DATA-DRIVEN OFFLINE OPTIMIZATION FOR ARCHITECTING HARDWARE ACCELERATORS

Anonymous authors
Paper under double-blind review

ABSTRACT

To attain higher efficiency, the industry has gradually reformed towards application-specific hardware accelerators. While such a paradigm shift is already starting to show promising results, designers need to spend considerable manual effort and perform large number of time-consuming simulations to find accelerators that can accelerate multiple target applications while obeying design constraints. Moreover, such a "simulation-driven" approach must be re-run from scratch every time the set of target applications or design constraints change. An alternative paradigm is to use a "data-driven", offline approach that utilizes logged simulation data, to architect hardware accelerators, without needing any form of simulations. Such an approach not only alleviates the need to run time-consuming simulation, but also enables data reuse and applies even when set of target applications changes. In this paper, we develop such a data-driven offline optimization method for designing hardware accelerators, dubbed PRIME, that enjoys all of these properties. Our approach learns a conservative, robust estimate of the desired cost function, utilizes infeasible points and optimizes the design against this estimate without any additional simulator queries during optimization. PRIME architects accelerators—tailored towards both single- and multi-applications—improving performance upon stat-of-the-art simulation-driven methods by about 1.54× and 1.20×, while considerably reducing the required total simulation time by 93% and 99%, respectively. In addition, PRIME also architects effective accelerators for unseen applications in a zero-shot setting, outperforming simulation-based methods by 1.26×.

1 INTRODUCTION

The death of Moore's Law (Esmaeilzadeh et al., 2011) and its spiraling effect on the semiconductor industry have driven the growth of specialized hardware accelerators. These specialized accelerators are tailored to specific applications (Yazdanbakhsh et al., 2021a; Reagen et al., 2017; Nardi et al., 2019; Shi et al., 2020). To design specialized accelerators, designers first spend considerable amounts of time developing simulators that closely model the real accelerator performance, and then optimize the accelerator using the simulator. While such simulators can automate accelerator design, this requires a large number of simulator queries for each new design, both in terms of simulation time and compute requirements, and this cost increases with the size of the design space (Yazdanbakhsh et al., 2021b; Shi et al., 2020; Hegde et al., 2021). Moreover, most of the accelerators in the design space are typically infeasible (Hegde et al., 2021; Yazdanbakhsh et al., 2021a) because of build errors in silicon or compilation/mapping failures. When the target applications change or a new application is added, the complete simulation-driven procedure is generally repeated. To make such approaches efficient and practically viable, designers typically "bake-in" constraints or otherwise narrow the search space, but such constraints can leave out high-performing solutions (Dave et al., 2019; Parashar et al., 2019; Chatarasi et al., 2020).

An alternate approach, proposed in this work, is to devise a *data-driven* optimization method that only utilizes a database of previously tested accelerator designs, annotated with measured performance metrics, to produce new optimized designs *without* additional active queries to an explicit silicon or a cycle-accurate simulator. Such a data-driven approach provides three key benefits: **(1)** it significantly shortens the recurring cost of running large-scale simulation sweeps, **(2)** it alleviates the need to explicitly bake in domain knowledge or search space pruning, and **(3)** it enables data re-use by empowering the designer to optimize accelerators for new unseen applications, by the virtue of effective generalization. While data-driven approaches have shown promising results in biology (Fu

& Levine, 2021; Brookes et al., 2019; Trabucco et al., 2021a), using offline optimization methods to design accelerators has been challenging primarily due to the abundance of infeasible design points (Yazdanbakhsh et al., 2021a; Hegde et al., 2021) (See Figure 3 and Appendix Figure 11).

The key contribution of this paper is a data-driven approach, PRIME, to automatically architect high-performing application-specific accelerators by using only previously collected offline data. PRIME learns a robust surrogate model of the task objective function from an existing offline dataset, and finds high-performing application-specific accelerators by optimizing the architectural parameters against this learned surrogate function, as shown in Figure 1. While naïvely learned surrogate functions usually produces poor-performing, out-of-distribution designs that appear quite optimistic under the learned surrogate (Kumar & Levine, 2019; Brookes et al., 2019; Trabucco et al., 2021a). The robust surrogate in PRIME is explicitly trained to prevent overestimation on "adversarial" designs that would be found during optimization. Furthermore, in contrast to prior works that discard infeasible points (Hegde et al., 2021; Trabucco et al., 2021a), our proposed method instead incorporates infeasible points when learning the conservative surrogate by treating them as additional negative samples. During evaluation, PRIME optimizes the learned conservative surrogate using a discrete optimizer.

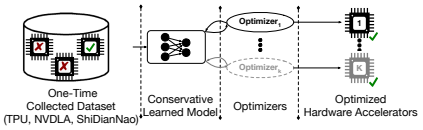


Figure 1: Overview of PRIME. We use a one-time collected dataset of prior hardware accelerator designs, including TPU-style (Yazdanbakhsh et al., 2021b), NVDLA-style (Nvidia, 2021a), and ShiDianNao-style (Du et al., 2015) accelerators to train a conservative surrogate model, which can then be used to design accelerators to meet desired goals and constraints.

Our results show that PRIME architects hardware accelerators that improve over the best design in the training dataset, on average, by 2.46× (up to 6.7×) when specializing for a single application. In this case, PRIME also improves over the best conventional simulator-driven optimization methods by 1.54× (up to 6.6×). These performance improvements are obtained while reducing the total simulation time to merely 7% and 1% of that of the simulator-driven methods for single-task and multi-task optimization, respectively. More importantly, a contextual version of PRIME can design accelerators that are *jointly optimal* for a set of *nine* applications without requiring any additional domain information. In this challenging setting, PRIME improves over simulator-driven methods, which tend to scale poorly as more applications are added, by 1.38×. Finally, we show that the surrogates trained with PRIME on a set of training applications can be readily used to obtain accelerators for *unseen* target applications, without any retraining on the new application. Even in this *zero-shot* optimization scenario, PRIME outperforms simulator-based methods that require re-training and active simulation queries by up to 1.67×. In summary, PRIME allows us to effectively address the shortcomings of simulation-driven approaches, (1) significantly reduces the simulation time, (2) enables data reuse and enjoys generalization properties, and (3) does not require domain-specific engineering or search space pruning. To facilitate further research in leveraging machine learning in architecting hardware accelerators, we also release the dataset in our experiments (Appendix B.3), consisting of many accelerator design points (both infeasible and feasible).

2 BACKGROUND ON HARDWARE ACCELERATORS

The goal of specialized hardware accelerators—Google TPUs (Jouppi et al., 2017; Gupta & Akin, 2020), Nvidia GPUs (Nvidia, 2021b), GraphCore (GraphCore, 2021)—is to improve the performance of specific applications, such as machine learning models. To design such accelerators, architects typically create a parameterized design and sweep over parameters using simulation.

Target hardware accelerators. Our primary evaluation uses an industry-grade and highly parameterized template-based accelerator following prior work (Yazdanbakhsh et al., 2021b). This template enables architects to determine the organization of various components, such as compute units, memory cells, memory, etc., by searching for these configurations in a discrete design space. Some ML applications may have large memory requirements (e.g., large language models (Brown et al., 2020)) demanding sufficient on-chip memory resources, while others may benefit from more compute blocks. The hardware design workflow directly selects the values of these parameters. In addition to this accelerator and to further show the generality of our method to other accelerator design problems, we evaluate two distinct dataflow accelerators with different search spaces, namely

NVDLA-style (Nvidia, 2021a) and ShiDianNao-style (Du et al., 2015) from Kao et al. (2020) (See Section 6 and Appendix C for a detailed discussion; See Table 6 for results).

How does an accelerator work? We briefly explain the computation flow on our template-based accelerators (Figure 2) and refer the readers to Appendix C for details on other accelerators. This template-based accelerator is a 2D array of processing elements (PEs). Each PE is capable of performing matrix multiplications in a single instruction multiple data (SIMD) paradigm (Gove et al., 1993). A controller orchestrates the data transfer (both activations and model parameters) between off-chip DRAM memory and the on-chip buffers and also reads in and manages the instructions (e.g. convolution, pooling, etc.) for execution. The computation stages on such accelerators start by sending a set of activations to the compute lanes, executing them in SIMD manner, and either storing the partial computation results or offloading them back into off-chip memory. Compared to prior works (Hegde et al., 2021; Du et al., 2015; Kao et al., 2020), this parameterization is unique—it includes multiple compute lanes per each PE and enables SIMD execution model within each compute lane—and yields a distinct accelerator search space accompanied by an end-to-end simulation framework. Appendix C elaborates on other accelerators evaluated in this work.

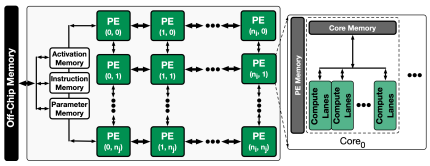


Figure 2: An industry-level machine learning accelerator Yazdanbakhsh et al. (2021b).

3 Problem Statement, Training Data and Evaluation Protocol

Our template-based parameterization maps the accelerator, denoted as $\mathbf{x}$, to a discrete design space, $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$, and each $\mathbf{x}_i$ is a discrete-valued variable representing one component of the microarchitectural template, as shown in Table 1 (See Appendix C for the description of other accelerator search spaces studied in our work). A design maybe be infeasible due to various reasons, such as a compilation failure or the limitations of physical implementation, and we denote the set of all such feasibility criterion as $\text{Feasible}(\mathbf{x})$. The feasibility criterion depends on both the target software and the underlying hardware, and it is not easy to identify if a given $\mathbf{x}$ is infeasible without explicit simulation. We will require our optimization procedure to not only learn the value of the objective function but also to learn to navigate through a sea of infeasible solutions to high-performing feasible solutions $\mathbf{x}^*$ satisfying $\text{Feasible}(\mathbf{x}^*) = 1$.

Our training dataset $\mathcal{D}$ consists of a modest set of accelerators $\mathbf{x}_i$ that are randomly sampled from the design space and evaluated by the hardware simulator. We partition the dataset $\mathcal{D}$ into two subsets, $\mathcal{D}_{\text{feasible}}$ and $\mathcal{D}_{\text{infeasible}}$. Let $f(\mathbf{x})$ denote the desired objective (e.g., latency, power, etc.) we intend to optimize over the space of accelerators $\mathbf{x}$. We do not possess functional access to $f(\mathbf{x})$, and the optimizer can only access $f(\mathbf{x})$ values for accelerators $\mathbf{x}$ in the feasible partition of the data, $\mathcal{D}_{\text{feasible}}$. For all infeasible accelerators, the simulator does not provide any value of $f(\mathbf{x})$. In addition to satisfying feasibility, the optimizer must handle explicit constraints on parameters such as area and power (Flynn & Luk, 2011). In our applications, we impose an explicit area constraint, $\text{Area}(\mathbf{x}) \leq \alpha$, though additional explicit constraints are also possible. To account for different constraints, we formulate this task as a constrained optimization problem. Formally:

$$\begin{aligned} &\min_{\mathbf{x}} \; f(\mathbf{x}) \quad \text{s.t.} \quad \text{Area}(\mathbf{x}) \leq \alpha, \quad \text{Feasible}(\mathbf{x}) = 1 \\ &\qquad \text{on} \;\; \mathcal{D} = \mathcal{D}_{\text{feasible}} \cup \mathcal{D}_{\text{infeasible}} = \{(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_N, \mathbf{y}_N)\} \cup \{\mathbf{x}'_1, \cdots, \mathbf{x}'_{N'}\} \end{aligned} \tag{1}$$

While Equation 1 may appear similar to other standard black-box optimization problems, solving it over the space of accelerator designs is challenging due to the large number of infeasible points, the need to handle explicit design constraints, and the difficulty in navigating the non-smooth landscape (See Figure 3 and Figure 10 in the Appendix) of the objective function.

What makes optimization over accelerators challenging? Compared to other domains where model-based optimization methods have been applied (Brookes et al., 2019; Trabucco et al., 2021a), optimizing accelerators introduces a number of practical challenges. First, accelerator design spaces typically feature a narrow manifold of feasible accelerators within a sea of infeasible points (Nardi et al., 2019; Shi et al., 2020; Gelbart et al., 2014), as visualized in Figure 3 and Appendix (Figure 11). While some of these infeasible points can be identified via simple rules (e.g. estimating chip area usage), most infeasible points correspond to failures during compilation or hardware simulation.

Table 1: The accelerator design space parameters for the primary accelerator search space targeted in this work. The maximum possible number of accelerator designs (including feasible and infeasible designs) is 452,760,000. PRIME only uses a small randomly sampled subset of the search space.

Accelerator Parameter	# discrete values	Accelerator Parameter	# discrete values
# of PEs-X	10	# of PEs-Y	10
PE Memory	7	# of Cores	7
Core Memory	11	# of Compute Lanes	10
Instruction Memory	4	Parameter Memory	5
Activation Memory	7	DRAM Bandwidth	6

Table 2: The description of the applications, their domains, number of (convolutions, depth-wise convolutions, feed-forward) XLA ops, model parameter size, instruction sizes in bytes, number of compute operations.

Name	Domain	# of XLA Ops (Conv, D/W, FF)	Model Param	Instr. Size	# of Compute Ops.
MobileNetEdge	Image Class.	(45, 13, 1)	3.87 MB	476,736	1,989,811,168
MobileNetV2	Image Class.	(35, 17, 1)	3.31 MB	416,032	609,353,376
MobileNetV3	Image Class.	(32, 15, 17)	5.20 MB	1,331,360	449,219,600
M4	Object Det.	(32, 13, 2)	6.23 MB	317,600	3,471,920,128
M5	Object Det.	(47, 27, 0)	2.16 MB	328,672	939,752,960
M6	Object Det.	(53, 33, 2)	0.41 MB	369,952	228,146,848
U-Net	Image Seg.	(35, 0, 0)	3.69 MB	224,992	13,707,214,848
t-RNN Dec	Speech Rec.	(0, 0, 19)	19 MB	915,008	40,116,224
t-RNN Enc	Speech Rec.	(0, 0, 18)	21.62 MB	909,696	45,621,248

These infeasible points are generally not straightforward to formulate into the optimization problem and requires simulation (Shi et al., 2020; Parashar et al., 2019; Yazdanbakhsh et al., 2021a).

Second, the optimization objective can exhibit high sensitivity to small variations in some architecture parameters (Figure 10b) in some regions of the design space, but remain relatively insensitive in other parts, resulting in a complex optimization landscape. This suggests that optimization algorithms based on local parameter updates (e.g., gradient ascent, evolutionary schemes, etc.) may have a challenging task traversing the nearly flat landscape of the objective, which can lead to poor performance.

Training dataset. We used an offline dataset $\mathcal{D}$ of (accelerator parameters, latency) via random sampling from the space of 452M possible accelerator configurations. Our method is only provided with a relatively modest set of feasible points ($\leq$ 8000 points) for training, and these points are the *worst-performing* feasible points across the pool of randomly sampled data. This dataset is meant to reflect an easily obtainable and an application-agnostic dataset of accelerators that could have been generated once and stored to disk, or might come from real physical experiments. We emphasize that no assumptions or domain knowledge about the application use case was made during dataset collection. Table 2 depicts the list of target applications, evaluated in this work, includes three variations of MobileNet (Gupta & Akin, 2020; Sandler et al., 2018; Howard et al., 2019), three in-house industry-level models for object detection (M4, M5, M6; names redacted to prevent anonymity violation), a U-net model (Ronneberger et al., 2015), and two RNN-based encoder-decoder language models (Graves, 2012; He et al., 2019; Sainath et al., 2020; Li et al., 2021). These applications span the gamut from small models, such as M6, with only 0.4 MB model parameters that demands less on-chip memory, to the medium-sized models ($\geq$ 5 MB), such as MobileNetV3 and M4 models, and large models ($\geq$ 19 MB), such as t-RNNs, hence requiring larger on-chip memory.

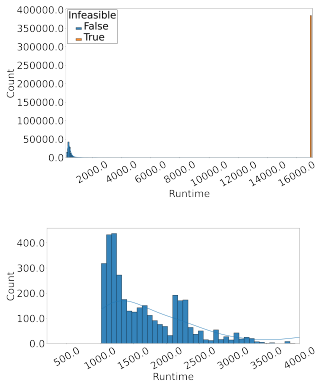


Figure 3: Top: histogram of infeasible (right orange bar with large score values) and feasible (left cluster of bars) data points; **Bottom:** zoomed-in histogram focused on feasible points highlighting the variable latencies.

Evaluation protocol. To compare state-of-the-art simulator-driven methods and our data-driven method, we limit the number of feasible points (costly to evaluate) that can be used by any algorithm to equal amounts. We still provide infeasible points to any method and leave it up to the optimization method to use it or not. This ensures our comparisons are fair in terms of the amount of data available to each method. However, it is worthwhile to note that in contrast to our method where *worse-quality* data points from small offline dataset are used, the simulator-driven methods have an inherent advantage because they can steer the query process towards the points that are more likely to be better in terms of performance. Following prior work (Brookes et al., 2019; Trabucco et al.,

2021a;b), we evaluate each run of a method by first sampling the top $n = 256$ design candidates according to the algorithm's predictions, evaluating all of these under the ground truth objective function and recording the performance of the best accelerator design. The final reported results is the median of ground truth objective values across five independent runs.

4 PRIME: ARCHITECTING ACCELERATORS VIA CONSERVATIVE SURROGATES

As shown in Figure 4, our method first learns a conservative surrogate model of the optimization objective using the offline dataset. Then, it optimizes the learned surrogate using a discrete optimizer. The optimization process does not require access to a simulator, nor to real-world experiments beyond the initial dataset, except when evaluating the final top-performing $n = 256$ designs (Section 3).

Learning conservative surrogates using logged offline data. Our goal is to utilize a logged dataset of feasible accelerator designs labeled with the desired performance metric (e.g., latency), $\mathcal{D}_{\text{feasible}}$, and infeasible designs, $\mathcal{D}_{\text{infeasible}}$ to learn a mapping $f_\theta : \mathcal{X} \to \mathbb{R}$, that maps the accelerator configuration $\mathbf{x}$ to its corresponding metric y. This learned surrogate can then be optimized by the optimizer. While a straightforward approach for learning such a mapping is to train it via supervised regression, by minimizing the mean-squared error $\mathbb{E}_{\mathbf{x}_i, y_i \sim \mathcal{D}}[(f_\theta(\mathbf{x}_i) - y_i)^2]$, prior work (Kumar & Levine, 2019; Kumar et al., 2020; Trabucco et al., 2021a) has shown that such predictive models can arbitrarily overestimate the value of an unseen input $\mathbf{x}_i$. This can cause the optimizer to find a solution $\mathbf{x}^*$ that performs poorly in the simulator but looks promising under the learned model. We empirically validate this overestimation hypothesis and find it to confound the optimizer in on our problem domain as well (See Figure 12 in Appendix). To prevent overestimated values at unseen inputs from confounding the optimizer, we build on COMs (Trabucco et al., 2021a) and train $f_\theta(\mathbf{x})$ with an additional term that explicitly maximizes the function value $f_\theta(\mathbf{x})$ at unseen $\mathbf{x}$ values. Such unseen designs $\mathbf{x}$, where the learned function $f_\theta(\mathbf{x})$ is likely to be overestimated, are "negative mined" by running a few iterations of an approximate stochastic optimization procedure that aims to maximize f_θ in the inner loop. This procedure is analogous to adversarial training (Goodfellow et al., 2014). Equation 2 formalizes this objective:

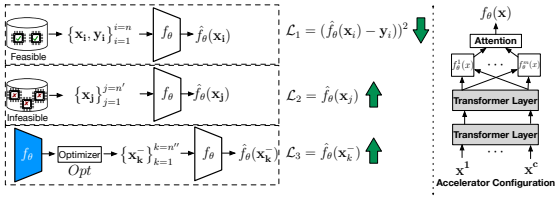


Figure 4: Overview of PRIME which trains a conservative surrogate $f_\theta(\mathbf{x}_i)$ using Equation 3. Our neural net model for $f_\theta(\mathbf{x})$ utilizes two transformer layers (Vaswani et al., 2017), and a multi-headed architecture which is pooled via a soft-attention layer.

$$\theta^* := \arg\min_\theta \; \mathcal{L}(\theta) := \mathbb{E}_{\mathbf{x}_i, y_i \sim \mathcal{D}_{\text{feasible}}} \left[(f_\theta(\mathbf{x}_i) - y_i)^2 \right] - \alpha \mathbb{E}_{\mathbf{x}_i^- \sim \text{Opt}(f_\theta)} \left[f_\theta(\mathbf{x}_i^-) \right]. \quad (2)$$

$\mathbf{x}_i^-$ denotes the negative samples produced from an optimizer $\text{Opt}(\cdot)$ that attempts to maximize the current learned model, f_θ. We will discuss our choice of Opt in the Appendix Section B.

Incorporating design constraints via infeasible points. While prior work (Trabucco et al., 2021a) simply optimizes Equation 2 to learn a surrogate, this is not enough when optimizing over accelerators, as we will also show empirically (Appendix A.1). This is because explicit negative mining does not provide any information about accelerator design constraints. Fortunately, this information is provided by infeasible points, $\mathcal{D}_{\text{infeasible}}$. The training procedure in Equation 2 provides a simple way to do incorporate such infeasible points: we simply incorporate $\mathbf{x}_i' \sim \mathcal{D}_{\text{infeasible}}$ as additional negative samples and maximize the prediction at these points. This gives rise to our final objective:

$$\min_\theta \; \mathcal{L}^{\text{inf}}(\theta) := \mathcal{L}(\theta) - \beta \mathbb{E}_{\mathbf{x}_i' \sim \mathcal{D}_{\text{infeasible}}} \left[f_\theta(\mathbf{x}_i') \right] \quad (3)$$

Multi-model optimization and zero-shot generalization. One of the central benefits of a data-driven approach is that it enables learning powerful surrogates that generalize over the space of applications, potentially being effective for new unseen application domains. In our experiments, we evaluate PRIME on designing accelerators for multiple applications denoted as $k = 1, \cdots, K$, jointly or for a novel unseen application. In this case, we utilized a dataset $\mathcal{D} = \{\mathcal{D}_1, \cdots, \mathcal{D}_K\}$, where each $\mathcal{D}_k$ consists of a set of accelerator designs, annotated with the latency value and the feasibility criterion for a given application k. While there are a few overlapping designs in different parts of the dataset annotated for different applications, most of the designs only appear in one part. To train a single conservative surrogate $f_\theta(\mathbf{x})$ for multiple applications, we extend the training procedure

in Equation 3 to incorporate *context vectors* $\mathbf{c}_k \in \mathbb{R}^d$ for various applications driven by a list of application properties in Table 2. The learned function in this setting is now conditioned on the context $f_\theta(\mathbf{x}, \mathbf{c}_k)$. We train f_θ via the objective in Equation 3, but in expectation over all the contexts and their corresponding datasets: $\min_\theta \mathbb{E}_{k\sim[K]} \left[\mathcal{L}_k^{\text{inf}}(\theta)\right]$. Once such a contextual surrogate is learned, we can either optimize the average surrogate across a set of contexts $\{\mathbf{c}_i, \mathbf{c}_2, \cdots, \mathbf{c}_n\}$ to obtain an accelerator that is optimal for multiple applications simultaneously on an average ("multi-model" optimization), or optimize this contextual surrogate for a novel context vector, corresponding to an unseen application ("zero-shot" generalization). In this case, PRIME is not allowed to train on any data corresponding to this new unseen application. While such zero-shot generalization might appear surprising at first, note that the context vectors are not simply one-hot vectors, but consist of parameters with semantic information, which the surrogate can generalize over.

Optimizing the learned conservative surrogate. Prior work (Yazdanbakhsh et al., 2021a) has shown that the most effective optimizers for accelerator design are meta-heuristic/evolutionary optimizers. We therefore choose to utilize, firefly (Yang, 2010; Yang & Deb, 2010; Liu et al., 2013) to optimize our conservative surrogate. This algorithm maintains a set of optimization candidates (a.k.a. "fireflies") and jointly update them towards regions of low objective value, while adjusting their relative distances appropriately to ensure multiple high-performing, but diverse solutions. We discuss additional details in Appendix B.1.

Cross validation: which model and checkpoint should we evaluate? Similarly to supervised learning, models trained via Equation 3 can overfit, leading to poor solutions. Thus, we require a procedure to select which hyperparameters and checkpoints should actually be used for the design. This is crucial, because we cannot arbitrarily evaluate as many models as we want against the simulator. While effective methods for model selection have been hard to develop in offline optimization (Trabucco et al., 2021a;b), we devised a simple scheme using a validation set for choosing the values of α and β (Equation 3), as well as which checkpoint to utilize for generating the design. For each training run, we hold out the best 20% of the points out of the training set and use them *only* for cross-validation as follows. Typical cross-validation strategies in supervised learning involve tracking validation error (or risk), but since our model is trained conservatively, its predictions may not match the ground truth, making such validation risk values unsuitable for our use case. Instead, we track Kendall's ranking correlation between the predictions of the learned model $f_\theta(\mathbf{x}_i)$ and the ground truth values y_i (Appendix B) for the held-out points for each run. We pick values of α, β and the checkpoint that attain the highest validation ranking correlation. We present the pseudo-code for PRIME (Algorithm 1) and implementation details in Appendix B.1.

5 RELATED WORK

Optimizing hardware accelerators has become more important recently. Prior works (Parsa et al., 2020; Iqbal et al., 2020; Sun et al., 2020; Nardi et al., 2019; Cong et al., 2018; Koeplinger et al., 2018; Balaprakash et al., 2016; Ansel et al., 2014; Hegde et al., 2021) mainly rely on expensive-to-query hardware simulators to navigate the search space. For example, HyperMapper (Nardi et al., 2019) targets compiler optimization for FPGAs by continuously interacting with the simulator in a design space with relatively few infeasible points. Mind Mappings (Hegde et al., 2021), optimizes software mappings to a fixed hardware provided access to millions of feasible points and throws away infeasible points during learning. Kao et al. (2020) utilizes reinforcement learning against a simulator to optimize the parameters of a set of simple accelerators. In contrast, our data-driven approach, PRIME , does not only learn a conservative surrogate using offline data but can also effectively leverage information from the large number of infeasible points and is effective with just a few thousand feasible points. In addition, to the best of our knowledge, our work, is the first to demonstrate generalization to unseen applications for accelerator design, outperforming state-of-the-art online methods.

A popular approach for solving black-box optimization problems is model-based optimization (MBO) (Snoek et al., 2015; Shahriari et al., 2016; Snoek et al., 2012). Most of these methods fail to scale to high-dimensions, and have been extended with neural networks (Snoek et al., 2015; 2012; Kim et al., 2019; Garnelo et al., 2018b;a; Angermueller et al., 2020a; 2019; Mirhoseini et al., 2020). While these methods work well in the active setting, they are susceptible to out-of-distribution inputs (Trabucco et al., 2021b) in the offline, data-driven setting. To prevent this, offline MBO methods that constrain the optimizer to the manifold of valid, in-distribution inputs have been

developed Brookes et al. (2019); Fannjiang & Listgarten (2020); Kumar & Levine (2019). However, modeling the manifold of valid inputs can be challenging for accelerators. PRIME dispenses with the need for generative modeling, while still avoiding out-of-distribution inputs. PRIME builds on "conservative" offline RL and offline MBO methods that train robust surrogates (Kumar et al., 2020; Trabucco et al., 2021a). However, unlike these approaches, PRIME can handle constraints by learning from infeasible data and utilizes a better optimizer (See Appendix Table 7 for a comparison). In addition, while prior works area mostly restricted to a single application, we show that PRIME is effective in multi-task optimization and zero-shot generalization.

6 EXPERIMENTAL EVALUATION

Our evaluations aim to answer the following questions: **Q(1)** Can PRIME design accelerators tailored for a given application that are better than the best observed configuration in the training dataset, and comparable to or better than state-of-the-art simulation-driven methods under a given simulator-query budget? **Q(2)** Does PRIME reduce the total simulation time compared to other methods? **Q(3)** Can PRIME produce hardware accelerators for a family of different applications? **Q(4)** Can PRIME trained for a family of applications extrapolate to designing a high-performing accelerator for a new, unseen application, thereby enabling data reuse? Additionally, we ablate various properties of PRIME (Appendix A.6) and evaluate its efficacy in designing accelerators with distinct dataflow architectures, with a larger search space (up to 2.5×10^{114} possible candidates).

Baselines and comparisons. We compare PRIME against three online optimization methods that actively query the simulator: **(1)** evolutionary search with the firefly optimizer (Yazdanbakhsh et al., 2021a) ("Evolutionary"), which is the shown to outperform other online methods for accelerator design; **(2)** Bayesian Optimization ("Bayes Opt") (Golovin et al., 2017), **(3)** MBO (Angermueller et al., 2019). In all the experiments, we grant all the methods the same number of feasible points. Note that our method do not get to select these points, and use the same exact offline points across all the runs, while the online methods can actively select which points to query, and therefore require new queries for every run. "$\mathcal{D}$(Best in Training)" denotes the best latency value in the training dataset used in PRIME. We also present ablation results with different components of our method removed in Appendix A.6, where we observe that utilizing both infeasible points and negative sampling are generally important for attaining good results. Appendix A.1 presents additional comparisons to COMs Trabucco et al. (2021a)—which only obtains negative samples via gradient ascent on the learned surrogate and does not utilize infeasible points—and P3BO Angermueller et al. (2020b)—an state-of-the-art online method in biology.

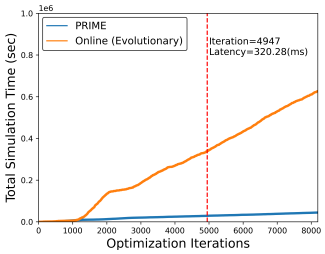


Figure 5: Comparing the total simulation time needed by PRIME and online evolutionary optimization on MobileNetEdge. Note that optimizing PRIME only requires about **7%** of the total simulation time of the online method.

Architecting application-specific accelerators. We first evaluate PRIME in designing specialized accelerators for each of the applications in Table 2. We train a conservative surrogate using the method in Section 4 on the logged dataset for each application separately. The area constraint α (Equation 1) is set to $\alpha = 29$ mm^2, a realistic budget for accelerators (Yazdanbakhsh et al., 2021a). Table 3 summarizes the results. On average, the best accelerators designed by PRIME outperforms the best accelerator configuration in the training dataset (last row Table 3), by 2.46×. PRIME also outperforms the accelerators in the best online method by 1.54× (up to 5.80× and 6.62× in t-RNN Dec and t-RNN Enc, respectively). Moreover, perhaps surprisingly, PRIME generates accelerators that are better than all the online optimization methods in 7/9 domains, and performs on par in several other scenarios (on average only 6.8% slowdown compared to the best accelerator with online methods in M5 and M6). These results indicates that offline optimization of accelerators using PRIME can be more data-efficient compared to online methods with active simulation queries. To answer **Q(2)**, we compare the total simulation time of PRIME and the best evolutionary approach from Table 3 on the MobileNetEdge domain. On average, not only that PRIME outperforms the best online method, but also considerably reduces the total simulation time by 93%, as shown in Figure 5. Even the total simulation time to the first occurrence of the final design that is eventually returned by the online methods is about 11× what PRIME requires to fine a better design. This indicates that data-driven PRIME is much more preferred in terms of the performance-time trade-off.

Table 3: Optimized objective values (i.e., latency in milliseconds) obtained by various methods for the task of learning accelerators specialized to a given application. Lower latency is better. **From left to right**: our method, online Bayesian optimization ("Bayes Opt"), online evolutionary algorithm ("Evolutionary"), and the best design in the training dataset. On average (last row), PRIME improves over the best in the dataset by 2.46× (up to 6.69× in t-RNN Dec) and outperforms best online optimization methods by 1.54× (up to 6.62× in t-RNN Enc). The best accelerator configurations identified is highlighted in bold.

Application	PRIME	Online Optimization			$\mathcal{D}$ (Best in Training)
		Bayes Opt	Evolutionary	MBO	
MobileNetEdge	**298.50**	319.00	320.28	332.97	354.13
MobileNetV2	**207.43**	240.56	238.58	244.98	410.83
MobileNetV3	**454.30**	534.15	501.27	535.34	938.41
M4	**370.45**	396.36	383.58	405.60	779.98
M5	208.21	201.59	**198.86**	219.53	449.38
M6	131.46	121.83	120.49	**119.56**	369.85
U-Net	**740.27**	872.23	791.64	888.16	1333.18
t-RNN Dec	**132.88**	771.11	770.93	771.70	890.22
t-RNN Enc	**130.67**	865.07	865.07	866.28	584.70
Geomean of PRIME's Improvement	1.0×	**1.58×**	**1.54×**	**1.61×**	**2.46×**

Table 4: Optimized average latency (the lower, the better) across multiple applications (up to ten applications) from diverse domains by PRIME and best online algorithms (Evolutionary and MBO) under different area constraints. Each row show the (Best, Median) of average latency across five runs. The geometric mean of PRIME's improvement over other methods (last row) indicates that PRIME is at least 21% better.

Applications	Area	PRIME (Ours)	Evolutionary (Online)	MBO (Online)
MobileNet (Edge,V2,V3)	29 mm^2	(310.21, 334.70)	(**315.72**, **325.69**)	(342.02, 351.92)
MobileNet (V2,V3), M5, M6	29 mm^2	(**268.47**, **271.25**)	(288.67, 288.68)	(295.21, 307.09)
MobileNet (Edge, V2, V3), M4, M5, M6	29 mm^2	(**311.39**, **313.76**)	(314.31, 316.65)	(321.48, 339.27)
MobileNet (Edge, V2, V3), M4, M5, M6, U-Net, t-RNN-Enc	29 mm^2	(**305.47, 310.09**)	(404.06, 404.59)	(404.06, 412.90)
MobileNet (Edge, V2, V3), M4, M5, M6, t-RNN-Enc	100 mm^2	(**286.45, 287.98**)	(404.25, 404.59)	(404.06, 404.94)
MobileNet (Edge, V2, V3), M4, M5, M6, t-RNN (Dec,Enc)	29 mm^2	(**426.65**, **426.65**)	(586.55, 586.55)	(626.62, 692.61)
MobileNet (Edge, V2, V3), M4, M5, M6, U-Net, t-RNN (Dec,Enc)	100 mm^2	(**383.57**, **385.56**)	(518.58, 519.37)	(526.37, 530.99)
Geomean of PRIME's Improvement	—	**(1.0×, 1.0×)**	**(1.21×, 1.20×)**	**(1.24×, 1.27×)**

Architecting accelerators for multiple applications. To answer **Q(3)**, we evaluate the efficacy of the contextual version of PRIME in designing an accelerator that attains the lowest latency averaged over a set of application domains. As discussed previously, the training data used does not label a given accelerator with latency values corresponding to each application, and thus, PRIME must extrapolate accurately to estimate the latency of an accelerator for a context it is not paired with in the training dataset. This also means that PRIME cannot simply return the accelerator with the best average latency and must run non-trivial optimization. We evaluate our method in seven different scenarios (Table 4), comprising various combinations of models from Table 2 and under different area constraints, where the smallest set consists of the three MobileNet variants and the largest set consists of nine models from image classification, object detection, image segmentation, and speech recognition. This scenario is also especially challenging for online methods since the number of jointly feasible designs is expected to drop significantly as more applications are added. For instance, for the case of the MobileNet variants, the training dataset only consists of a few (20-30) accelerator configurations that are jointly feasible and high-performing (Appendix B.2—Figure 8).

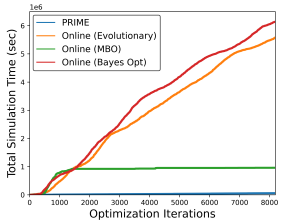


Figure 6: Comparing the total simulation time needed by PRIME and online methods on seven models (Area $\leq$ 100mm^2) . PRIME only requires about 1%, 6%, and 0.9% of the total simulation time of Evolutionary, MBO, and Bayes Opt, respectively, although PRIME outperforms the best online method by 41%.

Table 4 shows that, on average, PRIME finds accelerators that outperform the best online method by 1.2× (up to 41%). While PRIME performs similar to online methods in the smallest three-model scenario (first row), it outperforms online methods as the number of applications increases and the set of applications become more diverse. In addition, comparing with the best jointly feasible design point across the target applications, PRIME finds significantly better accelerators (3.95×). Finally, as the number of model increases the total simulation time difference between online methods and PRIME further widens (Figure 6). These results indicate that PRIME is effective in designing accelerators jointly optimized across multiple applications while reusing the same dataset as for the single-task, and scales more favorably than its simulation-driven counterparts. Appendix A.4 expounds on the details of the designed accelerators for nine applications, comparing our method and the best online method.

Table 5: Optimized objective values (i.e., latency in milliseconds) under zero-shot setting. Lower latency is better. From left to right: the applications used to train the surrogate model in PRIME the target applications for which the accelerator is optimized for, the area constraint of the accelerator, PRIME's (best, median) latency, and best online method's (best, median) latency. PRIME does not use any additional data from the target applications. On average (last row), PRIME yields optimized accelerator for target applications (with zero query to the target applications' dataset) with 1.26× (up to 1.66×) lower latency over the best online method. The best accelerator configurations identified is highlighted in bold.

Train Applications	Test Applications	Area	PRIME	Evolutionary (Online)
MobileNet (Edge, V3)	MobileNetV2	29 mm^2	(**311.39**, **313.76**)	(314.31, 316.65)
MobileNet (V2, V3), M5, M6	MobileNetEdge, M4	29 mm^2	(357.05, 364.92)	(**354.59**, **357.29**)
MobileNet (Edge,V2,V3), M4, M5, M6, t-RNN Enc	U-Net, t-RNN Dec	29 mm^2	(**745.87**, **745.91**)	(1075.91, 1127.64)
MobileNet (Edge,V2,V3),M4, M5, M6, t-RNN Enc	U-Net, t-RNN Dec	100 mm^2	(**517.76**, **517.89**)	(859.76, 861.69)
Geomean of PRIME's Improvement	—	—	(1.0×, 1.0×)	(**1.24**×, **1.26**×)

Table 6: Optimized objective values (i.e. total number of cycles) for two different dataflow architectures, NVDLA-style (Nvidia, 2021a) and ShiDianNao-style (Du et al., 2015), across three classes of applications. The maximum search space for the studied accelerators are $\approx 2.5\times10^{114}$. PRIME generalizes to other classes of accelerators with larger search space and outperforms the best online method by 1.06× and the best data seen in training by 3.75× (last column). The best accelerator configurations is highlighted in bold.

Applications	Dataflow	PRIME	Evolutionary (Online)	$\mathcal{D}$ (Best in Training)
MobileNetV2	NVDLA	$\mathbf{2.51\times10^7}$	2.70×10^7	1.32×10^8
MobileNetV2	ShiDianNao	$\mathbf{2.65\times10^7}$	2.84×10^7	1.27×10^8
ResNet50	NVDLA	$\mathbf{2.83\times10^8}$	3.13×10^8	1.63×10^9
ResNet50	ShiDianNao	$\mathbf{3.44\times10^8}$	3.74×10^8	2.05×10^9
Transformer	NVDLA	$\mathbf{7.8\times10^8}$	$\mathbf{7.8\times10^8}$	1.3×10^9
Transformer	ShiDianNao	$\mathbf{7.8\times10^8}$	$\mathbf{7.8\times10^8}$	1.5×10^9
Geomean of PRIME's Improvement	—	**1.0**×	**1.06**×	**3.75**×

Accelerating previously unseen applications ("zero-shot" optimization). Finally, we answer **Q(4)** by demonstrating that our data-driven offline method, PRIME enables effective data reuse by using logged accelerator data from a set of applications to design an accelerator for an unseen new application, without requiring any training on data from the new unseen application(s). We train a contextual version of PRIME using a set of "training applications" and then optimize an accelerator using the learned surrogate with different contexts corresponding to "test applications," without any additional query to the test application dataset. Table 5 shows, on average, PRIME outperforms the best online method by 1.26× (up to 66%) and only 2% slowdown in 1/4 cases. Note that the difference in performance increases as the number of training applications increases. These results show the effectiveness of PRIME in the zero-shot setting (more results in Appendix A.5).

Applying PRIME on other accelerator architectures and dataflows. Finally, to test the the generalizability of PRIME to other accelerator architectures Kao et al. (2020), we evaluate PRIME to optimize latency of two style of dataflow accelerators—NVDLA-style and ShiDianNao-style—across three applications (Appendix C details the methodology). As shown in Table 6, PRIME outperforms the online evolutionary method by 6% and improves over the best point in the training dataset by 3.75×. This demonstrates that PRIME is effective in optimizing accelerators with different dataflow architectures and can successfully optimize over extremely large hardware search spaces, just by learning from offline data.

7 DISCUSSION

In this work, we present a data-driven offline optimization method, PRIME to automatically architect hardware accelerators. Our method learns a conservative surrogate of the objective function by leveraging infeasible data points to better model the desired objective function of the accelerator using a one-time collected dataset of accelerators, thereby alleviating the need for time-consuming simulation. Our results show that, on average, our method outperforms the best designs observed in the logged data by 2.46× and improves over the best simulator-driven approach by about 1.54×. In the more challenging setting of designing accelerators jointly optimal for multiple applications or for new, unseen applications, zero-shot, PRIME outperforms simulator-driven methods by 1.2×, while reducing the total simulation time by 99%. Utilizing PRIME for software-hardware co-optimization is an interesting avenue for future work.

REPRODUCIBILITY STATEMENT

We provide complete experimental details regarding the application domains, accelerators considered, hyperparameters for our method and our cross-validation strategy in Appendix B. Additionally, we also open-source our dataset, in anonymous folder with this submission and provide python notebooks to parse and utilize the dataset. More information is provided in Appendix B.3. We also open source the code to ensure the reproducibility of our results.

REFERENCES

Christof Angermueller, David Dohan, David Belanger, Ramya Deshpande, Kevin Murphy, and Lucy Colwell. Model-based reinforcement learning for biological sequence design. In *International Conference on Learning Representations*, 2019.

Christof Angermueller, David Belanger, Andreea Gane, Zelda Mariet, David Dohan, Kevin Murphy, Lucy Colwell, and D Sculley. Population-based black-box optimization for biological sequence design. *arXiv preprint arXiv:2006.03227*, 2020a.

Christof Angermueller, David Belanger, Andreea Gane, Zelda Mariet, David Dohan, Kevin Murphy, Lucy Colwell, and D Sculley. Population-based black-box optimization for biological sequence design. *arXiv preprint arXiv:2006.03227*, 2020b.

Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. OpenTuner: An extensible framework for program autotuning. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pp. 303–316, 2014.

Prasanna Balaprakash, Ananta Tiwari, Stefan M Wild, Laura Carrington, and Paul D Hovland. AutoMOMML: Automatic multi-objective modeling with machine learning. In *International Conference on High Performance Computing*, pp. 219–239. Springer, 2016.

David Brookes, Hahnbeom Park, and Jennifer Listgarten. Conditioning by adaptive sampling for robust design. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2019. URL http://proceedings.mlr.press/v97/brookes19a.html.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Prasanth Chatarasi, Hyoukjun Kwon, Natesh Raina, Saurabh Malik, Vaisakh Haridas, Tushar Krishna, and Vivek Sarkar. MARVEL: A decoupled model-driven approach for efficiently mapping convolutions on spatial dnn accelerators. *arXiv preprint arXiv:2002.07752*, 2020.

Jason Cong, Peng Wei, Cody Hao Yu, and Peng Zhang. Automated accelerator generation and optimization with composable, parallel and pipeline architecture. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2018.

Shail Dave, Youngbin Kim, Sasikanth Avancha, Kyoungwoo Lee, and Aviral Shrivastava. Dmazerunner: Executing perfectly nested loops on dataflow accelerators. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–27, 2019.

Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. ShiDianNao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pp. 92–104, 2015.

Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *2011 38th Annual international symposium on computer architecture (ISCA)*, pp. 365–376. IEEE, 2011.

Clara Fannjiang and Jennifer Listgarten. Autofocused oracles for model-based design. *arXiv preprint arXiv:2006.08052*, 2020.

Michael J Flynn and Wayne Luk. *Computer system design: system-on-chip*. John Wiley & Sons, 2011.

Justin Fu and Sergey Levine. Offline model-based optimization via normalized maximum likelihood estimation. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=FmMKSO4e8JK.

Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and S. M. Ali Eslami. Conditional neural processes. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018a.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J. Rezende, S. M. Ali Eslami, and Yee Whye Teh. Neural processes. *CoRR*, abs/1807.01622, 2018b. URL http://arxiv.org/abs/1807.01622.

Michael A Gelbart, Jasper Snoek, and Ryan P Adams. Bayesian optimization with unknown constraints. *arXiv preprint arXiv:1403.5607*, 2014.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1487–1495, 2017.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Robert J Gove, Keith Balmer, Nicholas K Ing-Simmons, and Karl M Guttag. Multi-processor reconfigurable in single instruction multiple data (simd) and multiple instruction multiple data (mimd) modes and method of operation, May 18 1993. US Patent 5,212,777.

GraphCore. GraphCore. https://www.graphcore.ai/, 2021. Accessed: 2021-05-16.

Alex Graves. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*, 2012.

Suyog Gupta and Berkin Akin. Accelerator-aware neural network design using automl. *arXiv preprint arXiv:2003.02838*, 2020.

Yanzhang He, Tara N Sainath, Rohit Prabhavalkar, Ian McGraw, Raziel Alvarez, Ding Zhao, David Rybach, Anjuli Kannan, Yonghui Wu, Ruoming Pang, et al. Streaming end-to-end speech recognition for mobile devices. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6381–6385. IEEE, 2019.

Kartik Hegde, Po-An Tsai, Sitao Huang, Vikas Chandra, Angshuman Parashar, and Christopher W Fletcher. Mind mappings: enabling efficient algorithm-accelerator mapping space search. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 943–958, 2021.

Andrew Howard and Suyog Gupta. Introducing the Next Generation of On-Device Vision Models: MobileNetV3 and MobileNetEdgeTPU. https://ai.googleblog.com/2019/11/introducing-next-generation-on-device.html, 2020.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1314–1324, 2019.

Md Shahriar Iqbal, Jianhai Su, Lars Kotthoff, and Pooyan Jamshidi. Flexibo: Cost-aware multi-objective optimization of deep neural networks. *arXiv preprint arXiv:2001.06588*, 2020.

Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pp. 1–12, 2017.

Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 622–636. IEEE, 2020.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SkE6PjC9KX.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

David Koeplinger, Matthew Feldman, Raghu Prabhakar, Yaqi Zhang, Stefan Hadjis, Ruben Fiszel, Tian Zhao, Luigi Nardi, Ardavan Pedram, Christos Kozyrakis, et al. Spatial: A language and compiler for application accelerators. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 296–311, 2018.

Aviral Kumar and Sergey Levine. Model inversion networks for model-based optimization. *arXiv preprint arXiv:1912.13464*, 2019.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 754–768, 2019.

Bo Li, Anmol Gulati, Jiahui Yu, Tara N Sainath, Chung-Cheng Chiu, Arun Narayanan, Shuo-Yiin Chang, Ruoming Pang, Yanzhang He, James Qin, et al. A better and faster end-to-end model for streaming asr. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5634–5638. IEEE, 2021.

Changnian Liu, Yafei Tian, Qiang Zhang, Jie Yuan, and Binbin Xue. Adaptive firefly optimization algorithm based on stochastic inertia weight. In *2013 Sixth International Symposium on Computational Intelligence and Design*, volume 1, pp. 334–337. IEEE, 2013.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746*, 2020.

Luigi Nardi, David Koeplinger, and Kunle Olukotun. Practical design space exploration. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 347–358. IEEE, 2019.

Nvidia. NVDLA deep learning accelerator. http://nvdla.org, 2021a. Accessed: 2021-10-01.

Nvidia. Nvidia. https://www.nvidia.com/en-us/, 2021b. Accessed: 2021-05-16.

Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W Keckler, and Joel Emer. Timeloop: A systematic approach to dnn accelerator evaluation. In *ISPASS*. IEEE, 2019.

Maryam Parsa, John P Mitchell, Catherine D Schuman, Robert M Patton, Thomas E Potok, and Kaushik Roy. Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design. *Frontiers in Neuroscience*, 14:667, 2020.

Ali Razavi, Aäron van den Oord, Ben Poole, and Oriol Vinyals. Preventing posterior collapse with delta-vaes. *arXiv preprint arXiv:1901.03416*, 2019.

Brandon Reagen, José Miguel Hernández-Lobato, Robert Adolf, Michael Gelbart, Paul Whatmough, Gu-Yeon Wei, and David Brooks. A case for efficient accelerator design space exploration via bayesian optimization. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6. IEEE, 2017.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Tara N Sainath, Yanzhang He, Bo Li, Arun Narayanan, Ruoming Pang, Antoine Bruguier, Shuo-yiin Chang, Wei Li, Raziel Alvarez, Zhifeng Chen, et al. A streaming on-device end-to-end model surpassing server-side conventional model quality and latency. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6059–6063. IEEE, 2020.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104:148–175, 2016.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Zhan Shi, Chirag Sakhuja, Milad Hashemi, Kevin Swersky, and Calvin Lin. Learned hardware/software co-design of neural accelerators, 2020.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, 2012. URL http://dl.acm.org/citation.cfm?id=2999325.2999464.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *Proceedings of the 32nd International Conference on Machine Learning*. PMLR, 2015.

Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Jiancheng Lv. Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE Transactions on Cybernetics*, 2020.

Brandon Trabucco, Aviral Kumar, Xinyang Geng, and Sergey Levine. Conservative objective models for effective offline model-based optimization. In *International Conference on Machine Learning*, pp. 10358–10368. PMLR, 2021a.

Brandon Trabucco, Aviral Kumar, Xinyang Geng, and Sergey Levine. Design-bench: Benchmarks for data-driven offline model-based optimization, 2021b. URL https://openreview.net/forum?id=cQzf26aA3vM.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Xin-She Yang. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

Xin-She Yang and Suash Deb. Eagle strategy using lévy walk and firefly algorithms for stochastic optimization. In *Nature inspired cooperative strategies for optimization (NICSO 2010)*, pp. 101–111. Springer, 2010.

Amir Yazdanbakhsh, Christof Angermueller, Berkin Akin, Yanqi Zhou, Albin Jones, Milad Hashemi, Kevin Swersky, Satrajit Chatterjee, Ravi Narayanaswami, and James Laudon. Apollo: Transferable architecture exploration. *arXiv preprint arXiv:2102.01723*, 2021a.

Amir Yazdanbakhsh, Kiran Seshadri, Berkin Akin, James Laudon, and Ravi Narayanaswami. An evaluation of edge tpu accelerators for convolutional neural networks. *arXiv preprint arXiv:2102.10423*, 2021b.

Appendices

A ADDITIONAL EXPERIMENTS

In this section, we present additional experiments compared to the method of Trabucco et al. (2021a)), present some additional results obtained by jointly optimizing multiple applications (Appendix A.2), provide an analysis of the designed accelerators (Appendix A.4) and finally, discuss how our trained conservative surrogate can be used with a different evaluation time constraint (Appendix A.3).

A.1 COMPARISON TO OTHER BASELINE METHODS

Comparison to COMs. In this section, we perform a comparative evaluation of PRIME to the COMs method Trabucco et al. (2021a). Like several offline reinforcement learning algorithms (Kumar et al., 2020), our method, PRIME and COMs are based on the key idea of learning a conservative surrogate of the desired objective function, such that it does not overestimate the value of unseen data points, which prevents the optimizer from finding accelerators that appear promising under the learned model but are not actually promising under the actual objective. The key differences between our method and COMs are: **(1)** PRIME uses an evolutionary optimizer ($\mathrm{Opt}(\cdot)$) for negative sampling compared to gradient ascent of COMs, which can be vastly beneficial in discrete design spaces as our results show empirically, **(2)** PRIME can explicitly learn from infeasible data points provided to the algorithm, while COMs does not have a mechanism to incorporate the infeasible points into the learning of surrogate. To further assess the importance of these differences in practice, we run COMs on three tasks from Table 3, and present a comparison our method, COMs, and Standard method in Table 7. The "Standard" method represents a surrogate model without utilizing any infeasible points. On average, PRIME outperforms COMs by 1.17× (up to 1.24× in M6).

Table 7: Optimized objective values (i.e., latency in milliseconds) obtained by PRIME and COMs (Trabucco et al., 2021a) when optimizing over single applications (MobilenetV2, MobilenetV3 and M6), extending Table 3. Note that PRIME outperforms COMs. However, COMs improves over baseline "Standard" method (last column).

Application	PRIME (Ours)	COMs	Standard
MobileNetV2	**207.43**	251.58	374.52
MobileNetV3	**454.30**	485.66	575.75
M6	**131.46**	163.94	180.24
Geomean of PRIME's Improvement	**1.0×**	**1.17×**	**1.46×**

Comparison to generative offline MBO methods. We provide a comparison between PRIME and prior offline MBO methods based on generative models (Kumar & Levine, 2019). We evaluate model inversion networks (MINs) (Kumar & Levine, 2019) on our accelerator data. However, we were unable unable to train a discrete objective-conditioned GAN model to 0.5 discriminator accuracy on our offline dataset, and often observed a collapse of the discriminator. As a result, we trained a $\delta-$VAE (Razavi et al., 2019), conditioned on the objective function (i.e., latency). A standard VAE (Kingma & Welling, 2013) suffered from posterior collapse and thus informed our choice of utilizing a $\delta-$VAE. The latent space of a trained objective-conditioned $\delta-$VAE corresponding to accelerators on a held-out validation dataset (not used for training) is visualized in the t-SNE plot in the figure on the right. This is a 2D t-SNE of the accelerators configurations (§Table 1). The color of a point denotes the latency value of the corresponding accelerator configuration, partitioned into three bins. Observe that while we would expect these objective conditioned models to disentangle accelerators with different objective values in the latent space, the models we trained did not exhibit such a structure, which will hamper optimization. While our method PRIME could also benefit from a generative optimizer (i.e., by using a generative optimizer in place of $\mathrm{Opt}(\cdot)$ with a conservative surrogate), we leave it for future work to design effective generative optimizers on the accelerator manifold.

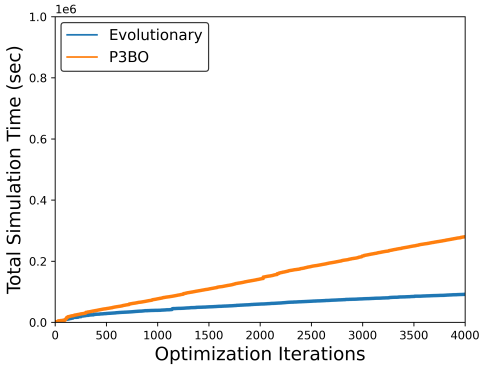


Figure 7: Comparing the total simulation time needed by the P3BO and Evolutionary method on MobileNetEdge. Note that, not only the total simulation time of P3BO is around 3.1× higher than the Evolutionary method, but also the latency of final optimized accelerator is around 18% for MobileNetEdge. The total simulation time of our method is around 14% of the Evolutionary method (See Figure 5).

Comparison to P3BO. We perform a comparison against P3BO method, the state-of-the-arts online method in biology (Angermueller et al., 2020b). On average, PRIME outperforms the P3BO method by 2.5× (up to 8.7× in U-Net). In addition, we present the comparison between the total simulation runtime of the P3BO and Evolutionary methods in Figure 7. Note that, not only the total simulation time of P3BO is around 3.1× higher than the Evolutionary method, but also the latency of final optimized accelerator is around 18% for MobileNetEdge. On the other hand, the total simulation time of PRIME for the task of accelerator design for MobileNetEdge is lower than both methods (only 14% of the Evolutionary method as shown in Figure 5).

Table 8: Optimized objective values (i.e., latency in milliseconds) obtained by PRIME and P3BO (Angermueller et al., 2020b) when optimizing over single applications (MobileNetEdge, M4, t-RNN Dec, t-RNN Enc, and U-Net). On average, PRIME outperforms P3BO by 2.5×.

Application	**PRIME (Ours)**	**P3BO**
MobileNetEdge	**298.50**	376.02
M4	**370.45**	483.39
U-Net	**740.27**	771.70
t-RNN Dec	**132.88**	865.12
t-RNN Enc	**130.67**	1139.48
Geomean of PRIME's Improvement	**1.0×**	**2.5×**

A.2 LEARNED SURROGATE MODEL REUSE FOR ACCELERATOR DESIGN

Extending our results in Table 4, we present another variant of optimizing accelerators jointly for multiple applications. In that scenario, the learned surrogate model is reused to architect an accelerator for a subset of applications used for training. We train a contextual conservative surrogate on the variants of MobileNet (Table 2) as discussed in Section 4, but generated optimized designs by only optimizing the average surrogate on only two variants of MobileNet (MobileNetEdge and MobileNetV2). This tests the ability of our approach PRIME to provide a general contextual conservative surrogates, that can be trained *only once* and optimized multiple times with respect to different subsets of applications. Observe in Table 9, PRIME architects high-performing accelerator configurations (better than the best point in the dataset by 3.29× – last column) while outperforming the online optimization methods by 7%.

A.3 LEARNED SURROGATE MODEL REUSE UNDER DIFFERENT DESIGN CONSTRAINT

We also test the robustness of our approach in handling variable constraints at test-time such as different chip area budget. We evaluate the learned conservative surrogate trained via PRIME under a reduced value of the area threshold, α, in Equation 1. To do so, we utilize a variant of rejection sampling – we take the learned model trained for a default area constraint $\alpha = 29\ \mathrm{mm}^2$ and then reject

Table 9: Optimized objective values (i.e., latency in milliseconds) obtained by our PRIME when using the jointly optimized model on three variants of MobileNets and use for MobileNetEdge and MobileNetV2 for different dataset configurations. PRIME outperforms the best online method by **7%** and finds an accelerator that is **3.29**× better than the best accelerator in the training dataset (last row). The best accelerator configuration is highlighted in bold.

Applications	PRIME			Standard	Online Optimization		$\mathcal{D}$ (Best in Training)
	All	-Opt	-Infeasible		Bayes Opt	Evolutionary	
(MobileNetEdge, MobileNetV2)	**253.85**	297.36	264.85	341.12	275.21	271.71	834.68

Table 10: Optimized objective values (i.e., latency in milliseconds) obtained by various methods for the task of learning accelerators specialized to MobileNetEdge under chip area budget constraint 18 mm^2 reusing the already learned model by our method for MobileNetEdge (shown in Table 3). Lower latency/runtime is better. From left to right: our method, our method without negative sampling ("PRIME-Opt") and without utilizing infeasible points ("PRIME-Infeasible"), standard surrogate ("Standard"), online Bayesian optimization ("Bayes Opt"), online evolutionary algorithm ("Evolutionary") and the best design in the training dataset. Note that PRIME improves over the best in the dataset by 12%, outperforms the best online optimization method by 4.4%. The best accelerator configuration is highlighted in bold.

Applications	PRIME			Standard	Online Optimization		$\mathcal{D}$ (Best in Training)
	All	-Opt	-Infeasible		Bayes Opt	Evolutionary	
MobileNetEdge, Area $\leq$ 18 mm^2	**315.15**	433.81	351.22	470.09	331.05	329.13	354.13

Table 11: Per application latency for the best accelerator design suggested by PRIME and the Evolutionary method according to Table 4 for multi-task accelerator design (nine applications and area constraint 100 mm^2). PRIME outperforms the Evolutionary method by 1.35×.

Applications	Latency (ms)		Improvement of PRIME over Evolutionary
	PRIME	Evolutionary (Online)	
MobileNetEdge	**288.38**	319.98	1.10×
MobileNetV2	**216.27**	255.95	1.18×
MobileNetV3	**487.46**	573.57	1.17×
M4	**400.88**	406.28	1.01×
M5	248.18	**239.18**	0.96×
M6	164.98	**148.83**	0.90×
U-Net	1268.73	**908.86**	0.71×
t-RNN Dec	**191.83**	862.14	5.13×
t-RNN Enc	**185.41**	952.44	4.49×
Average (Latency in ms)	`383.57`	`518.58`	`1.35`×

all optimized accelerator configurations which do not satisfy a reduces area constraint: $\text{Area}(\mathbf{x}) \leq \alpha_0 = 18\ \text{mm}^2$. Table 10 summarizes the results for this scenario for the MobileNetEdge (Gupta & Akin, 2020) application under the new area constraint ($\alpha = 18\ \text{mm}^2$). A method that produces diverse designs which are both high-performing and are spread across diverse values of the area constraint are expected to perform better. As shown in Table 10, PRIME provides better accelerator than the best online optimization from scratch with the new constraint value by 4.4%, even when PRIME does not train its conservative surrogate with this unseen test-time design constraint. Note that, when the design constraint changes, online methods generally need to restart the optimization process from scratch and undergo costly queries to the simulator. This would impose additional overhead in terms of total simulation time (§ Figure 5 and Figure 6). However, the results in Table 10 shows that our learned surrogate model can be reused under different test-time design constraint eliminating additional queries to the simulator.

A.4 ANALYSIS OF DESIGNED ACCELERATORS

In this section, we overview the best accelerator configurations that PRIME and the Evolutionary method identified for multi-task accelerator design (See Table 4), when the number of target applications are nine and the area constraint is set to 100 mm^2. The average latencies of the best accelerators found by PRIME and the Evolutionary method across nine target applications are **383.57 ms** and **518.58 ms**, respectively. In this setting, our method outperforms the best online method by **1.35**×.

Table 12: Optimized accelerator configurations (See Table 1) found by PRIME and the Evolutionary method for multi-task accelerator design (nine applications and area constraint 100 mm^2). Last row shows the accelerator area in mm^2. PRIME reduces the overall chip area usage by 1.97×. The difference in the accelerator configurations are shaded in gray.

Accelerator Parameter	Parameter Value	
	PRIME	Evolutionary (Online)
# of PEs-X	4	4
# of PEs-Y	6	8
# of Cores	64	128
# of Compute Lanes	4	6
PE Memory	2,097,152	1,048,576
Core Memory	131,072	131,072
Instruction Memory	32,768	8,192
Parameter Memory	4,096	4,096
Activation Memory	512	2,048
DRAM Bandwidth (Gbps)	30	30
Chip Area (mm^2)	**46.78**	**92.05**

Table 11 shows per application latencies for the accelerator suggested by our method and the Evolutionary method. The last column shows the latency improvement of PRIME over the Evolutionary method. Interestingly, while the latency of the accelerator found by our method for MobileNetEdge, MobileNetV2, MobileNetV3, M4, t-RNN Dec, and t-RNN Enc are better, the accelerator identified by the online method yields lower latency in M5, M6, and U-Net.

To better understand the trade-off in design of each accelerators designed by our method and the Evolutionary method, we present all the accelerator parameters (See Table 1) in Table 12. The accelerator parameters that are different between each of the designed accelerator are shaded in gray (e.g. # of PEs-Y, # of Cores, # of Compute Lanes, PE Memory, Instruction Memory, and Activation Memory). Last row of Table 12 depicts the overall chip area usage in mm^2. PRIME not only outperforms the Evolutionary algorithm in reducing the average latency across the set of target applications, but also reduces the overall chip area usage by **1.97**×. Studying the identified accelerator configuration, we observe that PRIME trade-offs compute (**64 cores vs. 128 cores**) for larger PE memory size (**2,097,152 vs. 1,048,576**). These results show that PRIME favors PE memory size to accommodate for the larger memory requirements in t-RNN Dec and t-RNN Enc (See Table 2 Model Parameters) where large gains lie. Favoring larger on-chip memory comes at the expense of lower compute power in the accelerator. This reduction in the accelerator's compute power leads to higher latency for the models with large number of compute operations, namely M5, M6, and U-Net (See last row in Table 2). M4 is an interesting case where both compute power and on-chip memory is favored by the model (6.23 MB model parameters and 3,471,920,128 number of compute operations). This is the reason that the latency of this model on both accelerators, designed by our method and the Evolutionary method, are comparable (400.88 ms in PRIME vs. 406.28 ms in the online method).

A.5 COMPARISON WITH ONLINE METHODS IN ZERO-SHOT SETTING

We evaluated the Evolutionary (online) method under two protocols for the last two rows of Table 5: first, we picked the best designs (top-performing 256 designs similar to the PRIME setting in Section 4) found by the evolutionary algorithm on the training set of applications and evaluated them on the target applications and second, we let the evolutionary algorithm continue simulator-driven optimization on the target applications. The latter is unfair, in that the online approach is allowed access to querying more designs in the simulator. Nevertheless, we found that in either configuration, the evolutionary approach performed worse than PRIME which does not access training data from the target application domain. For the area constraint 29 mm^2 and 100 mm^2, the Evolutionary algorithm reduces the latency from 1127.64 → 820.11 and 861.69 → 552.64, respectively, although still worse than PRIME. In the second experiment in which we *unfairly* allow the evolutionary algorithm to continue optimizing on the target application, the Evolutionary algorithm suggests worse designs than Table 5 (e.g. 29 mm^2: 1127.64 → 1181.66 and 100 mm^2: 861.69 → 861.66).

Application	PRIME			Standard	Online Optimization		$\mathcal{D}$ (Best in Training)
	All	-Opt	-Infeasible		Bayes Opt	Evolutionary	
MobileNetEdge	**298.50**	435.40	322.20	411.12	319.00	320.28	354.13
MobileNetV2	**207.43**	281.01	214.71	374.52	240.56	238.58	410.83
MobileNetV3	**454.30**	489.45	483.96	575.75	534.15	501.27	938.41
M4	**370.45**	478.32	432.78	1139.76	396.36	383.58	779.98
M5	208.21	319.61	246.80	307.57	201.59	**198.86**	449.38
M6	131.46	197.70	162.12	180.24	121.83	**120.49**	369.85
U-Net	**740.27**	740.27	765.59	763.10	872.23	791.64	1333.18
t-RNN Dec	**132.88**	172.06	135.47	136.20	771.11	770.93	890.22
t-RNN Enc	**130.67**	134.84	137.28	150.21	865.07	865.07	584.70

Table 13: Optimized objective values (i.e., latency in milliseconds) obtained by various methods for the task of learning accelerators specialized to a given application. Lower latency/runtime is better. From left to right: our method, our method without negative sampling ("PRIME-Opt") and without utilizing infeasible points ("PRIME-Infeasible"), standard surrogate ("Standard"), online Bayesian optimization ("Bayes Opt"), online evolutionary algorithm ("Evolutionary") and the best design in the training dataset. Note that, in all the applications PRIME improves over the best in the dataset, outperforms online optimization methods in 7/9 applications and the complete version of PRIME generally performs best. The best accelerator designs are in bold.

A.6 PRIME ABLATION STUDY

Here we ablate over variants of our method: (1) Opt was not used for negative sampling ("PRIME-Opt" in Table 13) (2) infeasible points were not used ("PRIME-Infeasible" in Table 13). As shown in Table 13, the variants of our method generally performs worse compared to the case when both negative sampling and infeasible data points are utilized in training the surrogate model.

B DETAILS OF PRIME

In this section, we provide training details of our method PRIME including hyperparameters and compute requirements and details of different tasks.

B.1 HYPERPARAMETER AND TRAINING DETAILS

Algorithm 1 outlines our overall system for accelerator design. PRIME parameterizes the function $f_\theta(\mathbf{x})$ as a deep neural network as shown in Figure 4. The architecture of $f_\theta(\mathbf{x})$ first embeds the discrete-valued accelerator configuration $\mathbf{x}$ into a continuous-valued 640-dimensional embedding via two layers of a self-attention transformer (Vaswani et al., 2017). Rather than directly converting this 640-dimensional embedding into a scalar output via a simple feed-forward network, which we found a bit unstable to train with Equation 3, possibly due to the presence of competing objectives for a comparison), we pass the 640-dimensional embedding into M different networks that map it to M different scalar predictions $(f^i_\theta(\mathbf{x}))_{i=1}^M$. Finally, akin to attention (Vaswani et al., 2017) and mixture of experts (Shazeer et al., 2017), we train an additional head to predict weights $(w_i)_{i=1}^M \geq 0$ of a linear combination of the predictions at different heads that would be equal to the final prediction: $f_\theta(\mathbf{x}) = \sum_{i=1}^K w_i f^i_\theta(\mathbf{x})$. Such an architecture allows the model to use different predictions $f^i_\theta(\mathbf{x})$, depending upon the input, which allows for more stable training. To train $f_\theta(\mathbf{x})$, we utilize the Adam (Kingma & Ba, 2015) optimizer. Equation 3 utilizes a procedure Opt that maximizes the learned function approximately. We utilize the same technique as Section 4 ("optimizing the learned surrogate") to obtain these negative samples. We periodically refresh Opt, once in every 20K gradient steps on $f_\theta(\mathbf{x})$ over training.

The hyperparameters for training the conservative surrogate in Equations 3 and its contextual version are as follows:

- **Architecture of** $f_\theta(\mathbf{x})$. As indicated in Figure 4, our architecture takes in list of categorical (one-hot) values of different accelerator parameters (listed in Table 1), converts each parameter into 64-dimensional embedding, thus obtaining a 10×64 sized matrix for each accelerator, and then runs two layers of self-attention (Vaswani et al., 2017) on it. The resulting 10×64 output is flattened to a vector in $\mathbb{R}^{640}$ and fed into $M = 7$ different prediction networks that give rise to $f^1_\theta(\mathbf{x}), \cdots, f^M_\theta(\mathbf{x})$, and an additional attention 2-layer feed-forward network (layer sizes $= [256, 256]$) that determines weights $w_1, \cdots, w_M$, such that $w_i \geq 0$ and $\sum_{i=1}^M w_i = 1$. Finally the output is simply $f_\theta(\mathbf{x}) = \sum_i w_i f^i_\theta(\mathbf{x})$.
- **Optimizer/learning rate for training** $f_\theta(\mathbf{x})$. Adam, $1e-4$, default $\beta_1 = 0.9$, $\beta_2 = 0.999$.

Algorithm 1 Training the conservative surrogate in PRIME

Initialize a neural model $f_{\theta_0}(\mathbf{x})$ and a set $M = 23$ of negative particles to be updated by the firefly optimizer $\{\mathbf{x}_1^-(0), \cdots, \mathbf{x}_i^-(0), \mathbf{x}_M^-(0)\}$ to random configurations from the design space.
for iteration $i = 0, 1, 2, 3, \ldots$ until convergence **do**
  for firefly update step $t = 0, 1, \ldots, 4$ ▷ **Inner loop do**
    Update the M fireflies according to the firefly update rule in Equation 5,
    towards maximizing $f_{\theta_i}(\mathbf{x})$ according to: ▷ **Negative mining**
$$\mathbf{x}_i(ti+1) = \mathbf{x}_i(ti) + \beta(\mathbf{x}_i(ti) - \mathbf{x}_j(ti)) + \alpha\epsilon_{ti}$$
  end for
  Find the best firefly found in these steps to be used as the negative sample:
$$\mathbf{x}_i^- = \arg\min\{f_{\theta_i}(\mathbf{x}_1^-(ti)), \cdots, f_{\theta_i}(\mathbf{x}_M^-(ti))\}$$ ▷ **Find negative sample**
  Run one gradient step on θ_i using Equation 3 with $\mathbf{x}_i^-$ as the negative sample
  if $i\%p == 0$, (p = 20000), then: ▷ **Periodically reinitialize the optimizer**
    Reinitialize firefly particles $\{\mathbf{x}_1^-(0), \cdots, \mathbf{x}_i^-(0), \mathbf{x}_M^-(0)\}$ to random designs.
end for
Return the final model $f_{\theta^*}(\mathbf{x})$

- **Validation set split.** Top 20% high scoring points in the training dataset are used to provide a validation set for deciding coefficients α, β and the checkpoint to evaluate.
- **Ranges of α, β.** We trained several $f_\theta(\mathbf{x})$ models with $\alpha \in [0.0, 0.01, 0.1, 0.5, 1.0, 5.0]$ and $\beta \in [0.0, 0.01, 5.0, 0.1, 1.0]$. Then we selected the best values of α and β based on the highest Kendall's ranking correlation on the validation set. Kendall's ranking correlation between two sets of objective values: $S = \{y_1, y_2, \cdots, y_N\}$ corresponding to ground truth latency values on the validation set and $S' = \{y'_1, y'_2, \cdots, y'_N\}$ corresponding to the predicted latency values on the validation set is given by τ equal to:
$$\tau = \frac{\sum_{i,j}^{N,N} \mathbb{I}[(y_i - y_j)(y'_i - y'_j) > 0] - \sum_{i,j}^{N,N} \mathbb{I}[(y_i - y_j)(y'_i - y'_j) \leq 0]}{N \cdot (N-1)}. \quad (4)$$
- **Clipping $f_\theta(\mathbf{x})$ during training**. Equation 3 increases the value of the learned function $f_\theta(\mathbf{x})$ at $\mathbf{x} = \mathbf{x}_0 \in \mathcal{D}_{\text{infeasible}}$ and $\mathbf{x}^- \sim \text{Opt}(f_\theta)$. We found that with the small dataset, these linear objectives can run into numerical instability, and produce $+\infty$ predictions. To avoid this, we clip the predicted function value both above and below by ± 10000.0, where the valid range of ground-truth values is $\mathcal{O}(1000)$.
- **Negative sampling with** $\text{Opt}(\cdot)$. As discussed in Section 4, we utilize the firefly optimizer for both the negative sampling step and the final optimization of the learned conservative surrogate. When used during negative sampling, we refresh (i.e., reinitialize) the firefly parameters after every $p = 20000$ gradient steps of training the conservative surrogate, and run $t = 5$ steps of firefly optimization per gradient step taken on the conservative surrogate.
- **Details of firefly:** The initial population of fireflies depends on the number of accelerator configurations ($\mathcal{C}$) following the formula $10 + \lfloor(\mathcal{C}^{1.2} + \mathcal{C}) \times 0.5\rceil$. In our setting with ten accelerator parameters (See Table 1), the initial population of fireflies is 23. We use the same hyperparameters: $\gamma = 1.0, \beta_0 = 1.0$, for the optimizer in all the experiments and never modify it. The update to a particular optimization particle (i.e., a firefly) $\mathbf{x}_i$, at the t-th step of optimization is given by:
$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \beta(\mathbf{x}_i(t) - \mathbf{x}_j(t)) + \text{ i.i.d. Gaussian noise}, \quad (5)$$
  where $\mathbf{x}_j(t), j \neq i$ is a different firefly that achieves a better objective value compared to $\mathbf{x}_i$ and the function β is given by: $\beta(r) = \beta_0 e^{-\gamma r^2}$.

B.2 Details of Architecting Accelerators for Multiple Applications Simultaneously

Now we will provide details of the tasks from Table 4 where the goal is to architect an accelerator which is jointly optimized for multiple application models. For such tasks, we augment data-points for each model with the context vector c_k from Table 2 that summarizes certain parameters for each

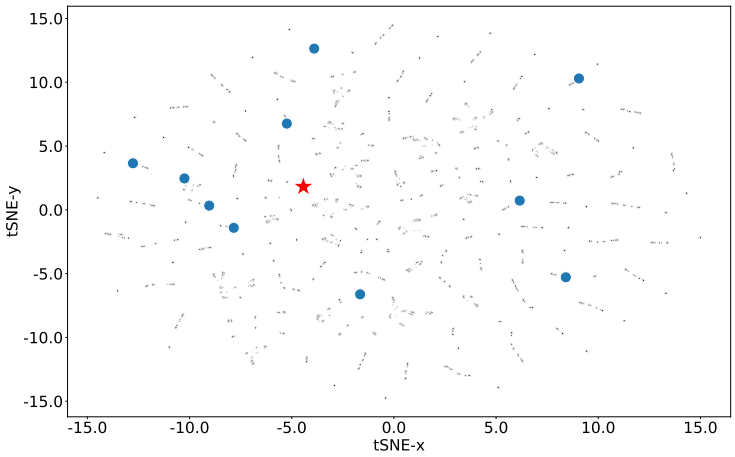


Figure 8: tSNE plot of the joint dataset and randomly sampled infeasible data points. The blue points show the accelerator configurations that are jointly feasible for all the applications. The highlighted point with red star shows the best design proposed by PRIMEThe rest of the points show the infeasible points.

application. For entries in this context vector that have extremely high magnitudes (e.g., model parameters and number of compute operations), we normalize the values by the sum of values across the applications considered to only encode the relative scale, and not the absolute value which is not required. To better visualize the number of feasible accelerators for joint optimization, Figure 8 show the tSNE plot (raw architecture configurations are used as input) of high-performing accelerator configurations. The blue-colored dots are the jointly feasible accelerators in the combined dataset, and note that these data points are no more than 20-30 in total. The highlighted red star presents the best design suggested by PRIME with average latency of 334.70 (Table 4). This indicates that this contextual, multi-application problem poses a challenge for data-driven methods: these methods need to produce optimized designs even though very few accelerators are jointly feasible in the combined dataset. Despite this limitation, PRIME successfully finds more efficient accelerator configurations that attain low latency values on each of the applications jointly, as shown in Table 4.

B.3 DATASET RELEASE

To reproduce the results presented in this paper and to further facilitate research in model-based optimization for accelerator design, we release a much larger dataset across various accelerator configurations. We collected these data points using a detailed industrial simulator Yazdanbakhsh et al. (2021b). Along with the dataset, we release a script to parse the dataset. The dataset can be found at the following anonymized link (clickable): PRIME Dataset. [The link is: `https://drive.google.com/drive/folders/1DUfA_GfZPPu7esS7-c34G8N8uCTWg7N8?usp=sharing`]

B.4 DATASET SENSITIVITY TO ACCELERATOR PARAMETERS

We visualize the sensitivity of the objective function (e.g. latency) with respect to the changes in certain accelerator parameters, such as memory size (Table 1), in Figure 10b, illustrating this sensitivity. As shown in the Figure, the latency objective that we seek to optimize can exhibit high sensitivity to small variations in the architecture parameters, making the optimization landscape particularly ill-behaved. Thus, a small change in one of the discrete parameters, can induce a large change in the optimization objective. This characteristic of the dataset further makes the optimization task challenging.

C OVERVIEW OF ACCELERATORS AND SEARCH SPACE

This section briefly discuss the additional accelerators (similar to (Kao et al., 2020)) that we evaluate in this work, namely NVDLA (Nvidia, 2021b) and ShiDianNao (Du et al., 2015), and their corresponding search spaces.

NVDLA: Nvidia Deep Learning Accelerator NVDLA (Nvidia, 2021a) is an open architecture inference accelerator designed and maintained by Nvidia. In compared to other inference accelerators, NVDLA is a weight stationary accelerator. That is, it retains the model parameters on each processing

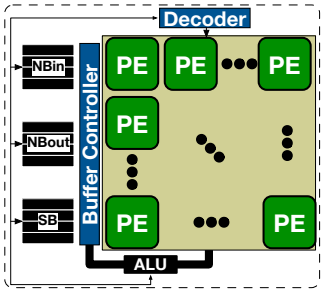
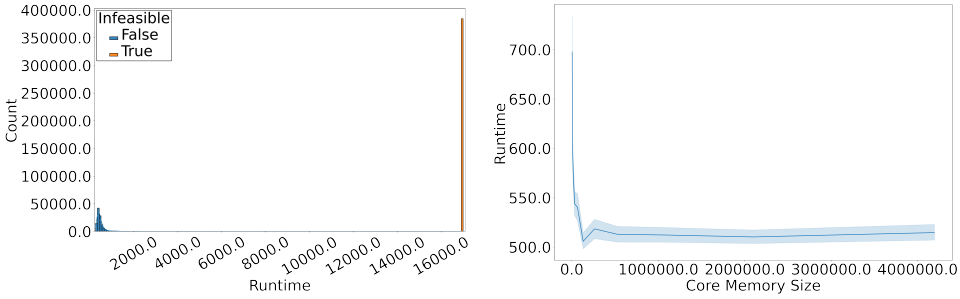


Figure 9: Overview of ShiDianNao dataflow accelerator. This dataflow accelerators exhibits an output-stationary dataflow where it keeps the partial results stationary within each processing elements (PEs).



(a) **(b)**

Figure 10: The (a) histogram of infeasible (orange bar with large score values)/feasible (blue bars) data points and (b) the sensitivity of runtime to the size of core memory for the MobileNetEdge (Howard & Gupta, 2020) dataset.

elements and parallelizes the computations across input and output channels. NVDLA-style dataflow accelerators generally yield better performance for the computations of layers at the later processing stages. This is because these layers generally have larger model parameters that could benefit from less data movement associated to the model parameters.

ShiDianNao: Vision Accelerator Figure 9 shows the high-level schematic of ShiDianNao accelerator (Du et al., 2015). ShiDianNao-style dataflow accelerator is an output-stationary accelerator. That is, it keeps the partial results inside each PE and instead move the model parameters and input channel data. As such, in compared to NVDLA-style accelerators, ShiDianNao provides better performance for the computations of the layers with large output channels (generally first few layers of a model).

Search space of dataflow accelerators. We follow a similar methodology as (Kao et al., 2020) to evaluate additional hardware accelerators, discussed in the previous paragraphs. We use MAESTRO (Kwon et al., 2019), an analytical cost model, that supports the performance modeling of various dataflow accelerators. In this joint accelerator design and dataflow optimization problem, the total number of parameters to be optimized is up to 106—the tuple of (# of PEs, Buffers) per per model layer—with each parameter taking one of 12 discrete values. This makes the hardware search space consist of $\approx 2.5 \times 10^{114}$ accelerator configurations. We also note that while the method proposed in (Kao et al., 2020) treats the accelerator design problem as a sequential decision making problem, and uses reinforcement learning techniques, PRIME simply designs the whole accelerator in a single step, treating it as a model-based optimization problem.

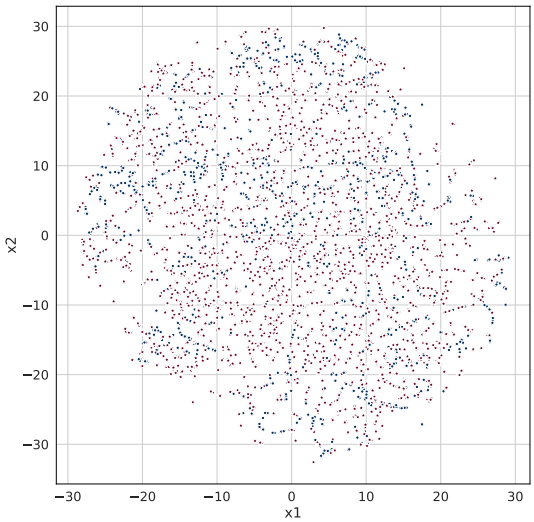


Figure 11: tSNE plot of the infeasible and feasible hardware accelerator designs. Note that feasible designs (shown in blue) are embedded in a sea of infeasible designs (shown in red), which makes this a challenging domain for optimization methods.

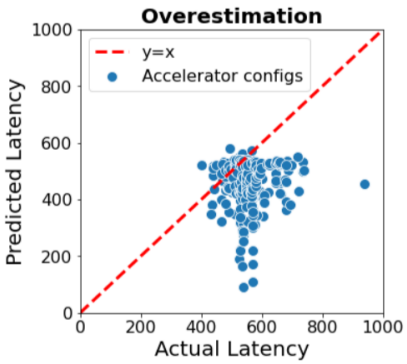


Figure 12: To verify if the overestimation hypothesis–that optimizing an accelerator against a naïve standard surrogate model is likely to find optimizers that appear promising under the learned model, but do not actually attain low-latency values–in our domain, we plot a calibration plot of the top accelerator designs found by optimizing a naïvely trained standard surrogate model. In the scatter plot, we represent each accelerator as a point with its x-coordinate equal to the actual latency obtained by running this accelerator design in the simulator and the y-coordinate equal to the predicted latency under the learned surrogate. Note that for a large chunk of designs, the predicted latency is much smaller than their actual latency (i.e., these designs lie beneath the $y = x$ line in the plot above). This means that optimizing designs under a naïve surrogate model is prone to finding designs that appear overly promising (i.e., attain lower predicted latency values), but are not actually promising. This confirms the presence of the overestimation hypothesis on our problem domain.