P2W: From Power Traces to Weights Matrix - An Unconventional Transfer Learning Approach

Roozbeh Siyadatzadeh
s.r.siyadatzadeh@liacs.leidenuniv.nl
LIACS, Leiden University
The Netherlands

Fatemeh Mehrafooz
LIACS, Leiden University
The Netherlands

Nele Mentens
LIACS, Leiden University
The Netherlands
Department of Electrical Engineering, KU Leuven
Belgium

Todor Stefanov
LIACS, Leiden University
The Netherlands

Abstract

The deployment of machine learning (ML) models in embedded systems on a chip (SoCs) is transforming fields like healthcare and autonomous vehicles. A primary challenge in training such embedded ML models is the scarcity of publicly available high-quality training data. Transfer learning addresses this by using knowledge from existing ML models. However, traditional approaches require direct access to these models, which is often infeasible for models on embedded SoCs. We introduce a novel transfer learning approach that extracts weights from an existing ML model running on an embedded SoC without direct access. Our method captures power consumption measurements from the SoC during ML model execution and translates them into an approximated weights matrix to initialize a new ML model. This enhances learning efficiency and predictive performance, especially when training data is limited. Our approach can effectively increase the accuracy of the new ML model up to three times compared to classical training methods using the same limited training data.

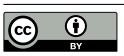
CCS Concepts

• **Computer systems organization** → **Embedded hardware**; **Embedded software**; • **Computing methodologies** → *Neural networks*.

Keywords

Embedded systems, Embedded machine-learning, Transfer Learning

ACM Reference Format:
Roozbeh Siyadatzadeh, Fatemeh Mehrafooz, Nele Mentens, and Todor Stefanov. 2026. P2W: From Power Traces to Weights Matrix - An Unconventional Transfer Learning Approach. In *The 41st ACM/SIGAPP Symposium on Applied Computing (SAC '26), March 23–27, 2026, Thessaloniki, Greece.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3748522.3779939



SAC '26, Thessaloniki, Greece

ACM ISBN 979-8-4007-2294-3/2026/03
https://doi.org/10.1145/3748522.3779939

1 Introduction

The advent and subsequent evolution of machine learning (ML) technologies have impacted various aspects of modern life, notably in fields like healthcare, autonomous vehicles, and cybersecurity [21]. However, one of the primary challenges in the ML domain is the gathering of relevant, high-quality data for training an ML model for a specific task. This is because of several reasons such as data confidentiality, high cost, time, and effort of data gathering [6]. One way to address this challenge is to use the knowledge encapsulated in existing ML models that are used for another related task, and to optimize and fine-tune these models to make them suitable for our new task. Such process is known as transfer learning [33].

Despite the clear benefits of transfer learning, a significant issue persists: it is not always easy to find a publicly available well-trained model to start from. For example, it is relatively easy to find a model on the internet for classifying simple objects, but it is not as easy to find a well-trained model for some classes of medical tasks due to several reasons, such as privacy and the expenses associated with such models. This lack of publicly available well-trained models for a specific class of tasks is even more severe in the domain of embedded systems on a chip (SoCs) for ML. For such SoCs, the design of a suitable ML model is impacted by limitations in resources and other requirements to make the model suitable for a specific kind of embedded SoC. These factors often prevent the full realization of the potential benefits of transfer learning in the embedded SoC domain.

Although public availability of well-trained ML models suitable for transfer learning in the embedded systems domain is limited, there are embedded SoCs everywhere that are running such ML models for a wide variety of tasks. The problem is that the general public does not have direct access to the models inside these chips and cannot benefit from the model's knowledge by utilizing conventional transfer learning approaches that require direct access to ML models.

Therefore, in this paper, we present a proof-of-concept of a novel unconventional transfer learning approach called **P2W**. Our approach facilitates the transfer of knowledge encapsulated in a relevant well-trained neural network, running inside an embedded SoC, to a new neural network. P2W enables effective training of the new neural network model to perform new similar tasks. This is done by measuring the power consumption of the SoC while it is running

the ML model. We take multiple power traces with different input data and analyze them with the help of an Encoder-Decoder Deep Neural Network (EDNN). Through such power trace analysis we extract, to a large extent, the knowledge of the well-trained ML model in the form of an *approximated* weights matrix. This matrix serves as foundational knowledge to start training new ML models. By initializing the new models with the approximated weights matrix (i.e., transferring the knowledge) and further training them with a limited amount of data, we obtain new well-trained ML models that can perform the same or related tasks very well.

The main novel contributions of the paper can be summarized as follows:

- We propose an unconventional transfer learning approach which circumvents the traditional barriers for researchers and developers of ML models related to the acquisition of high-quality data for training, in sensitive domains such as healthcare for example, where data availability is often restricted due to privacy/ethical concerns. By extracting encoded knowledge in the form of an approximated weights matrix, purely from power consumption traces, researchers/developers can transfer this knowledge to new models without requiring any access to sensitive information.
- We introduce a new power analysis technique that utilizes an `EDNN` to translate power traces into an approximated weights matrix.
- Using a limited training dataset, we evaluate our P2W approach by comparing the accuracy of an ML model trained with and without P2W. All experiments are conducted on a real hardware platform to ensure practical applicability. Our results show that with P2W and fine-tuning, we improve the model's average accuracy from 37% to 92%.

2 Related Work

This section provides an overview of the most relevant related works, divided into two categories: Power Analysis techniques and Transfer Learning approaches.

Power Analysis: Power analysis techniques have been used to reverse engineer neural networks by analyzing timing, power, and electromagnetic (EM) data to reveal details such as activation functions and network structures [1, 7]. For example, Wang et al. [26] deduced DNN models in in-memory computing systems through power analysis. Other studies [29, 30] collected voltage, current, and EM data to infer model structures, and [5] utilized cache timing side channels for insights into DNN architectures. Multi-step attack frameworks combine these approaches to extract even more information, for example, Gao et al [2] uses a multi-stage power analysis to identify layer types and activation functions, ultimately reconstructing complete DNN models on FPGA-based accelerators.

Our new power analysis technique differs from these by generating an approximated weights matrix directly from power traces, which existing techniques do not provide, as they focus mainly on uncovering the DNN model structure. In contrast, our focus is on transferring as much knowledge as possible, encapsulated in a DNN, via the approximated weights matrix.

Transfer Learning: Transfer learning tackles the challenge of limited training data. It uses data from a related source domain to enhance model performance. This approach works even when feature spaces and data distributions differ from traditional assumptions [28]. It has been applied in text sentiment classification [17], image classification [11], human activity recognition [3], software defect detection [15], and multi-language text classification [32, 31]. Such conventional transfer learning approaches typically require direct access to a fully functional ML model and its parameters and coefficients, which is not always feasible. In contrast, our unconventional P2W transfer learning approach does not require direct access to such parameters and coefficients because it utilizes power traces to extract this information. Thus, P2W expands the applicability of transfer learning to scenarios where direct access to ML models is not feasible.

3 Background

In our P2W approach, we use *Power Analysis*. This method leverages variations in a processing system's power consumption during operation. It extracts internal data and computational states. Initially developed for analyzing cryptographic systems, power analysis has expanded its applications to fields such as neural networks [5].

At the core of such power analysis is the observation that power consumption patterns are data-dependent. The total power consumption $P(t)$ of an embedded SoC at any time t can be expressed as:

$$P(t) = P_{\text{static}} + P_{\text{dynamic}}(t) \tag{1}$$

where P_{static} is the static power consumption of the chip due to leakage currents within the circuit integrated in the chip and other constant factors, and $P_{\text{dynamic}}(t)$ is the dynamic power consumption of the chip, at time t, that depends on the switching activity of the integrated circuit within the chip.

The dynamic power consumption of a chip, implemented using the today's most commonly used CMOS integrated circuit technology, is given by:

$$P_{\text{dynamic}}(t) = \sum_{i=1}^{n} \alpha_i C_i V_{\text{dd}}^2 f \tag{2}$$

where α_i is the switching activity factor, i.e., the probability of a 0-to-1 transition, of the i-th wire within the integrated CMOS circuit, C_i is the load capacitance at the i-th wire, V_{dd} is the supply voltage of the circuit, f is the circuit's clock frequency, and n is the number of switching wires within the circuit at time t. Both α_i and C_i depend on the data being processed by the circuit, thus the power consumption varies with the data due to circuit's switching activity.

In neural networks, computations involve multiply-accumulate (MAC) operations:

$$y = \sum_i w_i x_i + b \tag{3}$$

where w_i are the weights, x_i are the inputs of neurons, and b is the bias. These operations are executed by arithmetic and logic units that are CMOS circuits in an embedded SoC. The bit-level representations of w_i and x_i influence the circuits' switching activity, affecting the dynamic power consumption. Thus, the instantaneous power consumption during a MAC operation can be modeled as:

$$P_{\mathrm{MAC}}(t) = \kappa \cdot \mathrm{HW}(w_i x_i) + n(t) \quad (4)$$

where κ is a proportionality constant, $\mathrm{HW}(w_i x_i)$ is the Hamming weight (number of ones) of the product $w_i x_i$ at time t, and $n(t)$ represents the circuit's noise. Variations in weights w_i affect the product $w_i x_i$ and thus $\mathrm{HW}(w_i x_i)$, leading to observable changes in $P_{\mathrm{MAC}}(t)$. By supplying known inputs x_i and observing the corresponding power consumption $P_{\mathrm{MAC}}(t)$, we can attribute variations in $P_{\mathrm{MAC}}(t)$ to the unknown weights w_i. Therefore, by power analysis, it is possible to find the relationship between operations inside a SoC and the power trace of the SoC, which helps in extracting valuable information about the neural network's internal parameters (e.g., weights) without direct access to the neural network model running inside the SoC.

4 The P2W Approach

In this section, we describe our unconventional transfer learning approach in detail. The approach is based on the following assumptions. First, we assume that we only have a small dataset, D_{small}, which does not contain sufficient data samples to train an ML model from scratch and to achieve high accuracy. Second, we assume that we have an embedded device with a *target SoC* containing a programmable processor core that runs a *target ML model* having the knowledge we want to transfer. We can neither reprogram the target SoC nor look at the target ML model inside. The publicly available information we have about the target model is its type such as, multilayer perceptron (MLP) or convolutional neural network (CNN) and its topology. For example, ARM Cortex-M/A based SoCs are widely used, known for their affordability and low power consumption, making them ideal for embedded machine learning applications [20, 13]. In addition, manufacturers of embedded devices typically deploy ML models from publicly available libraries like TinyML [27], STM32 model zoo [24], and others. Even if this were not the case, as discussed in Section 2, existing studies demonstrate the feasibility of extracting ML model architectures through power analysis, making access to the ML model architecture a realistic assumption. Nevertheless, manufacturers often utilize proprietary training datasets and custom training methods, resulting in proprietary trained ML models.

Also, we assume that we can acquire a clone of the aforementioned target SoC without any ML model inside, and we can easily (re-)program this *clone SoC* and experiment with it. This is feasible because well-established SoC manufacturers provide affordable and easy-to-use HW/SW development kits for their popular SoCs used in many embedded devices. For example, Microchip, STMicro, and others sell evaluation and prototyping boards with SoCs based on ARM Cortex-M/A series of processor cores [14, 23].

Considering the above assumptions, the main goal of P2W is to transfer knowledge from a well-trained target ML model, running inside a target SoC, to a new ML model, establishing a basis for transfer learning, and then fine-tune this new model for another related task using the small dataset D_{small}. We perform this transfer by analyzing power traces obtained from the target SoC running the well-trained ML model, with the help of an `EDNN`. Figure 1 illustrates the overall workflow of our P2W approach that consists of three phases. In Phase 1, we train an `EDNN` model that maps power traces to weights matrices, using the clone SoC. In Phase 2, we capture a power trace from the target SoC running the target ML model and use the trained `EDNN` to obtain an approximated weights matrix. In Phase 3, we initialize a new ML model with the approximated weights and fine-tune it using the small dataset D_{small}.

In the rest of this section, we explain all three phases of our P2W approach in more detail.

4.1 Phase 1: `EDNN` Model Construction

Phase 1 consists of two steps: `Data Preparation` and `Training`. In the `Data Preparation` step, we prepare dataset D for training the `EDNN` model. To create this dataset, we deploy an ML model on the clone SoC and enable it to perform inference. Since we aim to find a relationship between the coefficients within the ML model and the power traces during inference, the accuracy and performance of these ML models are not important. Therefore, we initialize these ML models with random weights and capture power traces while performing inference. Also, from now on, we refer to these ML models as "surrogate ML model".

We input a single random data sample to be processed by the clone SoC running the surrogate ML model and capture the SoC's power consumption from the start to the end of the inference process for this random data sample, thereby obtaining the corresponding power trace. The input data sample selected randomly must remain the same throughout the entire process, in both Phase 1 and Phase 2.

To manage computational complexity, we reduce the length of the power traces using Principal Component Analysis (PCA) [25]. PCA converts a large set of correlated variables, such as millions of samples in the power trace, into a smaller set of uncorrelated variables known as principal components, effectively retaining the most significant features of the original dataset. We deploy the surrogate ML model on the clone SoC multiple times to obtain dataset D. Each time, we use the same random input data sample and a different set of weights y', capturing a new power trace x' during inference. After dataset D is created, the `Training` step starts. As illustrated in Figure 1, we use D to train an `EDNN` model including a convolutional encoder that receives data as input and a convolutional decoder that transforms the encoder's output.

Algorithm 1 details the `Data Preparation` (lines 4-13) for creating dataset D and the `Training` step (line 11) to train the `EDNN` to achieve an accuracy of θ (line 4). The algorithm takes *surML* and *genML*, representing the surrogate model and the `EDNN` model, respectively, both initialized with some initial weights. The other inputs are the required accuracy θ for the `EDNN` and N, the maximum number of times the algorithm is repeated to achieve θ.

In line 5, the surrogate ML model *surML* is initialized with random weights. In line 6, *surML* is deployed to the clone SoC. In line 7, the power trace x' is captured during inference, following the MAC power profile in Eq. (4). In line 9, all coefficients of *surML* are put in the weights matrix y' using $Coefficients\_to\_Matrix()$. In line 10, the pair (x', y') is added to dataset D. In line 11, *genML* is trained with dataset D, and its current accuracy is updated.

The behavior of the function $Coefficients\_to\_Matrix()$, used in line 9, is detailed in lines 15-27. This function takes a surrogate model *surML* as input and returns a matrix y' containing

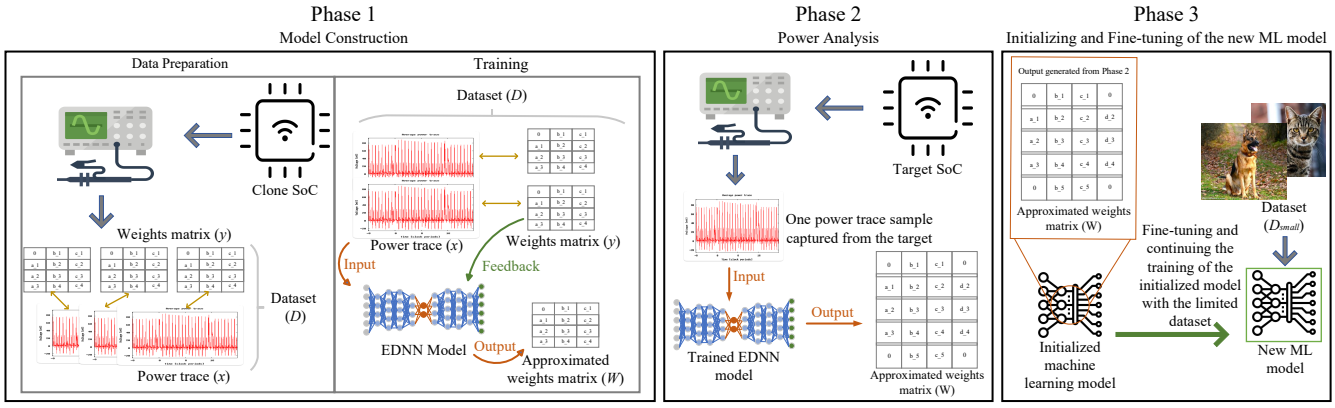


Figure 1: High-level overview of the phases in the P2W transfer learning approach.

Algorithm 1: Dataset Preparation and EDNN training

1 **Input:** $surML, genML, N, \theta$
2 **Output:** $genML$
3 $acc = 0; k = 1;$
4 **while** $acc < \theta \wedge k \leq |N|$ **do**
5 $surML \leftarrow$ `Init`$(surML, random\_generator())$;
6 Deploy $surML$ onto clone SoC;
7 $x' \leftarrow$ Capture power during inference;
8 `// Generating the weights matrix`
9 $y' \leftarrow$ `Coefficients_to_Matrix`$(surML)$;
10 $D \leftarrow D + (x', y')$;
11 $(acc, genML) \leftarrow$ `Train`$(genML, D)$;
12 $k = k + 1$;
13 **end**
14 **return** $genML$;
15 **Function** *Coefficients_to_Matrix(surML)*:
16 $y' \leftarrow \emptyset; j = 0;$
17 **for** $l \in surML.Layers$ **do**
18 **for** $n \in l$ **do**
19 $z = 0;$
20 **for** $c \in n$ **do**
21 $y'[j][z] = c;$
22 $z = z + 1;$
23 **end**
24 $j = j + 1;$
25 **end**
26 **end**
27 **return** y';

all coefficients of $surML$. Within this function, lines 17-26 contain three nested loops. The first loop iterates over each layer l of $surML$. The second loop iterates over each filter/neuron n in layer l. The innermost loop iterates over each coefficient c in filter/neuron n. In line 21, each coefficient c is added to the weights matrix y'. An example of the output matrix y' from the function

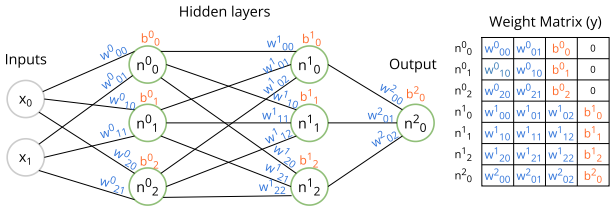


Figure 2: An example of a weights matrix construction.

$Coefficients\_to\_Matrix()$ applied to an MLP model is shown in Figure 2. The MLP model consists of four layers, where the first layer contains 2 inputs, the second and third layers consist of 3 neurons each, and the last one is the output. In the output weights matrix y', each row contains the weights and bias from a specific neuron n^i_j, where i is the layer index and j is the neuron index within that layer.

4.2 Phase 2: EDNN-based Power Analysis

In Phase 2, the goal is to use the trained EDNN to extract knowledge from the target SoC. We capture one power trace x from the target SoC running the target ML model and feed it into the trained EDNN constructed in Phase 1. The EDNN translates power trace x into weights matrix W, whose elements approximate the weights and biases of the target ML model.

4.3 Phase 3: Initialization and Fine-tuning

In Phase 3, we aim to train a new ML model using transfer learning. We initialize the new ML model with the approximated weights matrix W, generated by the EDNN in Phase 2. Following the initialization, we fine-tune the new ML model with dataset D_{small}. Recall that D_{small} does not contain a sufficient number of data samples to successfully train a randomly initialized ML model from scratch and achieve high accuracy. Although the initialization of the new ML model just with W can transfer an acceptable amount of knowledge to it, continuing the improvement of the new ML model by fine-tuning it with the dataset D_{small} helps the new ML model achieve a

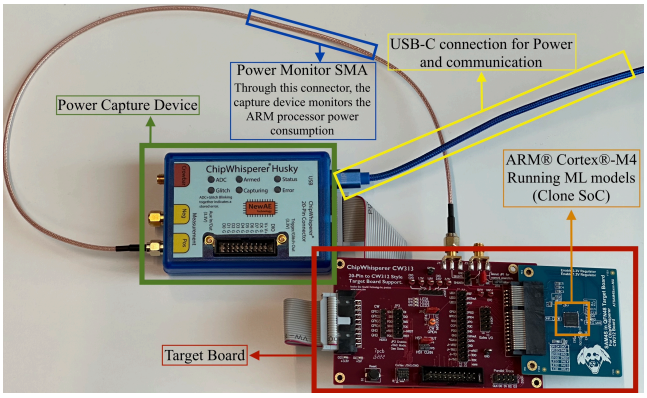


Figure 3: ChipWhisperer Husky Setup for ARM Processor Power Analysis

higher level of accuracy, which can be comparable to the accuracy of the target ML model.

5 Evaluation

In this section, we evaluate our P2W approach. The evaluation focuses particularly on two core aspects: the accuracy of the EDNN models obtained in Phase 1 and used in Phase 2, and the overall performance of the new ML models obtained after the fine-tuning in Phase 3. First, we discuss our experimental setup concerning the surrogate ML models (Phase 1), the EDNN models (Phase 1 and Phase 2), and the datasets D_{small} (Phase 3) in Section 5.1, Section 5.2, and Section 5.3, respectively. This is followed by the evaluation of the P2W approach in Section 5.4, and Section 5.6.

5.1 Surrogate ML Models and Power Trace Capturing

In our experiments, we utilize two surrogate ML models, called Model 1 and Model 2, each given as input *surML* to Algorithm 1. These models are two different fully connected neural networks, i.e., MLP networks with different topologies, that are initialized with random weights in Line 5 of Algorithm 1.

Model 1: This surrogate ML model is designed to perform binary classification. The model has one input layer followed by four fully connected hidden layers. The first hidden layer consists of 128 neurons and employs the ReLU activation function. The subsequent three hidden layers have 64 neurons each. Also, the model has an output layer with a single neuron utilizing the Sigmoid activation function.

Model 2: This surrogate ML model is designed to perform ternary classification. The model is a modified version of Model 1 where the number of neurons in the second hidden layer is increased to 128 and the output layer contains three neurons corresponding to the three classes.

In Lines 6–7 of Algorithm 1, we need to record the power trace while the randomly-weighted surrogate model *surML* runs on the clone SoC. In order to do this measurement, we use the setup, shown in Figure. 3. We use the ChipWhisperer-Husky as the capture unit and a CW313 SAM4S board as the target, with an ARM® Cortex®-M4 power rail sampled at 13 MS/s and 12-bit resolution [16]. All measurements are taken on real hardware in a lab with typical electromagnetic noises, demonstrating that the method works under non-ideal conditions.

As we discussed earlier in Section 4.1 of the proposed method, in order to keep the EDNN input size manageable, we apply PCA to every captured trace and keep only the first $k = 1024$ principal components, where k is the reduced dimension of each trace after PCA. The value of k was chosen once from the knee point of the scree-plot obtained on an independent calibration batch and then fixed for all subsequent experiments.

Finally, the aforementioned repetitive random initialization of surrogate models **Model 1** and **Model 2** with capturing power traces results in one dataset D containing 2000 distinct pairs (x', y') and another one containing 2900 distinct pairs, respectively. Every dataset D is partitioned and used as follows: 75% used for training the EDNN model, 20% dedicated to testing it, and 5% reserved for validation.

5.2 EDNN Models and Training Parameters

We construct and train our EDNN model in Line 11 of Algorithm 1 using PyTorch [18], primarily thanks to its flexibility and performance. The EDNN topology, given as input *genML* to Algorithm 1, consists of two parts: Encoder and Decoder. Recall that the purpose of the trained EDNN is to translate a one-dimensional (1D) power trace x into a two-dimensional (2D) weights matrix W (Section 4.2).

Encoder Topology: The encoder part is designed to process a 1D input array of size 1024 elements, where the array represents a power trace. The encoding part starts with a fully connected layer containing 256 neurons followed by a sequence of three 1D convolutional layers. Each of these layers utilizes a kernel with a size of 4 and a stride of 1. The first convolutional layer contains 256 filters, focusing on extracting mid-level features from the input. It is succeeded by a layer with 128 filters. The final convolutional layer, equipped with 64 filters, completes the feature extraction process. ReLU is used as the activation function in convolutional layers.

Decoder Topology: The decoder part takes as an input the features extracted by the encoder part. The decoder part starts with two sequentially arranged 1D transposed convolutional layers, designed to upscale the input feature maps while enhancing spatial details. The first of these layers consists of 256 filters, employs a kernel with a size of 5 and a stride of 1, and incorporates activation function ReLU for non-linearity. The second transposed convolutional layer, employs a kernel with a size of 4 and a stride of 1, has 128 filters, aiming at refining the feature maps. The feature maps are processed by a linear (fully connected) layer whose number of neurons has to match the size of the 2D output weights matrix. This layer is crucial for reshaping the decoder's output to its final dimensions.

Training parameters: Using the EDNN model topology described above, two different EDNN models *genML* are constructed by executing Algorithm 1 two times, each time with a different surrogate ML model, given as input *surML*, and with an input value of $\theta = 85\%$ and $N = 3500$. The value of N was determined experimentally. Increasing N beyond 3500 did not further improve model accuracy

Table 1: Summary of small datasets D_{small} utilized in PHASE 3 for fine-tuning.

Dataset	Classes	Size	Features	Acc	Ref
EEG	2 classes	22	19	31%	[34]
Diabetes	2 classes	45	9	43%	[22]
Sleep	3 classes	150	13	40%	[12]

in our experiment, whereas using fewer samples led to a noticeable drop in accuracy.

The surrogate ML models **Model 1** and **Model 2**, described in Section 5.1, are used for this purpose. When Algorithm 1 is executed with *surML* = **Model 1**, the EDNN training performed in Line 11 is carried out for 180 epochs and when *surML* = **Model 2** the training is carried out for 210 epochs. A batch size of 100 and a learning rate of 0.001 are employed for all training epochs. We utilize the mean squared error loss function to quantitatively measure the difference between the actual and predicted weights matrices. The Adam optimizer is used during the training thanks to its adaptability and efficiency [4]. To mitigate the risk of overfitting and ensure the model robustness, a dropout mechanism with a rate of 0.5 is applied after each transposed convolutional operation during the training. In our setting, each element $(x', y') \in D$ (Section 5.1) is treated as a supervised training example, where x' is the PCA-compressed power trace and y' is the corresponding weights matrix constructed from the surrogate model coefficients. The EDNN is therefore trained to translate 1D power traces into 2D weights matrices by minimizing the mean-squared error between the predicted and true matrices.

5.3 Datasets D_{small}

In this section, we introduce the datasets D_{small} used for the fine-tuning in PHASE 3. Recall that our P2W approach assumes that D_{small} does not contain a sufficient number of data samples to successfully train a randomly initialized ML model from scratch and to achieve high accuracy. We confirm this by showing and discussing a set of experiments that evaluate the accuracy of new ML models trained with D_{small} only.

Table 1 refers to three datasets (column Ref), where the EEG and Diabetes datasets contain data samples belonging to two classes, and the Sleep dataset contains samples belonging to three classes. It should be noted that these original datasets are relatively large and balanced, which means that within each dataset all classes have (almost) the same number of data samples. In order to perform experiments showing the performance of P2W in scenarios where we do not have access to a sufficient number of data samples, we use small batches of data samples taken from these large datasets. Thus, whenever we refer to D_{small} in our experiments, it means a small balanced batch of samples from these datasets. The number of data samples in the small batches is mentioned in the Size column of Table 1. The Features column indicates the length of each sample in bytes.

To confirm that the number of data samples in the aforementioned datasets D_{small} is not sufficient to train a new ML model from scratch and achieve high (acceptable) accuracy, we take 20 randomly selected D_{small} datasets and one *test dataset* from each

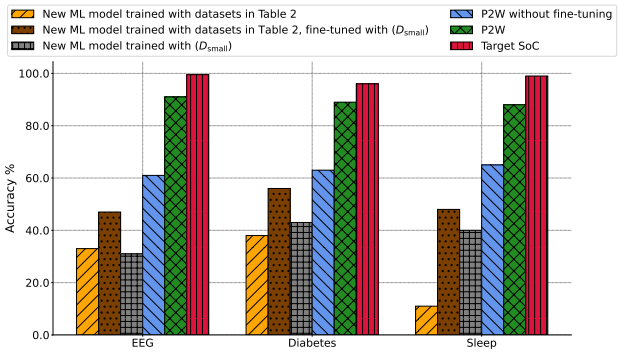


Figure 4: Accuracy of new ML models and target ML models performing EEG, Diabetes, and Sleep classification tasks.

large dataset (EEG, Diabetes, and Sleep). We take randomly initialized ML models that have the same topology as **Model 1** and **Model 2**, described in Section 5.1, and train them using the D_{small} datasets. For example, Model 2 is trained with the 20 randomly selected D_{small} datasets from the large Sleep dataset. After each training round, we check the accuracy of the model with the corresponding *test dataset* and take the average over the 20 rounds. The average accuracy is reported in the Acc column of Table 1 and is below 43% for all datasets D_{small}. Such low accuracy clearly indicates that the number of data samples (column Size) in our datasets D_{small} is not sufficient to train a new ML model from scratch and achieve acceptable accuracy on unseen data.

5.4 Accuracy Evaluation of the EDNN Models

To evaluate the quality of the approximated weights matrices generated by our EDNN models, we apply P2W with all three phases discussed in Section 4, but without fine-tuning in PHASE 3. In PHASE 1, we utilize two surrogate ML models, **Model 1** and **Model 2**, to create two datasets D and to train two separate EDNN models using Algorithm 1. All these actions are described in Section 5.1 and 5.2.

In PHASE 2, which is the power analysis in P2W, we use the two EDNN models and three target SoCs. Each target SoC runs a well-trained target ML model, whose knowledge we are interested to transfer to a new ML model using one of the EDNNs. By experimenting with three target ML models, we evaluate the EDNNs in three different scenarios. In the first two scenarios the well-trained target ML models perform EEG binary classification and Diabetes binary classification, respectively. for both scenarios, the EDNN used in PHASE 2 is the one constructed in PHASE 1 using surrogate **Model 1**. In the third scenario, the well-trained target ML model performs Sleep ternary classification, and the EDNN used is the one constructed in PHASE 1 using surrogate **Model 2**.

Each of the three well-trained target ML models runs on the CW313 SAM4S board which is our target SoC in the three scenarios. This board is part of the same ChipWhisperer platform, introduced in Section 5.1. In PHASE 2, while each of the target ML models is performing inference, we capture one power trace from the target SoC and feed the power trace to the corresponding EDNN. The EDNN generates an approximated weights matrix for the target model.

After the aforementioned Phase 2, we obtain three approximated weights matrices (one for every scenario) and we initialize three new ML models with the matrices in Phase 3. In order to evaluate the quality of the approximated weights matrices generated by the EDNNs, we perform a comparison between the accuracy of the target ML models and the new ML models initialized with the matrices. In this way, we evaluate how much knowledge has been transferred from a target ML model to a new ML model with the goal of performing the same classification task as the target model. For this comparison, we test the accuracy of the target and new models by inferring the data samples in the corresponding *test datasets* described in Section 5.3. These test datasets are unseen by both the target and new ML models.

Figure 4 summarizes the results from our comparison experiment. The horizontal axis shows the three scenarios with the corresponding classification tasks. Each bar represents the accuracy of the models. The gray bars show the average accuracy (column Acc in Table 1) of the new ML models trained only with selected D_{small}. The blue bars show the average accuracy of the new ML models initialized with the approximated weights matrix generated by the EDNNs (P2W without fine-tuning). The red bars show the accuracy of the target ML models running within the target SoC.

As can be seen in Figure 4, the average accuracy of the new ML models trained only with selected D_{small} (gray bars) is too low compared to the target ML models accuracy (red bars). This comparison is also another proof for the fact that datasets D_{small} are too small and insufficient to train a new ML model from scratch. Now, let us analyze the average accuracy of the new ML models initialized with the approximated weights matrices generated by the EDNNs – see the blue bars in Figure 4. These bars show that utilizing weights produced by the EDNNs results in ML models with a notable increase in inference accuracy compared to the gray bars. Although this average accuracy is not as high as the accuracy of the target ML models (the red bars), it indicates that our EDNN-generated weights are non-random and meaningful in the sense that a large amount (more than 60%) of the knowledge in the target ML models has been transferred successfully to the new ML models via the approximated weights matrices. This fact underscores the efficacy of the EDNN models in generating viable approximated weights matrices that could be used as a good starting point to continue the training and fine-tuning of the new ML model to achieve a higher level of accuracy. This will be demonstrated by the experiments described in the next section.

5.5 Performance Comparison to New ML Models Without P2W

For further evaluation of the quality and effectiveness of the approximated weight matrices generated by our EDNN models, we perform a comparison between the accuracy of the new ML models initialized with approximated weight matrices (discussed in Section 5.4) and ML models trained on datasets similar to our target tasks without P2W.

The list of datasets that we used to train the new ML models are available in Table 2. In this table, each dataset is characterized by four main attributes. The first attribute is Size, representing the number of data samples in each dataset. The second attribute is Features, indicating the length of each sample in bytes. The third attribute is Classes, denoting the number of classes associated with each dataset. The last attribute is Chunks, showing the number of chunks, created from each dataset. In addition, Table 2 shows that the datasets are grouped based on their number of classes, thereby forming two groups, where one group consists of datasets associated with 2 classes and the other with 3 classes.

Table 2: Datasets utilized for training similar ML models.

Dataset	Size	Features	Classes	Chunks	Ref
ECG	109k	188	2	100	[8]
Heart Disease	319k	16	2	310	[9]
Blood Pressure	1k	50	3	10	[19]

We perform this evaluation in two different scenarios. In the first scenario, we train three separate ML models only with these similar datasets listed in Table 2, without P2W, and then evaluate their accuracy. For the second scenario, we fine-tune the aforementioned three ML models with the small datasets D_{small} introduced in Section 5.3 and evaluate their accuracy.

In the first scenario, using the three datasets in Table 2, we train three separate ML models as follows: 1) ECG classification, 2) Heart disease detection, and 3) Blood pressure detection. Then, we evaluate the accuracy of each model with the test datasets mentioned in Section 5.3. Specifically, we test the ECG classification model with the EEG test dataset, the Heart disease detection model with the Diabetes test dataset, and the Blood pressure detection model with the Sleep classification test dataset. The yellow bars in Figure 4 show the results of this evaluation. As we can see, for all three target tasks (EEG, Diabetes, Sleep), the accuracy of these models (yellow bars) on our target tasks is lower than that of the new ML models initialized with weight matrices generated by our EDNN models (blue bars). In the Sleep classification scenario, which is a ternary classification task, the result is worse. In this case, the accuracy is even much lower than the model that is only trained with datasets D_{small} (gray bar).

In the second scenario of this evaluation, we fine-tune the aforementioned three ML models (ECG classification, Heart disease detection, Blood pressure detection) with datasets D_{small}, similar to the process in Phase 3 of P2W. The brown bars in Figure 4 show the average accuracy of these three models after the fine-tuning process. As we can see, the results (brown bars) show improvements compared to the yellow bars, but they are still lower than the accuracy of the new ML models initialized with the weight matrices generated by our EDNN models (blue bars). This set of evaluations shows the quality and effectiveness of the weight matrices generated by our EDNN model.

5.6 Performance Evaluation of P2W, Fine-Tuned with D_{small}

As mentioned in Sections 5.4 and 5.5, the initialization of the new ML models with the approximated weights matrices could be a good starting point for further training and fine-tuning in Phase 3 with the final goal of obtaining highly accurate new ML models using our proposed P2W transfer learning approach. To continue with the training and fine-tuning of the initialized ML models, we use

the small datasets D_{small} introduced in Section 5.3. It is important to note that we use these small datasets to highlight the effectiveness of our P2W approach in scenarios with very limited data availability for training.

After training and fine-tuning the initialized ML models with datasets D_{small}, we check the average accuracy of the final new ML models trained with P2W, shown with the green bars in Figure 4, by using the corresponding *test datasets* (Section 5.3). Comparing the green bars with the other bars in the figure, we see that new ML models, trained only with D_{small} (gray bars) or obtained only by initialization with the approximated weights matrices (blue bars) or even trained with a similar dataset and fine-tuned with D_{small} (brown bars) have much lower accuracy compared to the new ML models obtained with our P2W transfer learning approach (green bars). On the other hand, as shown in Figure 4, the accuracy of the new ML models obtained with P2W is still a bit lower than the accuracy of the target ML models (red bars). The reason is twofold: (1) it is practically impossible to transfer 100% of the knowledge from the target ML models to the new ML models via the approximated weights matrices generated from power traces; (2) the training/fine-tuning of the models in Phase 3 is performed with small/limited datasets D_{small} that are assumed to be the only datasets available to the user of our P2W approach.

Nevertheless, the green bars in Figure 4 clearly indicate that new ML models, obtained by P2W, achieve accuracy comparable to the corresponding target ML models. Moreover, in contrast to the target ML models, these new ML models are fully open and available to any user of P2W for further (re-)use, e.g., deployment of the new ML models on different computing platforms and in different application scenarios, fine-tuning them further to perform different tasks, etc.

6 Discussion

In this section, we discuss the limitations and future work concerning our P2W approach.

Limitations: In the first two paragraphs of Section 4, we explain our assumptions regarding P2W together with real-world examples for each assumption to show that P2W is realistic in several scenarios. However, the P2W approach is not very effective in scenarios where some of these assumptions are not true. For example, in order to deploy P2W, access to the power supply of the target SoC is needed, which might not always be the case. Nevertheless, for embedded devices, it is realistic that potential users are in the vicinity of the device and the power supply or even own the device. Moreover, instead of measuring the power consumption, similar results can be achieved by measuring the electromagnetic emanation of the chip, which can be done at a distance from the chip [10].

Another limitation is that the clone SoC needs to be of the exact same type as the target SoC. For broadly used embedded SoCs, it is not a problem to find suitable clone SoCs, as explained in Section 4, but for niche applications or exotic devices, it is more difficult to apply P2W.

Finally, P2W requires that there is prior knowledge on the topology of the target ML model. If this is not the case, the technique cannot be applied. To mitigate this limitation, a preliminary power consumption measurement on the target SoC can be performed in order to learn about the type of ML model that is used, as has been shown in related work [26, 1]. Moreover, for certain applications, it is commonly known which ML topologies are being deployed.

Future work: We plan to study the effects of the complexity of very large ML models within embedded devices on power traces. Although ML models utilized in embedded SoCs are typically not large, the scalability of the P2W approach to very large models could be a valuable path for future research. Additionally, as we evaluated P2W on SoCs with an ARM processor inside, exploring the effectiveness of P2W on different types of hardware, such as FPGAs and GPUs, is a promising direction for future research. Given the parallel processing nature of FPGAs and GPUs, it is beneficial to study how they affect power traces in reflecting the relationships and patterns related to ML model computations.

7 Conclusions

This paper presents P2W, an unconventional transfer learning approach for ML models. Our approach is useful for scenarios where direct access to the coefficients (weights and biases) of existing ML models is not feasible and there is only a relatively small dataset available to train a new ML model from scratch. P2W translates a power trace, captured from an embedded SoC running a target ML model inside, to a weights matrix by utilizing an encoder-decoder deep neural network. This weights matrix approximates the coefficients of the target ML model and is used to initialize a new ML model, thereby performing transfer learning from the target to the new ML model.

Experimenting with relatively small training datasets, we evaluate the effectiveness of our P2W approach by comparing the accuracy of a new ML model trained with and without using P2W.

Our experimental results show that after training the model with our P2W approach, we are able to improve the initial 37% accuracy of the model, trained with only a relatively small dataset, to 92% when P2W is utilized including fine-tuning with the same dataset. These results clearly indicate that new ML models, obtained by P2W, achieve very high predictive accuracy.

Acknowledgments

This work was supported by European Union (NeuroSoC project - Horizon Europe Grant Agreement n°101070634)

References

[1] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2019. Csi nn: reverse engineering of neural network architectures through electromagnetic side channel. In *Proceedings of the 28th USENIX Conference on Security Symposium* (SEC'19). USENIX Association, Santa Clara, CA, USA, 515–532. ISBN: 9781939133069.

[2] Ya Gao, Haocheng Ma, Mingkai Yan, Jiaji He, Yiqiang Zhao, and Yier Jin. 2023. NNLeak: An AI-Oriented DNN Model Extraction Attack through Multi-Stage Side Channel Analysis. In *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. (Dec. 2023), 1–6. Retrieved Apr. 21, 2025 from.

[3] Maayan Harel and Shie Mannor. 2011. Learning from multiple outlooks. In *Proceedings of the 28th International Conference on International Conference on Machine Learning* (ICML'11). Omnipress, Bellevue, Washington, USA, 401–408. ISBN: 9781450306195.

[4] Esraa Hassan, Mahmoud Y. Shams, Noha A. Hikal, and Samir Elmougy. 2023. The effect of choosing optimizer algorithms to improve computer vision tasks: a comparative study. *Multimedia Tools and Applications*, 82, 11, (May 2023), 16591–16633.

[5] Weizhe Hua, Zhiru Zhang, and G. Edward Suh. 2018. Reverse engineering convolutional neural networks through side-channel information leaks. In

Proceedings of the 55th Annual Design Automation Conference (DAC '18) Article 4. Association for Computing Machinery, San Francisco, California, 6 pages. ISBN: 9781450357005.

[6] Amelia Jiménez-Sánchez, Shadi Albarqouni, and Diana Mateus. 2018. Capsule networks against medical imaging data challenges. In *Intravascular Imaging and Computer Assisted Stenting and Large-Scale Annotation of Biomedical Data and Expert Label Synthesis*. Springer International Publishing, Cham, 150–160. ISBN: 978-3-030-01364-6.

[7] Raphaël Joud, Pierre-Alain Moëllic, Simon Pontié, and Jean-Baptiste Rigaud. 2023. Like an open book: read neural network architecture with simple power analysis on 32-bit microcontrollers. In *Smart Card Research and Advanced Applications: 22nd International Conference, CARDIS 2023, Amsterdam, The Netherlands, November 14–16, 2023, Revised Selected Papers*, 256–276.

[8] Mohammad Kachuee, Shayan Fazeli, and Majid Sarrafzadeh. 2018. Ecg heartbeat classification: a deep transferable representation. In *2018 IEEE International Conference on Healthcare Informatics (ICHI)*. IEEE, (June 2018).

[9] Kamil Pytlak. [n. d.] Indicators of Heart Disease (2022 UPDATE). en. (). Retrieved Nov. 30, 2024 from https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease.

[10] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Advances in Cryptology — CRYPTO' 99*. Michael Wiener, (Ed.) Springer Berlin Heidelberg, Berlin, Heidelberg, 388–397. ISBN: 978-3-540-48405-9.

[11] B. Kulis, K. Saenko, and T. Darrell. 2011. What you saw is not what you get: domain adaptation using asymmetric kernel transforms. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition* (CVPR '11). IEEE Computer Society, USA, 1785–1792. ISBN: 9781457703942.

[12] Laksika Tharmalingam. [n. d.] Sleep Health and Lifestyle Dataset. en. (). Retrieved Mar. 29, 2024 from https://www.kaggle.com/datasets/uom190346a/sleep-health-and-lifestyle-dataset.

[13] Ioan Lucan Orăşan, Ciprian Seiculescu, and Cătălin Daniel Căleanu. 2022. A brief review of deep neural network implementations for arm cortex-m processor. *Electronics*, 11, 16. https://www.mdpi.com/2079-9292/11/16/2545.

[14] microchip. [n. d.] Evaluation Boards | Microchip Technology. en. (). Retrieved Jan. 4, 2025 from https://www.microchip.com/en-us/tools-resources/evaluation-boards.

[15] Jaechang Nam et al. 2015. Heterogeneous defect prediction. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (ESEC/FSE 2015). Association for Computing Machinery, Bergamo, Italy, 508–519. ISBN: 9781450336758.

[16] Colin O'flynn and Zhizhang Chen. 2014. Chipwhisperer: an open-source platform for hardware embedded security research. In *Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers 5*. Springer, 243–260.

[17] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22, 10, 1345–1359.

[18] 2019. *Pytorch: an imperative style, high-performance deep learning library. Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 12 pages.

[19] PAVAN BODANKI. [n. d.] Blood Pressure Data for disease Prediction. en. (). Retrieved Feb. 12, 2024 from https://www.kaggle.com/datasets/pavanbodanki/blood-press.

[20] Plumerai. [n. d.] Plumerai wins MLPerf Tiny 1.1 AI benchmark for microcontrollers again | Plumerai Blog. (). Retrieved Nov. 29, 2023 from https://blog.plumerai.com/2023/06/mlperf-tiny-1.1/.

[21] Iqbal H. Sarker, Asif Irshad Khan, Yoosef B. Abushark, and Fawaz Alsolami. 2023. Internet of things (iot) security intelligence: a comprehensive overview, machine learning solutions and research directions. *Mobile Networks and Applications*, 28, 1, (Feb. 2023), 296–312.

[22] J W Smith, J E Everhart, W C Dickson, W C Knowler, and R S Johannes. 1988. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*. Kaggle, 261–265.

[23] STMicroelectronics. [n. d.] STM32 MCU Developer Zone - STMicroelectronics. en. (). Retrieved Jan. 4, 2025 from %7Bhttps://www.st.com/content/st-com/en/stm32-mcu-developer-zone.html%7D.

[24] STMicroelectronics. 2023. STMicroelectronics – STM32 model zoo. original-date: 2023-01-10T13:58:28Z. (Nov. 2023). Retrieved Nov. 29, 2023 from https://github.com/STMicroelectronics/stm32ai-modelzoo.

[25] Alaa Tharwat. 2016. Principal component analysis-a tutorial. *International Journal of Applied Pattern Recognition*, 3, 3, 197–240.

[26] Ziyu Wang, Fan-hsuan Meng, Yongmo Park, Jason K. Eshraghian, and Wei D. Lu. 2023. Side-channel attack analysis on in-memory computing architectures. *IEEE Transactions on Emerging Topics in Computing*, 1–13.

[27] Pete Warden and Daniel Situnayake. 2019. *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media.

[28] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. 2016. A survey of transfer learning. *Journal of Big Data*, 3, 1, (May 2016), 9.

[29] Yun Xiang et al. 2020. Open dnn box by power side-channel attack. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67, 11, 2717–2721.

[30] Kota Yoshida, Takaya Kubota, Mitsuru Shiozaki, and Takeshi Fujino. 2019. Model-extraction attack against fpga-dnn accelerator utilizing correlation electromagnetic analysis. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 318–318.

[31] Joey Zhou, Sinno Pan, Ivor Tsang, and Yan Yan. 2014. Hybrid heterogeneous transfer learning through deep learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28, 1, (June 2014).

[32] Joey Tianyi Zhou, Ivor W. Tsang, Sinno Jialin Pan, and Mingkui Tan. 2019. Multi-class heterogeneous domain adaptation. *Journal of Machine Learning Research*, 20, 57, 1–31.

[33] Zhuangdi Zhu, Kaixiang Lin, Anil K. Jain, and Jiayu Zhou. 2023. Transfer learning in deep reinforcement learning: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45, 11, 13344–13362.

[34] Igor Zyma, Sergii Tukaev, Ivan Seleznov, Ken Kiyono, Anton Popov, Mariia Chernykh, and Oleksii Shpenkov. 2019. Electroencephalograms during mental arithmetic task performance. *Data*, 4, 1.