LitePred: Transferable and Scalable Latency Prediction for Hardware-Aware Neural Architecture Search

Chengquan Feng, *University of Science and Technology of China;* Li Lyna Zhang, *Microsoft Research;* Yuanchi Liu, *University of Science and Technology of China;* Jiahang Xu and Chengruidong Zhang, *Microsoft Research;* Zhiyuan Wang, *University of Science and Technology of China;* Ting Cao and Mao Yang, *Microsoft Research;* Haisheng Tan, *University of Science and Technology of China*

https://www.usenix.org/conference/nsdi24/presentation/feng-chengquan

This paper is included in the Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation.

April 16–18, 2024 • Santa Clara, CA, USA

978-1-939133-39-7

Open access to the Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation is sponsored by

LitePred: Transferable and Scalable Latency Prediction for Hardware-Aware Neural Architecture Search

Chengquan Feng[1*] Li Lyna Zhang[2‡] Yuanchi Liu[1] Jiahang Xu[2] Chengruidong Zhang[2]
Zhiyuan Wang[1] Ting Cao[2] Mao Yang[2] Haisheng Tan[1‡]

[1]University of Science and Technology of China [2]Microsoft Research

Abstract

Hardware-Aware Neural Architecture Search (NAS) has demonstrated success in automating the design of affordable deep neural networks (DNNs) for edge platforms by incorporating inference latency in the search process. However, accurately and efficiently predicting DNN inference latency on diverse edge platforms remains a significant challenge. Current approaches require several days to construct new latency predictors for each one platform, which is prohibitively time-consuming and impractical.

In this paper, we propose LitePred, a lightweight approach for accurately predicting DNN inference latency on new platforms with minimal adaptation data by transferring existing predictors. LitePred builds on two key techniques: (i) *a Variational Autoencoder (VAE) data sampler* to sample high-quality training and adaptation data that conforms to the model distributions in NAS search spaces, overcoming the out-of-distribution challenge; and (ii) *a latency distribution-based similarity detection method* to identify the most similar pre-existing latency predictors for the new target platform, reducing adaptation data required while achieving high prediction accuracy. Extensive experiments on 85 edge platforms and 6 NAS search spaces demonstrate the effectiveness of our approach, achieving an average latency prediction accuracy of 99.3% with less than an hour of adaptation cost. Compared with SOTA platform-specific methods, LitePred achieves up to 5.3% higher accuracy with a significant 50.6× reduction in profiling cost. Code and predictors are available at https://github.com/microsoft/Moonlit/tree/main/LitePred.

1 Introduction

Hardware-aware Neural Architecture Search (NAS) has achieved remarkable success in automating the design of hardware-friendly deep neural networks (DNNs) in many tasks [8, 9, 22, 36, 51, 53, 56, 57]. This holds particular importance for crafting models suited to resource-limited edge platforms, such as mobile phones. However, to design low-latency models for diverse edge platforms, it remains of significant challenge to estimate accurately the inference latency of numerous models, which depends on multiple factors such as hardware, inference frameworks, and data precision [52].

Direct on-device measurement is expensive and impractical due to the very large search space of possible models. Consequently, many works [6, 7, 9, 18, 31, 54] have been proposed to predict the inference latency based on the given model architecture. However, these methods face two limitations. First, they typically rely on random sampling and require a significant amount of training data to achieve accurate predictions. This process is time-consuming, requiring *several days of data collection for a single platform* [18, 54]. Second, existing approaches focus on platform-specific prediction without considering the latency change due to various dynamics such as new hardware, various inference frameworks or new versions, and different data precision. We take nn-Meter [54], a cutting-edge approach, as an example. By developing latency predictors for the Xiaomi12 CPU platform using Onnxruntime (ORT) [35], it achieves an impressive 99.8% accuracy on the MobileNetV3 search space [8, 22]. However, the predictor drops to *0 accuracy* when the hardware is switched to a Xiaomi11 CPU. Thus, these approaches require rebuilding predictors for every new platform, which is prohibitively expensive and limits the applicability of hardware-aware NAS. This is especially challenging given the diverse edge platforms, which include a large number of mobile devices (e.g., 8318 heterogeneous smartphones [17]) and various inference frameworks (e.g., TFLite [20], ORT, OpenVINO [25], NCNN [60]).

Although a few platform-agnostic prediction works [30, 33] have attempted to address the high cost, they encounter various limitations. HELP [30] trains a latency predictor with meta learning [29, 47], but it conducts model-level prediction, which requires expensive redesigning and retraining of the meta predictor for new NAS search spaces. OneProxy [33] trains a latency monotonicity model to predict model latency rankings on different platforms. However, it only predicts latency rankings on new platforms, rather than the actual

*Work was done during the internship at Microsoft Research
‡Corresponding authors, (lzhani@microsoft.com and hstan@ustc.edu.cn)

values, which is often mandatory in practical deployments.

In this work, we propose LitePred, a lightweight approach for accurately predicting inference latency in hardware-aware NAS that eliminates the expensive rebuilding process required for new platforms. To handle the diverse model graphs in different NAS search spaces, LitePred performs kernel-level prediction and computes the model latency as the sum of the predicted latencies of all kernels [‡]. The key idea of LitePred is to identify the most similar pre-existing latency predictors for each kernel on new platform, and then finetune them with just a few adaptation samples to achieve high prediction accuracy. This makes LitePred a cost-effective solution for predicting latency, eliminating the bottleneck in hardware-aware NAS.

At the core of LitePred lies the principle that *knowledge from a pre-existing latency predictor for one platform can be transferred to new platforms that share similarities*. This is based on the fact that latency depends on key factors and their relationships, which can be learned by an accurate predictor. When a new platform shares similarities with a previous one, we can transfer the existing knowledge and adapt the predictor to capture new dynamics. For example, when using the same TFLite version 2.1, we can transfer a latency predictor trained on a Xiaomi11 CPU to a Pixel 6 CPU. By reusing the knowledge learned for TFLite 2.1, we only need a few adaptation data to learn behaviors on the Pixel 6 CPU.

LitePred maintains a knowledge pool of existing latency predictors, each being a 16-layer Multilayer Perceptron (MLP) network. Initially, it constructs base predictors for warmup platforms and stores them in the pool. When targeting a new platform, LitePred detects its kernels and identifies the most similar predictor for each kernel from the pool. These predictors are then finetuned with a small amount of adaptation data from the new platform to achieve accurate predictions.

LitePred faces two technical challenges. **(1)** First, constructing initial latency predictors and finetuning existing predictors require effective data collection. However, latency-dominant kernels, such as Conv and DWConv kernels, exhibit a multi-dimensional joint distribution in NAS search spaces, with dimensions that are highly correlated. This makes random and adaptive data sampling methods [54] ineffective due to the out-of-distribution problem, resulting in a large amount of useless training data and low accuracy. Direct sampling from search spaces can cause data leakage and limit the generalization ability in new NAS search spaces. **(2)** Second, the similarity between existing predictors and new platforms greatly affects the number of adaptation samples and the final prediction accuracy. However, the high diversity between different edge platforms, many of which are black boxes, poses a challenge in effectively detecting the most similar predictor.

VAE data sampler. To address the first challenge, we leverage the concept of Variational AutoEncoder (VAE) [27] and introduce the VAE data sampler. It consists of an encoder-decoder network, where the encoder compresses the multi-dimensional joint distribution in the search spaces into a latent space with a multivariate Gaussian distribution [19]. The decoder then reconstructs the Gaussian distribution back into the original distribution. By sampling from this distribution and decoding the data using the decoder, we can generate new training data that conforms to the original multi-dimensional distribution, addressing the out-of-distribution problem while preserving generalizability in new model search spaces.

Similar predictor detection. To identify the most similar pre-existing latency predictors for a target platform, we propose a *latency distribution-based similarity detection* method. The key idea is to compare the real latency distribution on the target platform with the predicted latency distributions by pre-existing latency predictors. For each kernel, we create a small representative set reflecting platform-specific optimizations. Then, we compute the distribution similarity by calculating the Kullback-Leibler divergence [13] between the real and predicted latencies for each latency predictor in the knowledge pool. The predictor with the highest similarity is selected.

We extensively evaluate LitePred on 85 edge platforms, including 10 hardware, 10 CPU frequencies, 5 commonly used edge inference frameworks, and 2 data precisions (FP32 and INT8). Before LitePred, platform-specific approaches take several days of data collection for a single platform, limiting evaluations to only a few platforms. To the best of our knowledge, we are the first to evaluate on such a wide range of platforms, with requiring $\sim$1 hour of measurements on each platform.

We summarize our key contributions as follows:

- We propose LitePred, the first to transfer pre-existing latency predictors and achieve accurate latency prediction on new edge platforms with a profiling cost of less than 1 hour.
- We introduce two key techniques: a VAE data sampler to collect effective multi-dimensional data and a latency-distribution similarity detector that identifies the most similar pre-existing latency predictors for black-box platforms.
- Extensive experiments on various platforms and 6 NAS search spaces demonstrate LitePred's effectiveness, achieving 99.3% accuracy and outperforming state-of-the-art latency prediction baselines. LitePred significantly improves accuracy on new edge platforms compared to platform-agnostic baselines, with up to a 77.5% improvement, and achieves up to +5.3% higher accuracy against platform-specific baselines while reducing profiling cost by 50.6×.
- By integrating LitePred with NAS, we discover better models with superior accuracy and lower latency than MobileNets. Our models surpass MobileNetV2, achieving an impressive up to 4.4% higher accuracy on ImageNet.

[‡]A kernel represents an execution unit, it may be either a single primitive operator or a fusion of multiple fused operators, similar to nn-Meter [54].

(a) Same device equipped with various inference frameworks					
Xiaomi11	TFLite 2.1	TFLite 2.7	ORT	NCNN	Mindspore
	289.4ms	271.8ms	451.8ms	165.2ms	209.2ms
(b) Different devices under the same inference framework					
TFLite2.1	Xiaomi11CPU	Xiaomi11GPU	Xiaomi12CPU	Pixel5CPU	Pixel5CPU*
	289.4ms	36.3ms	244.4ms	300.1ms	568.6ms
(c) Different data precision under the same platform					
Xiaomi11	TFLite 2.1		TFLite 2.7		
	FP32	INT8	FP32	INT8	
	289.4ms	75.4 ms	271.8ms	77.8ms	

Table 1: The vastly different inference latency of ResNet50 on diverse platforms. *: we set a lower CPU frequency.

2 Background and Motivations

2.1 Factors that impact the latency

To investigate factors impacting model latency on diverse edge platforms, we start by conducting measurements. We monitor different platforms, varying devices, CPU frequency, inference frameworks, and data precision. Our sample model is ResNet50, and we deploy it on each platform to measure the inference latency. The results in Table 1 show that model latency is heavily dependent on the following factors:

Devices. It has been widely observed that the same model can exhibit varying inference latencies on different devices. As shown in Table 1(b), ResNet50 runs 8.0× faster on a GPU than a CPU on the Xiaomi 11. Surprisingly, even among mobile phones with ARM CPUs, latency differences for the same model can exceed 10%.

Inference frameworks: The effectiveness of framework optimizations and how well they align with the underlying hardware can significantly impact the overall latency. As shown in Table 1(a), on Xiaomi11 CPU, deploying ResNet50 with NCNN yields a 1.7×, 1.6×, 2.7× and 1.3× faster speed than TFLite 2.1, TFLite 2.7, ORT and Mindspore Lite [24], respectively. Notably, even using the same TFLite framework, different versions can introduce significant latency differences.

CPU frequency. In addition to the device type and inference frameworks, CPU frequency has a significant impact on inference latency, as shown in Table 1(b). Our experiments on the Pixel5 demonstrates that even minor changes in CPU frequency (2.2 GHz to 1.9 GHz) can substantially affect latency.

Data precision. Finally, edge platforms support different data precisions, which lead to different memory and computation costs, thereby affecting inference latency. Our experiments focus on the widely supported FP32 and INT8, as shown in Table 1(c). When we switch the data precision of ResNet50 from FP32 to INT8, we observe a 3.8× and 3.5× reduction in latency in TFLite 2.1 and TFLite 2.7, respectively.

In summary, these results indicate that *any alteration in the device, framework, CPU frequency, or data precision within a platform can significantly affect model inference latency.*

2.2 Challenges to platform-specific prediction

However, these findings pose a significant challenge to current platform-specific latency prediction approaches [6, 7, 18, 31, 32, 54]. Trained predictors cannot be directly applied to new platforms, requiring a costly rebuilding process for adaptation.

Predicted Platform		New Platforms			
Xiaomi12 CPU, ORT		Xiaomi11 CPU, ORT		Xiaomi12 CPU, TFLite 2.1	
Accuracy	RMSE	Accuracy	RMSE	Accuracy	RMSE
99.84%	12.2 ms	**0%**	**57.6 ms**	**0%**	**220.6 ms**

Table 2: Directly apply nn-Meter's predictors on new platform results in a poor prediction accuracy of 0%.

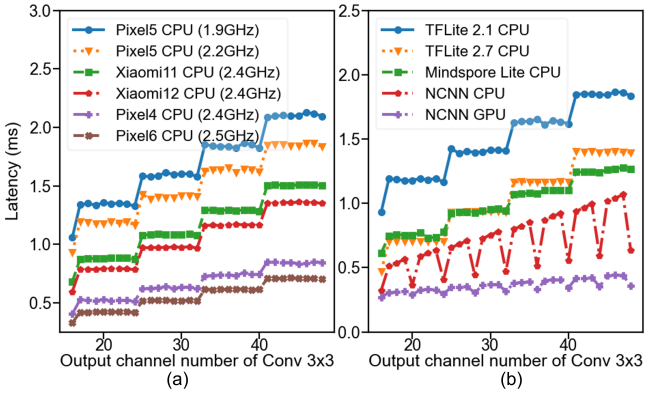

Figure 1: (a) Under the same TFLite 2.1, mobile ARM CPUs exhibit a similar staircase latency pattern; (b) On Xiaomi11, various frameworks exhibit similar latency patterns. Config: HW=56, Stride S=1, Input channel C_{in}=16.

Poor prediction accuracy on new platforms. nn-Meter [54] leads in device-specific latency prediction. By building predictors with 42k training data on Xiaomi12 CPU with ORT, it achieves an impressive 99.8% accuracy in predicting MobileNetV3 search space [8]. However, such highly accurate predictors perform poorly on new platforms. Table 2 shows a significant drop in accuracy to 0 when transitioning to Xiaomi11 CPU or switching to TFLite 2.1 framework.

The reason is that the objective of device-specific latency prediction is to minimize the regression errors between the predicted latency, y', and the actual latency, y. As a result, when there are significant changes in the actual latency, y, on a new platform, these device-specific predictors experience a large regression error and become unreliable.

Expensive rebuilding cost. The accuracy drop to 0 mandates an expensive rebuild of the platform-specific latency prediction method, demanding substantial training data collection on the new platform. Unfortunately, this data collection process can be extremely time-consuming, taking several days to complete for a single platform. For instance, nn-Meter usually collects around 42k kernel data points, requiring 2.5 days on a Google Pixel 4 CPU. Such high cost makes platform-specific approaches unfeasible and unscalable for handling the vast number of edge devices and various inference frameworks.

2.3 Opportunities

Observations and insights. Although numerous factors can impact model latency, we observe that similar latency behavior patterns persist across diverse platforms. This is primarily because many inference frameworks share a common goal of

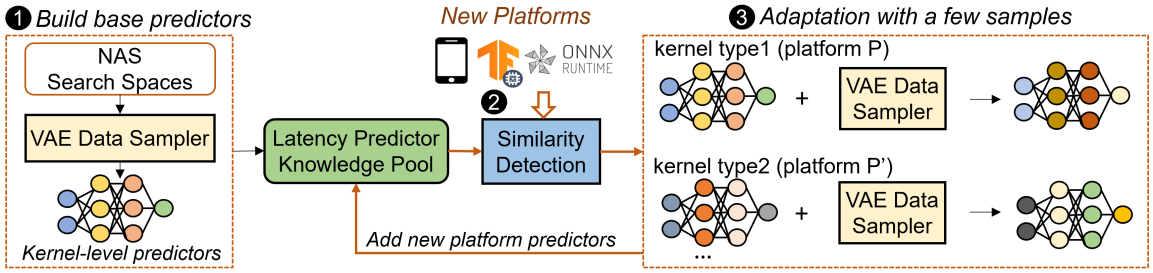

Figure 2: The overview of LitePred. (1) LitePred starts with building accurate base predictors for warmup platforms and stores them in a knowledge pool. (2) For a new platform, we match the most similar latency predictors for each kernel from knowledge pool through similarity detection. (3) We adapt identified latency predictors with a few samples from the new platform.

maximizing the utilization of underlying device resources to minimize DNN inference latency. Consequently, some critical optimization techniques are commonly applied across these frameworks. An example is operator fusion, which combines multiple operators into a single one without storing intermediate results in memory. Moreover, many frameworks leverage Neon optimizations [3] specifically designed for ARM CPUs. TFLite and Mindspore employ CPU kernels optimized for the ARM Neon instruction set, while NCNN goes a step further with assembly-level optimization for ARM Neon.

To showcase this, we conduct two experiments. In the first experiment, we keep the inference framework fixed at TFLite 2.1 and measure the latency of Conv 3x3 on various mobile phones. As shown in Fig. 1(a), despite significant absolute latency differences among various mobile CPUs, they all exhibit the same staircase latency pattern as output channels increases. In the second experiment, we explore whether similar patterns exist between various inference frameworks. Fig. 1(b) show that TFLite 2.1, TFLite 2.7 and Mindspore Lite all exhibit a very similar staircase latency pattern on CPU, while NCNN CPU and NCNN GPU display a different step pattern.

We thus reveal two key observations: *(i): mobile hardware of the same type exhibit similar latency patterns; (ii) despite varying optimizations and implementations, there exist inference frameworks display similar latency patterns.*

Opportunities. These observations motivate us to develop platform-transferable latency predictors that can eliminate the need for a costly rebuilding process when introducing new devices or frameworks in the target platform. Our insight is to train latency predictors to well learn the latency patterns for a specific platform and then transferring the shared common knowledge to a new platform. By finetuning the latency predictor with a few training samples to adapt to any new factors in the platform, we can achieve high prediction accuracy.

3 System Design

Overview. The observations in Section § 2 motivate LitePred, a lightweight approach for predicting the latency of arbitrary DNN models, through transferring pre-existing latency predictors with minimal adaptation cost across diverse edge platforms. Fig. 2 illustrates the system overview.

To start, we select a few edge platforms (e.g., 5) as warmup ones and build a set of precise kernel-level latency predictors from scratch (§ 4). Utilizing a VAE data sampler for high-quality training data collection, these predictors learn accurate latency patterns for each warmup platform and are stored in a "knowledge pool". When targeting a new platform, we perform kernel detection [54] to identify all possible kernels. Then, we identify the most similar latency predictor for each kernel type from the knowledge pool and finetune them swiftly using VAE data sampler with just a few adaptation data (§ 5). We use these finetuned latency predictors to predict kernel latencies, and sum them up as the final model latency.

Design choice. Our ultimate objective is to effectively serve hardware-aware NAS on a wide range of diverse edge platforms with LitePred. Specifically, LitePred aims to accurately predict the inference latency of any DNN models on the target platform within NAS search spaces. Guided by this goal, we design the system based on the following principles.

• *LitePred predicts latencies for DNN models in NAS search spaces*. In theory, DNN models could be arbitrary, resulting in a vast number of possibilities that complicate latency prediction. Yet, many models are inferior in accuracy and therefore disregarded. In our work, we focus on top-tier model search spaces in NAS, crafted by AI experts and known for their success in finding accurate models. We collect 5 CNN NAS search spaces and 1 vision transformer NAS search space for our final evaluation of latency prediction. These search spaces include a vast number of 10^{20} high-quality models.

• *The accuracy of initial base latency predictors is crucial for successful transfer across devices.* Our approach starts by training dedicated, accurate latency predictors for warmup platforms, which learn the latency patterns resulting from various inference optimizations. These predictors serve as a source of knowledge for transferring to new platforms. If the base predictor fails to achieve high accuracy, transferred predictors will have inferior performance. For example, a more accurate predictor (90% base accuracy) outperforms an inferior one (67.1% base accuracy) with 1.6× less adaptation data to achieve the same finetuned accuracy.

• *LitePred detects the most similar pre-existing latency predictors.* Instead of randomly selecting a predictor from the

knowledge pool, LitePred uses a similarity-based approach to identify the most similar predictor for a new platform. This allows for the reuse of learned knowledge and leads to reduced adaptation data samples and lower measurement costs. Intuitively, the most similar predictor needs the least amount of adaptation data and delivers the best prediction accuracy.

• *We predict latency at the kernel-level to simplify predictors transfer to new platforms.* While HELP [30] and One-Proxy [33] use model graphs to predict latency, LitePred takes a kernel-level latency approach. We divide a model into kernels and sum up their predicted latencies as the model latency, since kernels are sequentially executed on edge platforms. This is advantageous as kernels are the basic scheduling units in inference frameworks, covering framework optimizations (e.g., operator fusion) and specific kernel algorithms (e.g., Winograd convolution [28]). Latency prediction at the kernel level simplifies cross-platform predictor detection and requires less adaptation data than model-level prediction.

Technical challenges. LitePred faces two major technical challenges: *(i) Given the exponentially large NAS search spaces, how to efficiently collect high-quality data for training base predictors and finetuning pre-existing predictors? (ii) How to find most similar predictors and adapt them on new platforms with minimal profiling costs?*

4 Build Accurate Base Latency Predictors

In this section, we introduce our approach for training base latency predictors on warmup platforms and propose VAE data sampler to efficiently collect high-quality training data.

4.1 NAS search spaces and the challenges

We begin by studying model distributions in NAS search spaces. Then, we discuss the challenges in data collection.

NAS search space collection. We collect 5 widely-used CNN and 1 vision transformer NAS search space, including OFA-MobileNetV3 [22], ProxylessNAS [9], OFA-ResNet [8], BigNAS [53], FBNetV3 [14] and AutoFormer [10]. These search spaces are of high quality (i.e., models have the potential for high accuracy) and specifically tailored to edge-regime DNNs. In total, the 6 search spaces contain an impressive of 10^{20} different models, representing a vast prediction scope.

Multi-dimensional distribution of kernel configurations. As elaborated in Section § 3, LitePred builds latency predictors at kernel level. To construct these kernel latency predictors, we need to collect a large amount of training data consisting of configuration and latency pairs for each kernel type. However, this presents a practical challenge due to the vast configuration space of common CNN model kernels, such as Conv+bn+relu. This kernel has five primary configuration dimensions (*input height and width HW, kernel size K, strides S, input channels C_{in}, and output channels C_{out}*), resulting in a large number of possible configurations. Profiling on-device latency for every possible configuration on every target platform can be prohibitively expensive.

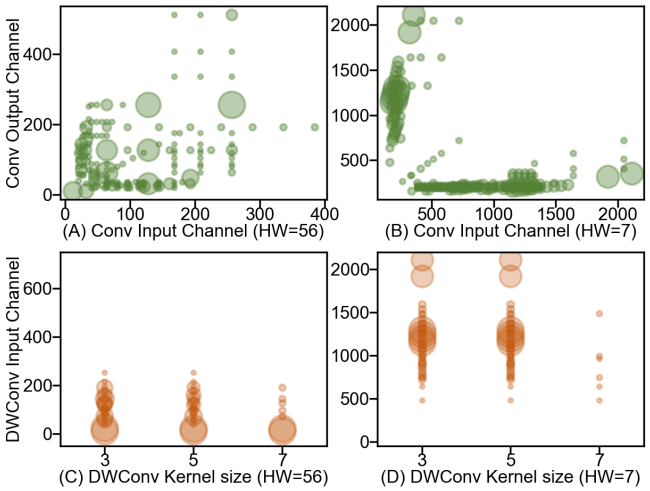

Figure 3: The multi-dimensional distributions of Conv/DW-Conv configurations in 5 widely-used NAS search spaces. (AB): C_{in} vs. C_{out} exhibit different patterns under different HW. (CD): The kernel size KS and C_{in} of DWConv also exhibit different distributions under different HW. Larger circle size indicates that the configurations have larger frequency.

Fortunately, we observe that each kernel configuration displays a unique and smaller multi-dimensional joint distribution in NAS search space. To illustrate this, we collect all the configurations $(HW, K, S, C_{cin}, C_{out})$ for Conv and DWConv kernels from 5 CNN search spaces. As shown in Fig. 3(AB), the values of C_{in} and C_{out} of Conv kernels under different HW display distinct distributions. Fig. 3(CD) demonstrates that kernel size KS also has a different distribution under varying channel numbers C_{in} and input size HW. *This allows us to gather training data conforming to the distribution, rather than collecting data for all possible configurations.*

Challenges in collecting high-quality training data. However, gathering training data aligned with the multi-dimensional distribution poses a challenge. Random sampling can result in the collection of irrelevant data and lower accuracy. nn-Meter introduces an adaptive data sampler that constructs a probability distribution per dimension and performs independent sampling. However, since the dimensions are highly correlated (Fig. 3), independent sampling can still trigger out-of-distribution issues. While uniform sampling from search spaces resolves this, it introduces data leakage problems by revealing evaluation data during training, limiting the generalizability to new search spaces. Therefore, we call for a new sampling approach. In next section, we introduce VAE data sampler to address all these challenges.

4.2 Efficient VAE data sampler

To collect high-quality training and adaptation data, we divide the process into two specific tasks: *(i)* learning the multi-dimensional joint distribution of different kernels, and *(ii)* generating new data based on the distribution. In this section, we take inspiration from the concept of Variational Autoencoders (VAE) [27] and propose a VAE data sampler.

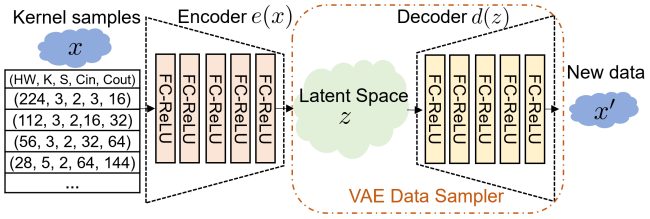

Figure 4: Our VAE data sampler for collecting training data.

VAEs are a type of generative neural network, originally popularized for their applications in image and text generation tasks [40, 41, 43]. They learn to represent complex data distributions by encoding input data into a lower-dimensional latent space and then decoding it back to the original space. VAEs comprise two primary components: an encoder network that maps input data to the latent space, and a decoder network that reconstructs the input data from this latent space.

Inspired by the encoding and decoding process of VAEs, we propose training an encoder-decoder VAE model to learn the multi-dimensional distribution for each kernel. Then, we utilize the trained decoder as our data sampler, generating new samples that adhere to each kernel's distribution within the NAS search space. Next, we will discuss the application of VAEs for sampling data to train kernel latency predictors.

VAE model design. Fig. 4 illustrates the design of our VAE model, which consists of a 5-layer Linear encoder and a 5-layer Linear decoder with a hidden dimension of 256. The latent space dimension size is empirically set to 128. For each kernel type (e.g., Conv+bn+relu), we carefully sample several configurations as input x, representing its distribution in search spaces. The encoder maps input x to the latent space z, which is assumed to follow a multivariate Gaussian distribution [19]. The decoder then maps from the latent space back to the input space, generating data x' that adhere to the same distribution as the original input x. Note that x and x' follow the same distribution, but with minimal values overlap.

Input data x. To train the VAE model, it is crucial to collect representative configurations from search spaces, which characterizes the distribution for each kernel and acts as the target distribution for the input data x, which the VAE aims to learn.

However, collecting such data is non-trivial, as many latency-dominated configurations have low frequencies in NAS search spaces. For instance, Conv and DWConv kernels' latency is significantly larger in the first 3 layers, contributing up to 44% of model latency. Accurate prediction of these configurations is crucial for precise latency prediction. However, they represent only 1.52% in NAS search spaces, with most configurations in the middle layers. Naive uniform sampling can result in the omission of latency-dominating data.

We tackle this challenge by performing data normalization on latency-dominated large kernels. We assign a larger weight to Conv and DWConv related kernels. Also, since the number of kernel configurations differs for each search space, we collect the x proportionally based on their relative quantities.

Training the VAE model. After collecting the representative distribution x for each kernel type, the next step is to train the VAE model that learns the multi-dimensional distribution for each kernel and generates new configurations x' based on it. The training objective aims to minimize the distance between the decoder output x' and the encoder input x.

We follow the original VAE work [27] and use two loss functions: a Mean Squared Error (MSE) reconstruction loss [2] and a Kullback-Leibler (KL) divergence [1]. The reconstruction loss measures the difference between the decoder output x' and the original input data x. The KL divergence measures the difference between the encoder output distribution in the latent space and the standard normal distribution. We train the VAE model to minimize the total loss function. After the training is done, the decoder is able to map the latent space z back to the original kernel configuration distribution.

Generating kernel configuration data via the decoder. Once the VAE is trained, we can employ the decoder to generate new kernel configuration data x' that closely aligns with the distribution x in NAS search spaces. As illustrated in Fig. 4, to generate N configuration data for Conv+bn+relu kernel, we first sample N vectors from the latent space, which follows a multivariate Gaussian distribution. We then pass these N vectors through the decoder, resulting in the generation of N new configurations for Conv+bn+relu. Note that the original decoded configuration data x' are continuous values. We apply a straightforward round-to-nearest strategy, mapping continuous values to the closest valid discrete value.

4.3 Build transferable base latency predictors

The above VAE data sampler efficiently samples high-quality training data for each kernel. This section outlines designing latency predictor models that accurately predict latency on warmup platforms using the collected data. Moreover, these predictors are designed for easy transfer to new platforms.

Conventional platform-specific prediction methods [6, 7, 11, 54] typically rely on decision-tree-based machine learning regressors to create latency predictors. For example, models like RandomForest and XGBoost regression are often used, which fit the training data by minimizing the difference between actual and predicted latency values. However, these decision-tree-based regressors require a complete retraining process, making it impossible to reuse fitted predictors for transferring to new platforms.

To achieve accurate and transferable latency prediction, we introduce a DNN model. DNN models have demonstrated strong performance in transfer learning tasks [39, 59]. Specifically, our latency model is a 16-layer Multilayer Perceptron (MLP) network, a small model with $\sim$1 million parameters.

We now describe the training process. For each kernel, we measure the latency of VAE sampled kernel configurations on warmup platforms. The prediction features are kernel configurations, FLOPs, and parameter size, while the corresponding inference latency is the label. Our training objective is to minimize the Mean Absolute Percentage Error (MAPE)

loss between the predicted and actual latency values. We use MAPE loss because we follow previous work [18, 54], which uses ±10% accuracy as the evaluation metric. This metric calculates the percentage of kernels with predicted latency within ±10% error. Minimizing MAPE loss aligns with our goal of maximizing ±10% accuracy.

LitePred trains base predictors for five randomly selected warmup platforms listed in Table 3 and stores them in the knowledge pool, as illustrated in Fig. 2. In the following section, we will describe our method for reusing and transferring these existing predictors to new platforms.

5 Transfer Predictors to New Platforms

For a new edge platform, LitePred addresses the challenge of identifying the most similar kernel predictors from the knowledge pool. It then transfers these pre-existing predictors to the new platform using minimal adaptation data.

5.1 Similar predictor detection for each kernel

Identifying the most similar latency predictor for each kernel on new platforms is challenging as most edge platforms are black boxes. OneProxy [33] suggests using Spearman's Rank Correlation Coefficient (SRCC) to assess the statistical dependence between latency rankings of models. However, this method has limitations. First, the SRCC is based on the total model latency, which might not accurately identify the most similar platform. In contrast, our experiments shows that the most similar platform predictor varies by the kernel type. Second, it depends on random model sampling to evaluate platform similarity, which is typically sparse and fails to reflect specialized optimizations on the target platform.

Overview of our approach. In our work, we introduce a lightweight latency-distribution based similarity detection approach. Instead of focusing on the similarity of the entire model, we detect similarity at the kernel level since on-device optimizations are usually implemented at this level. *A pre-existing kernel latency predictor is considered similar for the target platform if its predicted latency displays a similar distribution to the real latency.* To achieve this, we design a small set of representative kernel configurations that capture both kernel distribution and platform-specific optimizations, rather than randomly sampling or relying solely on the VAE data sampler. This enables us to identify the most similar latency predictors more effectively while keeping the approach lightweight.

Fig. 5 illustrates the overall process. We design a small set of representative configurations for each kernel, and measure the actual latency of under these configurations on the platform, denoted as Y_r. For each latency predictor in the knowledge pool, we predict the latency of these configurations as Y_p. We then calculate the similarity score using KL divergence between the actual latency Y_r and predicted latency Y_p. Finally, we return the predictor with lowest KL divergence (i.e., the highest similarity score).

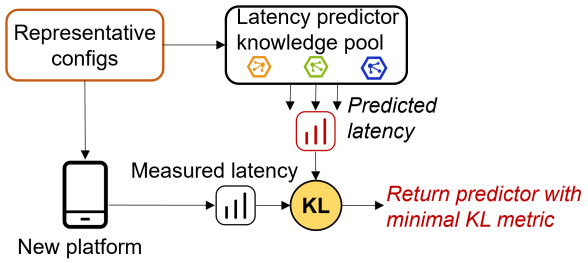

Figure 5: Our proposed similarity detection to identify the most similar predictor for each kernel type on new platforms.

Representative data for computing similarity. The effectiveness and cost of our similarity detection technique heavily rely on the quality of representative data. We now introduce how we design it for computing predictor similarity between diverse platforms. We consider two types of kernel configurations: *(i)* configurations in the search spaces that reflect the underlying distribution; and (ii) specifically designed configurations that capture the latency patterns on the target platform.

Type (i) data collection employs our VAE data sampler. To collect type (ii) data that reflects specific on-device optimizations, we adopt a simple but effective fine-grained approach. This is inspired by Fig. 1, where it reveals that device-specific staircase latency patterns necessitate dense profiling to uncover. In contrast, VAE and random samplers exhibit sparse coverage, which can distort the profiled latency patterns.

Specifically, we select an initial configuration from search spaces per kernel. We fix all dimensions except one, and perform fine-grained sampling on that dimension. We sample 16 continuous points in the channel number dimension and enumerate {1, 3, 5, 7} in the kernel size dimension. For example, for Conv+bn+relu kernel with the configuration (56, 3, 2, 16, 16), we fix the HW, K, S and C_{in} dimensions, and generate 16 continuous points (e.g., 16 to 32) in the output channel dimension. Then, we fix the HW, S, C_{in}, and C_{out} dimensions, and enumerate {1, 3, 5, 7} in the kernel size dimension.

Similarity score. To compute the similarity of latency distributions between each pre-existing predictor and a target platform, we use the Kullback-Leibler (KL) divergence, a widely used statistical metric for comparing probability distributions. First, we obtain the real kernel latency for representative configurations as Y_r. Next, for each pre-existing predictor i that is trained or finetuned for a different platform, we obtain its predicted latencies Y_p^i under the representative configurations. Finally, we compute the KL divergence between the probability distributions P and Q that correspond to Y_p^i and Y_r, respectively. The KL divergence is defined as follows:

$$D_{KL}(P||Q) = \sum_{y_r \in Y_r, y_p \in Y_p} P(y_p) \log \frac{P(y_p)}{Q(y_r)}$$

For each kernel, the predictor with the lowest KL divergence is chosen for adaptation to the new target platform. Note that the cost of our similarity detection method comes from the profiling overhead of the representative configurations, which typically takes less than 10 minutes.

5.2 New platform adaptation

After identifying the most similar latency predictors for all kernels, we introduce the method for adapting these predictors to the new platform through a finetuning process.

Our method, illustrated in Fig. 2, begins by sampling adaptation configurations on the target platform using a VAE data sampler and measuring their latencies. We then finetune the identified predictors using the sampled data, updating their weights to improve accuracy on the new platform.

Increasing adaptation data typically improves prediction accuracy but raises profiling costs. Our experimental results indicate that a small number of samples (less than 500) can achieve high accuracy when finetuning on a new platform. This is because we identified very similar latency predictors, which greatly reduces the required amount of adaptation data. As a result, our approach is significantly more cost-effective than traditional platform-specific prediction methods.

6 Implementation

Training the VAE. For large kernels such as Conv+bn+relu and DWConv+bn+relu, we collect 2000 configurations to train the VAE model as described in Sec.§ 4.2. For smaller kernels such as Squeeze&Excitation (SE) [23] and bn+relu, we collect 500 configurations. Our VAE model is implemented based on the Pytorch implementation [44]. We train the model for 2k epochs. During each training step, we randomly sample a batch of 16 configurations based on their frequency. We use the Adam optimizer [26] with a learning rate of 0.001 and decay rates of (0.9, 0.999).

Build base latency predictors with VAE decoder. After completing VAE training, we utilize the learned latent space and the decoder to gather training data for constructing base latency predictors for each kernel type. Specifically, to sample N configurations, we generate N 128-dimensional vectors from multivariate Gaussian distribution and feed them into the decoder. The N decoder outputs are used as training data. We sample up to N=10k configurations for Conv and DWConv related kernels and N=1000 for small kernels.

For each kernel type, we generate the corresponding model graph based on the sampled configurations. Then, we measure the inference latency on 5 random warmup platforms to gather training data for building base latency predictors. Each predictor is trained for 350 epochs using the AdamW optimizer with a cosine learning rate scheduler, and the initial learning rate is set to 0.001. The training cost is feasible. On an Nvidia RTX 2080Ti, it takes 26 minutes to train latency predictors for large kernels and only 6 minutes for small kernels.

Transfer to new platforms. For a new platform, we first conduct kernel detection [54] to identify all possible kernels. Then, we perform similarity detection for each kernel to find the most similar predictor from the knowledge pool. Specifically, we identify similar predictors for Conv and DWConv kernels using 400 representative data. For small kernels, we reuse the detected platform for Conv kernels and use the corresponding predictor, as small kernels usually have same platform detection results as Conv kernels.

Device	CPU	GPU	CPU Frequency
Pixel 4	Qualcomm Snapdragon 855	Adreno 640	2.4GHz, 2.1GHz
Pixel 5	Qualcomm Snapdragon 765G	Adreno 620	2.2GHz, 1.9GHz
Pixel 6	Google Tensor SoC	Mali-G78	2.5GHz, 2.2GHz
Xiaomi 11	Qualcomm Snapdragon 888	Adreno 660	2.4GHz, 2.1GHz
Xiaomi 12	Qualcomm Snapdragon 8 Gen 1	Adreno 730	2.4GHz, 2.1GHz
Inference engines	TFLite 2.1, TFLite 2.7, NCNN, Mindspore Lite, Onnxruntime		
Precision	FP32, INT8		

Table 3: Our **85** evaluated platforms, including 10 different hardware and CPU frequencies, 5 popular inference frameworks on edge and 2 data precision.

To finetune predictors, we use our VAE data sampler to generate a few configurations and measure their latency. We sample 100 configurations for small kernels and 500 for large kernels like Conv and DWConv. If accuracy is unsatisfactory, we iteratively sample 500 more data points per round until the desired accuracy is reached. During finetuning, we use detected kernel weights as initial weights and finetune for 300 epochs. We follow the same training settings as the base predictor, but with a smaller learning rate of 0.0005.

7 Evaluation

7.1 Experiment Setup

Platforms. We evaluate LitePred on a wide variety of edge platforms, as detailed in Table 3. Our evaluation includes 10 mobile hardware (CPU and GPU), 5 popular inference engines for edge devices, and 2 data precision options (FP32 and INT8 for CPU devices). Besides, we test each CPU device under two frequencies. In total, we evaluate 85 different platforms, a significantly larger number than previous works. This large-scale evaluation is achieved by our proposed lightweight, scalable, and transferable latency prediction paradigm. Without it, the cost of such an evaluation would be extremely expensive.

Evaluation datasets. To evaluate the effectiveness of LitePred in hardware-aware NAS, we build latency datasets using the 6 high-quality NAS search spaces (ref Section 4.1). We randomly sample 4k models from each search space and measure their latency on our 85 platforms. In total, the dataset contains 1.86 million model and latency pairs.

For each platform, we measure the model inference latency by performing a warmup of 10 inference runs and then calculating the average latency over 50 subsequent inference runs. For CPU platforms, we set the CPU frequency to the target value and measure the corresponding latency. For INT8 precision, we measure the INT8 latency on TFLite platforms, as TFLite has good support for this precision. Specifically, we use TFLite's official tools [46] to quantize models to INT8 precision and follow the standard process to measure latency.

Comparison baselines. We compare LitePred with state-of-the-art latency prediction methods by implementing two strong baselines: (1) nn-Meter [38, 54], a platform-specific method; and (2) HELP [30], a platform-agnostic method.

Metrics. We evaluate three metrics: Root Mean Square Error (RMSE), prediction accuracy, and profiling cost. RMSE quantifies the errors between predicted and real latency. Since RMSE can be highly influenced by the range of real latency values, we also report prediction accuracy. Following related works [18, 54] , we measure prediction accuracy as the percentage of models whose predicted latency error falls within $\pm 5\%$ and $\pm 10\%$ of the real measured latency. For profiling cost, we report both the number of sampled kernels and the total on-device measurement time cost.

7.2 Key findings

Before presenting detailed results, we summarize our valuable findings and insights gained from extensive experimentation:
(i): Despite differences in hardware, software inference engine implementations, data precision, and CPU frequencies among edge platforms, LitePred successfully transfers pre-existing latency predictors and achieves 99.3% accurate latency prediction across diverse edge platforms.
(ii): On most edge platforms, LitePred requires $<$1 hour of adaptation profiling cost. Compared to state-of-the-art platform-specific baselines, it achieves up to 5.3% higher accuracy and reduces profiling costs by 50.6×.
(iii): The detected most similar predictor's corresponding platform varies depending on the kernel type. (Table 5).
(iv): The data required for adaptation depends on the similarity between the detected platform predictor and the target platform. A highly similar predictor requires minimal data, while less similar predictors demand more data for finetuning.
(v): By enabling hardware-aware NAS, LitePred facilitates the development of efficient DNN models for various edge platforms. Remarkably, our searched models outperform MobileNetV2 by up to 4.4% in accuracy on the ImageNet dataset.

7.3 Evaluation on diverse edge platforms

7.3.1 Comparison with baseline methods

We demonstrate the effectiveness of LitePred by comparing with state-of-the-art latency prediction baselines.
Setup. We select 4 out of 85 platforms for comparison and use the higher frequency of the CPU device. By default, we use FP32 data precision unless stated otherwise. Since HELP performs model-level latency prediction, a separate meta latency predictor needs to be designed and trained for each search space. For simplicity, we choose MobileNetV3 search space, which HELP already supports, as the evaluation dataset.

For nn-Meter, we use the official code [38] to sample training data and train kernel latency predictors for each target platform. For HELP, we initially adapt its meta predictor using 10 randomly sampled models from target platform. However, the accuracy is poor. Therefore, we add two edge platforms (i.e., Pixel5 CPU with MindSpore, and Xiaomi11 GPU with NCNN) to retrain the meta latency predictor for better prediction. During evaluation, we increase the number of adaptation models and allow the profiling cost to be the same as ours. We refer to this improved implementation as HELP*.

Platform	Method	Train Data	Cost	RMSE	Prediction Acc ±5%	Prediction Acc ±10%
Xiaomi11 CPU Mindspore	HELP	10 models	12.44s	6.6 ms	11.5%	22.5%
	HELP*	1030 models	0.35h	4.1 ms	39.3%	48.7%
	nn-Meter	234997 kernels	16.23h	0.8 ms	78.0%	98.9%
	Ours	4800 kernels	**0.35h**	**0.4 ms**	**95.4%**	**100%**
Xiaomi11 CPU NCNN	HELP	10 models	10.87s	9.5 ms	15.4%	23.0%
	HELP*	3000 models	0.88h	6.7 ms	37.1%	49.1%
	nn-Meter	169305 kernels	20.17h	0.4 ms	96.4%	100%
	Ours	11400 kernels	**0.88h**	**0.3 ms**	**99.5%**	**100%**
Pixel 5 GPU TFLite 2.7	HELP	10 models	2.66s	1.2 ms	13.9%	28.0%
	HELP*	2500 models	0.62h	0.8 ms	51.6%	61.1 %
	nn-Meter	104996 kernels	7.94h	0.8 ms	37.7%	95.8%
	Ours	11900 kernels	**0.62h**	**0.3 ms**	**95.9%**	**99.9%**
Pixel 5 GPU NCNN	HELP	10 models	16.41s	12 ms	7.9%	16.8%
	HELP*	2100 models	0.96h	7.5 ms	33.5%	50.8 %
	nn-Meter	397384 kernels	48.60h	1.6 ms	52.2%	94.7%
	Ours	17400 kernels	**0.96h**	**0.9 ms**	**92.6%**	**100%**

Table 4: LitePred outperforms both state-of-the-art platform-specific and platform-agnostic baselines by achieving higher prediction accuracy with significantly lower sampling costs.

Results and analysis. Table 4 summarizes the comparison results. LitePred consistently outperforms both platform-specific and platform-agnostic prediction baselines by achieving higher prediction accuracy and lower RMSE, all while requiring significantly lower sampling costs. Compared to platform-agnostic methods, LitePred conducts much more precise latency prediction on new platforms, with over 92% of models whose predicted latency error falls within $\pm 5\%$ of the real latency. In contrast, HELP achieves only 12% accuracy on average, and the improved HELP* achieves 40.4%.

Furthermore, LitePred achieves much higher $\pm 5\%$ prediction accuracy than platform-specific baseline while reducing prediction costs to within 1 hour. Compared to nn-Meter, LitePred speedups the prediction cost by 46.4×, 22.9×, 12.8× and 50.6× on the four platforms, respectively. These results demonstrate the remarkable effectiveness of LitePred in terms of both prediction accuracy and efficiency.

7.3.2 Transfer to diverse new platforms

We now evaluate the effectiveness of LitePred in latency prediction on a wider range of new platforms by transferring pre-existing predictors. Due to space limit, we select 14 different edge platforms, covering a variety of hardware types, frequencies, data precisions, and inference frameworks.
Setup. We conduct two experiments to demonstrate that LitePred can adapt well to any new edge platforms. **(a)**: we select the most similar kernel predictors from knowledge pool, regardless of the hardware or inference engines for the detected predictors. **(b)**: To demonstrate that LitePred can transfer latency predictors across different hardware and inference engines, we exclude the predictors with the same inference engines as target platform from the knowledge pool and select the most similar predictors from the remaining ones.
Results and analysis. Table 5 summarizes the detailed results, including the detected similar kernel predictors, the required data and time cost for adaptation, and the prediction accuracy over our benchmark dataset on each target platform.

Platform	Similar Platforms		#Adaptation		Prediction Accuracy	
	Conv kernel	DWConv kernel	Data	Cost	±5%Acc	±10%Acc
(a) Selecting most similar kernel predictors from the whole knowledge pool						
Xiaomi11CPU, ORT	Xiaomi12 CPU, ORT	Xiaomi12 CPU, ORT	1400	0.48h	90.5%	98.9%
Pixel5 GPU, NCNN	Xiaomi11 GPU, NCNN	Xiaomi11 GPU, NCNN	17400	0.96h	84.3%	99.1%
Xiaomi11 CPU, MindSpore	Pixel5 CPU, MindSpore	Xiaomi12 CPU, MindSpore	4800	0.35h	90.4%	99.9%
Xiaomi11 GPU, TFLite 2.7	Xiaomi12 GPU, TFLite 2.7	Xiaomi12 GPU, TFLite 2.7	11000	0.17h	83.7%	98.6%
Xiaomi11 CPU, NCNN	Xiaomi11 CPU, MindSpore	Pixel5 CPU, NCNN	11400	0.88h	80.3%	98.9%
Pixel6 CPU, TFLite 2.1	Xiaomi12 CPU, TFLite 2.1	Xiaomi12 CPU, TFLite 2.1	3500	0.16h	79.4%	100%
Pixel5 CPU, TFLite 2.7	Xiaomi11 CPU, TFLite 2.7	Xiaomi11 CPU, TFLite 2.7	3400	0.13h	79.6%	99.2%
Xiaomi12 CPU, TFLite 2.7, INT8	Xiaomi11 CPU, ORT	Pixel5 GPU, TFLite 2.7	3100	0.05h	95.7%	100%
(b) Similarity detection of kernel predictors *Excluding* same inference frameworks						
Xiaomi11 CPU, ORT	Pixel5 CPU, MindSpore	Pixel5 GPU, NCNN	2400	0.72h	84.2%	99.2%
Xiaomi12 GPU, TFLite 2.7	Pixel5 GPU, NCNN	Xiaomi12 CPU, MindSpore	16100	0.22h	79.4%	98.7%
Xiaomi11 CPU, Mindspore	Pixel5CPU, TFLite 2.7	Pixel5GPU, TFLite 2.7	9700	0.80h	98.1%	99.2%
Pixel5 GPU, NCNN	Xiaomi12 CPU, TFLite 2.1	Xiaomi11 CPU, ORT	18500	1.73h	86.5%	99.3%
Xiaomi12 CPU, TFLite 2.1, low Freq	Xiaomi11 GPU, NCNN	Pixel5 CPU, MindSpore	1800	0.18h	94.7%	100%
Xiaomi12 CPU, TFLite 2.1	Pixel4 CPU, TFLite 2.7	Pixel5 CPU, MindSpore	1800	0.10h	97.6%	99.9%

Table 5: Transferable latency prediction of LitePred on diverse new platforms. LitePred accurately predicts the inference latency of models across five different CNN NAS search spaces, with minimal adaptation cost (0.05 to 1.73 hours) on new platforms.

Table 5 demonstrates that LitePred achieves superior accuracy in predicting latency on diverse new edge platforms equipped with varying hardware, inference engines, data precision, and frequencies. Remarkably, we achieve an average of 99.3% transfer accuracy, with 87.0% of models having prediction errors within a negligible 5% margin.

Furthermore, not only does LitePred accurately predict inference latency on unseen new platforms, but it also requires only ∼4,000 adaptation data points for finetuning all kernel predictors across most of the platforms we evaluated. This leads to <1 hour of measurement overhead, which is a significant improvement over platform-specific methods that typically require 1-3 days of measurement for a single platform.

In Table 5, we observe that more adaptation data is needed for finetuning kernel predictors on GPUs with NCNN/TFLite 2.7 and CPUs with NCNN. This is due to two main reasons. First, models generate 4 new kernels in the NCNN framework: Conv+bn+swish/hswish and DWConv+bn+swish/hswish, requring the use of similar predictors and more adaption data points. Second, edge GPU platforms typically require more adaptation data for Conv and DWConv related kernels due to specific optimizations on various GPUs, making it challenging to find a highly similar platform. Thus, more data is needed for the predictor to learn these optimizations.

7.4 Ablation study

We now conduct ablation studies to assess the effectiveness of each of our techniques.

The effectiveness of VAE data sampler. An effective data sampler is crucial for improving prediction accuracy and avoiding useless data. To evaluate the effectiveness of our VAE data sampler, we compare it with the state-of-the-art adaptive data sampler proposed in nn-Meter on 4 platforms.

Platform	Method	Conv Acc.	DWConv Acc.
Xiaomi11 CPU MindSpore	Adaptive data sampler	84.9%	52.8%
	VAE data sampler	**91.4%**	**93.6%**
Xiaomi11 CPU NCNN	Adaptive data sampler	81.5%	95.5%
	VAE data sampler	**88.8%**	**98.3%**
Pixel5 GPU TFLite 2.7	Adaptive data sampler	61.6%	86.7%
	VAE data sampler	**76.7%**	**89.1%**
Pixel5 GPU NCNN	Adaptive data sampler	65.6%	79.6%
	VAE data sampler	**87.1%**	**81.7%**

Table 6: Under the same sampling budget of 10k data points, VAE data sampler outperforms state-of-the-art methods with achieving much higher latency prediction accuracy.

We sample 10k configurations using both data samplers to train the latency predictor for Conv and DWConv kernels.

Table 6 presents a comparison of the achieved prediction accuracy. Our VAE data sampler consistently outperforms the adaptive data sampler, achieving much higher prediction accuracy on all four platforms. This is because we can sample configurations that conform to the multi-dimensional distribution in NAS search spaces. In contrast, the adaptive data sampler is constrained to align solely with individual dimensions, leading to the out-of-distribution issue.

The effectiveness of our similarity detection technique. We evaluate whether our method detects the most similar latency predictor for target platform. We set up two baselines: *(i)* OneProxy [33], choosing the predictor with the highest SRCC metric by comparing predicted and real latency rankings for random kernel configurations, and *(ii)* Random, selecting a predictor from the knowledge pool randomly. For the selected kernel predictor, we use VAE data sampler to generate the same adaptation data for finetuning. A more similar predictor is expected to have higher accuracy after finetuning on the

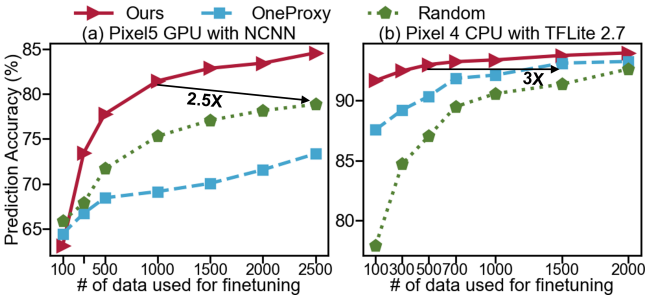

Figure 6: Finetuned accuracy of Conv kernel on new platforms. By our similarity detection, we achieve higher accuracy with 2.5× less adaptation data than baseline methods.

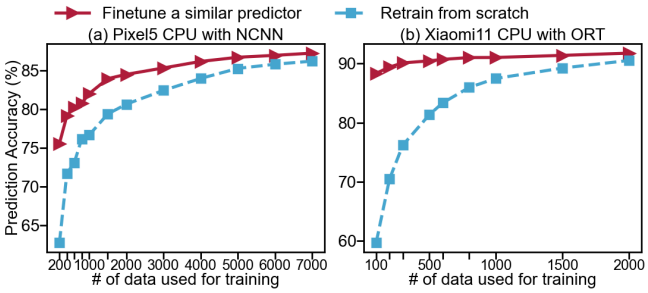

Figure 7: Prediction accuracy of Conv kernel. Finetuning a similar pre-trained predictor yields higher accuracy than training a new predictor from scratch on the same platform.

new platform and require less adaptation data.

Fig. 6 compares the finetuned accuracy of Conv kernel predictor on two platforms. Our method consistently identify the most similar predictors for the target platform, resulting in significantly higher accuracy across varying amounts of adaptation data. When achieving same accuracy, our detected predictors require 2.5× and 3× less adaptation data than those required by random selection and OneProxy, respectively.

The effectiveness of finetuning pre-existing predictors. With sufficient time, it's possible to train kernel latency predictors from scratch for a new platform using VAE data sampler. Our experiment shows that finetuning a similar pre-trained predictor yields higher prediction accuracy with less training data. As shown in Fig. 7, finetuning leads to faster convergence and better accuracy. This finding is consistent with the intuition of our work, as a similar predictor can already capture some latency behaviors for the target platform.

Transfer cost analysis. We now analyze the transferring costs in Table 5. Our results show that major adaptation costs come from finetuning Conv and DWConv kernels. Conv finetuning uses **51.1%** of the adaptation data, and DWConv uses **32.5%**. Small kernels need only 100 points, while SE kernels need slightly more, ranging from 300 to 700 points.

We further analyze the impact of varying amounts of adaptation data on prediction accuracy. We select two platforms: Pixel5 GPU with NCNN, which required the largest amount of data, and Xiaomi12 CPU with TFLite 2.1, which only required 1800 samples. The results, shown in Fig. 8, indicate

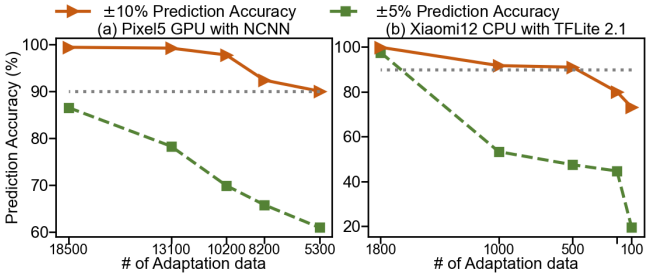

Figure 8: Model-level prediction Acc. vs. data amounts.

a large drop of ±5% accuracy when reducing the adaptation data, followed by a more gradual decline at ±10% accuracy. If we set a relaxed threshold of 90% ±10% accuracy as acceptable number, then we can reduce the amount of adaptation data by 3.5× and 3.6× for the two platforms, respectively.

7.5 Hardware-aware NAS with LitePred

We now showcase how LitePred effectively supports hardware-aware NAS in finding accurate, low-latency DNNs for various edge platforms. We integrate LitePred with a state-of-the-art NAS approach, called OFA [8], and conduct latency-constrained search for 4 different edge platforms. It is worth noting that we test 2 edge GPU platforms, which are rarely evaluated in current hardware-aware NAS due to the challenges of accurately obtaining latency on them. Specifically, we search 5k model architectures for each given latency constraint and select the model with highest validation accuracy on ImageNet 2012 dataset [15]. We then evaluate the test accuracy of the final model and measure the on-device latency.

Table 7 compares the best searched model accuracy with MobileNetV2 [42] and MobileNetV3 [22], which are state-of-the-art lighweight CNNs designed for edge platforms. Results show that OFA with LitePred delivers better models than MobileNets, achieving higher accuracy on ImageNet and lower latency on 4 diverse edge platforms. Our searched models surpass MobileNetV2 by up to 4.4% higher ImageNet accuracy. Our results proves LitePred's value as a tool for hardware-aware NAS in designing higher accuracy models that meet specific latency constraints on diverse edge platforms.

7.6 LitePred on Transformer models

We demonstrate LitePred's generalization ability on other DNN architectures by assessing transfer latency prediction on vision transformers [10, 16, 48]. We test on mobile CPUs with TFLite platforms, which have good support for transformer models. We build latency predictors for five vision transformer sub-modules: MultiHeadAttention, PatchEmbedding, MLP, LayerNormalization, and Linear layers. Specifically, TensorFlow's optimizations for MultiHeadAttention remove the need for fine-grained kernel detection.

Table 8 shows the latency prediction accuracy for AutoFormer [10] search space. The high transferable prediction accuracy showcases LitePred's effectiveness in transferring latency predictors for vision transformers across edge platforms with varying devices, data precision, and CPU frequencies.

Method	Pixel6 CPU, TFLite 2.1		Xiaomi11 CPU, Mindspore		Pixel5 GPU, NCNN		Xiaomi11 GPU, TFLite 2.7	
	Top1 Acc. ↑	Latency ↓	Top1 Acc. ↑	Latency ↓	Top1 Acc. ↑	Latency ↓	Top1 Acc. ↑	Latency ↓
MobileNetV2	72.0	45.9ms	72.0	22.7ms	72.0	31.3ms	72.0	5.0 ms
OFA [8] + LitePred	**76.4**	**44.4 ms**	**73.4**	**21.8 ms**	**75.3**	**31.1 ms**	**75.8**	**5.0 ms**
MobileNetV3× 0.75	73.3	29.7ms	73.3	24.4ms	73.3	34.4ms	73.3	4.0 ms
OFA [8] + LitePred	**74.8**	**29.6 ms**	**74.5**	**23.9 ms**	**76.0**	**34.3 ms**	**74.4**	**3.9 ms**
MobileNetV3	75.2	37.2ms	75.2	33.4ms	75.2	30.3ms	75.2	4.7ms
OFA [8] + LitePred	**75.5**	**36.8 ms**	**75.6**	**33.2 ms**	**75.6**	**29.8 ms**	**75.5**	**4.6 ms**

Table 7: Hardware-aware NAS search results on ImageNet 2012 dataset [15]. By integrating LitePred into OFA, we achieve superior accuracy compared to MobileNets across various edge platforms.

Platform	Similar Platform	Time Cost	Prediction Acc ±5%	Prediction Acc ±10%
Xiaomi11 CPU TFLite 2.7	Xiaomi11 CPU TFLite 2.1	0.05h	100%	100%
Xiaomi12 CPU TFLite 2.1	Pixel5 CPU TFLite 2.7, LowFreq	0.08h	83.9%	99.9%
Xiaomi12 CPU TFLite 2.7, INT8	Pixel5 CPU TFLite 2.7, LowFreq	0.02h	41.4%	99.9%

Table 8: Transferable latency prediction of LitePred on vision transformer NAS search space [10].

8 Related Works

Platform-specific latency prediction. Most previous latency prediction approaches [6, 7, 18, 31, 32, 54] are designed to build platform-specific latency predictors. Notable examples include BRP-NAS [18] and nn-Meter [54]. BRP-NAS [18] uses graph convolutional networks (GCN) to train a GCN latency predictor. However, it requires the entire model graph as input, which makes it necessary to redesign and retrain the model for new hardware-aware NAS search spaces, leading to expensive and time-consuming processes. nn-Meter [54] tackles this challenge by building kernel-level predictors. However, all these approaches require significant time costs and efforts to rebuild the predictors for new platforms, as they rely on platform-specific data collection and predictor training.

Platform-adaptive latency prediction. Recently, a few works propose to construct platform-agnostic latency predictors [30, 34, 37]. HELP [30] builds a meta latency predictor that incorporates hardware embeddings. However, meta-training requires numerous latency measurements on a wide variety of heterogeneous platforms and faces challenges in generalizing to new unseen devices. OneProxy [34] exploits latency monotonicity across diverse devices to predict DNN latency rankings on new unseen platforms. However, many hardware-aware NAS approaches [8, 9, 45, 56, 57] require actual latency values rather than rankings.

Hardware-aware NAS. Hardware-aware NAS approaches [8, 9, 49, 50, 53, 55, 57] aim to design efficient DNN models that balance accuracy and latency. However, the vast model search space makes latency measurement costly. Most NAS works [21, 49, 50] use FLOPs as an efficiency metric, but it's an inaccurate proxy for latency. Recent works such as ProxylessNAS [9] and OFA [8] employ a layer-wise latency predictor, but ignore latency changes caused by graph optimizations. Also, most of the evaluated platforms are limited to cloud platforms. LitePred provides rapid and accurate latency predictions on diverse edge platforms, facilitating the design of more efficient DNN models for edge environments.

9 Discussion

Comparison with Cost Models in DNN Compilers. Many deep learning compilers [4, 5, 11, 12, 58] build cost models to predict the execution time of different code implementations on a given hardware platform. They typically rely on complex feature engineering to build decision-tree-based regression models. For instance, TVM [11] employs XGBoost to make predictions based on a diverse set of features, including memory access and data reuse ratio, along with embedded features like AST. However, as many edge inference frameworks are closed source, these code-based methods are infeasible. LitePred differentiates itself from these approaches by predicting model latency solely based on the model configurations.

Generalization Ability. Currently, LitePred focuses on predicting inference latency for CNNs and vision transformers on commercial edge platforms. If a new edge platform is significantly different from our knowledge pool, it may be necessary to sample more adaptation data and train like starting from scratch. Our approach can be easily extended to other model types, such as language transformers. Generalization to cloud platforms has not been validated due to potential concurrency in kernel execution. We leave this as future work.

10 Conclusion

In this work, we propose LitePred, a lightweight latency prediction approach that can accurately predict the inference latency of DNN models on a new edge platform based on a small amount of extra measurements. LitePred incorporates a VAE data sampler to collect high-quality training and adaptation data. By transferring the most similar pre-existing latency predictors, LitePred achieves accurate predictions with an adaptation cost of less than 1 hour. Extensive experiments on 85 edge platforms and 6 NAS search spaces demonstrate the effectiveness of LitePred, achieving an impressive prediction accuracy of 99.3% and a remarkable 50.6× reduction in profiling cost compared with state-of-the-art baselines.

11 Acknowledgement

We are thankful to the anonymous NSDI reviewers and our shepherd, Hong Xu, for their constructive feedback. The work of Chengquan Feng, Yuanchi Liu, Zhiyuan Wang and Haisheng Tan are supported by the National Science Foundation of China under Grant No. 62132009.

References

[1] Pytorch kl-divergence loss. https://pytorch.org/docs/stable/generated/torch.nn.KLDivLoss.html?highlight=kl+divergence.

[2] Pytorch mseloss. https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html.

[3] ARM. An introduction to the armv8 instruction sets. https://developer.arm.com/documentation/den0024/a/An-Introduction-to-the-ARMv8-Instruction-Sets, 2019.

[4] Riyadh Baghdadi, Massinissa Merouani, Mohamed-Hicham Leghettas, Kamel Abdous, Taha Arbaoui, Karima Benatchba, et al. A deep learning based cost model for automatic code optimization. *Proceedings of Machine Learning and Systems*, 3:181–193, 2021.

[5] Riyadh Baghdadi, Jessica Ray, Malek Ben Romdhane, Emanuele Del Sozzo, Abdurrahman Akkas, Yunming Zhang, Patricia Suriana, Shoaib Kamil, and Saman Amarasinghe. Tiramisu: A polyhedral compiler for expressing fast and portable code. In *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization*, 2019.

[6] Noureddine Bouhali, Hamza Ouarnoughi, Smail Niar, and Abdessamad Ait El Cadi. Execution time modeling for cnn inference on embedded gpus. Association for Computing Machinery, 2021.

[7] Halima Bouzidi, Hamza Ouarnoughi, Smail Niar, and Abdessamad Ait El Cadi. Performance prediction for convolutional neural networks on edge gpus. In *Proceedings of the 18th ACM International Conference on Computing Frontiers*, CF '21, page 54–62. Association for Computing Machinery, 2021.

[8] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *ICLR*, 2020.

[9] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[10] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12270–12280, 2021.

[11] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Tvm: An automated end-to-end optimizing compiler for deep learning. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, 2018.

[12] Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Learning to optimize tensor programs. *Advances in Neural Information Processing Systems*, 31, 2018.

[13] THOMAS M. COVER and JOY A. THOMAS. *Elements of Information Theory*. 1991.

[14] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, and Joseph E. Gonzalez. Fbnetv3: Joint architecture-recipe search using neural acquisition function. *CoRR*, abs/2006.02049, 2020.

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society, 2009.

[16] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[17] DroidChart. You can currently find 8318 smartphones from 238 brands, 2023.

[18] Lukasz Dudziak, Thomas Chau, Mohamed Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas Lane. Brp-nas: Prediction-based nas using gcns. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems (Neurips)*, volume 33, pages 10480–10490. Curran Associates, Inc., 2020.

[19] Nathaniel R Goodman. Statistical analysis based on a certain multivariate complex gaussian distribution (an introduction). *The Annals of mathematical statistics*, 34(1):152–177, 1963.

[20] Google. Tensorflow lite. https://tensorflow.google.cn/lite/. Accessed 2022-12-14.

[21] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling.

In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*, pages 544–560. Springer, 2020.

[22] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. In *ICCV*, 2019.

[23] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[24] Huawei. Mindspore. https://www.mindspore.cn/lite/. Accessed 2022-12-14.

[25] Intel. Deploy high-performance deep learning inference, openvino. https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit.html, 2019.

[26] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

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

[28] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4013–4021, 2016.

[29] Hayeon Lee, Eunyoung Hyung, and Sung Ju Hwang. Rapid neural architecture search by learning to generate graphs from datasets. *ArXiv*, abs/2107.00860, 2021.

[30] Hayeon Lee, Sewoong Lee, Song Chong, and Sung Ju Hwang. Hardware-adaptive efficient latency prediction for nas via meta-learning. *Advances in Neural Information Processing Systems*, 34:27016–27028, 2021.

[31] Zhuojin Li, Marco Paolieri, and Leana Golubchik. Predicting inference latency of neural architectures on mobile devices. Association for Computing Machinery, 2023.

[32] Liang Liu, Mingzhu Shen, Ruihao Gong, Fengwei Yu, and Hailong Yang. Nnlqp: A multi-platform neural network latency query and prediction system with an evolving database. 2022.

[33] Bingqian Lu, Jianyi Yang, Weiwen Jiang, Yiyu Shi, and Shaolei Ren. One proxy device is enough for hardware-aware neural architecture search. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(3):1–34, 2021.

[34] Bingqian Lu, Jianyi Yang, Weiwen Jiang, Yiyu Shi, and Shaolei Ren. One proxy device is enough for hardware-aware neural architecture search. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(3):1–34, 2021.

[35] Microsoft. onnxruntime, 2022.

[36] Bert Moons, Parham Noorzad, Andrii Skliar, Giovanni Mariani, Dushyant Mehta, Chris Lott, and Tijmen Blankevoort. Distilling optimal neural networks: Rapid search in diverse spaces. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12229–12238, 2021.

[37] Saeejith Nair, Saad Abbasi, Alexander Wong, and Mohammad Javad Shafiee. Maple-edge: A runtime latency predictor for edge devices. In *CVPR EVW workshop*, pages 3660–3668, 2022.

[38] Microsoft Research nn Meter Team. nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices, 2021.

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

[40] Jialun Peng, Dong Liu, Songcen Xu, and Houqiang Li. Generating diverse structure for image inpainting with hierarchical vq-vae. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10775–10784, 2021.

[41] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019.

[42] 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 (CVPR)*, 2018.

[43] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. A hybrid convolutional variational autoencoder for text generation. *arXiv preprint arXiv:1702.02390*, 2017.

[44] A.K Subramanian. Pytorch-vae. https://github.com/AntixK/PyTorch-VAE, 2020.

[45] Chen Tang, Li Lyna Zhang, Huiqiang Jiang, Jiahang Xu, Ting Cao, Quanlu Zhang, Yuqing Yang, Zhi Wang, and Mao Yang. Elasticvit: Conflict-aware supernet training for deploying fast vision transformer on diverse mobile devices. *arXiv preprint arXiv:2303.09730*, 2023.

[46] Tensorflow. Post-training integer quantization. https://www.tensorflow.org/lite/performance/post_training_integer_quant, 2023.

[47] Sebastian Thrun and Lorien Y. Pratt. Learning to learn: Introduction and overview. In Sebastian Thrun and Lorien Y. Pratt, editors, *Learning to Learn*, pages 3–17. Springer, 1998.

[48] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers and distillation through attention. In *International Conference on Machine Learning*, volume 139, pages 10347–10357, July 2021.

[49] Dilin Wang, Chengyue Gong, Meng Li, Qiang Liu, and Vikas Chandra. Alphanet: Improved training of supernet with alpha-divergence. In *ICML*, 2021.

[50] Dilin Wang, Meng Li, Chengyue Gong, and Vikas Chandra. Attentivenas: Improving neural architecture search via attentive sampling. In *Conference on Computer Vision and Pattern Recognition*, 2021.

[51] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. HAT: hardware-aware transformers for efficient natural language processing. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7675–7688. Association for Computational Linguistics, 2020.

[52] Xudong Wang, Li Lyna Zhang, Yang Wang, and Mao Yang. Towards efficient vision transformer inference: A first study of transformers on mobile devices. In *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications*, 2022.

[53] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *ECCV*, 2020.

[54] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, page 81–93, New York, NY, USA, 2021. ACM.

[55] Li Lyna Zhang, Youkow Homma, Yujing Wang, Min Wu, Mao Yang, Ruofei Zhang, Ting Cao, and Wei Shen. Swiftpruner: Reinforced evolutionary pruning for efficient ad relevance. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 3654–3663, 2022.

[56] Li Lyna Zhang, Xudong Wang, Jiahang Xu, Quanlu Zhang, Yujing Wang, Yuqing Yang, Ningxin Zheng, Ting Cao, and Mao Yang. Spaceevo: Hardware-friendly search space design for efficient int8 inference. *arXiv preprint arXiv:2303.08308*, 2023.

[57] Li Lyna Zhang, Yuqing Yang, Yuhang Jiang, Wenwu Zhu, and Yunxin Liu. Fast hardware-aware neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[58] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, et al. Ansor: Generating {High-Performance} tensor programs for deep learning. In *14th USENIX symposium on operating systems design and implementation (OSDI 20)*, pages 863–879, 2020.

[59] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

[60] Nihui Zuo Zhang. Tencent: Ncnn: a high-performance neural network inference computing framework optimized for mobile platforms. https://github.com/Tencent/ncnn/, 2019.