2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD) | 978-1-6654-3166-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/MLCAD52597.2021.9531299

Feeding Hungry Models Less: Deep Transfer Learning for Embedded Memory PPA Models

Special Session

Felix Last
Technical University of Munich and Intel Germany
mail@felixlast.de

Ulf Schlichtmann
Technical University of Munich
ulf.schlichtmann@tum.de

***Abstract*—Supervised machine learning requires large amounts of labeled data for training. In power, performance and area (PPA) estimation of embedded memories, every new memory compiler version is considered independently of previous versions. Since the data of different memory compilers originate from similar domains, transfer learning may reduce the amount of supervised data required by pre-training PPA estimation neural networks on related domains. We show that provisioning times of PPA models for new compiler versions can be reduced significantly by exploiting similarities across versions and technology nodes. Through transfer learning, we shorten the time to provision PPA models for new compiler versions by 50% to 90%, which speeds up time-critical periods of the design cycle. This is achieved by requiring less than 6,500 ground truth samples for the target compiler to achieve average estimation errors of 0.35% instead of 13,000 samples. Using only 1,300 samples is sufficient to achieve an almost worst-case (98th percentile) error of approximately 3% and allows us to shorten model provisioning times from over 40 days to less than one week.**

***Index Terms*—electronic design automation, machine learning, memory compilers, regression, artificial neural networks, transfer learning, few-shot learning, deep learning**

I. Introduction

The relative impact of memories on the overall power, performance, and area (PPA) of systems-on-chip (SoCs) continues to increase. Due to the growing number of memory compiler parameters, the task of tuning these parameters to fulfill design requirements while achieving optimal PPA is highly complex. Efficient design space exploration requires rapid estimation which memory compilers cannot provide due to long run times. As previously shown in [1], supervised neural networks may be trained to estimate PPA quickly and accurately, independently of technology node and compiler vendor. However, the models' data hunger, i.e. the amount of training data required to attain satisfactory performance, is immense: for neural networks to model a release of ten compilers with high accuracy, datasheets for around 100,000 memories need to be compiled, which takes more than 40 days. Aside from high computational costs, this leads to a significant delay between the release of a new compiler version and the provisioning of PPA models which enable design optimization. With fast-paced design cycles and frequently released memory compiler versions, it is the goal of this work to explore ways in which the provisioning time of memory compiler PPA models can be shortened while retaining high accuracy.

Rather than considering model training for each compiler version as an isolated task, we propose leveraging the knowledge learned from related tasks, such as previous compiler versions as well as compilers of different technology nodes. A branch of meta-learning, the idea of transfer learning describes the transfer of knowledge acquired for a "source domain" to a different but similar "target domain" [2], [3]. The focus on model performance on a single target domain distinguishes transfer learning from multi-task learning, where a single model performs multiple tasks at once [3]–[5].

The potential benefits of transfer learning include a better initial model, less time (or data) required to achieve the same performance as well as better final model performance and improved generalization [3], [6]. On the other hand, dissimilar domains and other issues may cause transfer learning to degrade performance on the target domain, a problem which is referred to as "negative" transfer learning [3].

The remainder of this article is organized as follows: In Section II, we define and formalize the task of transfer learning for memory compiler PPA estimation. Section III describes related research from the fields of machine learning and electronic design automation. In Section IV, we propose to use parameter transfer by pre-training networks on data from other compilers and compiler versions before fine-tuning them on the target domain. We present our experiments and results in Section V before concluding this work in Section VI.

II. Problem Definition

Embedded memories are ubiquitous in circuits, with modern chip products using between 500 and 10,000 such memories. The regular structure of bit cell arrays facilitates scaling and design automation. These factors led to the development of memory compilers, which generate memories for a large range of sizes according to parameters provided by the designer. Foundries and electronic design automation (EDA) vendors provide compilers for different types of memories, such as

static random access memories (SRAMs), read-only memories (ROMs), and register files (RFs), as well as with different PPA focus, such as high density or high speed. Compilers are often silicon-verified and characterized for several PPA metrics, e.g. area, timing, and power. Due to the breadth of products and usage profiles, vendors aim to characterize PPA for a large and growing number of process, voltage, and temperature (PVT) corners. While compiler run times may be no more than 30 minutes per memory when only a single product's PVT sign-off corners (i.e. less than five) are required, requesting PPA for many PVT corners significantly increases compiler run times. Because our models are used in the design of many different products, we require a broad range of PVT corners. We assume a compiler run time of one hour per memory, which is a conservative lower bound.

Memory compiler parameters include system parameters, which affect the interface of the memory (e.g. number of words, number of bits), and architectural parameters (e.g. number of banks, column multiplexers, periphery voltage), which affect only PPA. Architectural parameters can thus not be derived directly from system requirements and are not known a priori, but must be tuned to achieve optimal PPA trade-offs for the product's usage profile. Even when system parameters are fixed, there are still 100 to 1,000 possible combinations of architectural parameter values to choose from for a single memory. Given the large number of memories on a product, comparing even a few options for each by obtaining PPA from compiler runs is expensive. To explore the design space efficiently, we employ models which can rapidly estimate PPA for any combination of compiler parameters.

Collecting training data for supervised PPA models requires memory compilers to be run for different parameterizations. Modern memory compilers can generate memories for more than 100 million parameter combinations (system and architectural), so covering even a small portion of their input parameter space is challenging. For reliable optimization, low estimation error (i.e. high accuracy) is required. To reach estimation errors below 5% in 98% of the cases (i.e. almost worst-case), we observed that training data sets of approximately 10,000 memories are required, depending on compiler complexity. Assume that the generation of memories with many PVT corners takes one hour and resources allow parallelization of up to 100 compiler runs. Then, generating the ground truth data required to train a PPA model for a single compiler takes more than four days (100 hours). In practice, memory compiler releases are often bundled, so that ten or more compilers are released at the same time. With the same resources, generating sufficient amounts of data for all compilers of a bundle thus takes approximately 40 days (1000 hours).

Note that these estimates of PPA model provisioning time assume substantial CPU and RAM resources as well as seamless automation of the full data generation process. Furthermore, model training, data quality inspection, and tasks surrounding the integration of compilers from different vendors are ignored by this calculation. Given the frequent release of new compilers by different vendors for various technology nodes, resources are clearly pushed to the limit. Moreover, designers need to create and test their designs. Optimizing and estimating PPA for the latest compiler versions soon after their release thus is crucial for efficient design cycles.

The memory compiler PPA prediction problem is formalized as follows:

$$\underset{\hat{y}}{\operatorname{argmin}} \sum_{x \in X_{train}} L(y(x), \hat{y}(x)) \tag{1}$$

where X_{train} is the set of memories used for training, and y represents PPA as a function of memory x. Function $\hat{y}(x)$ approximates $y(x)$ and is described by the weights and biases of a feed-forward neural network. L is the loss function used to express the deviation of $\hat{y}(x)$ from $y(x)$.

In order to reduce model provisioning time, the most effective lever is the amount of training data generated. Let X_{train_t} be a reduced subset of X_{train}. Then the minimization problem given in equation 1 remains, but we add the following constraints:

$$\begin{aligned} |X_{train_t}| - |X_{train}| &< 0 \\ \sum_{x \in X_{train_t}} L(y(x), \hat{y}_t(x)) - \sum_{x \in X_{train}} L(y(x), \hat{y}(x)) &\leq 0 \end{aligned} \tag{2}$$

where the first constraint represents the reduced amount of training data for the target domain t, and the second constraint represents our goal to maintain or reduce the estimation error L when training on the subset.

We focus on transfer learning between compilers where the input parameters, i.e. the model's features, are structurally identical. Likewise, the structure of PPA variables, i.e. the model's target variables, is assumed to be identical. In general, the domains (compilers) are assumed to be highly similar. The direction of correlation between certain inputs and outputs is expected to be consistent. For example, the number of banks increases memory area while decreasing cycle time. This correlation can be observed independently of the compiler version or technology node. However, the value distribution and the slope of the correlation may naturally differ. Another potential difference between domains is which PVT corners are available; newer compiler versions often expand the set of characterized PVT corners.

III. Related Work

Transfer learning is applied across application domains using various approaches. Inductive transfer learning is the branch of transfer learning which addresses supervised learning problems. Within inductive transfer learning, the authors of [4] distinguish four types of techniques: Relational knowledge transfer is applied to data with interrelations. Instance-based transfer re-uses source domain data during target domain training; often, misleading source domain samples are identified and removed beforehand. Feature-representation-transfer transforms the target domain features to match the distribution of the source domain, so that an existing model for the source domain may be used to obtain target domain estimates. Lastly, parameter transfer copies model parameters learned on the

source domain to another model, which is then trained (fine-tuned) on the target domain. Parameter transfer is commonly referred to as pre-training in the context of neural networks.

In deep learning, pre-trained models are the most common approach [5] with especially widespread use in the domains of image recognition, natural language processing, and character recognition (e.g. [7]–[9]). In the realm of regression tasks solved with neural networks, time-series prediction prevails as the most common task for which new approaches are frequently published (e.g. [10], [11]). However, all of the aforementioned works usually address specialized network architectures for the respective domain. Therefore, their results cannot be assumed to apply directly to PPA regression.

The authors of [12] cite several cross-domain machine learning approaches for EDA, most of which are based on reinforcement learning. However, not all mentioned "cross-domain" approaches are transfer learning approaches. [13] estimate the performance of applications on different field-programmable gate array (FPGA) platforms to circumvent the long run time of high-level synthesis (HLS) tools. They use a single model to predict different domains (i.e. devices), which is enabled by feeding properties of the target domain which are known a priori. No such a priori properties are available in the case of memory compilers. On the other hand, [14] aim to tune the parameters of FPGA HLS. To this end, they train neural networks to estimate the area and performance of different benchmark applications which are synthesized to FPGA designs. Their work is a successful application of pre-training for the transfer of knowledge gained from one application to another. An interesting aspect is that the feature space differs between applications, where different HLS parameters are to be tuned. Lastly, [15] successfully apply transfer learning to estimate delay and area of synthesis flows, transferring knowledge between different technology nodes and circuit designs. The modeling of the synthesis flows is done using the time-series technique of long short-term memory (LSTM) networks. Although the results of transferring knowledge between technology nodes are encouraging for our work, these findings cannot be applied directly given the recurrent network architecture. To the best of our knowledge, transferring knowledge from previous compiler versions or other technology nodes to facilitate the estimation of embedded memory PPA has not been proposed previously.

IV. Proposed Solution

We propose applying transfer learning through two steps: In the first step, a base network is trained on the source domains, which may consist of a single or multiple memory compilers from previous versions. Second, the learned network parameters are transferred to a second network which is trained or fine-tuned on a target domain dataset. In many works, only some layers are re-trained while parameters on the other layers remain fixed. The stated goal of such approaches is to save computational resources, which is relevant for the extremely large and deep networks common in some domains. It has been shown by [9] that fine-tuning more layers consistently leads to a lower estimation error in the target domain, which is why we opt to fully retrain the network. Our method is described in Algorithm 1.

Algorithm 1 Transfer learning for memory PPA estimation

function TRANSFERLEARN(X_s, Y_s, X_t, Y_t) ▷ s(ource), t(arget)
  $n_X \leftarrow$ countFeatures(X_s)
  $i \leftarrow 0$
  while $i < n_X$ **do** ▷ normalize each feature
    $x_{min} \leftarrow min(\{min(X_{s,*,i}), min(X_{t,*,i})\})$
    $x_{max} \leftarrow max(\{max(X_{s,*,i}), max(X_{t,*,i})\})$
    $X_{source,i} \leftarrow \frac{X_{s,i} - x_{min}}{x_{max} - x_{min}}$
    $X_{t,i} \leftarrow \frac{X_{t,i} - x_{min}}{x_{max} - x_{min}}$
    $i \leftarrow i + 1$
  end while
  $n_Y \leftarrow$ countFeatures(Y_s)
  $i \leftarrow 0$
  while $i < n$ **do** ▷ normalize each target variable
    $y_{min} \leftarrow min(\{min(Y_{s,i}), min(Y_{t,i})\})$
    $y_{max} \leftarrow max(\{max(Y_{s,i}), max(Y_{t,i})\})$
    $Y_{s,i} \leftarrow \frac{Y_{s,i} - y_{min}}{y_{max} - y_{min}}$
    $Y_{t,i} \leftarrow \frac{Y_{t,i} - y_{min}}{y_{max} - y_{min}}$
    $i \leftarrow i + 1$
  end while
  $m \leftarrow$ initializeModelParameters()
  $X_{\text{train}}, X_{\text{val}} \leftarrow$ splitData(X_s)
  $Y_{\text{train}}, Y_{\text{val}} \leftarrow$ splitData(Y_s)
  $m_{X_{\text{train}}} \leftarrow$ countObservations(X_{train})
  $m_{X_{\text{val}}} \leftarrow$ countObservations(X_{val})
  converged $\leftarrow 0$
  while converged $\neq 1$ **do** ▷ pre-train
    $e_{\text{train}} \leftarrow \sum_{j \in \{0,\ldots,m_{X_{\text{train}}}\}} \text{loss}(m(X_{\text{train},j,*}), Y_{\text{train},j,*})$
    $m \leftarrow$ backpropagate($\nabla e_{\text{train}}, m$)
    $e_{\text{val}} \leftarrow \sum_{j \in \{0,\ldots,m_{X_{\text{val}}}\}} \text{loss}(m(X_{\text{val},j,*}), Y_{\text{val},j,*})$
    converged $\leftarrow$ hasNotImproved(e_{val})
  end while
  $X_{\text{train}}, X_{\text{val}} \leftarrow$ splitData(X_t)
  $Y_{\text{train}}, Y_{\text{val}} \leftarrow$ splitData(Y_t)
  $m_{X_{\text{train}}} \leftarrow$ countObservations(X_{train})
  $m_{X_{\text{val}}} \leftarrow$ countObservations(X_{val})
  converged $\leftarrow 0$
  while converged $\neq 1$ **do** ▷ fine-tune
    $e_{\text{train}} \leftarrow \sum_{j \in \{0,\ldots,m_{X_{\text{train}}}\}} \text{loss}(m(X_{\text{train},j,*}), Y_{\text{train},j,*})$
    $m \leftarrow$ backpropagate($\nabla e_{\text{train}}, m$)
    $e_{\text{val}} \leftarrow \sum_{j \in \{0,\ldots,m_{X_{\text{val}}}\}} \text{loss}(m(X_{\text{val},j,*}), Y_{\text{val},j,*})$
    converged $\leftarrow$ hasNotImproved(e_{val})
  end while
  return m
end function

As features, all compiler parameters which affect PPA are used. Additionally, process, voltage, and temperature are included as features, which has the advantage of easing transfer between compilers which were characterized for different PVT

corners and enables the model to exploit their numerical values for more accurate predictions [16]. The target variables consist of five PVT-dependent PPA variables dynamic power (read and write), static power (standby leakage), timing (access and cycle time). PVT-independent variables, such as area and aspect ratio, are excluded from our analysis for simplicity. In practice, a separate model would be trained for these variables. Features are standardized to zero mean and unit variance, whereas targets are preprocessed by applying the natural logarithm (which yields a more normal distribution for static power) and min-max scaling into the range $[0, 1]$.

The minima and maxima for scaling of the target variables may be determined either individually for source and target domains, or jointly. Our results indicate that joint scaling is preferred. Note that when the full source domain dataset is unavailable, minima and maxima of the source domain data can be re-used to scale the target domain data. These are typically stored with pre-trained models in order to post-process estimates, i.e. transform estimates to original scale.

The network structure used in this work follows a partial sharing architecture with six layers. The first half of the layers is shared among all targets, whereas the latter half consists of non-shared weights and biases. This architecture allows the exploitation of commonalities (target-target correlations) through the shared layers and specific feature-target correlations through non-shared layers [17]. Each layer consists of 340 rectified linear unit (ReLU) units, where non-shared layers have 340 units per target (1700 units). Network weights are initialized using Xavier method with uniform sampling [18] [1], whereas biases are initialized with zeros.

All networks are trained by minimizing the log-cosh loss function, which is well suited for regression [17] using an Adam optimizer [20] with a batch size of 100. Adam parameters are left at default settings with a learning rate of 0.001 and hyperparameters $\beta_1 = 0.9; \beta_2 = 0.999; \epsilon = 10^{-7}$. Convergence is determined through early stopping [21] by evaluating the estimation error on the validation set. If there is no improvement for 100 epochs, training is stopped and those parameters which led to the best validation set performance are restored. Early stopping is applied to both pre-training (using source domain validation data) and fine-tuning (using target domain validation data), as early experiments indicated that this leads to optimal transfer learning performance. Note that for an unbiased final evaluation, a completely held-out test set is used which is separate from validation data.

We evaluate two scenarios with different source domains. In both cases, the target domain is the same high-density register file compiler ("target compiler"). In scenario "predecessor", we pre-train models on the data of a single compiler, which is the direct predecessor of the target compiler. This compiler is expected to have the greatest similarity with the target compiler. Scenario "cross-tech" utilizes source domain data from the same compiler in another technology node. For these two scenarios, Figure 1 shows kernel density estimates of the dynamic power and static power distributions of source and target domains. The distributions are compared after applying joint scaling and normalization as described above.

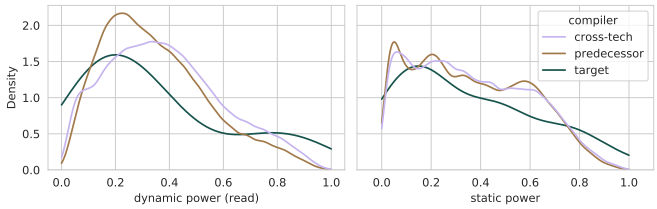

Fig. 1. Distribution of PPA variables for different compiler version datasets

V. Results

To express the estimation error as a percentage, we employ an unsigned symmetric percentage error (SPE) proposed by [22] to avoid the bias of the mean absolute percentage error (MAPE) [23]. The SPE is calculated after denormalizing the data back to original scale. As suggested by [1], aggregating errors for each target variable using quantiles avoids disproportionally weighting the long right tail of the error distribution. The arithmetic mean is used to aggregate further.

Figure 2 shows the error rate as a function of the relative amount of target domain data used for the two evaluated scenarios. The amount is presented as a fraction of the total available target domain data (approximately 13,000 memories). Each point represents the estimation error of a model trained on the target domain after parameter transfer from a source domain model, averaged across three repetitions with different data splits and weight initialization seed. The whiskers represent the standard deviation. All scenarios (cross-tech, predecessor, baseline) were evaluated on the same data splits and weight initializations to ensure comparability. The horizontal line represents a baseline model which was trained only on the target domain, i.e. without transfer learning. The baseline model was trained with 100% of the available target domain data. Although the remainder of this section evaluates mean results across all five target variables, the discussed observations are representative of the distributions of individual target variables unless otherwise indicated.

Using more target domain data leads to a lower estimation error. For the predecessor scenario, we observe that with around 50% of target domain data are used, the baseline (0.37) is surpassed at 0.34 (92% of baseline error). Therefore, when the goal is to reduce the amount of target domain data used without sacrificing accuracy, transfer learning reduces the amount of target domain data required by approximately 50%. Using transfer learning with more than 50% of target domain data reduces the error beyond the baseline. The 100% model achieves an estimation error of 0.30 (81% of baseline). Using less than 50% of target domain data compromises accuracy: When 10% of target domain data used, an average error of 0.57 is achieved (154% of baseline), while the 30% model attains

[1] In our experiments, He initialization [19] led learning to quickly stagnate ("dying ReLU")

an average estimation error of 0.42 (114% of baseline). The pre-trained model without fine-tuning (see Table I) attains an average error of 2.19 (592% of baseline).

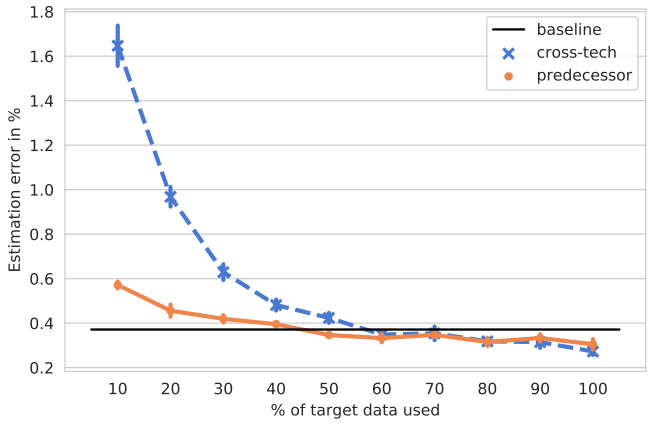

Fig. 2. Average estimation error for models fine-tuned with varying amounts of target domain data after pre-training on all source domain data for two different source scenarios. The baseline is trained only on target domain data (no pre-training).

Considering the cross-tech scenario, Figure 2 reveals a similar, albeit steeper trend. The baseline (same as in predecessor scenario) is outperformed with 60% of target domain data at 0.35 (94% of baseline). However, using less target domain data drastically deteriorates accuracy. With 10% of target domain data used we observe an estimation error of 1.64 (443% of baseline). The high estimation error of the pre-trained model (see Table I) of 72.02 helps to explain this phenomenon: the source domain (a compiler for another technology node) is much less similar to the target domain than the direct predecessor. It is therefore in line with expectations that a model pre-trained on this domain requires more target domain data before reaching sufficient accuracy. Nevertheless, given enough target domain data, the prior imposed by the source domain is overcome and the estimation error outperforms the baseline. When all target domain data is used, an error of 0.27 is observed (72% of baseline).

Figure 3 illustrates the almost worst-case estimation errors, i.e. the 98th percentile. The baseline model achieves an error of less than 1.91 for 98% of test set observations. With 60% of target domain data, the predecessor model slightly outperforms the baseline error at 1.90. The models trained with 10% and 100% of data respectively achieve errors of 3.18 (166% of baseline) and 1.62 (85% of baseline) in the 98th percentile.

The almost worst-case estimation errors for the cross-tech scenario exhibit a very similar distribution as for the average error, which is much steeper than that of the predecessor scenario. The model trained on 10% of target domain data attains an estimation error of 8.35 (437% of baseline). Using 60% of target domain data, the model outperforms the baseline with 1.80, while the model trained with 100% of target domain data reaches an estimation error of 1.37 (72% of baseline).

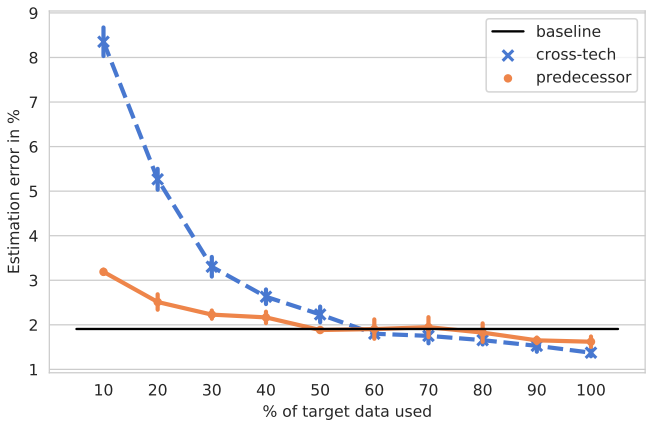

Fig. 3. 98th-percentile estimation error (almost worst-case) for two different source scenarios. The baseline is trained only on target domain data (no pre-training).

Using 90% or 100% of target domain data, the cross-tech model achieves lower average estimation errors than the predecessor model. One possible explanation for this effect is that, while the closely related predecessor source domain is more quickly adapted to the target domain, the cross-tech source domain allows the model to learn to generalize better. In support of this hypothesis, the effect is observed even earlier in the almost worst-case analysis, where the cross-tech model outperforms the predecessor model when using at least 60% of target domain data. The worst-case error is more sensitive to a model's generalization performance, hence this supports the hypothesis of better generalization obtained through a less similar source domain. We observe that the advantage of the cross-tech model is most pronounced for dynamic power (read and write) and access time. For static power, we observe this effect to a much lesser extend. In the case of cycle time, the predecessor model significantly outperforms the cross-tech model for all target domain data shares.

Table I shows the average (50th percentile) and almost worst-case (98th percentile) errors for pre-trained and fine-tuned models for different amounts of target domain data. In both cases, fine-tuning on 10% of target domain data drastically reduces the average estimation error from the pre-trained model. The model pre-trained on the cross-tech source domain exhibits a large average error of 72.02, whereas the model pre-trained on the predecessor source domain starts out with a lower error of 2.19. The pre-trained predecessor model also achieves a low 98th percentile error of 25.52, while the cross-tech model exhibits a very high error of 1061.66. This illustrates the relative dissimilarity between the target domain and the cross-tech source domain. The first 10% of target domain data lead to a sharp reduction in average and almost worst-case estimation error in both scenarios. The predecessor source domain outperforms the cross-tech source domain when very little target domain data is used, while the cross-tech model provides better errors for 100% of target domain data.

TABLE I
ESTIMATION ERROR (MEAN AND STANDARD DEVIATION) FOR DIFFERENT SCENARIOS AND AMOUNTS OF TARGET DOMAIN DATA USED

	% of target data used	cross-tech estimation error	predecessor estimation error
median	0	72.02 ± 9.58	$\mathbf{2.19 \pm 0.05}$
	10	1.65 ± 0.11	$\mathbf{0.57 \pm 0.02}$
	100	$\mathbf{0.27 \pm 0.02}$	0.30 ± 0.03
98th %tile	0	1061.66 ± 426.58	$\mathbf{25.52 \pm 6.05}$
	10	8.35 ± 0.39	$\mathbf{3.19 \pm 0.06}$
	100	$\mathbf{1.37 \pm 0.10}$	1.62 ± 0.15

VI. CONCLUSION AND OUTLOOK

Our experiments have shown that transfer learning is effective for reducing the amount of target domain data required by "hungry" PPA models of memory compilers. Through pre-training models on different source domains (direct predecessor and same compiler on another technology node), competitive estimation errors on the target domain can be achieved with less data. We have shown that pre-training on the direct predecessor, 50% less target domain data are required to avoid compromising accuracy. Based on this figure, PPA model provisioning times of approximately 40 days without transfer learning could be halved. On the other hand, only 10% of target domain data are sufficient to fulfill our quality requirements of less than 5% error in 98% of cases. Through transfer learning and the improved data and network structure used throughout this work, model provisioning times of less than a week are thus made feasible.

When direct predecessor versions are unavailable, the evaluated cross-tech transfer scenario promises data reductions of around 40% without compromising accuracy. Although the much steeper learning curve dismisses the possibility of extremely little target domain data, models fine-tuned on 30% of target domain data achieve sub-5% almost worst-case estimation errors. Therefore, provisioning times of less than two weeks appear realistic even when no direct predecessor is available for pre-training. Notably, when given a lot of target domain data, any of the evaluated transfer scenarios will outperform the baseline approach which does not leverage knowledge from related domains.

We further conclude that, depending on the goal of transfer learning, different source domains are preferred. For error reduction, it appears that a more distanced domain (cross-tech) leads to better generalization, but a closely related domain (predecessor) provides better results when data reduction is the objective. For our use case, data reduction is most desirable. Thus, we resort to data from another technology node only when no direct predecessor is available.

On the other hand, we plan to explore if and how data from multiple sources can be incorporated effectively: How does training order affect model accuracy? Should all sources be combined in a single training loop or be trained separately, perhaps in order of similarity to the target domain? It also remains to be researched how the suitability of a source domain can be assessed without evaluating models pre-trained on each available source. Measures of distributional distance (e.g. relative entropy) could be relevant indicators in this regard. Lastly, we aim to explore the possibility of including less closely matching source domains, where e.g. compiler parameters differ or fewer PPA variables are available. Such scenarios, which require input or output layers to be adapted and trained from scratch, remain to be investigated.

REFERENCES

[1] F. Last, M. Haeberlein, and U. Schlichtmann, "Predicting Memory Compiler Performance Outputs Using Feed-forward Neural Networks," *ACM TODAES*, vol. 25, no. 5, Jul. 2020.

[2] J. Vanschoren, "Meta-Learning," in *Automated Machine Learning: Methods, Systems, Challenges*, ser. SSCML. Springer International Publishing, 2019.

[3] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010.

[4] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE TKDE*, vol. 22, no. 10, Oct. 2010.

[5] J. Lu, V. Behbood *et al.*, "Transfer learning using computational intelligence: A survey," *Knowledge-Based Systems*, vol. 80, May 2015.

[6] S. Thrun, "Is Learning The n-th Thing Any Easier Than Learning The First?" in *NIPS*, vol. 8. MIT Press, 1996.

[7] M. Simon, E. Rodner, and J. Denzler, "Imagenet pre-trained models with batch normalization," *arXiv p rint arXiv:1612.01452*, 2016.

[8] X. Qiu, T. Sun *et al.*, "Pre-trained models for natural language processing: A survey," *Science China Technological Sciences*, vol. 63, no. 10, Oct. 2020.

[9] D. C. Cireşan, U. Meier, and J. Schmidhuber, "Transfer learning for Latin and Chinese characters with Deep Neural Networks," in *The 2012 IJCNN*, Jun. 2012.

[10] R. Ye and Q. Dai, "A novel transfer learning framework for time series forecasting," *Knowledge-Based Systems*, vol. 156, Sep. 2018.

[11] T. Mallick, P. Balaprakash *et al.*, "Transfer Learning with Graph Neural Networks for Short-Term Highway Traffic Forecasting," in *25th ICPR*, Jan. 2021.

[12] G. Huang, J. Hu *et al.*, "Machine Learning for Electronic Design Automation: A Survey," *ACM TODAES*, vol. 26, no. 5, Jun. 2021.

[13] H. M. Makrani, H. Sayadi *et al.*, "XPPE: cross-platform performance estimation of hardware accelerators using machine learning," in *24th ASPDAC*. ACM, Jan. 2019.

[14] J. Kwon and L. P. Carloni, "Transfer Learning for Design-Space Exploration with High-Level Synthesis," in *MLCAD '20*. ACM, Nov. 2020.

[15] C. Yu and W. Zhou, "Decision Making in Synthesis cross Technologies using LSTMs and Transfer Learning," in *MLCAD '20*. ACM, Nov. 2020.

[16] F. Last and U. Schlichtmann, "Partial Sharing Neural Networks for Multi-Target Regression on Power and Performance of Embedded Memories," in *MLCAD '20*. ACM, Nov. 2020.

[17] O. Reyes and S. Ventura, "Performing Multi-Target Regression via a Parameter Sharing-Based Deep Network," *International Journal of Neural Systems*, vol. 29, no. 09, Nov. 2019.

[18] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *13th AISTATS*. JMLR Workshop and Conference Proceedings, Mar. 2010.

[19] K. He, X. Zhang *et al.*, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," 2015.

[20] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Dec. 2014.

[21] R. Caruana, S. Lawrence, and C. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *NIPS*, 2001.

[22] S. K. Morley, T. V. Brito, and D. T. Welling, "Measures of Model Performance Based On the Log Accuracy Ratio," *Space Weather*, vol. 16, no. 1, 2018.

[23] C. Tofallis, "A Better Measure of Relative Prediction Accuracy for Model Selection and Model Estimation," Social Science Research Network, Tech. Rep., Jul. 2014.