LEAPER: Fast and Accurate FPGA-based System Performance Prediction via Transfer Learning

Gagandeep Singh[a] Dionysios Diamantopoulos[b] Juan Gómez-Luna[a]
Sander Stuijk[c] Henk Corporaal[c] Onur Mutlu[a]
[a]ETH Zürich [b]IBM Research Europe, Zurich [c]Eindhoven University of Technology

Machine learning has recently gained traction as a way to overcome the slow accelerator generation and implementation process on an FPGA. It can be used to build performance and resource usage models that enable fast early-stage design space exploration. However, these models suffer from three main limitations. First, training requires large amounts of data (features extracted from design synthesis and implementation tools), which is cost-inefficient because of the time-consuming accelerator design and implementation process. Second, a model trained for a specific environment cannot predict performance or resource usage for a new, unknown environment. In a cloud system, renting a platform for data collection to build an ML model can significantly increase the total-cost-ownership (TCO) of a system. Third, ML-based models trained using a limited number of samples are prone to overfitting. To overcome these limitations, we propose LEAPER, a *transfer learning*-based approach for prediction of performance and resource usage in FPGA-based systems. The key idea of LEAPER is to transfer an ML-based performance and resource usage model trained for a low-end edge environment to a new, high-end cloud environment to provide fast and accurate predictions for accelerator implementation. Experimental results show that LEAPER (1) provides, on average across six workloads and five FPGAs, 85% accuracy when we use our transferred model for prediction in a cloud environment with *5-shot learning* and (2) reduces design-space exploration time for accelerator implementation on an FPGA by 10×, from days to only a few hours.

1. Introduction

The need for energy efficiency and flexible acceleration of workloads has boosted the widespread adoption of field-programmable gate arrays (FPGAs) [1–4] in both edge and cloud computing. Past works [1–12] show that FPGAs can be employed effectively to accelerate a wide range of applications, including graph processing, databases, neural networks, weather forecasting, and genome analysis.

An FPGA is highly configurable as its circuitry can be tailored to perform any task. However, FPGA developers face two main issues while designing an accelerator. First, the large configuration space of an FPGA and the complex interactions among its configuration options cause many developers to explore optimization techniques in an ad-hoc manner [13, 14]. Second, FPGA programming leads to low productivity because of the time-consuming accelerator design and implementation process [15]. Therefore, a common challenge that past works have faced is how to evaluate the performance (and resource usage) of an accelerator implementation in a reasonable amount of time [16]. To overcome this problem, researchers have recently employed machine learning (ML)-based models [16–26] to predict the performance and resource usage of a given accelerator implementation quickly. However, these ML-based models have three fundamental issues that reduce their usability.

First, these ML-based predictors are trained for specific workloads, fixed hardware, and/or a set of inputs. Therefore, we cannot reuse these models for a previously *unseen* workload or a different FPGA platform because the trained model does not have a notion of the new, unknown environment.[1] Therefore, traditional ML-based models have limited *reusability*.

Second, ML-based models require a large number of samples to construct a useful performance predictor. Collecting such a large number of samples is often very time-consuming due to the very long accelerator implementation cycle on an FPGA, especially in a cloud computing environment where data collection could be costly.

Third, ML-based models trained using a limited number of samples are prone to serious *overfitting* problems (i.e., model matches to the training data too closely) [27], limiting model generalization.

Our goal is to overcome these three issues of ML-based models for prediction of FPGA performance and resource usage. To this end, we present LEAPER. Our key idea is to leverage an ML-based performance and resource usage model trained for a low-end edge environment to predict performance and resource usage of an accelerator implementation for a new, high-end cloud environment.

LEAPER[2] employs a transfer learning-based approach (also called *few-shot learning* [28]). This technique is based on the idea that algorithms, similar to humans, can learn from past experiences and *transfer* knowledge to the resolution of previously-unknown tasks. Concretely, LEAPER transfers an ML-based model trained on an edge FPGA-based system to a new, high-end cloud FPGA-based system. Using an edge FPGA-based system for training the ML-based model provides three major benefits over using a high-end FPGA. First, since edge devices are small, FPGA bitstream generation is faster compared to generating bitstream for a high-end FPGA. Second, edge FPGAs are cheaper and more affordable. Third, a high-end FPGA often requires integration with a server-grade host CPU, which can be costly or not possible for many users. Therefore, using low-cost and broadly available edge systems for training data collection can greatly facilitate the generation of ML-based predictors for performance and resource usage of FPGA-based systems.

LEAPER consists of three main steps. First, LEAPER uses *design of experiments* (*DoE*) [29], a technique to extract representative training data from a small number of experimental runs. Second, LEAPER trains an ML-based model (*base model*) to predict performance or resource usage for an accelerator implementation on a low-end edge environment. Third, LEAPER transfers the trained base model to a new, high-end *cloud* environment (a new FPGA or a new application) with only a few

[1]We consider a new workload or a new FPGA platform as a new environment.
[2]We call our mechanism LEAPER because it allows us to hop or "leap" between machine learning models.

new training samples (between 5 to 10 samples) from the new environment.

Figure 1 compares the traditional ML-based approach (ⓐ) to LEAPER (ⓑ). Using the traditional ML-based approach, we would need to create two separate prediction models, one for the low-end edge environment and another one for the high-end cloud environment, each one requiring a large number of samples. LEAPER transfers a previously trained model to a new, unknown environment using transfer learning with only a few training samples.

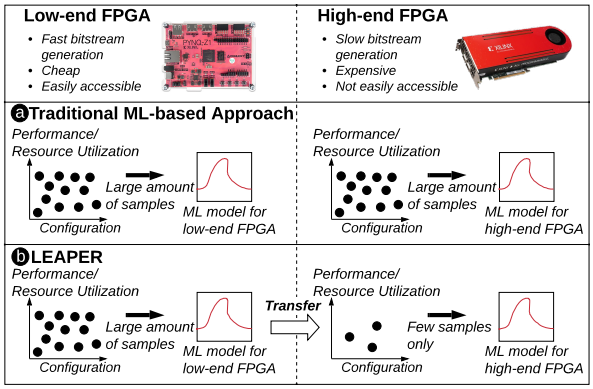

Figure 1: Traditional ML-based approach vs. LEAPER.

Key results. We demonstrate LEAPER across *five* state-of-the-art, high-end, cloud FPGA-based platforms with *three* different interconnect technologies on *six* real-world applications. We present two keys results. First, LEAPER achieves, on average across six evaluated workloads and five FPGAs, 85% accuracy when we use use our transferred model for prediction in a cloud environment. Second, LEAPER greatly reduces (up to 10×) the training overhead by transferring a *base model*, trained for a low-end, edge FPGA platform, to predict performance or resource usage for an accelerator implementation on a new, unknown high-end environment rather than building a new model from scratch.

This work makes the following **major contributions**:

1. We introduce LEAPER, a new transfer learning-based framework for prediction of performance and resource usage in FPGA-based systems.
2. Unlike state-of-the-art works [16–22] in FPGA modeling that use deep neural networks, we show that classic non-neural network-based models are enough to build an accurate predictor to evaluate an accelerator implementation on an FPGA.
3. We conduct an in-depth evaluation of LEAPER on real cloud systems with various FPGA configurations, showing that LEAPER can develop cheaper, faster, and highly accurate models.

2. LEAPER

LEAPER is a performance and resource estimation framework to *transfer* ML models [30,31] across: (1) different FPGA-based platforms for a single application, and (2) different applications on the same platform. In this section, we describe the main components of the framework. First, we give an overview of LEAPER (Section 2.1). Second, we describe the target cloud FPGA-based platform (Section 2.2). Third, we discuss accelerator optimization options and application features used for training an ML model (Section 2.3). Fourth, we briefly describe the base model building (Section 2.4). Fifth, we explain the key component of LEAPER to build cloud models: the transfer learning technique (Section 2.5). Sixth, we describe LEAPER's transfer learning algorithm (Section 2.6).

2.1. Overview

Figure 2 shows the key components of LEAPER. It consists of two parts: (1) base model building (low-end environment) and (2) target model building (high-end environment).

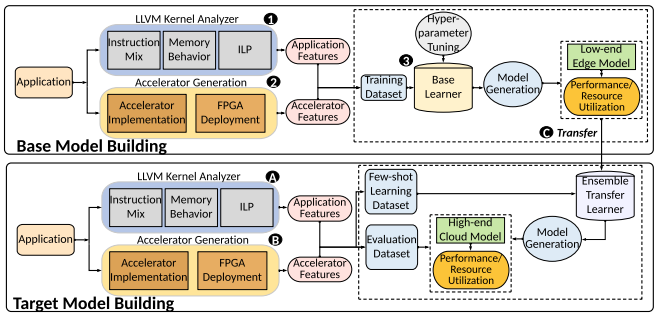

Figure 2: Overview of LEAPER.

Base model building. LEAPER's base model building consists of three phases. The first phase (❶ in Figure 2) is an LLVM-based [32] kernel analysis phase (Section 2.3), which extracts architecture-independent workload characteristics. We characterize in a microarchitecture-independent manner by using a specialized plugin of the LLVM compiler framework [33]. This type of characterization excludes any hardware dependence and captures the inherent characteristics of workloads.

In the second phase ❷, we generate accelerator implementations to gather accelerator implementation responses (performance and resource utilization) for training. Once the accelerator design has been implemented, the resulting FPGA-based accelerator is deployed in a system with a host CPU. We employ the *design of experiments* (DoE) technique [29] to select a small set of accelerator optimization configurations that well represent the entire space of accelerator optimization configurations (c_{doe}) to build a highly accurate *base learner*. By using DoE, we minimize the number of experiments needed to gather training data for LEAPER while ensuring good quality training data. Then, we run the c_{doe} configurations on the deployed FPGA-based system to gather samples for training our base model. The generated responses along with application properties from the first phase and the accelerator optimization parameters form the input to our base learner. In the third phase ❸, we train our base learner (Section 2.4) using ensemble learning [34]. During this phase, we perform additional tuning of our ML algorithm's hyper-parameters.[3] Once trained, the framework can predict the performance and resource usage on the base environment (a low-end edge system) of previously-unseen configurations, which are not part of the c_{doe} configurations used during the training.

Target model building. LEAPER's target model building consists of three phases. The first and the second phases (Ⓐ and Ⓑ in Figure 2) are the same LLVM-based kernel analysis and the accelerator generation phases as in base model building, respectively. We perform this step to create our *few-shot* learning dataset (c_{tl}), which is used to adapt the low-end edge

[3]Hyper-parameters are sets of ML algorithm variables that can be tuned to optimize the accuracy of the prediction model.

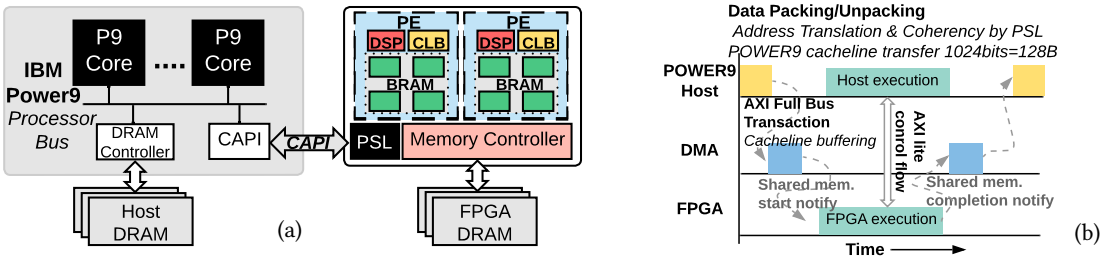

Figure 3: (a) Experimental cloud platform: High-end cloud FPGA platform with IBM® POWER9 CPU. (b) Execution timeline with data flow sequence from the host to the FPGA.

model to the target cloud environment.[4] In the final phase ❻, we train our ensemble transfer learner (Section 2.5) to leverage the low-end edge model to perform predictions for a new target environment (new application or FPGA).

2.2. Target Cloud FPGA-based Platform

Figure 3a shows the high-level overview of our FPGA-based cloud platform. An FPGA is connected to a server system based on an IBM POWER9 processor using the Coherent Accelerator Processor Interface (CAPI) [35]. Our design consists of multiple processing elements (PEs) that interact with the host system through the power service layer (PSL), which is the CAPI endpoint on the FPGA. A PE can utilize the on-chip FPGA elements such as DSP (digital signal processor), CLB (configurable logic block)[5], and BRAM (block RAM) to implement an application.

Figure 3b shows the execution timeline from our host CPU to the FPGA board. We make use of CAPI in a coarse-grained way as we offload the entire application to the FPGA. CAPI ensures that a PE accesses the entire CPU memory with the minimum number of memory copies between the host and the FPGA, e.g., avoiding the intermediate buffer copies that traditional PCIe-based DMA invokes [36, 37]. However, depending on the application, the CAPI protocol can be employed in a fine-grained algorithm-hardware co-design, like in *ExtraV* [38], which exploits the fine-grained communication capability of CAPI. On task completion, the PE notifies the host CPU via the AXI lite interface [39] and transfers back the results via CAPI-supported DMA transactions.

2.3. Application Features and Accelerator Optimization Options

The ML feature vector used for training our base and target models is composed of two parts: application features and accelerator optimization options.

Application features. We include inherent application features in our training dataset. For each application kernel k processing a dataset d, we obtain an application profile $p(k, d)$. $p(k, d)$ is a vector where each parameter is a statistic about an application feature. Table 1 lists the main application features we extract by using the LLVM-based PISA analysis tool [33]. Ultimately, the application profile p has 395 features, which includes all the sub-features of each metric we consider. We perform feature selection to select the 100 most important features to analyze the behavior of an application.

[4]We show via experiments (Section 4) that up to 5 samples (*5-shot*) are enough to learn the characteristics of a new environment.

[5]CLB is the fundamental component of an FPGA, made up of look-up-tables (LUTs) and flip-flops (FF).

Table 1: Main application features extracted from LLVM.

Application Feature	Description
Instruction Mix	Fraction of each instruction type (integer, floating point, memory read, memory write, etc.)
ILP	Instruction-level parallelism on an ideal machine.
Data/Instruction Reuse Distance	For a given distance δ, probability of reusing one data element/instruction (in a certain memory location) before accessing δ other unique data elements/instructions (in different memory locations).
Register Traffic	Average number of registers per instruction.
Memory Footprint	Total memory size used by the application.

Accelerator optimization options. Table 2 describes commonly used high-level synthesis (HLS) [40] pragmas to optimize an accelerator design on an FPGA. These optimization options constitute a part of our ML feature vector for training. We use eight optimization options. First, *loop pipelining* (PL) optimizes a loop to overlap different loop operations. Second, *loop unrolling* (UR) creates multiple copies of a loop for parallel execution. PL and UR aim to increase the processing throughput of an accelerator implementation. Third, to enable simultaneous memory accesses, *array partitioning* (PR) divides an array into smaller units of desired partitioning factor and maps them to different memory banks. This optimization produces considerable speedups but consumes more resources. Fourth, *inlining* (IL) ensures that a function is instantiated as dedicated hardware core. Fifth, *dataflow* (DF) optimization exploits task-level parallelism to allow parallel execution of tasks. Sixth, *burst read* (BR) controls burst reads from the host to the accelerator. Seventh, *burst writes* (BW) controls burst write to the host from an accelerator. Eighth, *FPGA frequency* (FR) constrains an accelerator implementation to a specific clock frequency. It affects not only the performance but also the resource usage of an implementation. For instance, to meet the FPGA timing requirements, the FPGA tool inserts registers between flip-flops, which increases resource usage.

Table 2: Accelerator optimization options used in training.

Optimization	Description
Loop pipelining (PL)	Enabled/disabled
Loop unrolling (UR)	Unrolling factor (Factor: $2^n, 1 \leq n \leq 6$)
Array partitioning (PR)	Block/cyclic/complete (Factor: $2^n, 1 \leq n \leq 6$))
Inlining (IL)	Enabled/disabled function inlining
Dataflow (DF)	Task level pipelining
Burst read (BR)	Read data burst from the host
Burst write (BW)	Write data burst to the host
FPGA frequency (FR)	Four-different frequency levels for an FPGA logic (50 MHz, 100 MHz, 150 MHz, and 200 MHz)

In total, our optimization options for a particular application leads up to 4,608 configurations. However, the actual optimization space of an application depends on the specific application characteristics (see Table 4). For example, we include loop unrolling in the optimization space when an

application contains loops that can be unrolled.

2.4. Base Learner Training

The third phase of base model training is the training of the base learners. We use an *ensemble* of two non-linear base learners that can capture the intricacies of accelerator implementation by predicting the execution time or the resource usage. Our first algorithm is the *random forest* (RF) [31], which is based on *bagging* [41]. We use RF to avoid a complex feature-selection scheme since RF embeds automatic procedures that are able to screen many input features [42]. Starting from a root node, RF constructs a tree and iteratively grows the tree by associating a node with a splitting value for an input feature to generate two child nodes. Each node is associated with a prediction of the target metric equal to the observed mean value in the training dataset for the input subspace that the node represents. Our second learner is *gradient boosting* [43]. Gradient boosting aims to *boost* the accuracy of a weak learner by using other learners to correct its predictions. Bagging [44] reduces model variance, and boosting decreases errors [45]. Therefore, we use RF and gradient boosting together to increase the predictive power of our final trained base model. In a machine learning task, $\mathcal{X}$ represents the feature space with label $\mathcal{Y}$, where a machine learning model is responsible for estimating a function $f : \mathcal{X} \rightarrow \mathcal{Y}$. LEAPER uses *base learners* to predict the performance (or resource usage) $\mathcal{Y}$ for a tuple *(p, c)* that belongs to the ML feature space $\mathcal{X}$, where *c* is a set of accelerator optimization options for an application profile *p(k, d)*.

The training dataset for our base model has three parts: (1) an application configuration vector *p(k, d)*, (2) an accelerator optimization option *c*, and (3) responses corresponding to each pair *(p, c)*. To gather the accelerator responses, we run each application *k* belonging to the training set $\mathbb{T}$ with an input dataset *d* on an FPGA-based platform while using an accelerator optimization *c*. This way, we obtain the execution time for the tuple *(p, c)*, which we can use as a *label* ($\mathcal{Y}$) for training our base learner for performance prediction. We build a similar model to predict resource usage, where we use the resource usage ($\eta_{\{BRAM,FF,LUT,DSP\}}$) of the tuple *(p(k, d), c)* as a *label* when we train our base learner for resource usage. After it is trained, our base learner it can predict the execution time (or resource usage) ($\hat{f}_s : \mathcal{X}_s \rightarrow \mathcal{Y}_s$) of tuples *(p(k, d), c)* that are *not* in the training set.

We improve base learner performance by tuning the algorithm's hyper-parameters [46]. Hyper-parameter tuning can provide better performance estimates for some applications. First, we perform as many iterations of the cross-validation process as hyper-parameter combinations. Second, we compare all the generated models by evaluating them on the testing set, and select the best one.

2.5. Target Model Building via Transfer Learning

LEAPER provides the ability to transfer trained a performance (or resource usage) model across different environments. LEAPER defines a target environment τ_t as an environment for which we wish to build a prediction model $\hat{f}_t$ where data collection is expensive, and a source environment τ_s as an environment for which we can *cheaply* collect many samples to build an ML model f_s. In our case, τ_s is a low-cost edge FPGA-based system, while τ_t is a high-cost cloud FPGA-based system. LEAPER then transfers the ML model for τ_s to τ_t using an ensemble transfer model $\hat{h}_t$.

Transfer learner. In transfer learning, a weak relationship between the base and the target environment can decrease the predictive power of the target environment model. This degradation is referred to as a *negative transfer* [47]. To avoid this, we use an ensemble model trained on the transfer set (i.e., the *few-shot learning* dataset in Figure 2) as our transfer learners (TLs). We use non-linear transfer learners because, based on our analysis (Section 5), non-linear models are able to better capture the non-linearity present in the accelerator performance and optimization options. Our first TL is based on TrAdaBoost [27], a boosting algorithm that fuses many weak learners into one strong predictor by adjusting the weights of training instances. The motivation behind such an approach is that by fusing many weak learners, boosting can improve the overall predictions in areas where the individual learners did not perform well. We use Gaussian process regression [48] as our second TL. It is a Bayesian algorithm that calculates the probability distribution over all the appropriate functions that fit the data. To transfer a trained model, we train both TrAdaBoost and Gaussian process regresssion, and select the best performing TL.

2.6. LEAPER Transfer Learning Algorithm

Algorithm 1 presents LEAPER's transfer learning approach. We provide as input the: (1) f_s model trained for a low-end environment, and (2) sub-sampled few-shot learning dataset c_{tl}. We initialize the training loop to the maximum value (line 1) to run LEAPER until convergence (line 2). We normalize the input feature vector to have all the different features to be on the same scale (line 3). Using the normalized input data, LEAPER trains the ensemble of TLs (line 4) that transforms the performance or resource usage model of a source environment f_s to the target environment's performance or resource usage model f_t. We use c_{tl} to generate a transfer learner $\hat{h}_t$ (line 6). We choose the TL that has the lowest mean relative error (line 7). Finally, we use $\hat{h}_t$ to transfer predictions from f_s to produce f_t (line 9) by performing a non-linear transformation (line 10).

Algorithm 1: LEAPER's transfer learning algorithm.

Input: (1) Base model (f_s) trained on the edge environment, (2) Sub-sampled *few-shot learning* dataset $c_{tl} \subset c_{doe}$ from the base and the target environment
Output: Target cloud model $f_t : \mathcal{X}_t \rightarrow \mathcal{Y}_t$

1 **Initialize:** Maximum number of iterations M
2 **while** $M \neq 0$ **do**
3 Normalize the feature vector
4 Train ensemble transfer learners (TL) with c_{tl}
5 Find the candidate TL:
6 $\hat{h}_t : \mathcal{X}_{tl} \rightarrow \mathcal{Y}_{tl}$ that minimizes the error over the $c_{doe} - c_{tl}$
7 Compute the mean relative error:
8 $\epsilon_{mre} = \frac{1}{c_{doe}-c_{tl}} \sum_{i=1}^{c_{doe}-c_{tl}} \frac{|y_t^{acc} - y_t^{pred}|}{y_t^{acc}}$
9 Use identified $\hat{h}_t$ to transform predictions of f_s:
10 $f_t = \hat{h}_t(f_s)$ where $f_s : \mathcal{X}_s \rightarrow \mathcal{Y}_s$
11 $M \leftarrow M - 1$
12 **end**
13 **return** f_t

3. Evaluation Methodology

3.1. Hardware Platform

Table 3 summarizes the system details of our low-end edge environment and high-end cloud environment. We select the widely available PYNQ-Z1 board [49] with XC7Z020-1CLG400C FPGA [50] as the low-end FPGA platform to build base model. We use the accelerator coherency port (ACP) [51] with PCIe Gen2 to attach FPGA-based accelerators to the ARM Cortex-A9 CPU [52] present in PYNQ-Z1. We use the Nimbix cloud [53] with CAPI-based FPGA system attached to a server-grade IBM POWER9 CPU system as our high-end cloud environment. Nimbix uses KVM (Kernel Virtual Machine) [54] for Linux virtualization and OpenStack [55] as middleware. We evaluate five state-of-the-art, high-end, cloud FPGA boards (ADM8K5 [56], ADM9V3 [57], NSA241 [58], N250SP [59], and ADMKU3 [60]) using two different interconnect technologies (CAPI-1 and CAPI-2). We can derive from the indicative prices listed in the last column of Table 3 shows that the total cost of ownership (TCO) of a high-end cloud system can be more than 100× of that of the low-end system, and thus it can be cost-prohibitive to many users.

Table 3: System parameters and configuration.

Low-end Edge System				**Indicative Price**
	PYNQ-Z1 ZYNQ [49] XC7Z020-1CLG400C [50] with PCIe Gen2 [61] ARM Cortex-A9 @650MHz, dual-core 512MB DDR3 with 16-bit bus @ 1050Mbps			$299[6]
Nimbix Cloud [53] System with OpenStack [55] and KVM Hypervisor [54]				**Indicative Price**
Host Configuration	IBM® POWER9 AC922 [62] @2.3 GHz, 16 cores 4-way SMT [63], 32 KiB L1 cache,256 KiB L2 cache, 10 MiB L3 cache, 32GiB RDIMM DDR4 2666 MHz [64]			$55000-$75000[7]
FPGA Description				
Board	**FPGA Family**	**Device**	**Interface**	**Indicative Price**
ADM9V3 [57]	Virtex UltraScale	XCVU3P-2	CAPI-2	N/A
NSA241 [58]	Virtex UltraScale	XCVU9P-2	CAPI-2	N/A
N250SP [59]	Kintex UltraScale	KU15P-2	CAPI-2	N/A
ADMKU3 [60]	Kintex UltraScale	XCKU060-2	CAPI-1	N/A
ADM8K5 [56]	Kintex UltraScale	XCKU115-2	CAPI-1	N/A

[5] https://store.digilentinc.com/pynq-z1-python-productivity-for-zynq-7000-arm-fpga-soc/ (accessed on 2022-06-13)
[6] https://www.microway.com/product/ibm-power-systems-ac922/ (accessed on 2022-06-13)
N/A: Not available indicative price from an online store, but in the region of $2500-$5000 for our purchased on-prem cards.

3.2. Programming Toolflow

We use the Xilinx SDSoC [65] design tool for implementing an accelerator on the low-end edge environment τ_s and the Vivado HLS [40] with the IBM CAPI-SNAP framework[8] for the high-end cloud environment τ_t. The SNAP framework provides seamless integration of an accelerator [66] and allows the exchange of control signals between the host and the FPGA processing elements over the AXI lite interface [39].

3.3. Workloads

We evaluate LEAPER using six benchmarks (Table 4), which are hand-tuned for FPGA execution. These benchmarks cover several application domains, i.e., **(1) image processing**: histogram calculation (*hist*) [67], and canny edge detection (*cedd*) [67]; **(2) machine learning**: binary long short term memory (*blstm*) [10], digit recognition (*digit*) [68]; **(3) databases**: relational operation (*select*) [69]; and **(4) data reorganization**: stream compaction (*sc*) [67]. These kernels are specified in C/C++ code using high-level synthesis (HLS) that is compiled to the target FPGA device.

[8] https://github.com/open-power/snap

Table 4: Evaluated application description including their domain, major kernels, and the input dataset. For major kernels, we mention the optimization space where × represents the optimization being applied to multiple loops or elements.

Application	**Domain**	**Major Kernels**	**Dataset**	**Optimization Space**
blstm [10]	Machine learning	Hidden layer fw Hidden layer back Output layer	Fraktur OCR [70]	2×PL, 3×PR(2,4), IL, 2×UR 2×PL, IL, 2×UR PL, IL, UR, DF, BR, BW, FR
cedd [67]	Image proc.	Gaussian filter Sobel filter Suppression filter Hysteresis filter	Frame-354×626 1000 frames	PL, PR(2,4), IL, UR PL, PR(2,4), IL, UR PL, PR(2,4), IL, UR PL, IL, UR, DF, BR, BW, FR
digit [68]	Machine learning	Hamming dist. KNN voting	MNIST-18000 train 2000 test	2×PL, 3×PR(2,4), IL, 4×UR IL, BR, BW, FR
hist [67]	Image proc.	Histogram avg.	Input-1536×1024 Bins-256	PL, PR, IL, DF, BR, BW, UR, FR
sc [67]	Data reorg.	Count Compact	1048576 elements	PL, IL, DF, BR, BW, UR, FR
select [69]	Database	Selection	1048576 elements	PL, IL, DF, BR, BW, FR

3.4. Evaluation Metrics

LEAPER is used to transfer a trained model using *few-shot learning*. We then analyze the accuracy of the newly-built target model to predict the performance and resource usage of all the other configurations in the target environment. We evaluate the accuracy of the transferred model in terms of the mean relative error (ϵ_i) to indicate the proximity of the predicted value y_i' to the actual value y_i across N test samples. The mean relative error (MRE) is calculated with Equation 1.

$$MRE = \frac{1}{N}\sum_{i=1}^{N}\epsilon_i = \frac{1}{N}\sum_{i=1}^{N}\frac{|y_i' - y_i|}{y_i} \quad (1)$$

4. Results

4.1. Accuracy Analysis of the Transferred Model

Performance model transfer. Figure 4 shows LEAPER's accuracy for transferring from edge to cloud platforms. We make the following three observations. First, as we increase the number of labeled samples, the target model accuracy increases. However, the accuracy saturates and with 5-10 samples or *shots*, we can achieve an accuracy as high as 80-90%. Second, compared to applications with multiple complex kernels (*blstm*, *cedd*, *digit* in Figure 4(a), 4(b), and 4(c), respectively), simpler kernels (*hist*, *sc*, *select* in Figure 4(d), 4(e), and 4(f), respectively) can be more easily transferred using fewer samples. There are two reasons for this trend: (1) applications with multiple kernels have a larger optimization space. The large optimization space leads to more complex interactions that have compounding effects with other optimization options because we model multiple kernels rather than just a single kernel, and (2) simple kernels such as *sc* and *select* have been implemented using *hls stream* interfaces, where rather than storing intermediate data in local FPGA memories, we read streams of data, and hence certain complex optimizations (like array partitioning) cannot be applied. This leads to a change in the feature space for different environments. Third, similar environments are more amenable to transfer as the source, and target models are more closely related, i.e., we require small number of samples from the target environment to transfer a model. For example, transferring to an FPGA with CAPI1 interface (PCIe Gen3 with ∼ 3.3 GB/s bandwidth) from low-end PYNQ with PCIe Gen2 ∼ 1.2 GB/s bandwidth entails a smaller increase in bandwidth than moving to an FPGA with CAPI2 interface (PCIe Gen4 with ∼ 12.3 GB/s bandwidth).

Figure 5 shows LEAPER's accuracy for transferring ML models *across different applications*. We make three observations. First, as we increase the number of training

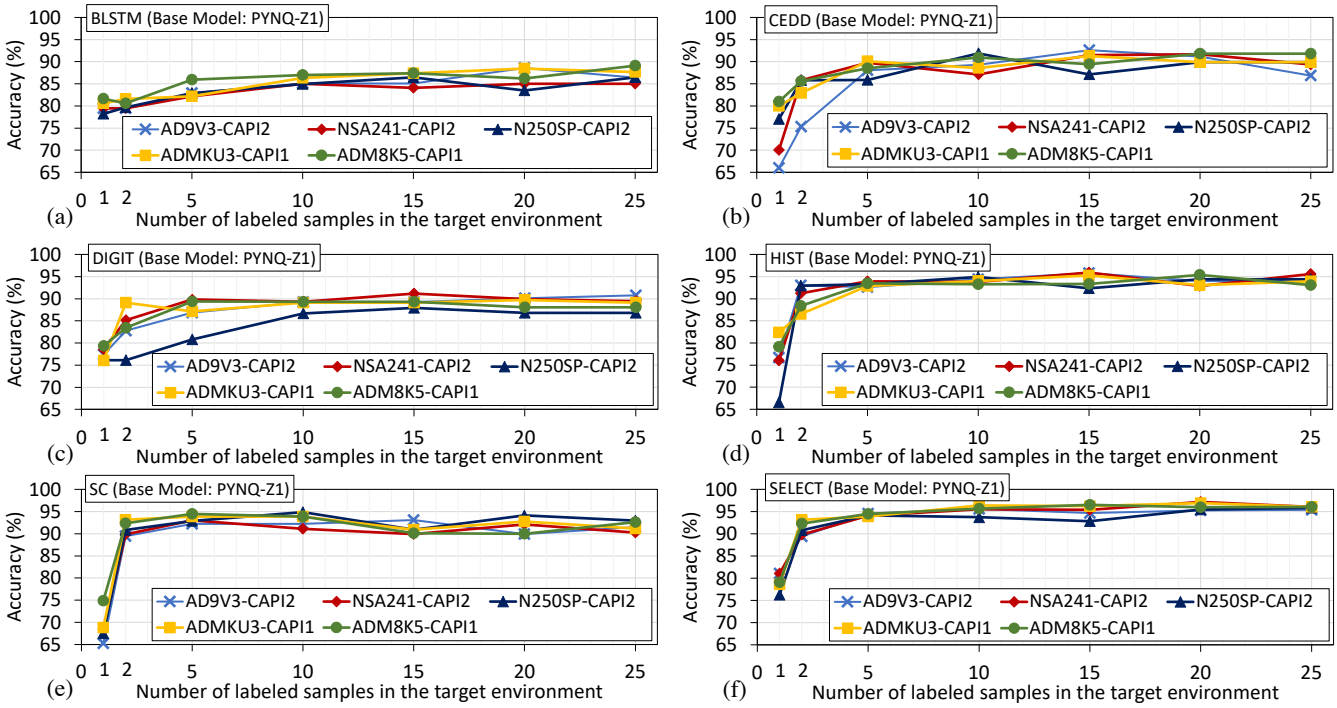

Figure 4: LEAPER's accuracy for platforms using different samples (horizontal axis) from the target platform.

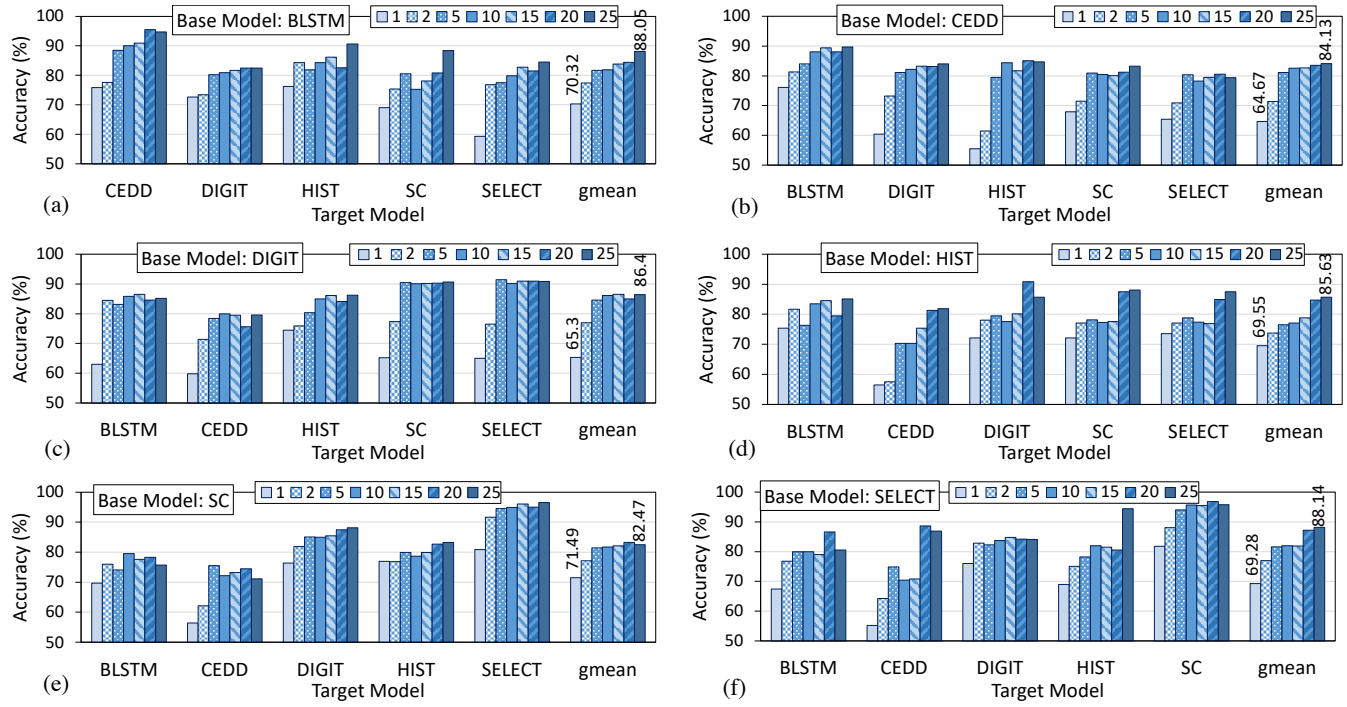

Figure 5: LEAPER's accuracy for transferring base models across various applications. The legends indicate the number of samples. Each plot represents a different application used as a base model. We *transfer* these base models, trained on the PYNQ-Z1 platform.

samples, the target model accuracy increases for most of the applications. This observation is inline with the observation we made while transferring models performance models across different cloud platforms (Figure 4). Second, the largest improvement in accuracy occurs when our sample size (c_{tl}) is between 2 to 10. In most cases, the accuracy saturates after 20 samples. Third, in some cases, we see a decrease in accuracy when increasing the number of samples, e.g., Figure 5(a) for `HIST`, `SC`, and `SELECT`. This result could be attributed to training with a small amount of data, which can sometimes lead to overfitting [27]. We conclude that LEAPER is effective at transferring models from edge to cloud

platforms and across applications.

Resource usage model transfer. By using LEAPER, we can also train a resource usage model on a *low-end* edge environment and transfer it to a high-end cloud environment. Figure 6 shows the accuracy of a target model trained by *5-shot* transfer learning for predicting a resource usage vector $\eta_{\{BRAM,FF,LUT,DSP\}}$. The reported accuracy is for the transferred model, i.e., using a base model (low-end FPGA) to predict a target model (high-end cloud FPGA) after transfer learning. In Figure 6(a), the horizontal axis depicts the target platform, while the base model is trained on a PYNQ-Z1 board. In the case of across application transfer (Figure 6(b)), the platform remains unchanged (PYNQ-Z1), while we use different applications to build the base model (horizontal axis).

We make three observations. First, the resource usage model shows low error rates for predicting BRAM and DSP usage. This is attributed to the fact that the technological configuration of these resources remains relatively unchanged across platforms (e.g., BRAM is implemented as 18 Kbits in both the source and target platforms). Second, flip-flops and look-up-tables have comparably lower accuracy because the configuration of CLB slices varies with the transistor technology and FPGA family. Third, while transferring across different applications (Figure 6(b)), we observe relatively low accuracy for DSP usage while transferring a base model trained on *hist*. This low accuracy is because the *hist* accelerator implementation does not use DSP units for logic. However, all other applications use DSP units. Therefore, the ML model trained on *hist* provides lower accuracy in other environments that use DSP units for accelerator implementation. We conclude that LEAPER can efficiently transfer resource usage models.

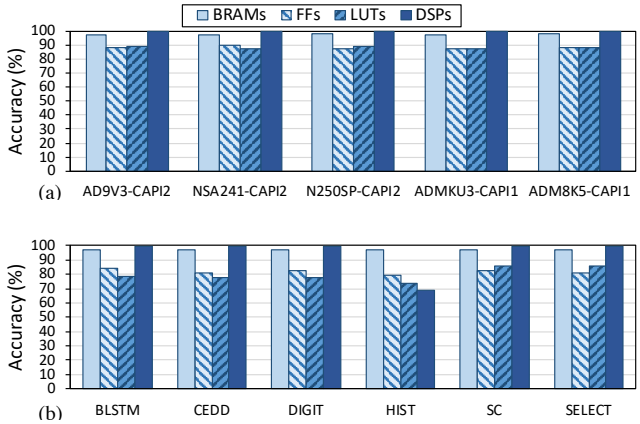

Figure 6: LEAPER's accuracy for transferring FPGA resource usage models through using (a) a base model trained on a low-end PYNQ-Z1 to different high-end target FPGA boards (horizontal axis), and (b) different applications as base models (horizontal axis) to all the target applications, on low-end PYNQ-Z1 board.

In Table 5, we report the performance and resource usage for our six applications both on a low-end system and a high-end ADMKU3 FPGA-based cloud system. We use LEAPER to obtain the performance and resource usage for the high-end cloud configuration.

4.2. Base Model Accuracy Analysis

We also evaluate the accuracy of our base model. The base model is trained on our low-end edge PYNQ board using c_{lhs} configurations sampled using the DoE technique. The base model can predict performance (or resource usage) outside the base model dataset (i.e, any configuration that is not a part of the DoE configuration space) c_{lhs}. To assess our base model, we use 30 previously unseen configurations that are not part of c_{lhs} on the base system, and we evaluate the mean relative error for all 30 unseen configurations on all six applications. Figure 7 shows the base model accuracy results. We also compare our base model to three other ML algorithms that are also trained using c_{lhs} configurations to predict performance and resource usage: XG-Boost (`XGB`) [71] based on Dai *et al.* [21], an artificial neural network (`ANN`) used by Makrani *et al.* [20] and a traditional decision tree (`DT`) [72].

Table 5: Execution time and resource usage for low-end edge configuration (PYNQ-Z1) and high-end cloud configuration (Nimbix *np8f1* instance).

Application	Config.	Exec (msec)	BRAM	DSP	FF	LUT
blstm	low-end	4200	80%	15%	24%	47%
	LEAPER	1245	62%	8%	12%	21%
cedd	low-end	10254	83%	37%	95%	97%
	LEAPER	2217	56%	3%	75%	94%
digit	low-end	2458	94%	33%	79%	85%
	LEAPER	873	84%	12%	24%	75%
hist	low-end	6173	94%	0%	11%	37%
	LEAPER	1104	67%	0%	5%	30%
sc	low-end	19306	82%	0.4%	12%	25%
	LEAPER	4018	91%	0.1%	12%	23%
select	low-end	18306	82%	0.4%	12%	25%
	LEAPER	3918	91%	0.1%	12%	23%

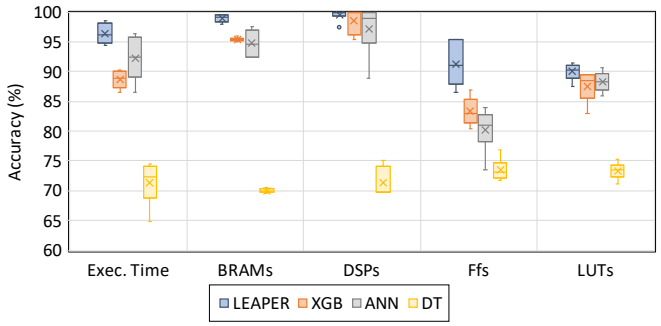

Figure 7: Average accuracy for performance (`Exec. Time`) and resource (`BRAM`, `DSP`, `FF`, `LUT`) usage predictions using LEAPER's base model and other machine learning techniques.

We make two major observations. First, on average, LEAPER is 8.1% (4.1%), 4.3% (5.1%), and 25.9% (23.9%) more accurate in terms of performance (resource usage) prediction than `XGB`, `ANN`, and `DT`, respectively. Second, `ANN` is 22.7% and 19.5% more accurate than *DT* for performance and resource usage prediction, respectively, but performs worse than LEAPER. `ANN` is not sample-efficient as it requires more training samples to learn [73]. We conclude that LEAPER's base model provides both high accuracy and sample-efficiency compared to other ML-based algorithms.

We also compare the performance of using different ML-based methods for transferring models across different platforms and across different applications. In Table 6, we show the average accuracy of LEAPER's ensemble of transfer learner and compare it to two different ML-based methods: (1) decision tree (`DT`) [72], and (2) adaBoost (`ADA`) [74]. We observe that LEAPER is on average 12.1% (10.6%) and 6.6% (7.7%) on average more accurate in transferring models across platforms (across applications) than `DT` and `ADA`, respectively. We conclude that an ensemble of transfer learner is better than using

a single transfer learner while transferring across different FPGA-based environments.

Table 6: Average accuracy (%) comparison of LEAPER with decision tree (DT) and adaBoost (ADA) as TL for *5-shot* transfer.

Environment	DT	ADA	LEAPER
Across Platform	77.7	83.2	89.8
Across Application	70.6	73.5	81.2

4.3. Target Cloud FPGA Model Building Cost

In Table 7, we mention the time and cost to build a model from scratch on a cloud environment using the traditional ML-based approach and compare it to using LEAPER to build a model for the cloud environment. If we build a model from scratch, then we need 50 sampled DoE configurations (c_{lhs}) for which the time and cost is mentioned in *DoE run (hours)* and *DoE cost*, respectively. Table 7 also includes the execution time on the ADMKU3 cloud platform ("Exec (msec)". While the process of synthesis and place and route (P&R) for the cloud FPGA, which is needed to obtain performance estimates in terms of maximum operating clock frequency and the resource usage, can be carried out offline, most of the cloud providers offer virtual machines (VMs) with all the appropriate software, IPs, and licenses needed to generate an FPGA image ready to be deployed at their cloud infrastructure (e.g., the Vivado AMI of AWS [75]). Therefore, we include the cost of the cloud environment ("Est. Cost ($)") for data collection.

By using LEAPER, the DoE runtime is amortized and, by using a few labeled samples c_{tl} ("5-shot (hours)") from the target platform, we can transfer a previously trained model and make predictions for all the other configurations for the target platform. We mention the the transfer time ("Transfer (msec)") for each model. As a result, quick exploration and significant time savings (at least 10.2×) are possible when transferring a model (i.e., "5-shot (hours)" + "Transfer (msec)") as compared to building a new model from scratch (i.e., "DoE run (hours)") + "Exec (msec)"). DoE reduces training samples from 500+ to 50, while *5-shot* transfer learning further reduces the number of samples to 5, so we achieve $\sim 100\times$ effective speedup compared to a traditional "brute-force" approach for data collection.

Table 7: DoE time for gathering sampled data points for a single CPU-FPGA platform ("DoE run (hours)"), DoE execution time on the deployed platform ("Exec (ms)"), Estimated Cost on a cloud platform ("Est. Cost ($)"), time for gathering 5 labeled samples ("5-shot (hours)"), LEAPER time including the transfer time ("Transfer (msec)"), "Speedup" over building a new model from scratch using only the DoE data.

Application	Traditional			LEAPER			
Name	DoE run (hours)	Exec (msec)	Est.Cost[7] ($)	5-shot (hours)	Transfer (msec)	Est.Cost[7] ($)	Speedup
blstm	135	455	168.7	13	55.6	16.2	10.4
cedd	124	295	155.0	12	26.5	15.0	10.3
digit	122	435	152.5	12	58.8	14.9	10.2
hist	97	45	121.2	9	17.1	11.3	10.8
sc	104	145	130.0	10	27.9	12.4	10.4
select	106	145	132.5	10	27.6	12.5	10.6

[7]The cost is estimated based on an enterprise online cost estimator [76]. Specifically, we selected an *n2* (8-core, 64GB RAM VM - 1.25$/h) for bitstream generation (x86) and an *np8f1* instance (160-thread 1TB RAM, ADM-PCIE-KU3 with CAPI-1 - 3$/h) for deployment.

5. Explainability: Why does LEAPER work?

To explain our results for transfer learning, we analyze the degree of *relatedness* between the source and target environments. We use two different analysis techniques: (1) divergence analysis [77], and (2) correlation analysis [78].

Divergence analysis. We use Jensen-Shannon divergence (JSD) [79] to measure the difference between two probability distributions of the source ($P(\tau_s)$) and the target environment ($P(\tau_t)$). The lower the JSD value, the more similar the target environment is to the source (i.e., if $D_{JSD}(P(\tau_t)||P(\tau_s) = 0$ implies that the distributions are identical and 1 indicates unrelated distributions). Table 8 shows the JSD analysis for transferring models between different applications. We make three main observations. First, JSD analysis confirms the trend observed from transferring application models (Figure 5), i.e., the more closely related the source and target applications, the fewer samples are required to train our non-linear transfer learners. Second, the higher the JSD between two applications, the lower the accuracy while transferring between those tasks. Third, for many applications JSD values is low, which indicates that we can easily transfer models between such environments using a few samples from the target environment.

Table 8: Jensen-Shannon Divergence (JSD) [79] between performance distributions of different applications. JSD measures statistical distance between two probability distributions.

		Base Learner					
		blstm	cedd	digit	hist	sc	select
Target Model	blstm	0.00	0.24	0.34	0.25	0.31	0.30
	cedd	0.24	0.00	0.49	0.54	0.41	0.40
	digit	0.34	0.49	0.00	0.25	0.21	0.21
	hist	0.25	0.54	0.25	0.00	0.25	0.24
	sc	0.30	0.40	0.21	0.24	0.00	0.05
	select	0.30	0.41	0.21	0.25	0.05	0.00

Correlation analysis. Correlation analysis measures the strength of linear correlation between two environments. We make four major observations. First, for different target hardware platforms, we have a high correlation of 0.76 to 0.97 between the source and target execution time, which indicates that the target model's performance behavior can accurately be learned using the source environment. Second, as we switch to a higher external bandwidth for the target platform (i.e., CAPI1 to CAPI2), the correlation becomes lower because the hardware change is much more *severe* coming from a low-end FPGA with limited external bandwidth. Whereas changing the technology node from one FPGA to another (e.g., changing from ADMKU3 board to AD9V3 board) leads to a smaller change in the environment because of the linear relation between technology node and performance. Third, the correlation between applications on a single platform is lower (0.45 to 0.9) because of the varying application characteristics and optimization space. Fourth, as the linear correlation is not 1 for all platforms, the use of a nonlinear transfer models is substantiated. We conclude that LEAPER learns differences in environments to accurately transfer FPGA-based system performance prediction models from one platform to another.

6. Discussion and Limitations

LEAPER's generality. LEAPER is a framework for building and transferring models from a small edge environment to any new, unknown FPGA-based environment. We demonstrate our approach using the cloud system as our target environment because cloud systems often use expensive, high-end FPGAs, e.g., Amazon AWS F1 cloud [80], Alibaba Elastic cloud [81], etc. We can, thus, achieve tangible gains in terms of cost, efficiency, and performance. However, LEAPER can be used to transfer models to any high-end FPGA system.

Effect of FPGA resource saturation. An FPGA gives us the flexibility to map a given operation to different potential resources. For example, we can map a multiplication operation

to either a CLB or a DSP slice. We can decide the mapping based on the operand width, i.e., if the operand width is smaller than DSP slice width, the operation is mapped to a CLB otherwise to a DSP unit. Currently, we do not consider mapping the same operation to different resources.

Transfer a model to a new platform and application simultaneously. In supervised learning, transferring both to a new platform and application at the same time would lead to sub-optimal results (as observed in [82]). This sub-optimal performance is because in such a scenario we would perform two types of transfer at the same time to (1) unknown hardware and (2) unknown application. We explicitly exclude this scenario in the current work.

7. Related Work

To our knowledge, LEAPER is the first work to leverage an ML-based performance and resource usage model trained for a low-end edge environment to predict the performance and resource usage of an accelerator implementation for a new, high-end cloud environment. FPGAs lead to very low productivity due to the time-consuming downstream accelerator implementation process. In this section, we describe other related works in ML-based modeling of FPGA, analytical modeling of FPGA, and transfer learning.

ML-based modeling of FPGAs. Recent works propose ML-based methods [16–25, 83] to overcome the issue of low productivity while designing FPGA-based accelerators. O'Neal *et al.* [16] use CPU performance counters to train several ML-based models to predict the performance and power consumption of an accelerator implementation. Makrani *et al.* [20] train a neural network-based model to predict application speedup across different FPGAs. Makrani *et al.* [17] and Dai *et al.* [21] use ML to predict resource usage for an accelerator implementation. However, these solutions become largely impractical once the platform, the application, or even the size of the workload changes. LEAPER proposes to reuse previously-built models for a low-end source environment on a high-end target environment through transfer learning. Unlike LEAPER, past works apply traditional, time-consuming brute-force techniques to collects training datasets.

Analytical modeling of FPGAs. Analytical models abstract low-level system details and provide quick performance estimates at the cost of accuracy. These approaches (e.g., [84–87]) analyze dataflow graphs and apply mathematical equations to approximate resource usage or performance after the HLS pre-implementation phase. Even though they enable quick early-stage design studies, however, analytical models are not able to model the intricacies of the complete implementation process [21]. Therefore, these approaches provide crude estimates of the actual performance. Moreover, these models require FPGA domain knowledge to form mathematical equations. In contrast, LEAPER does not require expert knowledge to construct equations. LEAPER learns from the training data (application features and accelerator optimization options) to consider the complete downstream accelerator implementation process and provides the capability to transfer models from an edge-FPGA to a high-end cloud FPGA environment.

Transfer learning. Recently, transfer learning [88–92] has gained traction to decrease the cost of learning by transferring knowledge between different tasks. Valov *et al.* [93] investigate the transfer of application models across different CPU-based environments using linear transformations. Jamshidi *et al.* [47] demonstrate the applicability of using nonlinear models to transfer CPU-based performance models. The works above influenced the design of LEAPER. In contrast to them, we: (1) focus on FPGA-based systems that, unlike a CPU-based system, have a different hardware architecture for every application and optimization strategy, and (2) use an ensemble of transfer learners that transfers accurate models to a target environment.

8. Conclusion

We introduce LEAPER, the first *transfer learning*-based approach for prediction of performance and resource usage in FPGA-based systems. LEAPER overcomes the inefficiency of traditional ML-based methods by leveraging an ML-based performance and resource usage model trained for a low-end edge environment to predict the performance and resource usage of an accelerator implementation for a new, high-end cloud environment.

Our experiments show that LEAPER is cheaper (with *5-shot*), faster (up to 10×), and highly accurate (on average 85%) at predicting performance and resource usage in a new, unknown target cloud environment than building models from scratch. We believe that LEAPER can open up new avenues for research on FPGA-based systems from edge to cloud computing, and hopefully, it will inspire the development of new modeling techniques for FPGAs.

Acknowledgments

We thank the SAFARI Research Group members for valuable feedback and the stimulating intellectual environment they provide. Special thanks to Florian Auernhammer and Raphael Polig for providing access to IBM systems. We appreciate valuable discussions with Kaan Kara. This work was performed in the framework of the Horizon 2020 program for the project "Near-Memory Computing (NeMeCo)". It is funded by the European Commission under Marie Sklodowska-Curie Innovative Training Networks European Industrial Doctorate (Project ID: 676240). We acknowledge the generous gifts of our industrial partners, especially Google, Huawei, Intel, Microsoft, VMware. This research was partially supported by the Semiconductor Research Corporation and the ETH Future Computing Laboratory.

References

[1] G. Singh *et al.*, "FPGA-based Near-Memory Acceleration of Modern Data-Intensive Applications," in *IEEE Micro*, 2021.
[2] J. van Lunteren *et al.*, "Coherently Attached Programmable Near-Memory Acceleration Platform and Its Application to Stencil Processing," in *DATE*, 2019.
[3] G. Singh *et al.*, "NARMADA: Near-Memory Horizontal Diffusion Accelerator for Scalable Stencil Computations," in *FPL*, 2019.
[4] G. Singh *et al.*, "Low Precision Processing for High Order Stencil Computations," in *Springer LNCS*, 2019.
[5] J. Jiang *et al.*, "Boyi: A Systematic Framework for Automatically Deciding the Right Execution Model of OpenCL Applications on FPGAs," in *FPGA*, 2020.
[6] D. S. Cali *et al.*, "SeGraM: A Universal Hardware Accelerator for Genomic Sequence-to-Graph and Sequence-to-Sequence Mapping," in *ISCA*, 2022.
[7] G. Dai *et al.*, "ForeGraph: Exploring Large-scale Graph Processing on Multi-FPGA Architecture," in *FPGA*, 2017.
[8] M. Alser *et al.*, "GateKeeper: A New Hardware Architecture for Accelerating Pre-Alignment in DNA Short Read Mapping," in *Bioinformatics*, 2017.
[9] M. Alser *et al.*, "SneakySnake: A Fast and Accurate Universal Genome Pre-Alignment Filter for CPUs, GPUs, and FPGAs," in *Bioinformatics*, 2020.
[10] D. Diamantopoulos *et al.*, "ecTALK: Energy Efficient Coherent Transprecision Accelerators—The Bidirectional Long Short-Term Memory Neural Network Case," in *COOL CHIPS*, 2018.

[11] K. Kara *et al.*, "ColumnML: Column-Store Machine Learning with On-The-Fly Data Transformation," in *VLDB*, 2018.
[12] G. Singh *et al.*, "Accelerating Weather Prediction using Near-Memory Reconfigurable Fabric," in *TRETS*, 2022.
[13] D. Diamantopoulos *et al.*, "Agile Autotuning of a Transprecision Tensor Accelerator Overlay for TVM Compiler Stack," in *FPL*, 2020.
[14] G. Singh *et al.*, "Modeling FPGA-Based Systems via Few-Shot Learning," in *FPGA*, 2021.
[15] K. O'Neal *et al.*, "Predictive Modeling for CPU, GPU, and FPGA Performance and Power Consumption: A Survey," in *VLSI*, 2018.
[16] K. O'Neal *et al.*, "HLSPredict: Cross Platform Performance Prediction for FPGA High-Level Synthesis," in *ICCAD*, 2018.
[17] H. M. Makrani *et al.*, "Pyramid: Machine Learning Framework to Estimate the Optimal Timing and Resource Usage of a High-Level Synthesis Design," in *FPL*, 2019.
[18] M. Ferianc *et al.*, "Improving Performance Estimation for FPGA-based Accelerators for Convolutional Neural Networks," in *ARC*, 2020.
[19] E. Ustun *et al.*, "Accurate Operation Delay Prediction for FPGA HLS Using Graph Neural Networks," in *ICCAD*, 2020.
[20] H. M. Makrani *et al.*, "XPPE: Cross-Platform Performance Estimation of Hardware Accelerators Using Machine Learning," in *ASP-DAC*, 2019.
[21] S. Dai *et al.*, "Fast and Accurate Estimation of Quality of Results in High-Level Synthesis with Machine Learning," in *FCCM*, 2018.
[22] J. Zhao *et al.*, "Machine Learning Based Routing Congestion Prediction in FPGA High-Level Synthesis," in *DATE*, 2019.
[23] Q. Yanghua *et al.*, "Improving Classification Accuracy of a Machine Learning Approach for FPGA Timing Closure," in *FCCM*, 2016.
[24] Z. Wang *et al.*, "Machine Leaming to Set Meta-Heuristic Specific Parameters for High-Level Synthesis Design Space Exploration," in *DAC*, 2020.
[25] A. Mahapatra *et al.*, "Machine-Learning Based Simulated Annealer Method for High Level Synthesis Design Space Exploration," in *ESLsyn*, 2014.
[26] G. Singh *et al.*, "NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning," in *DAC*, 2019.
[27] W. Dai *et al.*, "Boosting for Transfer Learning," in *ICML*, 2007.
[28] Y. Wang *et al.*, "Few-Shot Learning: A Survey," in *arxiv*, 2019.
[29] D. C. Montgomery, *Design and Analysis of Experiments*. John Wiley & Sons, 2017.
[30] P. H. Swain *et al.*, "The Decision Tree Classifier: Design and Potential," in *IEEE GE*, 1977.
[31] L. Breiman, "Random Forests," in *ML*, 2001.
[32] C. Lattner *et al.*, "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation," in *CGO*, 2004.
[33] A. Anghel *et al.*, "An Instrumentation Approach for Hardware-Agnostic Software Characterization," in *IJPP*, 2016.
[34] D. Opitz *et al.*, "Popular Ensemble Methods: An Empirical Study," in *JAIR*, 1999.
[35] J. Stuecheli *et al.*, "CAPI: A Coherent Accelerator Processor Interface," in *IBM JRD*, 2015.
[36] Y.-k. Choi *et al.*, "A Quantitative Analysis on Microarchitectures of Modern CPU-FPGA Platforms," in *DAC*, 2016.
[37] G. Singh *et al.*, "NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling," in *FPL*, 2020.
[38] J. Lee *et al.*, "ExtraV: Boosting Graph Processing Near Storage with a Coherent Accelerator," in *VLDB*, 2017.
[39] "AXI Reference Guide, https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf."
[40] "Vivado High-Level Synthesis, https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html."
[41] L. Breiman, "Bagging Predictors," in *ML*, 1996.
[42] G. Mariani *et al.*, "Predicting Cloud Performance for HPC Applications: A User-Oriented Approach," in *CCGRID*, 2017.
[43] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," in *Annals of statistics*, 2001.
[44] R. E. Schapire, "The Strength of Weak Learnability," in *ML*, 1990.
[45] S. Kotsiantis *et al.*, "Combining Bagging and Boosting," in *IJCIS*, 2004.
[46] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," in *JMLR*, 2011.
[47] P. Jamshidi *et al.*, "Transfer Learning for Improving Model Predictions in Highly Configurable Software," in *SEAMS*, 2017.
[48] C. E. Rasmussen, *Gaussian Processes in Machine Learning*. Springer, 2003.
[49] "PYNQ-Z1 Setup Guide, https://pynq.readthedocs.io/en/latest/getting_started/pynq_z1_setup.html."
[50] "Zynq-7000 SoC Data Sheet: Overview, https://docs.xilinx.com/v/u/en-US/ds190-Zynq-7000-Overview."
[51] "Accelerator Coherency Port, Cortex-A9 MPCore Technical Reference Manual, Arm Ltd., https://developer.arm.com/documentation/ddi0407/e/snoop-control-unit/accelerator-coherency-port."
[52] "Cortex-A9-ARM Developer, https://developer.arm.com/processors/cortex-a9."
[53] "Nimbix Cloud, https://www.nimbix.net, Accessed: 2020-06-13."
[54] "Kernel Virtual Machine, https://www.linux-kvm.org/page/main_page."
[55] "Openstack, https://www.openstack.org, Accessed: 2020-06-12."
[56] "ADM-PCIE-8K5-High-Performance Data Processing, https://www.alpha-data.com/dcp/products.php?product=adm-pcie-8k5."
[57] "ADM-PCIE-9V3-High-Performance Network Accelerator, https://www.alpha-data.com/dcp/products.php?product=adm-pcie-9v3."
[58] "NSA.241 FPGA accelerator card, http://www.semptian.com/proinfo/126.html."
[59] "Nallatech 250SP, http://www.nallatech.com/250sp."
[60] "ADM-PCIE-KU3-High-Performance Data Processing, https://www.alpha-data.com/dcp/products.php?product=adm-pcie-ku3."
[61] D. Mayhew *et al.*, "PCI Express and Advanced Switching: Evolutionary Path to Building Next Generation Interconnects," in *HOTI*, 2003.
[62] S. K. Sadasivam *et al.*, "IBM POWER9 Processor Architecture," in *IEEE Micro*, 2017.
[63] D. M. Tullsen *et al.*, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," in *ISCA*, 1995.
[64] RDIMM, https://www.micron.com/products/dram-modules/rdimm.
[65] "SDSoc Development envrionment, https://www.xilinx.com/products/design-tools/software-zone/sdsoc.html."
[66] A. Castellane *et al.*, "Enabling Fast and Highly Effective FPGA Design Process Using the CAPI SNAP Framework," in *ISC HPC*, 2019.
[67] J. Gómez-Luna *et al.*, "Chai: Collaborative Heterogeneous Applications for Integrated-architectures," in *ISPASS*, 2017.
[68] Y. Zhou *et al.*, "Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs," in *FPGA*, 2018.
[69] J. Gómez-Luna *et al.*, "In-Place Data Sliding Algorithms for Many-Core Architectures," in *ICPP*, 2015.
[70] M. R. Yousefi *et al.*, "Binarization-Free OCR for Historical Documents Using LSTM Networks," in *ICDAR*, 2015.
[71] T. Chen *et al.*, "XGBoost: A Scalable Tree Boosting System," in *SIGKDD*, 2016.
[72] W.-Y. Loh, "Classification and Regression Trees," in *WIREs DMKD*, 2011.
[73] D. Li *et al.*, "Processor Design Space Exploration via Statistical Sampling and Semi-Supervised Ensemble Learning," in *IEEE Access*, 2018.
[74] Y. Freund *et al.*, "A Decision-Theoretic Generalization of On-line Learning and An Application to Boosting," in *JCSS*. Elsevier, 1997.
[75] "Vivado 2020.2 Developer AMI, https://aws.amazon.com/marketplace/pp/xilinx-vivado-20202-developer-ami/b08pvvmhnq, Accessed: 2021-02-01."
[76] "Nimbix Cloud Price Calculator, https://www.nimbix.net/cloud-price-calculator, Accessed: 2020-06-13."
[77] B. Coutinho *et al.*, "Divergence Analysis and Optimizations," in *PACT*, 2011.
[78] J. Benesty *et al.*, "Pearson Correlation Coefficient," in *Noise Reduction in Speech Processing*, 2009.
[79] J. Lin, "Divergence Measures based on the Shannon Entropy," in *IEEE Trans. Inf. Theory*, 1991.
[80] "Amazon Web Services. AWS FPGA Developer AMI, https://aws.amazon.com/ec2/instance-types/f1, Accessed: 2020-06-10."
[81] "Alibaba cloud, www.alibabacloud.com, Accessed: 2020-06-10."
[82] N. Ardalani *et al.*, "Cross-Architecture Performance Prediction (XAPP) Using CPU Code to Predict GPU Performance," in *ISCA*, 2015.
[83] Q. Sun *et al.*, "Correlated Multi-objective Multi-fidelity Optimization for HLS Directives Design," in *TODAES*, 2022.
[84] G. Zhong *et al.*, "Lin-Analyzer: A High-level Performance Analysis Tool for FPGA-based Accelerators," in *DAC*, 2016.
[85] J. Zhao *et al.*, "COMBA: A Comprehensive Model-based Analysis Framework for High Level Synthesis of Real Applications," in *ICCAD*, 2017.
[86] Y.-k. Choi *et al.*, "HLScope+: Fast and Accurate Performance Estimation for FPGA HLS," in *ICCAD*, 2017.
[87] S. Huang *et al.*, "Analysis and Modeling of Collaborative Execution Strategies for Heterogeneous CPU-FPGA Architectures," in *ICPE*, 2019.
[88] H. Chen *et al.*, "Experience Transfer for the Configuration Tuning in Large-Scale Computing Systems," in *TKDE*, 2010.
[89] S. J. Pan *et al.*, "A Survey on Transfer Learning," in *IEEE TKDE*, 2009.
[90] U. Baumann *et al.*, "Classifying Road Intersections using Transfer-Learning on a Deep Neural Network," in *ITSC*, 2018.
[91] S. Bozinovski *et al.*, "The Influence of Pattern Similarity and Transfer Learning Upon Training of a Base Perceptron B2," in *Proceedings of Symposium Informatica*, 1976.
[92] L. Y. Pratt, "Discriminability-based Transfer Between Neural Networks," in *NIPS*, 1992.
[93] P. Valov *et al.*, "Transferring Performance Prediction Models Across Different Hardware Platforms," in *ICPE*, 2017.