Estimating Power Consumption of GPU Application using Machine Learning Tool

Gargi Alavani Prabhu, Tanish Desai, Sharvil Potdar, Nayan Gogari, Snehanshu Saha, Santonu Sarkar*
BITS Pilani K. K. Birla Goa Campus, India
{gargia,f20220053,f20220724,f20220116,snehanshus,santonus}@goa.bits-pilani.ac.in

***Abstract*—As Graphic Processing Units (GPU)s play an increasingly important role in High-Performance Computing (HPC) and data-intensive Machine Learning (ML) tasks, accurate power prediction is essential. Traditional methods, relying on architecture-specific models like DVFS and hardware counters, limit cross-architecture applicability of these models. We propose a static analysis framework that predicts an application's power usage across different NVIDIA GPU architectures without execution. Extensive experiments with state-of-the-art ML approaches show promising results, demonstrating generalizability in predicting power consumption for a newer architecture without the need for complete retraining.**[1]

***Index Terms*—GPGPU, Power Prediction, Static analysis, Ensemble learning, Domain Adaptation**

I. Introduction

Rapid advancement and widespread adoption of Graphics Processing Units (GPUs) in fields such as high-performance computing, artificial intelligence, and big data analytics have significantly increased the demand for efficient power management solutions [1], [2]. Consequently, there is a growing need for accurate power prediction tools that can help optimize energy usage without compromising performance [3], [4]. Popoular power and energy management tools use dynamic voltage and frequency scaling (DVFS) [5] and performance counters [6], [7] to accurately measure the amount of energy consumed by a GPU. Since these approaches collect runtime data from the GPU to *measure* the actual power consumption; they are not useful to *predict* the power an application will consume in the future. In this paper, we present a power prediction machine learning model based on static analysis, hardware features, and runtime data using microbenchmarks. Our contributions are as follows. 1. Since there is no publicly available data related to power consumption on GPUs for different types of application, we created a diverse set of benchmark CUDA kernels, including graph traversal, parallel algorithms, and linear algebra functions. As described in Section III we developed a data generation tool (Fig. 1), to collect power and energy data, generating approximately 4,000 datapoints per architecture. We made this tool and the dataset publicly available.[2]

2. We performed a principled data analytic approach to postulate that tree-based ML models are favorable for inference on power consumption in individual GPU architectures(Section IV). Our observation is consistent with the claims made in the seminal paper [8] that, for the low-tabular data regime, tree-based methods outweigh DL-based methods.

3. A key goal of our work is to obtain generalizability (see Section V, where a model trained on the existing GPU architecture should be able to predict power consumption on a newer GPU architecture. This is a challenging problem (owing to distribution shifts, unique power usage patterns, and dynamic power management systems) observed across multiple architectures (Fig.3).

[1]This research is partially supported by the New Faculty Seed Grant of BITS Pilani under Grant No.NFSG/GOA/2023/G0916.

[2]https://anonymous.4open.science/r/GPU-Power-Prediction–2D76/Datasets/

II. Related Work

Several popular studies have explored how DVFS [5], performance counters [9], and runtime application features [3] affect GPU power consumption. Since these approaches use architectural details and runtime data, one needs to actually execute an application on the GPU to estimate the power consumed. Chitakara et al. [9] proposed a multilinear regression to model power consumption using GPU performance counters (available only at runtime). Furthermore, some hardware counters might be unavailable or inconsistent across architecture, affecting the prediction model's accuracy and generalizability across multiple architectures [2].

Another notable work is "AccelWattch" [7] which uses simulation models and hardware counters. Although AccelWattch demonstrates reasonable accuracy, it requires a machine with a GPU to collect the traces of a CUDA kernel. Braun et al. [3] introduced a portable model to predict the execution time and power of the CUDA kernel using the Random Forest algorithm using hardware-independent features. However, these features are extracted through instrumentation by running the kernel.

Static analysis, such as the influence of software design on power consumption, has emerged as a promising technique [10]. Coplin et al. [11] demonstrated how optimizing a CUDA program can enhance energy efficiency. A static analysis-based approach for predicting GPU power consumption is demonstrated in [4]. Some of the features used in this study are similar to those in this work. However, the prediction models reported in [4] were based on a few GPU architectures with a limited dataset, which poses a constraint in the evaluation of ANN-based models.

III. Methodology

A. Overview of the Proposed Tool

The proposed tool uses static analysis techniques, hardware details, and machine learning models to provide accurate power estimates, as shown in Fig. 1. The NVCC compiler is used to compile the GPU application source code into PTX (Parallel Thread Execution) code. The Application Feature Generator module extracts code features relevant to power consumption from the generated PTX code. It also incorporates hardware-specific details and micro benchmarking data to enrich the feature set. This includes data such as the number of SM, memory bandwidth, clock frequency, and other relevant hardware metrics. The microbenchmarks utilized in this study are built using techniques similar to those presented in [12], [13]. The features used in this study, along with the details of feature engineering, are thoroughly described in our previous work [4].

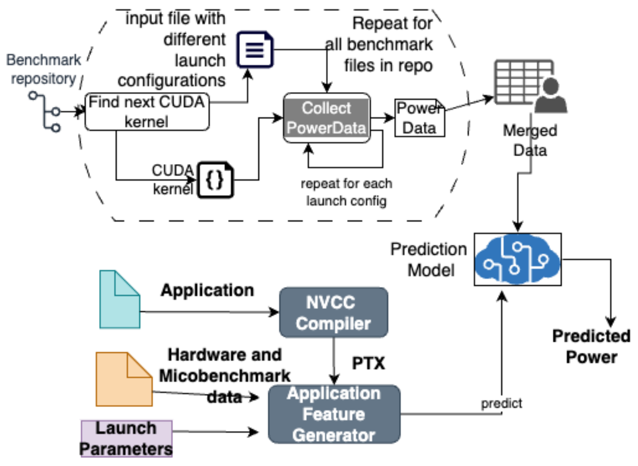
Benchmark repository
input file with different launch configurations
Repeat for all benchmark files in repo
Find next CUDA kernel
Collect PowerData
Power Data
Merged Data
CUDA kernel
repeat for each launch config
Prediction Model
Application
NVCC Compiler
PTX
Hardware and Micobenchmark data
Application Feature Generator
Launch Parameters
predict
Predicted Power

Fig. 1. Overview of Power Prediction Tool

A collection of feature vectors from various applications forms a feature matrix that serves as input to build machine learning models. The feature matrix is used to train machine learning models. These models learn to predict power consumption based on the patterns identified in the feature vectors. Once trained, the models are stored in a repository. Different models can be trained for different types of applications or hardware configurations to enhance prediction accuracy. The new feature vector is input into the trained machine learning models in the repository. The model predicts the power consumption of the application based on its learned patterns.

B. Dataset Collection

We created benchmark applications from the i) Rodinia [14] suite, ii) CUDA SDK [3]. Additionally,

[3]https://docs.nvidia.com/cuda/cuda-samples/index.html

we developed a set of linear algebra-based benchmarks such as relu, softmax, scalar multiplication, scalar addition, matrix determinant, matrix transpose, and a few others. Using our data collection tool shown in Figure 1, we executed these benchmarks on four NVIDIA GPU architectures: Tesla K80 (Kepler), Tesla V100 (Volta), RTX A4000 (Ampere), and RTX 4060 (Ada Lovelace) with various launch configurations and measured the actual power consumption. We used the NVIDIA Management Library (NVML). NVML, a C-based interface, is built for monitoring and managing various hardware states within NVIDIA Quadro and Tesla GPUs. NVML utilizes on-chip sensors for power measurement.

TABLE I
FEATURE IMPORTANCE AND PERCENTAGE SIMILARITY

Features	**Feature Importance**		**Percentage Similarity**
	SHAP	**Random Forest**	
avg_comp_lat	24.77%	29.34%	87.78%
avg_glob_lat	19.06%	21.69%	67.79%
reg_thread	12.34%	9.56%	100%
comp_inst_kernel	7.41%	5.07%	100%
cache_penalty	5.65%	5.55%	67.85%
grid_size	5.59%	3.64%	100%
misc_inst_kernel	4.61%	4.29%	100%
inst_issue_cycles	4.54%	5.42%	93.76%
glob_load_sm	3.85%	2.86%	76.80%
branch_inst	3.46%	4.21%	93.77%
glob_store_sm	3.27%	2.71%	76.99%
block_size	1.99%	2.86%	100%
glob_inst_kernel	1.65%	0.89%	100%
avg_shar_lat	1.08%	1.29%	92.28%
loop_iterations	0.73%	0.63%	100%

IV. BUILDING PREDICTION MODEL

To assess the accuracy of our power predictions, we compared the predicted power against the actual measurements. The dataset for these experiments is generated by combining the data of all four architectures. We conducted a systematic evaluation of classical machine learning and Artificial Neural Networks (ANN), presenting the models based on their design and performance. After observing a non-linear relationship between features and power consumption, along with a very low R-square score (0.25) for multilinear regression, we transitioned to a more complex machine learning algorithm.

Support Vector Regression (SVR) with Radial Basis Function (RBF) kernel did not yield impressive results as presented in Table II. ANNs [15] are highly favored for modeling complex, non-linear relationships within data, particularly in predictive tasks. We carried out extensive experiments with ANN, adjusting its architecture, dataset size, and learning rates to optimize performance.

A. ANN-Based Investigations

We conducted 5-fold cross-validation using ANNs with 3 hidden layers with fixed learning rate, exponential decay, and inverse time decay learning rates (Table II). The fixed learning rate was the one that performed the best among all three rates. It was also observed that a network with 256, 256, 128 layers and a 0.01 initial learning rate yielded better results than other combinations of layers and learning rates. From the loss function plots in Figs. 2 a and b, we observe that the training plots do not exhibit uniform smoothness and monotonicity, suggesting instability during training. Loss decreases rapidly initially and then stabilizes. These observations suggest that ANN may not be the most optimal approach for this work [8].

B. Other Machine Learning Approaches

We moved to tree-based algorithms beginning with Decision Trees which showed significant improvement over SVR, and results close to ANN. This led us to try ensemble approaches that have proved to be effective in tabular data [8]. Ensemble approaches improve performance and reduce variance using multiple models and their combined predictions [16]. Numerous approaches to ensemble learning exist in the literature [16] of which we experimented with tree-based ensemble algorithms such as Random Forest, Gradient Boost, Extra Trees, XGBoost and CatBoost. We have presented the results of all the methods for five runs, to emphasize that the results are consistent across different random test and train splits. As shown in Table II, we applied the Bonferroni method over five algorithm runs (only the best run is presented due to space constraints), we conclude that XGBoost, Extra Trees, Random Forest, and CatBoost significantly outperform the other algorithms. We plotted the loss for CatBoost and XGBoost (Fig.2), which demonstrates that tree-based ensemble methods have a smooth, monotonically decaying loss as

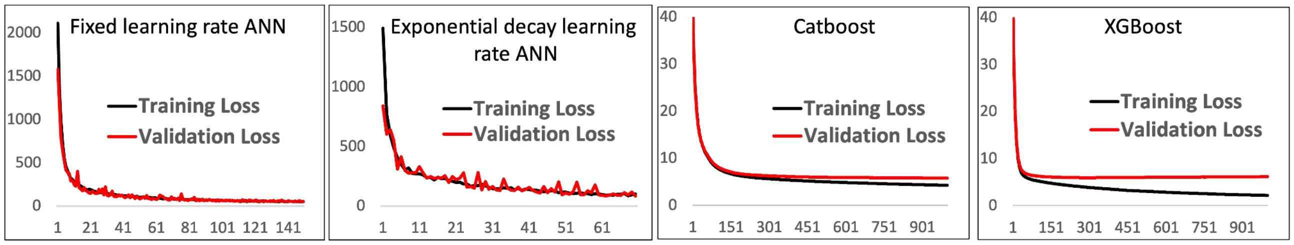

Fig. 2. Loss function plots for 4 models. X-axis is the number of epochs and Y-axis is the loss value

opposed to what was observed in ANN. Both the model show strong and stable performance.

TABLE II
MODEL PERFORMANCE METRICS ON COMBINE DATASET EXPERIMENTS

Model	RMSE	MSE	MAE	R^2
Catboost	5.91	34.99	3.89	0.9823
XGBoost	6.06	37.23	3.84	0.9820
Gradient Boost	11.27	122.62	8.20	0.9374
Extra Tree	6.39	39.69	3.59	0.9801
Decision Trees	7.63	59.86	4.16	0.9701
SVR	34.36	1180.84	22.08	0.4062
Random Forest	6.03	36.44	3.58	0.9820
ANN learning rate (0.001)	6.91	47.89	4.58	0.9759
ANN Exponential rate (0.01)	9.11	83.09	5.97	0.9581
ANN Inverse rate (0.01)	9.80	96.10	6.19	0.9516

V. TOWARDS GENERALIZABILTY

An important objective of this work is to build a generalizable model, constructed using data from older GPU architectures, that can predict power consumption on a newer architecture, without having to retrain the model again.

A. Examining Suitability of Current Model

To understand generalizability, we examined the model performance and noticed that it is possible to build a good predictive model (Section IV and Table II) when the train and test data are from the old and new architectures. However,

1 When we trained using datasets from older architecture(Tesla/Ampere and K80) and tested on newer architecture(Ada Lovelace), the results were significantly poor as seen in Table III.

2) When we employed AutoML [17] using the Keras Tuner library to optimize the hyperparameters and architectures of ANN models automatically, our results were still less than satisfactory with a very low R^2 (-18.19).

Thus, we conclude that the current model is *not naturally generalizable* and it is necessary to explore techniques such as transfer learning to achieve generalizability.

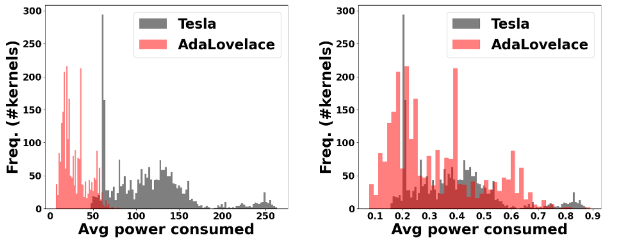

Fig. 3. Histogram: Left is without scaling, Right is after scaling

After a thorough investigation, we discovered that the poor performance of the older models in this experiment was attributed to a domain change in the data set caused by changes in architecture. The target variable, the average power, changes scale with different architectures, illustrated in the histograms of power for both Tesla and Ada Lovelace in Fig. 3. The first histogram is plotted with unscaled data. To assess the similarity between both distributions, we employed Earth Mover's Distance metric [18]. Without scaling the average power values, the similarity between the distributions was only 60.20%. However, after scaling the average power values by the maximum power at which each architecture can run, the similarity increased to 91.74%, as shown in the right histogram in Fig. 3 both distributions are almost overlapping. This indicates a strong need to develop models that can adapt to domain shifts. Next, we investigate domain adaptation techniques [19], [20] to overcome these limitations and enhance model generalization across different GPU architectures.

B. Domain Adaptation Approach

We propose a special type of domain adaptation technique that employs ANN due to their ability to model complex relationships in the data and their adaptability to different types of input features. Here

TABLE III
MODEL PERFORMANCE METRICS FOR GENERALIZABILITY

Model	**RMSE**	**MSE**	**MAE**	R^2
Gradient Boosting	57.56	3313.05	53.77	-15.8246
Extra Trees	61.13	3736.57	57.81	-17.9753
Random Forest	58.55	3427.81	55.79	-16.4074
Decision Tree	60.28	3634.04	53.24	-17.4547
XGBoost	67.25	4523.00	62.16	-21.9691
CatBoost	56.42	3182.96	54.02	-15.1639
SVR RBF	58.14	3380.43	55.24	-16.1668

we trained ANNs to learn features that are invariant across different GPU architectures, followed by fine-tuning them on the target GPU architecture.

Step 1: The model is trained using data from the three older architectures (source domain), excluding the newer architecture (Ada Lovelace, the target domain). Since power consumption data shows insignificant changes in the input feature distribution (Table I) but evidence of a non-negligible shift in the output distribution (power consumption) is observed (Fig. 3), our approach becomes suitable.

Let dataset $D_s = (x_i^s, y_i^s)_{i=1}^{N_s}$ where x_i^s represents the feature vector, and y_i^s is the target variable (power consumption) for the i-th CUDA kernel sample of older architectures. N_s is the number of samples in the source domain. The objective is to learn a function $f(x;\theta)$ that minimizes the loss $\mathcal{L}$ over the source domain as follows:

$$\theta^* = \min_{\theta} \frac{1}{N_s} \sum_{i=1}^{N_s} \mathcal{L}(f(\mathbf{x}_i^s;\theta), y_i^s) \tag{1}$$

where θ represents the parameters of the model. This pretraining is performed for 150 epochs with a batch size of 32.

Step 2: Once the data is trained on the older architecture, the next step is to freeze layers to preserve the learned features. The model f consists of three layers $f(x;\theta) = g(h(x;\theta_1);\theta_2)$, where $h(x;\theta_1)$ represents the first two layers with parameter θ_1 and $g(x;\theta_2)$ represents the third layer with parameter θ_2. After the initial training, we froze the first two layers i.e. θ_1 to retain the learned features from the source domain.

Step 3: Once the layers are frozen, the model is fine-tuned in the target domain, that is, a new architecture. Let $\mathcal{D}_t = \{(\mathbf{x}_i^t, y_i^t)\}_{i=1}^{N_t}$ be the dataset from the target domain, where N_t is the number of samples in the target domain. We divide $\mathcal{D}_t$ into a small subset $\mathcal{D}_t^{\text{train}}$ for fine-tuning (5% of $\mathcal{D}_t$) and $\mathcal{D}_t^{\text{test}}$ for evaluation (95% of $\mathcal{D}_t$). During fine-tuning, we adjust only the parameters θ_2, keeping θ_1 fixed. The fine-tuning is performed for 100 epochs with a batch size of 32 and a learning rate of 0.01. N_t^{train} is the number of samples in the fine-tuning subset of the dataset. Similar to Equation 1, the loss is minimised as follows:

$$\theta_2^* = \min_{\theta_2} \frac{1}{N_t^{\text{train}}} \sum_{i=1}^{N_t^{\text{train}}} \mathcal{L}(g(h(\mathbf{x}_i^t;\theta_1);\theta_2), y_i^t) \tag{2}$$

Step 4: After fine-tuning the subset of the new architecture, we then evaluated this model on the remaining 95% of the dataset $\mathcal{D}_t^{\text{test}}$. The resultant performance for N_t^{test} (number of samples in the test Ada Lovelace subset) is calculated as follows:

$$\text{Result} = \frac{1}{N_t^{\text{test}}} \sum_{i=1}^{N_t^{\text{test}}} \mathcal{L}(g(h(\mathbf{x}_i^t;\theta_1);\theta_2^*), y_i^t) \tag{3}$$

The results of this domain adaptation approach are promising as seen in Table IV. These promising initial results provide a strong foundation for future work using domain adaptation.

TABLE IV
DOMAIN ADAPTATION MODEL PERFORMANCE METRICS

Architecture	RMSE	MSE	MAE	R-square
(32,32,64)	6.81	46.44	4.51	0.7641
	5.92	34.99	4.12	0.8223
	6.25	39.07	4.33	0.8015
	7.57	57.27	5.67	0.7091
	6.31	39.83	4.25	0.7976

VI. CONCLUSION AND FUTURE DIRECTIONS

In this paper, we present a machine learning-based tool that analyzes a CUDA kernel and predicts its power consumption without the need to execute it on a NVIDIA GPU. A major challenge in doing so is that each GPU architecture has unique traits, such as memory hierarchy, core configurations, and power management, which complicates the creation of a generalized prediction model. In the future, it is necessary to adopt a hybrid approach that combines dynamic and static analysis can enhance precision by capturing these differences.

The next challenge is the nonavailability of any dataset or benchmark. We developed a data collection tool to generate a reasonable corpus from four GPU architectures to build the model. We have made the data and the collection tool publicly available. However, the dataset is still limited. In the future, we have to expand it, especially to employ ANNs like solutions for domain adaptation.

We have reported several important observations, such as distribution shifts, unique power usage patterns, and dynamic power management, that significantly impact model performance. We demonstrated that domain adaptation is one of the practical approaches to make the model generalizable. We first showed that CatBoost and XGBoost ensemble learning performs the best among other tree-based approaches, SVR and ANN when we combine data from all GPU architectures. Then we demonstrated that all of them including the best-performing ones falter when we train it on older architectures and test on newer architecture. Lastly, we showed that the domain adaption approach significantly improves the performance with respect to generalization. Future work includes improving prediction performance, purely from the perspective of addressing generalizability challenges.

REFERENCES

[1] D. Zhao, S. Samsi, J. McDonald, B. Li, D. Bestor, M. Jones, D. Tiwari, and V. Gadepally, "Sustainable supercomputing for ai: Gpu power capping at hpc scale," in *Proceedings of the 2023 ACM Symposium on Cloud Computing*, 2023, pp. 588–596.

[2] R. A. Bridges, N. Imam, and T. M. Mintz, "Understanding gpu power: A survey of profiling, modeling, and simulation methods," *ACM Computing Surveys*, vol. 49, pp. 41:1–41:27, 2016.

[3] L. Braun, S. Nikas, C. Song, V. Heuveline, and H. Fröning, "A simple model for portable and fast prediction of execution time and power consumption of gpu kernels," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 1, pp. 1–25, 2020.

[4] G. Alavani, J. Desai, S. Saha, and S. Sarkar, "Program analysis and machine learning–based approach to predict power consumption of cuda kernel," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 8, no. 4, 2023.

[5] S. K. Mandal, G. Bhat, J. R. Doppa, P. P. Pande, and U. Y. Ogras, "An energy-aware online learning framework for resource management in heterogeneous platforms," *ACM Transactions on Design Automation of Electronic Systems*, vol. 25, no. 3, 2020.

[6] G. Ali, M. Side, S. Bhalachandra, N. J. Wright, and Y. Chen, "Performance-aware energy-efficient gpu frequency selection using dnn-based models," in *Proceedings of the 52nd International Conference on Parallel Processing*, 2023, pp. 433–442.

[7] V. Kandiah, S. Peverelle, M. Khairy, J. Pan, A. Manjunath, T. G. Rogers, T. M. Aamodt, and N. Hardavellas, "Accelwattch: A power modeling framework for modern gpus," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2021, pp. 738–753.

[8] R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," *Information Fusion*, vol. 81, pp. 84–90, 2022.

[9] Y. Chitkara, "A review on statistical power modelling for a graphics processing unit (gpu)," in *2022 Sixth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*. IEEE, 2022, pp. 327–330.

[10] K. Kandalla, E. P. Mancini, S. Sur, and D. K. Panda, "Designing power-aware collective communication algorithms for infiniband clusters," in *2010 39th International Conference on Parallel Processing*. IEEE, 2010.

[11] J. Coplin and M. Burtscher, "Effects of source-code optimizations on gpu performance and energy consumption," in *Proceedings of the 8th Workshop on General Purpose Processing using GPUs*. ACM, 2015.

[12] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying gpu microarchitecture through microbenchmarking," in *International Symposium on Performance Analysis of Systems & Software*. IEEE, 2010.

[13] G. Alavani and S. Sarkar, "Inspect-gpu: A software to evaluate performance characteristics of cuda kernels using microbenchmarks and regression models," in *International Conference on Software Technologies*. Rome - Italy: SCITEPRESS, July 2023.

[14] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization*. IEEE, 2009.

[15] T. Hill, L. Marquez, M. O'Connor, and W. Remus, "Artificial neural network models for forecasting and decision making," *International journal of forecasting*, vol. 10, pp. 5–15, 1994.

[16] I. D. Mienye and Y. Sun, "A survey of ensemble learning: Concepts, algorithms, applications, and prospects," *IEEE Access*, vol. 10, 2022.

[17] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, 2021.

[18] L. J. G. Yossi Rubner, Carlo Tomasi, "The Earth Mover's Distance as a Metric for Image Retrieval," *International Journal of Computer Vision*, vol. 2, 2000.

[19] M. Long, Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks," in *International conference on machine learning*, 2015.

[20] A. De Mathelin, G. Richard, F. Deheeger, M. Mougeot, and N. Vayatis, "Adversarial weighting for domain adaptation in regression," in *IEEE 33rd International Conference on Tools with Artificial Intelligence*, 2021, pp. 49–56.