing the GPU frequency (vertical section). This inflection is seen for many of the contour lines, other than for the lowest GPU speeds $< 200MHz$.

We draw the *balanced ratio line* for time (solid red) on these contours as the best linear fit that pass through the balanced points on the contours, selected visually. The slope of these lines form the time balance ratio, for a given workload and memory frequency. These lines are shown for all three default workloads in Fig. 3c and their balanced ratios listed in Table 4.

There exists a power balanced ratio between lower CPU and GPU frequencies, and power scaling ratio between higher CPU and GPU frequencies for a DNN workload. At lower CPU and GPU frequencies (bottom left of Fig. 3b), we see a similar inflection point for the *power balanced ratio* β_p between CPU and GPU. This indicates that at low CPU frequency, increasing the GPU frequency beyond the balance ratio does not increase the power load, and vice versa. This is because when the CPU frequency is low, the pre-processing is slower and the GPU at higher frequencies exceeding the balance point is stalling on the CPU to provide new work. So the GPU utilization is low and the power load does not increase despite a higher clock speed.

But at higher frequencies for the CPU and GPU (top right), such stalls by CPU and GPU on each other is not as prominent and both their utilizations are relatively high. Here, rather than an inflection point we see an inverse relation between CPU and GPU frequencies for a given power whose slope is given by the *scaling ratio* $\sigma_p = \frac{\delta_c}{\delta_g}$, i.e., if the CPU frequency increases by δ_c, then the GPU frequency should reduce by $\sigma \cdot \delta_g$ to maintain the same power load. If both frequencies increase, then the power load increases. The balance and scaling ratios for power are shown as

	ResNet				MobileNet				YOLO			
Mem. Freq.	**Bal. Time**	**Bal. Pow.**	**Bal. Avg.**	**Scal. Pow.**	**Bal. Time**	**Bal. Pow.**	**Bal. Avg.**	**Scal. Pow.**	**Bal. Time**	**Bal. Pow.**	**Bal. Avg.**	**Scal. Pow.**
2133 MHz	0.9	1.1	1.0	-2.9	1.7	1.6	1.6	-2.5	3.0	1.6	2.3	-1.4
3199 MHz	1.1	1.1	1.1	-2.7	1.5	1.1	1.3	-2.3	2.4	1.8	2.1	-1.4

Fig. 4. Balance ratios between CPU and GPU for time, power and their average, for three DNN workloads.

red narrow-dashed and wide-dashed lines in Fig. 3c. We note that the scaling ratios appear similar across memory frequencies thouhg they vary across workloads.

The time and power balanced ratios may diverge, but tend to be Pareto adjacent. The time and power balanced ratios are close but may not be identical. While they are fairly similar for MobileNet and ResNet for a given memory speed (e.g., 1.7 and 1.6 for MobileNet at 2133MHz), they are divergent for YOLO. This is because YOLO is compute intensive and there is little sign of an inflection point for power even for low frequencies, and the scaling ratio dominates. Nonetheless, we observe that the Pareto front of time vs. power lies for the DNN workloads lie close to these balanced ratio lines. These Pareto points derived from Fig. 10b are shown using markers in the contour plots above, and as we see, they lie on or adjacent to the balance lines. This indicates that power modes at the balanced ratios are able to capture the best trade-off between time and power. This is understandable. At the balance point for time, increasing just the CPU or the GPU speed does not reduce the time, but for higher frequencies, it can cause the power to increase based on the scaling ratio. So staying on the balance point is the sweet-spot for achieving a given training time at the minimal possible CPU and GPU speeds, which in turn minimize the power load.

Increasing the memory frequency tends to reduce the balanced ratio between CPU and GPU frequencies. We see from Table 4 that increasing the memory speed from 2133MHz to 3199MHz either keeps the average balanced ratio similar (e.g., 1.0 to 1.1 for ResNet), or reduces it (1.6 to 1.3 for MobileNet). This drop is more prominent for models that have a higher FLOPs. This indicates that DNN workloads that are GPU-heavy, like MobileNet and YOLO, also perform more memory accesses, and are more sensitive to memory speeds than the CPU. While the power modes offer only four memory frequencies, this is still a factor to be considered. It is also intuitive that as the memory speed increases, the Pareto points in the contour plots shift to the top right, i.e., to a higher CPU and GPU frequencies, indicating that all three grow to support a faster training time. The power balanced ratios also remain similar or fall marginally. We further observe that these points fall in two tiers on the scatter plot - the lower tier points correspond to the higher 2 memory frequencies and take very little time, the higher tier points correspond to the lower 2 memory frequencies and take much more time. Although these should be close to the Pareto too, these two memory frequencies act as a bottleneck, limiting the achievable time.

The time and power balanced ratios vary across DNN workloads. As we can see in Fig. 3c and Table 4, the balanced point lines and ratios for time across DNN workloads is different, e.g., spanning 1.1, 1.5 and 2.4 for ResNet, MobileNet and YOLO at 2133MHz memory speed. This indicates that the DNN architecture get a different degree of benefit from CPU and GPU. This limits the genralizability of these ratios across various DNN workloads although it is more consistent within a workload.

3.3 Analytical Prediction Model

We sample points within this V-shaped region for every DNN and fit equations to them using multi-variate linear regression. In case of time, we perform log-log transformations on both input and output features and then perform linear regression. The linear regression equation is of the form: $a \cdot x1 + b \cdot x2 + c = T$ The input feature vector is *log(CPU frequency* $(x1)$*), log(GPU frequency* $(x3)$*)* and the output is $log(T)$. For power, we simply use linear regression without any transformations: $a \cdot x1 + b \cdot x2 + c = P$ The input feature vector is *CPU frequency* $(x1)$*, GPU frequency*

$(x3)$ and the output is P . We notice that these equations also have different coefficients across DNNs, indicating a workload-specific impact of frequencies on performance and power.

Since coming up with the balance lines for every DNN requires running a large number of power modes, we explore if shifting the equations obtained from one DNN to another using a small set of points ($50 - 60$ power modes) that obey the same ratio. For example, with ResNet's equations as the base, we identify all the power modes that fall in the V region for ResNet and use the time and power values for MobileNet for these same power modes to shift the equation. For this, we implement a simple single-layer Multi-Layer Perceptron (MLP) based regression model using the sklearn library. Multi-layer Perceptron [46] is a type of feed-forward network, where each node in a given layer is fully connected to all the nodes in the preceding and succeeding layers. Here, the weights of the MLP are the co-efficients a and b and the bias is c. These are initialized using the equations of the base model. We train the MLP for 200 epochs using data from the new DNN and verify that it converges. We do this for both the higher memory frequencies (2133 and 3199MHz). We plot a 2D projection of these transferred curves in Figures 14 (shown in Appendix) and use them for prediction. As we show later, these are not good for prediction.

4 ML-DRIVEN MODELING AND PREDICTION

While the analytical model proposed above is based on certain rules of thumb and observations on the balancing and scaling ratios, it does not translate well into good predictions or perfect optimizations, as we later see. Hence, we take an alternate data-drive approach 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 appear 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 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, a new hardware as well. This overhead is manifest primarily in the form of profiling time since the actual time to perform training itself is negligible, taking 15 *minutes* at most. ML models tend to be more accurate with more training samples, yet collecting profiling data for a growing number of power modes can take 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 the this data collection time starts being a large part of the multi-epoch DNN workload training time.

4.1 Neural Network (NN) Prediction Model per Workload

Neural Networks (NN) [35] 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 has 4 dense layers with 256, 128, 64 and 1 neurons, respectively. 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 for a power mode: CPU cores, CPU frequency, GPU frequency and memory frequency, while the output layer returns the predicted training time (or power) per minibatch. 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. We use the ReLu activation function for the first 3 dense layers, and a linear activation function for the final layer. 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 in between the 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 least validation loss) seen during training and use it as the final model.

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 4.4*k* samples for our Orin AGX for the default DNN workloads. (2) The second provides a much smaller number of training data, from 10 till 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 14h 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.

4.2 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.

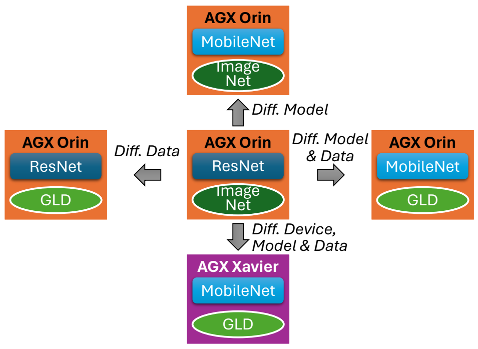


Fig. 5. Generalizability of PowerTrain.

Transfer learning [7, 8] is a popular technique in deep learning where a model developed for one task is applied on a different but related task. The core idea behind 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 Fig. 1). 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 linear 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. 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 (§ 5), we train the reference NN for one of the three default DNN workloads using the full corpus of 4.4*k* power modes as training data. 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

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

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

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 (Fig. 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. Results from these are presented next.

5 PREDICTION RESULTS

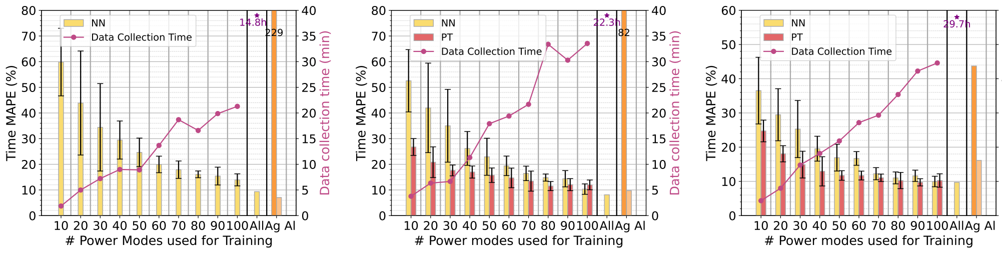
In this section, we present a detailed evaluation of PowerTrain for time and power predictions, and compare it with the Neural Network (NN) baseline using large and small sampling. These are evaluated on their Mean Absolute Percentage Error (MAPE) relative to the ground truth time taken and power for 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 1*s*, 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 range of 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.

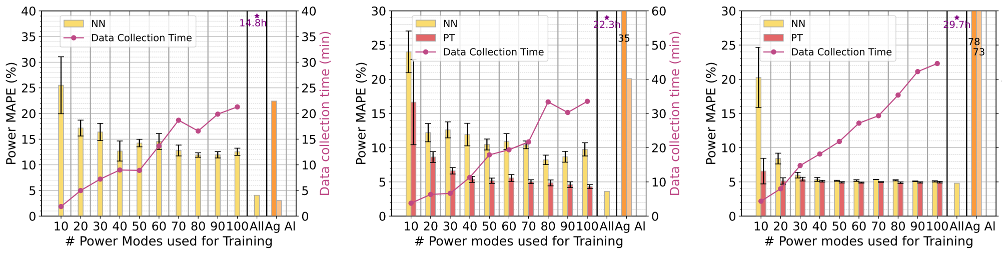
5.1 Choice of Base DNN workload for PowerTrain

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 4.4*k* training samples to allow reference model training. In Fig. 6, we compare the validation MAPE when using each of these three as 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.% 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



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

Fig. 7. Prediction error and profiling overheads for *time prediction* for different DNN workloads



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

Fig. 8. Prediction error and profiling overheads for *power prediction* for different DNN workloads

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.

5.2 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 transfered to the other two, with different number of training samples used for transfer learning. We also show the accuracy of the predictions using the Analytical model we had proposed in § 3. Figs. 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 even smaller 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 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 acccuracy 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 15mins for all 3 DNNs, and under 25mins 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.

By 100 power modes, PowerTrain reaches close to the optimum accuracy for time and power prediction, comparable to NN training on all 4.4k power modes. 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 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 10.18% for PT at 100 samples while the NN on all is a close 9.7%, and a similar trend seen for power prediction as well. This again demonstrates the ability of PT in being able to given higher accuracy with fewer samples.

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

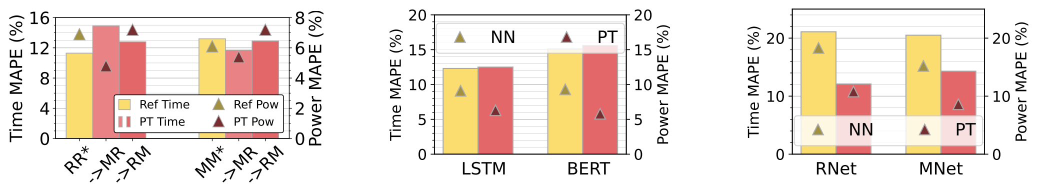
PT Power prediction has lower MAPEs (5-10%) as compared to time predictions (10-20%) across all DNNs. From Figures 8b and 7b, at 10 power mode samples for MobileNet, we observe a 2× lower MAPE for power as compared to time. This holds across DNN workloads, and for both PT and NN. At 50 power modes, which 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.

The predictions of the analytical model have a very high global and local MAPE. The local MAPE is calculated over the set of power modes that lie on or very close to the base model's (ResNet) balance lines i.e the V region (bar Al in Fig 7 and 8) The global MAPE is calculated over all possible predictions of the analytical model and is plotted as bar Ag in Fig 7 and 8. As seen, MobileNet has a global time MAPE of 82.06% and a local time MAPE of 9.73%, a global and local power MAPEs of 34.86% and 20.09%. Similarly, YOLO has local and global time MAPEs of 16.1% and 43.7%.

5.3 Generalization of PowerTrain

A vital benefit of PowerTrain is its ability to generalize to heterogeneous DNN workloads using few profiling samples to achieve a higher accuracy. We next evaluate this generalizability along three dimensions (Fig. 5), each being more diverse compared to the reference DNN workload trained: (1) Either the DNN architecture or dataset being different from the reference DNN workload, (2) Both the DNN architecture and dataset being different, and (3) The base 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.

5.3.1 Different DNN Architecture or Dataset from Reference. Here, PowerTrain uses two different reference DNN models: one trained on ResNet with its native dataset ImageNet (Table 2, 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



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

Fig. 9. PowerTrain generalization when reference model is transferred to new workloads and device.

DNN architecture or the dataset is different [8]. These combinations are listed as $RR* \longrightarrow RM$, when transferring to ResNet architecture with MobileNet's native dataset GLD, $RR* \longrightarrow MR$ when transferring to MobileNet architecture with ResNet's native dataset ImageNet, and similarly $MM* \longrightarrow MR$ and $MM* \longrightarrow RM$. So, either the architecture or dataset overlaps with the reference. We use 50 random power mode samples for transfering 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. In Fig.9a, having overlapping DNN architectures, $RR* \longrightarrow RM$ and $MM* \longrightarrow MR$, 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, whern the datasets overlap but the DNN architecture changes, i.e., $RR* \longrightarrow MR$ and $MM* \longrightarrow RM$, we see the time MAPE increase from 11.3% to 14.9% and 13.2% to 12.9%, respectively. 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.

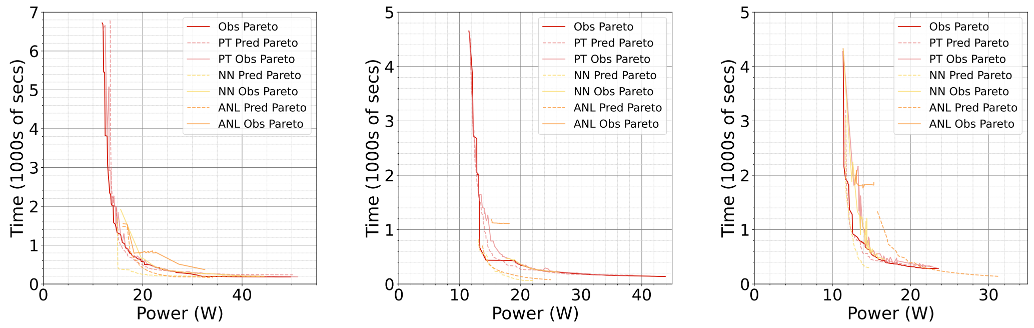
5.3.2 *Distinct DNNs Workloads.*

The earlier results in Figs. 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 2), and text based training datasets rather than images. Specifically, BERT (Bidirectional Encoder Representations from Transformers) Base Uncased [16] 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 [43]. 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 [18].

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 Fig. 9b in comparison with an NN model as basline trained using the same 50 power mode samples.

PowerTrain generalizes well to diverse DNNs with few samples and out-performs NN on power predictions. As seen from Fig. 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

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



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

Fig. 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

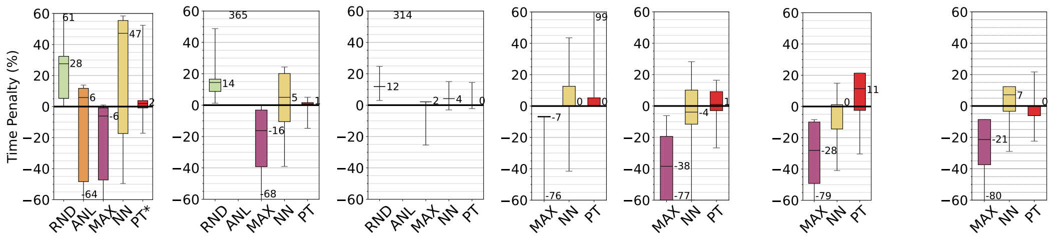
is visible for power predictions, where PT has a MAPE of 6.3% while NN is 9.1% for LSTM, and likewise, PT has a 3.5% better MAPE for BERT.

5.3.3 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 [39], which is the comparable previous generation top-end Jetson device. As shown in Table 1, 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. Fig. 9c, we see that when PT transfers from the reference DNN trained using ResNet workload on Orin to the same workload on Xavier (device along changes), we see a time prediction MAPE of 12% and power prediction MAPE of 11%. This is much better than training a NN on just 50 power mode samples of the DNN workload on Xavier, which reports a prediction error 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. 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 Nano series, or even different edge models such as Raspberry Pi. This is left to future work.

6 OPTIMIZING DNN WORKLOADS USING OUR PREDICTION MODELS

One of the applications of our time and power prediction models is to optimize DNN workload applications. 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} is within a user-specified budget P_b, i.e., Given m_{tr}, select $pm \in \mathbb{PM} \quad | \quad \min t_{tr} \quad \text{s.t.} \quad P_{tr} \leq P_b$.



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

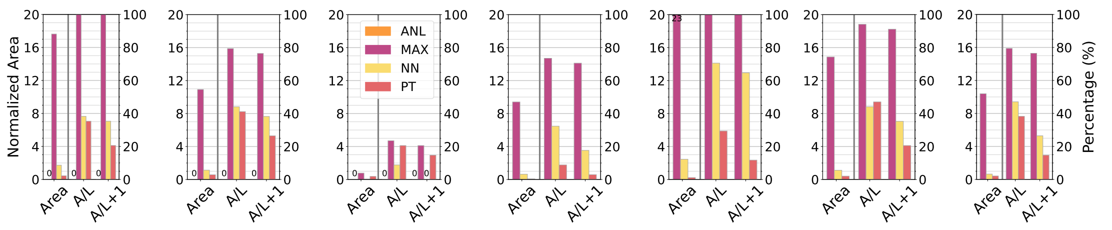
Fig. 11. *Time Penalty %* for model-based optimization relative to ideal Pareto time. Lower is better. *PT for ResNet indicates training of base model on full data.

As shown in Fig. 1, we use two prediction models to solve the optimization problem, using PowerTrain and the Analytical approach. For PowerTrain, we use the reference model trained using the ResNet workload on 4368 power modes, and tranfer 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* using the profiling information for 4.4k power modes, as shown in Fig. 10, 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 from these predictions, also shown in Fig. 10 (PT Pred and ANL Pred), and this predicted Pareto is used for the optimization decisions. We also report the matching ground-truth values for these predicted power modes on the Pareto, shown as PT Obs Pareto. The NN Prediction model using 50 samples serves as a baseline to compare against, and is also shown. *We see that the predicted and observe Paretos are close for PT while they deviate for NN and ANL.*

Baselines. We offer a few baselines approaches for solving the optimization problem to contrast against. **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** 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 use the **NN model** custom trained for each DNN workload using 50 random power modes, and use this to predict the Pareto front over all power modes and perform optimization. We have also evaluated a more targeted sampling based on the analytical modeling, but this performs only slightly better than random sampling, and therefore we do not report its results here.

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 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 of power in excess of the given budget for the given solution by a strategy (W per solution); the % of solutions for a strategy that exceed the power limit; and the % of solutions that exceed the limit but more than a 1-Watt threshold. There are no power errors for the observation based random sampling.



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

Fig. 12. Pareto Power Errors for different DNN workloads

PT optimization results in a lower time penalty as compared to NN in most cases. As seen from Figure 11, 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, PT has a lower median time penalty. PT does worse than NN only for ResNet + MN data.

PT optimization has the lowest normalized excess power area for most models and exceeds the power budget with $1W$ threshold $< 25\%$ of the time. In Figure 12, the normalized area with power exceeding the given user budget is lowest for PT across all DNN workloads except YOLO (6 out of 7 cases). Additionally, this budget+1W buffer is breached in $< 20\%$ of the solutions for 6 out of 7 DNNs. So PT's predictions and solutions help stay within the power budget most of the time.

Analytical modeling does not work well for optimization. In Fig. 11 and 12, the analytical modeling has a low time penalty and never exceeds power budget only for ResNet. This points to the adaptation of the ResNet curve fits to MobileNet and YOLO being poor. This demonstrates the limitations of analytical modeling for automated prediction, and rather suggests that rules of thumbs based on the balanced ratios and scaling ratio may offer better guidance to humans.

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 Figures 11 and 12, MAXN has negative time penalties for ResNet, MobileNet and YOLO, i.e., much faster than the optimal solution. But this is because of setting all frequencies to the maximum, thus violating the power limits often. In contrast, a Pareto from Random sampling is 12-28% slower than the optimal.

7 RELATED WORK

Energy, Power and Performance Modeling on Edge devices. A previous work [22] uses roofline modeling on Jetson TK1 and TX1 to understand and characterize the performance of CPU and GPU micro-benchmarks for matrix multiplication. MIRAGE [1] uses gradient tree boosting to predict runtime and power consumption for DNN inference on a Jetson Xavier AGX. Others [17, 24] have investigated the impact of frequencies and cores with the latency, power and energy for inferencing on the Jetson Nano. Abdelhafez and Ripeanu [2] 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 inferening or micro-benchmarks, and not DNN training. A recent study characterized training of DNNs on Jetson AGX, NX Xavier and Nano [41]. They offer initial insights on the opportunities for deep learning training on Jetson and some of the performance and power behavior. They also propose 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. We also go beyond and develop a balanced ratio for the CPU and GPU speeds.

Energy, Power and Performance Modeling on GPU servers. NeuralPower [9] uses polynomial regression to predict power, runtime and energy consumption of CNNs for inference on server-grade GPUs. Paleo [42] 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. Some [38] use linear regression to predict the GPU power consumption of CUDA kernels based on hardware performance counters. Others [50] build a power, performance and energy model for application kernels on desktop/server-grade GPUs. Recent work [33] build 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.

Optimization studies on the edge and server. Zeus [57] 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 [4] select optimal GPU frequency that lower power while minimizing the performance impact on server-grade GPUs. D3 [19] looks at optimizing the runtime accuracy trade-offs for DNN inference workloads in autonomous vehicle pipelines that have dynamically varying deadlines. ALERT [54] 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 [13] 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 [28] find the Pareto optimal scheduling of multiple deep learning applications in terms of the response time and energy consumption on smartphones. Mephesto [37] models memory contention for kernel placement on heterogeneous SoCs to achieve the desired energy performance tradeoff. 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.

8 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 workloads 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 also explore an analytical method to come up with a CPU:GPU balanced ratio and demonstrate that the points on this ratio are Pareto adjacent. However, this is not adequate and does poorly on prediction or optimization problems. 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.

If dynamic switching of power modes is supported in a future version of Jetsons without requiring the periodic reboot, this method can be done in an online fashion, allowing us to collect sampling data for ML training while the DNN workload runs productively. As future work, we also plan to explore reinforcement learning based methods to solve this problem. We also plan to solve variants of this problem, such as time and power prediction for concurrent training and inference workloads. We also plan to investigate how well this translates to devices such as GPU servers. There is also work to be done on better exploiting the balanced ratios.

REFERENCES

[1] Hazem A. Abdelhafez, Hassan Halawa, Mohamed Osama Ahmed, Karthik Pattabiraman, and Matei Ripeanu. 2021. MIRAGE: Machine Learning-based Modeling of Identical Replicas of the Jetson AGX Embedded Platform. In *2021 IEEE/ACM Symposium on Edge Computing (SEC)*.

[2] Hazem A. Abdelhafez and Matei Ripeanu. 2019. 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)*.

[3] Ahmed M. Abdelmoniem, Atal Narayan Sahu, Marco Canini, and Suhaib A. Fahmy. 2023. REFL: Resource-Efficient Federated Learning. In *Proceedings of the Eighteenth European Conference on Computer Systems*.

[4] Ghazanfar Ali, Mert Side, Sridutt Bhalachandra, Nicholas J. Wright, and Yong Chen. 2023. Performance-Aware Energy-Efficient GPU Frequency Selection using DNN-based Models. In *Proceedings of the 52nd International Conference on Parallel Processing*.

[5] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. 2022. Ekya: Continuous Learning of Video Analytics Models on Edge Compute Servers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*.

[6] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. In *Proceedings of Machine Learning and Systems*. 374–388.

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

[8] Stevo Bozinovski and Ante Fulgosi. 1976. The influence of pattern similarity and transfer learning upon training of a base perceptron b2. In *Proceedings of Symposium Informatica*, Vol. 3. 121–126.

[9] Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, and Diana Marculescu. 2017. Neuralpower: Predict and deploy energy-efficient convolutional neural networks. In *Asian Conference on Machine Learning*.

[10] Thyago P. Carvalho, Fabrízzio A. A. M. N. Soares, Roberto Vita, Roberto da P. Francisco, João P. Basto, and Symone G. S. Alcalá. 2019. A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering* 137 (2019).

[11] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. 2020. TiFL: A Tier-based Federated Learning System. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*.

[12] Sehyun Chun, Nima Hamidi Ghalehjegh, Joseph Choi, Chris Schwarz, John Gaspar, Daniel McGehee, and Stephen Baek. 2019. 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*.

[13] Ismet Dagli, Alexander Cieslewicz, Jedidiah McClurg, and Mehmet E. Belviranli. 2022. AxoNN: Energy-Aware Execution of Neural Network Inference on Multi-Accelerator Heterogeneous SoCs. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*.

[14] Arghya Kusum Das, Jaeki Hong, Sayan Goswami, Richard Platania, Kisung Lee, Wooseok Chang, Seung-Jong Park, and Ling Liu. 2017. Augmenting Amdahl's Second Law: A Theoretical Model to Build Cost-Effective Balanced HPC Infrastructure for Data-Driven Science. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*.

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

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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*.

[17] Anurag Dutt, Sri Pramodh Rachuri, Ashley Lobo, Nazeer Shaik, Anshul Gandhi, and Zhenhua Liu. 2023. Evaluating the energy impact of device parameters for DNN inference on edge. In *International Green and Sustainable Computing*.

[18] Hugging Face. 2021. Datasets : wikitext. https://huggingface.co/datasets/wikitext.

[19] Ionel Gog, Sukrit Kalra, Peter Schafhalter, Joseph E. Gonzalez, and Ion Stoica. 2022. D3: A Dynamic Deadline-Driven Approach for Building Autonomous Vehicles. In *Proceedings of the Seventeenth European Conference on Computer Systems*.

[20] J. Gray, G. Bell, and A. Szalay. 2006. Petascale Computational Systems. *Computer* 39, 01 (2006).

[21] J. Gray and P. Shenoy. 2000. Rules of thumb in data engineering. In *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073)*.

[22] Hassan Halawa, Hazem A. Abdelhafez, Andrew Boktor, and Matei Ripeanu. 2017. NVIDIA Jetson Platform Characterization. In *Euro-Par 2017: Parallel Processing*.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[24] Stephan Holly, Alexander Wendt, and Martin Lechner. 2020. Profiling Energy Consumption of Deep Neural Networks on NVIDIA Jetson Nano. In *2020 11th International Green and Sustainable Computing Workshops (IGSC)*.

[25] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. 2019. Searching for MobileNetV3. In *IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE.

[26] Deepthi Jallepalli, Navya Chennagiri Ravikumar, Poojitha Vurtur Badarinath, Shravya Uchil, and Mahima Agumbe Suresh. 2021. Federated Learning for Object Detection in Autonomous Vehicles. In *2021 IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService)*.

[27] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. 2023. *Ultralytics YOLOv8*. https://github.com/ultralytics/ultralytics

[28] Duseok Kang, Jinwoo Oh, Jongwoo Choi, Youngmin Yi, and Soonhoi Ha. 2020. Scheduling of deep learning applications onto heterogeneous processors in an embedded device. *IEEE Access* 8 (2020), 43980–43991.

[29] Awais Khan, Hyogi Sim, Sudharshan S. Vazhkudai, Ali R. Butt, and Youngjae Kim. 2021. An Analysis of System Balance and Architectural Trends Based on Top500 Supercomputers. In *The International Conference on High Performance Computing in Asia-Pacific Region*.

[30] Latif U. Khan, Ibrar Yaqoob, Nguyen H. Tran, S. M. Ahsan Kazmi, Tri Nguyen Dang, and Choong Seon Hong. 2020. Edge-Computing-Enabled Smart Cities: A Comprehensive Survey. *IEEE Internet of Things Journal* 7 (2020).

[31] Yongho Kim, Seongha Park, Sean Shahkarami, Rajesh Sankaran, Nicola Ferrier, and Pete Beckman. 2022. Goal-driven scheduling model in edge computing for smart city applications. *J. Parallel and Distrib. Comput.* 167 (2022).

[32] Young Geun Kim and Carole-Jean Wu. 2020. AutoScale: Energy Efficiency Optimization for Stochastic Edge Inference Using Reinforcement Learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1082–1096. https://doi.org/10.1109/MICRO50266.2020.00090

[33] Shijian Li, Robert J Walls, and Tian Guo. 2020. Characterizing and modeling distributed training with transient cloud gpu servers. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*.

[34] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. 2019. Edge Computing for Autonomous Driving: Opportunities and Challenges. *Proc. IEEE* 107, 8 (2019).

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

[36] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*.

[37] Mohammad Alaul Haque Monil, Mehmet E Belviranli, Seyong Lee, Jeffrey S Vetter, and Allen D Malony. 2020. Mephesto: Modeling energy-performance in heterogeneous socs and their trade-offs. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*. 413–425.

[38] Hitoshi Nagasaka, Naoya Maruyama, Akira Nukada, Toshio Endo, and Satoshi Matsuoka. 2010. Statistical power modeling of GPU kernels using performance counters. In *International Conference on Green Computing*.

[39] NVIDIA. 2018. Jetson Xavier Series. https://www.nvidia.com/en-in/autonomous-machines/embedded-systems/jetson-agx-xavier/.

[40] NVIDIA. 2022. Jetson AGX Orin Developer Kit. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/.

[41] Prashanthi S.K, Sai Anuroop Kesanapalli, and Yogesh Simmhan. December 2022. Characterizing the Performance of Accelerated Jetson Edge Devices for Training Deep Learning Models. *Proc. ACM Meas. Anal. Comput. Syst.* 6, 3 (December 2022).

[42] Hang Qi, Evan R Sparks, and Ameet Talwalkar. 2017. Paleo: A performance model for deep neural networks. In *5th International Conference on Learning Representations, ICLR*.

[43] Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. *arXiv preprint arXiv:1806.03822* (2018).

[44] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. 2020. MLPerf Inference Benchmark. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*.

[45] Crefeda Faviola Rodrigues, Graham Riley, and Mikel Luján. 2017. Fine-grained energy profiling for deep convolutional neural networks on the Jetson TX1. In *2017 IEEE International Symposium on Workload Characterization (IISWC)*.

[46] Frank Rosenblatt. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* 65, 6 (1958), 386.

[47] Nermin Samet, Samet Hicsonmez, and Emre Akbas. 2020. HoughNet: Integrating near and long-range evidence for bottom-up object detection. In *European Conference on Computer Vision (ECCV)*.

[48] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. 2017. Incremental learning of object detectors without catastrophic forgetting. In *Proceedings of the IEEE international conference on computer vision*. 3400–3409.

[49] Guthemberg Silvestre, Sébastien Monnet, Ruby Krishnaswamy, and Pierre Sens. 2012. Caju: A Content Distribution System for Edge Networks. In *Euro-Par Workshops*.

[50] Shuaiwen Song, Chunyi Su, Barry Rountree, and Kirk W. Cameron. 2013. A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*.

[51] A.S. Szalay, G. Bell, J. Vandenberg, A. Wonders, R. Burns, Dan Fay, J. Heasley, T. Hey, M. Nieto-SantiSteban, A. Thakar, C. van Ingen, and R. Wilton. 2009. GrayWulf: Scalable Clustered Architecture for Data Intensive Computing. In *2009 42nd Hawaii International Conference on System Sciences*.

[52] TensorFlow. 2022. TFF GLDv2. https://www.tensorflow.org/federated/api_docs/python/tff/simulation/datasets/gldv2/load_data.

[53] Yuanyishu Tian, Yao Wan, Lingjuan Lyu, Dezhong Yao, Hai Jin, and Lichao Sun. 2022. FedBERT: When Federated Learning Meets Pre-training. *ACM Trans. Intell. Syst. Technol.* 13, 4 (2022).

[54] Chengcheng Wan, Muhammad Santriaji, Eri Rogers, Henry Hoffmann, Michael Maire, and Shan Lu. 2020. ALERT: Accurate Learning for Energy and Timeliness. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*.

[55] Essam Wisam. 2022. Language Modeling with LSTMs in PyTorch. https://towardsdatascience.com/language-modeling-with-lstms-in-pytorch-381a26badcbf/.

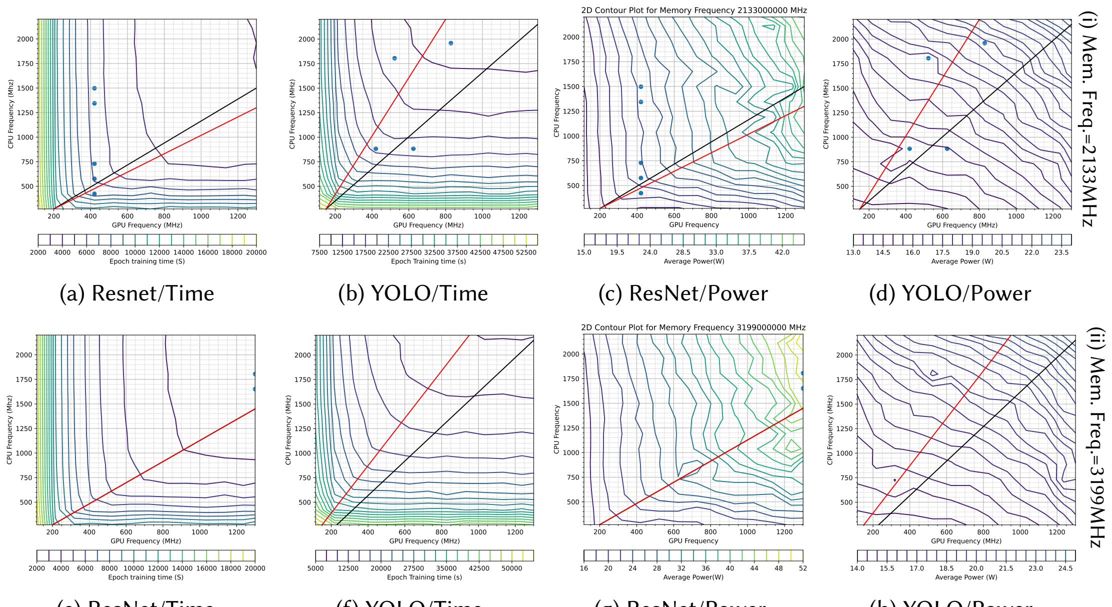
[56] Suwei Yang, Massimo Lupascu, and Kuldeep S. Meel. 2021. Predicting Forest Fire Using Remote Sensing Data And Machine Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 17 (2021).

[57] Jie You, Jae-Won Chung, and Mosharaf Chowdhury. 2023. Zeus: Understanding and Optimizing {GPU} Energy Consumption of {DNN} Training. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 119–139.

[58] Chit Wutyee Zaw, Shashi Raj Pandey, Kitae Kim, and Choong Seon Hong. 2021. Energy-Aware Resource Management for Federated Learning in Multi-Access Edge Computing Systems. *IEEE Access* 9 (2021).

[59] Michael Zhang, Chandra Krintz, and Rich Wolski. 2022. Sparta: Heat-Budget-Based Scheduling Framework on IoT Edge Systems. In *Edge Computing – EDGE 2021*.

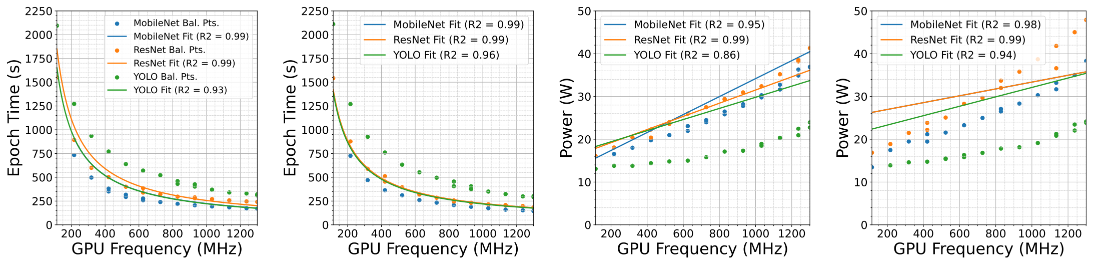
A APPENDIX



(a) Resnet/Time (b) YOLO/Time (c) ResNet/Power (d) YOLO/Power

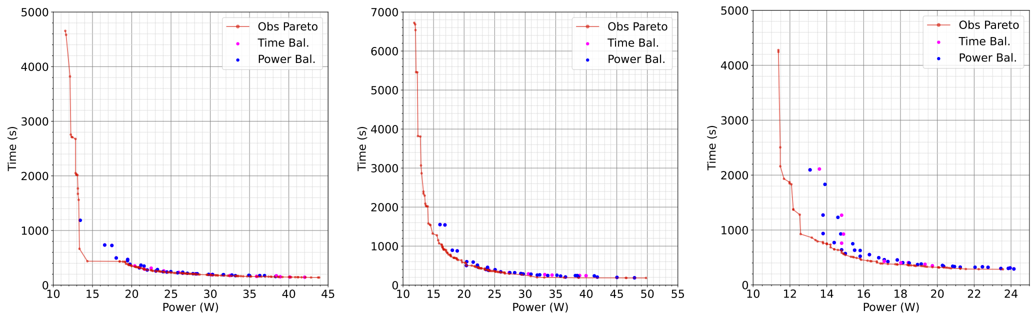
(e) ResNet/Time (f) YOLO/Time (g) ResNet/Power (h) YOLO/Power

Fig. 13. Contour plots of observed *training time*and *average power* per epoch for ResNet, MobileNet and YOLO across diverse CPU and GPU frequencies and 12 CPU cores for memory frequency of 2133 Mhz and 3199 Mhz. Red solid (time) and dashed (power) lines are drawn through the balance points between CPU and GPU frequency.



(a) Transferred Time Curves/Memory=2133MHz (b) Transferred Time Curves/Memory=3199MHz (c) Transferred Power Lines/Memory=2133MHz (d) Transferred Power Lines/Memory=3199MHz

Fig. 14. Transferred time and power curves for MobileNet, ResNet and YOLO.



(a) TL to MobileNet (b) ResNet NN on Full Data (c) TL to YOLO

Fig. 15. 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. Time and Power balance points for higher two memory frequencies are colored blue and magenta.