

ProfEdge: Efficient Construction of DNN Performance Evaluation Model on Edge Devices

WEILONG WANG, Southeast University, Nanjing, China
SONGTAO LU, Southeast University, Nanjing, China
JIAWEI LIU, Southeast University, Nanjing, China
BORUI LI*, Southeast University, Nanjing, China
ZHAODONG XU, Southeast University, Nanjing, China
SHUAI WANG, Southeast University, Nanjing, China

With the rapid development of the Internet of Things (IoT) and artificial intelligence (AI) technologies, edge computing emerges as a crucial computing paradigm. By processing data near the source, edge computing enables faster and more efficient intelligent services. However, edge devices usually limit computational resources, and existing DNN inference latency profiling methods often rely on internal model details or large-scale latency measurements, making them costly and unsuitable for black-box deployment scenarios. This paper proposes ProfEdge, a fast construction framework for deep neural network (DNN) latency profiling models based on Gaussian process regression and Bayesian optimization. ProfEdge adaptively samples real latency measurements under different batch-size states and tunes profiling-model hyperparameters to reduce construction cost while improving profiling accuracy. Specifically, ProfEdge builds an adaptive sampling module based on Gaussian process regression to locate high-error regions through coarse-grained sampling and dynamically refine the sampling process. It further designs a dynamic Bayesian optimization mechanism to improve the accuracy of the latency profiling model. Finally, ProfEdge constructs a cross-device performance mapping model to migrate an existing profiling model to a target device with lightweight stratified calibration, thereby avoiding full reconstruction of the target-device profiling model. Experiments on various edge devices and DNN models, including CNN-based and transformer-based workloads, show that ProfEdge reduces profiling errors by up to 80% and saves over 70% of profiling construction cost compared with existing methods. The cross-device migration results further demonstrate that ProfEdge can achieve competitive profiling accuracy on new devices with only a small number of target-device calibration samples.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; • **General and reference** → **Performance**.

Additional Key Words and Phrases: edge computing, inference latency profiling, edge AI, deep neural network.

1 Introduction

With the rapid development of the Internet of Things (IoT) and artificial intelligence (AI) technologies, deep neural network (DNN) models are widely applied in today's mobile and edge applications [6, 7, 19, 22, 29, 47, 48, 54], such as on-device video analytics, facial recognition, AR/VR, and others [4, 11, 20, 43, 46, 49, 53]. In many latency-sensitive applications, inference latency becomes a key indicator for running DNN models on various mobile and edge devices [17, 25, 30, 33, 35, 36, 44]. Accurately profiling the inference latency of neural network models is crucial for resource scheduling and management [10, 16, 24, 26, 31, 34, 45]. However, when deploying

Authors' Contact Information: Weilong Wang, Southeast University, Nanjing, China; e-mail: wang_wl@seu.edu.cn; Songtao Lu, Southeast University, Nanjing, Jiangsu, China; e-mail: seu_lst@seu.edu.cn; Jiawei Liu, Southeast University, Nanjing, Jiangsu, China; e-mail: liujoy@seu.edu.cn; Borui Li, Southeast University, Nanjing, Jiangsu, China; e-mail: libr@seu.edu.cn; Zhaodong Xu, Southeast University, Nanjing, Jiangsu, China; e-mail: xuzhdgyq@seu.edu.cn; Shuai Wang, Southeast University, Nanjing, Jiangsu, China; e-mail: shuaiwang@seu.edu.cn.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 1557-6051/2026/9-ART

<https://doi.org/10.1145/3847112>

DNN models in resource-constrained edge device environments, a core challenge lies in efficiently managing computational resources to meet the real-time requirements of applications. The inference latency of models under different batch sizes does not change linearly, as it is closely related to the computational characteristics of the underlying hardware. Therefore, constructing an accurate latency prediction model is crucial, as it can quantify the inference latency required for different batch sizes and provide key performance insights for resource schedulers.

However, the predictive ability of this latency prediction model is often limited by the sampling size. The larger the scale, the more complete the model training and the more accurate the evaluation. Exhaustively measuring the real inference latency of DNN models under all possible batch-size states is highly labor-intensive and expensive [10, 23, 26, 28, 31, 32, 34, 52]. Sampling refers to selecting a limited number of runtime states and measuring the real inference latency of a DNN model under these states. Specifically, each runtime state is mainly associated with a batch size, and each sampled point corresponds to a batch-size-latency pair. These sampled latency values are then used as training data to construct the latency profiling model. Such high costs hinder scalability and make measurement-based methods practically incapable of supporting the rapidly growing number of edge devices.

To address the aforementioned challenges, existing research primarily focuses on two key dimensions: one emphasizes how many latency measurements should be sampled, while the other prioritizes what should be sampled, such as model-level states or operator-level components. In terms of sampling scale, i.e., the proportion of measured batch-size states, most conventional approaches adopt large-scale data sampling, aiming to maximize the accuracy of latency profiling models. For instance, methods based on floating-point operations (FLOPs) are widely utilized to evaluate model efficiency [17, 43]. While these methods offer simplicity, they do not directly quantify latency, an inherent limitation, as FLOPs merely reflect computational complexity rather than actual runtime performance on specific hardware.

In terms of sampling targets, prevalent existing solutions predominantly focus on modeling and evaluating the individual operators of neural networks [12, 17, 51, 52]. This operator-centric modeling confers strong generalizability, as the end-to-end latency of a model is derived by analyzing the latency of its constituent operators and summing their contributions. However, these methods suffer from two critical drawbacks: (1) they heavily rely on the comprehensiveness of operator coverage in the latency profiling model; (2) their accuracy in cross-device or model migration scenarios remains insufficient and requires further refinement. Different from these architecture-aware or operator-level predictors, ProfEdge does not require detailed operator decomposition or architecture-specific feature extraction. It treats the deployed DNN as a black-box model and focuses on constructing an accurate latency profiling model with significantly reduced sampling cost for each target model-device pair. This design prioritizes black-box compatibility, low-cost latency measurement, and profiling accuracy for a given model-device pair.

Fortunately, when the device environment remains unchanged, the relationship between the model state (i.e., the batch size) and latency mostly follows a consistent pattern. However, the parallel computing capacity of different devices has an upper limit; thus, beyond a certain threshold, this pattern is disrupted or even redefined. As illustrated in Figure 1, we uniformly sample a subset of real latency data to construct a latency prediction model based on the random forest algorithm. By calculating the relative error between the predicted results and the actual latency values across all states, it is observed that the errors follow a Gaussian distribution, with small errors dominating and large errors occurring infrequently. Therefore, the states with small errors allow for reduced sampling regions, while those with large errors require expanded sampling regions. This approach maintains evaluation accuracy while reducing sampling costs. Furthermore, the sampling scale and the performance of latency profiling models are not perfectly proportional. This is attributed to the law of diminishing marginal returns governing the improvement of latency profiling model performance: as the sampling scale expands beyond a certain threshold, the incremental gain in model accuracy becomes increasingly modest.

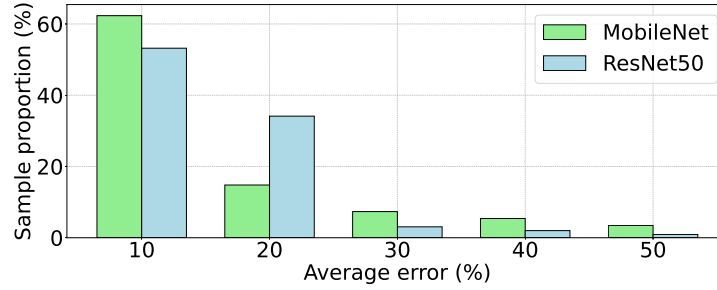


Fig. 1. The cost of model sampling and the error of model profiling results in Jetson Xavier NX.

Consequently, by tolerating negligible prediction errors, the computational and resource costs associated with model construction can be significantly reduced. On the other hand, most contemporary tasks involve migrating DNN models from the cloud or other edge devices to a target device. As such, the architectural details and latency characteristics of the DNN models are not entirely unknown a priori. Leveraging the latency evaluation results of the model on the source device, combined with an understanding of the configuration and performance discrepancies between the source and target devices, it is feasible to infer the model's latency on the target device. This transfer learning-inspired approach effectively mitigates the overhead of constructing a latency profiling model from scratch on the target device.

The above opportunities provide new methods for constructing latency evaluation models for DNN models, but also bring new challenges.

(1) How to quickly identify sampling states with high errors and select the optimal sampling region for them? For unknown models, it is impossible to determine which state patterns differ from others before sampling. Excessive or overly large sampling compromises the accuracy of latency evaluation, while insufficient sampling increases sampling costs.

(2) How can latency mapping relationships between different edge devices be accurately constructed? Different devices have varying configurations, and these configurations influence the latency of the models. Inaccurate mapping relationships affect the evaluation results of DNN model latency, necessitating the reconstruction of evaluation strategies and significantly impacting reconstruction costs.

To address the aforementioned issues, we propose an adaptive latency profiling model construction method based on Bayesian optimization and Gaussian process regression, named the ProfEdge. We employ a coarse-grained sampling strategy to identify regions with high errors. Here, high-error regions refer to groups of batch-size states where the current latency profiling model produces relatively large prediction errors compared with the real measured latency. These regions usually indicate that the current sampling data are insufficient to capture the local latency variation, and therefore require additional fine-grained sampling. Based on the Gaussian process regression theory, we adaptively partition these high-error regions and perform sampling. This approach ensures accuracy while reducing sampling costs. We design a model parameter optimization mechanism based on Bayesian optimization to further enhance the accuracy of the latency profiling model. Finally, by analyzing the affinity between different model architectures and device configurations, as well as the performance differences of the same model across devices with varying configurations, we construct a device migration-oriented model latency mapping method. This method improves the profiling accuracy of model latency without reconstructing a complete latency profiling model on the target device. Instead, it reuses the source-device profiling result and performs lightweight calibration using a small number of stratified target-device measurements.

We summarize our main contributions as follows:

- To the best of our knowledge, we are the first to propose a black-box, model-level adaptive latency profiling framework for device migration, ProfEdge. By leveraging adaptive sampling strategies and a device migration mapping mechanism, our approach ensures the accuracy of model latency evaluation while reducing sampling costs. Unlike architecture-aware or operator-level predictors, ProfEdge focuses on low-cost profiling for each deployed model-device pair without requiring operator decomposition or architecture-specific feature extraction.
- We propose an adaptive sampling optimization scheme based on the Gaussian process regression theory and Bayesian optimization. Gaussian process regression theory is used to adaptively locate high-error sampling regions, ensuring evaluation accuracy while minimizing sampling costs. Bayesian optimization is employed to adaptively optimize latency profiling model parameters, enhancing the accuracy of DNN model evaluation.
- We conduct experiments and evaluate ProfEdge using various DNN and transformer models on edge devices. The results demonstrate that, compared to existing methods, ProfEdge achieves significantly better profiling accuracy and training costs across different devices and DNN models.

The rest of this paper is organized as follows. Section 2 demonstrates background technology and the performance latency differences of DNN models under different hardware configurations. Section 3 presents the principles throughout our design and the system overview of ProfEdge. Section 4 presents the detailed design of ProfEdge. Section 5 finally presents the evaluation results. Section 6 summarizes the related works of ProfEdge. Finally, Section 7 concludes the paper.

2 Background and Motivation

In this section, we introduce the background knowledge relevant to this work and present the motivation behind our research.

2.1 Background

Edge computing. Edge computing [11, 22, 36, 39, 40] is a computational paradigm that processes and analyzes data at the edge of the network, close to the data source or end users. It brings computing, storage, and network resources closer to where data is generated, such as edge servers and gateway devices. This approach reduces the need to transmit data to the cloud or central data centers, thereby lowering latency, saving bandwidth, and enabling continuous data processing even in unstable or offline network conditions. Widely applied in industrial IoT, smart transportation, and other fields, edge computing provides efficient solutions for scenarios requiring high real-time performance.

DNN Model. Deep Neural Network (DNN) [8, 9] is a type of neural network structure in artificial intelligence characterized by multiple hidden layers. By simulating the connection patterns of neurons in the human brain, DNN performs layer-by-layer feature extraction and transformation on input data to learn complex patterns and relationships within the data. Each layer of neurons in a DNN processes the input data through weighted summation and activation functions, passing the results to the next layer. Through training on large amounts of labeled data, DNN continuously adjusts weights and biases to improve its ability to recognize data features. Since the landmark work of AlexNet, DNN experts spend many years developing numerous DNN models, including VGG, ResNet, MobileNet, and others.

Edge AI. Edge intelligence (Edge AI) [5, 18, 50] is the integration of edge computing and artificial intelligence technologies, where AI models and algorithms are deployed on edge devices or edge servers, enabling these devices to perform local data processing and intelligent decision-making. By running deep learning and other models on the edge, edge intelligence rapidly analyzes and infers real-time data without the need to transmit it to the cloud, significantly reducing latency and improving system responsiveness. Additionally, it enhances

Table 1. Specification comparison of edge computing devices.

Device ID	Name	CPU Cores	Memory	Dimensions
Device 1 (j1)	Jetson Xavier NX	6-core	8 GB	103mm × 90.5mm
Device 2 (j2)	Jetson AGX Xavier	8-core	32 GB	100mm × 87mm
Device 3 (s1)	Raspberry Pi 4B	4-core	4 GB	85mm × 56mm
Device 4 (p1)	i7-12700H + RTX 3060	14-core	16 GB	x86 edge device

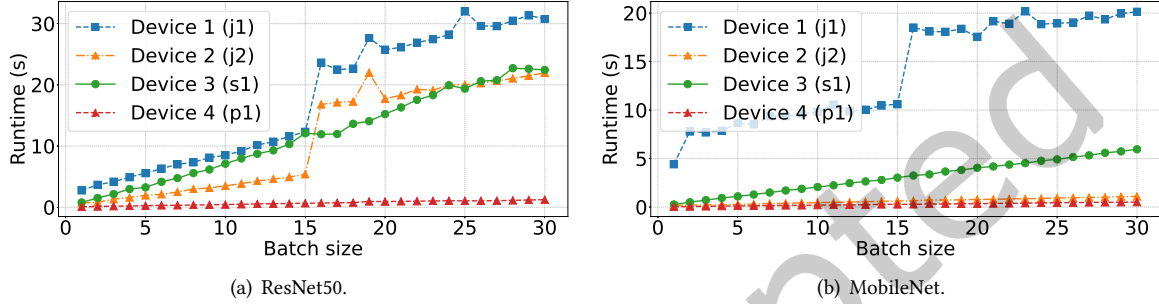


Fig. 2. The execution time of different models deployed on different devices.

data privacy and security, as sensitive data is processed locally, minimizing the risk of theft or leakage during transmission. Edge intelligence is widely applied in smart homes, intelligent security, autonomous driving, and other fields, providing efficient and reliable technical support for scenarios requiring real-time intelligent processing.

2.2 Motivation

Considering the variations in execution latency of different models across different devices, we deploy two typical DNN models on four devices with varying configurations. We set Device 1 as Jetson Xavier NX, Device 2 as Jetson AGX Xavier, Device 3 as Raspberry Pi 4B, and Device 4 as an x86 edge device equipped with an i7-12700H CPU and RTX 3060 GPU. These devices cover diverse architectures and computing capabilities, thereby better reflecting device heterogeneity in real edge deployment scenarios. The specific device information is shown in Table 1. The default DNN model object for testing is ResNet50. The default testing device is Device 1.

Given that different DNN models (e.g., ResNet50, MobileNet, DenseNet) often require batch processing to accelerate the inference of large volumes of data, we compare the performance of these models across different devices under various batch sizes.

As shown in Figure 2, when the same model is deployed on different devices, the execution latency varies significantly due to differences in computational capabilities and efficiency across devices. Moreover, the execution latency of the model does not increase linearly with the batch size. This is because different devices employ distinct memory management strategies, and memory is often a critical constraint limiting model inference latency. Additionally, the figure reveals that the performance advantages of different models vary across devices. For example, for the complex ResNet50 model, when the batch size is small, device 2 outperforms device 3; while when the batch size is large, device 3 outperforms device 2. And the model is not linearly related to latency and batch size on multiple devices. For the simple model MobileNet, devices 2 and 4 have the best performance, and on multiple devices, latency is linearly correlated with batch size. This discrepancy arises from the distinct

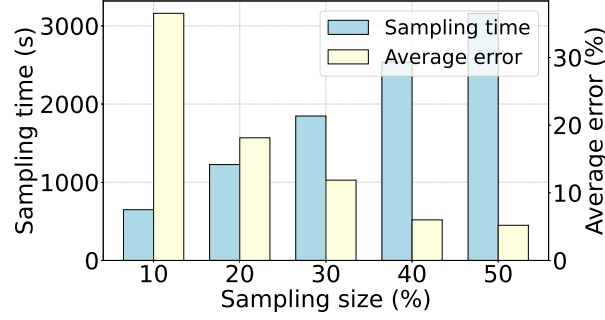


Fig. 3. Impact of sampling scale on latency profiling. The sampling scale denotes the sampling percentage, i.e., the proportion of candidate batch-size states selected for real latency measurement and used to train the latency profiling model.

architectural designs of the models, and the varying computational affinity of different device configurations towards these architectural paradigms.

Considering that most existing evaluation schemes employ random forest (RF) models to construct latency profiling models, we initially use the same random forest model while utilizing sampling data of different scales to build latency profiling models. Sampling refers to measuring the real inference latency of a DNN model under selected batch-size states, rather than sampling a continuous signal over time. For example, when the batch-size range is from 1 to 50, there are 50 theoretical latency samples in total. The sampling ratio denotes the proportion of actually measured samples among these candidate batch-size states. We then compare their construction time and inference accuracy. The latency profiling model construction time includes both latency measurement time and model training time.

As shown in Figure 3, the larger the sampling scale, the higher the evaluation accuracy of the model, but the longer the model construction time. Here, the sampling scale denotes the sampling percentage, i.e., the proportion of candidate batch-size states selected for real latency measurement and used to train the latency profiling model. The relationship between model construction time and accuracy is not strictly inversely proportional. There exists an “optimal scale” that ensures the model achieves relatively high evaluation accuracy while keeping the construction time relatively low. In other words, the optimal scale corresponds to an appropriate sampling percentage rather than a fixed absolute number of samples. Moreover, the “optimal scale” varies for different models deployed on different devices. This is partly due to differences in computational capabilities across devices and partly because of the varying computational affinity between different model architectures and devices.

In summary, our goal is to adaptively identify the “optimal scale” for model sampling, i.e., the appropriate sampling percentage over candidate batch-size states, for various models deployed on different devices. By constructing latency profiling models using sampling data at these “optimal scales”, we aim to achieve accurate profiling of DNN model latency while effectively reducing the latency profiling model construction time.

3 Overview

In this section, we delineate the overall architecture of ProfEdge. As shown in Figure 4, ProfEdge is primarily divided into two phases: construction and migration.

Construction: The latency profiling model involves training sample collection and model construction on Device 1. When a new DNN model is introduced, the system requires sampling the execution latency of the model under different states (e.g., varying batch sizes) to build an accurate latency profiling model. Considering the impact of low-scale sampling on model accuracy and the cost implications of high-scale sampling, we dynamically

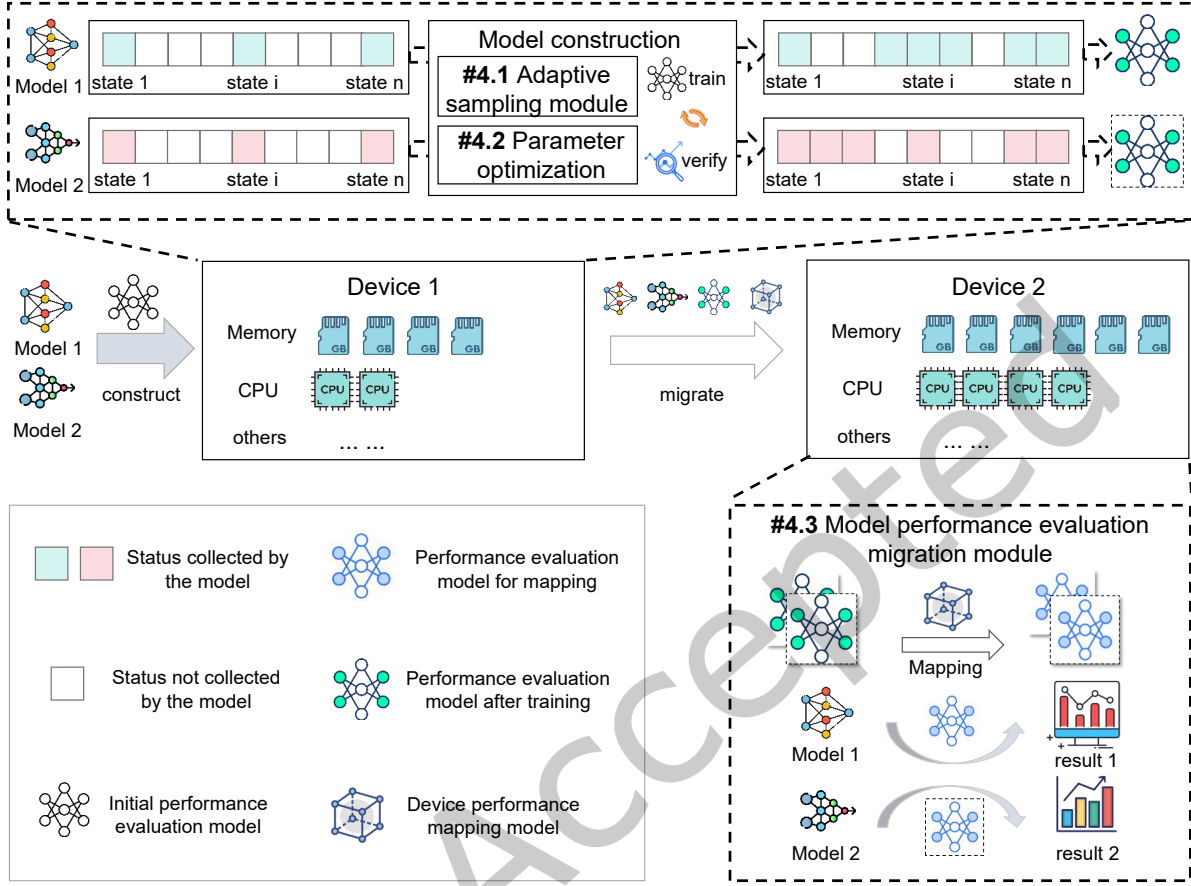


Fig. 4. The overview of ProfEdge

determine the sampling scale using a proposed adaptive sampling optimization scheme based on Gaussian process regression theory and Bayesian optimization. Initially, the stability of the model's performance is assessed through low-scale sampling. Subsequently, high-error states are adaptively identified using the Gaussian process regression theory, and the scale for local fine-grained sampling is determined (cf. Section 4.1). Furthermore, since the parameters of the latency profiling model itself influence its profiling accuracy, which in turn affects the selection of sampling scale and states, we dynamically optimize the parameters of the latency profiling model based on Bayesian optimization theory and the stability of model evaluation performance. This optimization enhances the accuracy of the latency profiling model and reduces sampling costs (cf. Section 4.2).

Migration: The performance latency profiling model is accurately migrated from Device 1 to Device 2. To further reduce the sampling cost of the latency profiling model, we design a device migration mapping mechanism. Considering that the computational logic of the model remains unchanged when its architecture is unchanged, while the computational performance of the model is closely coupled with the configuration of the device itself, we leverage the relationship between performance and device configuration, as well as the configuration mapping relationship between different devices, to construct a device mapping model (cf. Section 4.3). The latency profiling

model on Device 1 is directly migrated to Device 2, and its profiling results is adjusted using the inter-device mapping model, eliminating the need for sampling data to build a new latency profiling model.

In summary, ProfEdge reduces the construction cost of the latency profiling model during both the initial latency profiling model construction and the latency profiling model migration phases, while ensuring the accuracy of the latency profiling model.

4 Design

As discussed in the motivation section, one of the key reasons for the low sampling cost of ProfEdge is the implementation of the adaptive sampling scale size. This section describes how ProfEdge is designed to identify the optimal sampling regions and scales, and adaptively determine the optimal training parameters for the latency profiling model. Additionally, it explains how the relationship between configuration parameters and performance across heterogeneous devices is leveraged to avoid the reconstruction of the latency profiling model through transferable latency profiling models, thereby reducing sampling costs.

4.1 Adaptive sampling module based on Gaussian process regression

The construction of a latency profiling model involves two steps: (1) collecting training samples; (2) fitting the model's parameters using these samples. The final model accurately assesses the computational latency of a DNN model. There is a trade-off between the number of training samples and the latency profiling model's performance. More samples enhance the model's capabilities but increase data collection and training costs. It should be noted that the assessment capability of the model is not inversely proportional to the number of training samples. Moreover, different latency profiling models may have varying assessment errors for DNN models in different states. Thus, our aim is to achieve the highest sampling efficiency with the lowest possible cost while maintaining model evaluation accuracy. We formulate the problem as follows.

Problem. Assume there is a DNN model M_D to be evaluated, which has multiple states $B = \{b_1, b_2, \dots, b_n\}$. In this paper, a state refers to the runtime input state under which inference latency is measured, rather than a layer-level compute state or the execution progress at a timestamp. In our current implementation, each state is mainly defined by the batch size. Thus, b_i denotes a batch-size state, and each sampled training point is represented as (b_i, t_i) , where t_i is the real measured inference latency under batch size b_i . Accordingly, for the per-model latency profiling model M_P , the input feature is the batch-size state b_i , and the prediction target is the measured latency t_i . The runtime of the DNN model varies across these states, denoted as $T = \{t_1, t_2, \dots, t_n\}$, with a general trend that a larger batch size state b_i corresponds to a larger t_i . We need to find a $B^* \in B$ such that the latency profiling model M_P , trained on B^* , accurately predicts the inference latency t_i of M_D under each state b_i . Specifically, the predicted t_i should approach the true value t_i as closely as possible. Meanwhile, the sampling cost $T^* \in T$ needs to be minimized. Our problem objective is as follows:

$$\arg \min \sum_{j=1}^m t_j \quad (1a)$$

$$s.t. \ p_i = \left| \frac{M_P(b_i) - t_i}{t_i} \right| \leq P \quad (1b)$$

$$t_i \in T, \ t_j \in T^*, \ b_i \in B, \ b_j \in B^* \quad (1c)$$

where t_j denotes the actual execution latency of M_D in sampling state b_j . m is the number of state elements in B^* , with $m \leq n$. $M_P(b_i)$ represents the inference result of the latency profiling model M_P for the execution latency of M_D in state b_i . P indicates the acceptable profiling error range for M_P , which is set to 0.1 by default.

ProfEdge employs a generic random forest model [3] as the underlying latency profiling model M_P . To address the aforementioned sampling cost issue, ProfEdge leverages a Gaussian Process Regression model [13] to guide

sampling during model training. As a non-parametric regression approach, it makes no assumptions about data distribution or function form. Instead, it models by learning relationships in data, making it ideal for linking batch-size states to inference accuracy. To identify batch-size states with the poorest inference performance, modeling the batch-size state set and corresponding profiling accuracy is crucial. Here, the feature used by the latency profiling model is the batch-size state b_i , and the measured label is the corresponding real latency t_i . For this, we use $1 - p_i$ as the inference accuracy, which inversely gauges inference results accuracy via error. In our implementation, GPR is used to predict the profiling discrepancy of the current latency profiling model over different batch-size states. Specifically, the discrepancy is defined as the relative error between the latency predicted by M_P and the real measured latency. The adaptive sampling decision is mainly guided by the GPR-predicted discrepancy rather than only by the predictive variance. This choice is aligned with our objective: ProfEdge aims to reduce the profiling error under a limited sampling budget, rather than merely exploring statistically uncertain regions. Uncertainty-based sampling may select regions with high predictive variance, but these regions do not necessarily correspond to large profiling errors. In contrast, predicted discrepancy directly reflects where the current latency profiling model performs poorly and is therefore more suitable for selecting additional sampling regions.

The construction process of the latency profiling model M_P is divided into the following six stages.

(1) Develop initial random forest model M_P and Gaussian process regression model G . The performance of M_P depends on the quantity and quality of training data, while G relies on a well-defined kernel function, which balances fitting and generalization. Here, the radial basis function (RBF) is chosen as the kernel function, expressed as:

$$K(b_i, b_j) = \theta_0 \exp \left(-\frac{1}{2} \sum_{d=1}^D \eta_d (b_{i,d} - b_{j,d})^2 \right) + \theta_1 \quad (2)$$

In our implementation, the state feature is mainly the batch size, so $D = 1$ and $b_{i,1}$ corresponds to the batch-size value. The formulation is written in a general form so that additional runtime state features can be incorporated if needed. θ_0 represents the signal variance, which controls the overall amplitude of the kernel function. It reflects the overall intensity of the impact of the input state b_i on the computation delay t_i . η_d denotes the length-scale parameter, which governs the rate of change on each feature dimension. It indicates the sensitivity of each feature dimension d of the input state b_i to changes in computation delay t_i . θ_1 stands for the noise variance, which accounts for the noise in the data. It reflects the random fluctuations or errors during the measurement of computation delay t_i .

(2) Coarse-grained sampling and training set construction. Constructing $B^* = \{b_g, b_{2g}, \dots, b_k\}$, $k = \lfloor \frac{n}{g} \rfloor * g$, $k \leq n$. g represents the initial granularity size, which is set to 10 by default.

(3) Model training. First, input the initial sampling data into model M_P for training to obtain a basic latency profiling model. Then, train G based on the evaluation results of M_P and the actual results in B^* , aiming to estimate the hyperparameters $\theta_0, \eta_d, \theta_1$ in the kernel function and other model parameters. Different from directly modeling the real latency values, G is trained to model the profiling discrepancy of M_P . For each sampled state $b_i \in B^*$, we calculate the discrepancy as:

$$e_i = \left| \frac{M_P(b_i) - t_i}{t_i} \right|. \quad (3)$$

Then, G learns the mapping from the sampled state b_i to the discrepancy e_i , so that it can estimate which unsampled regions are likely to produce large profiling errors. We fit the training parameters by maximizing the likelihood function, which is calculated as follows:

$$q(t \mid B, \theta) = \mathcal{N}(t \mid 0, K + \sigma^2 I) \quad (4)$$

In this equation, the response vector t can be instantiated as the discrepancy vector $E = \{e_i\}$ when GPR is used to guide adaptive sampling. B stands for the matrix of input states for all training samples, containing each sampling point's DNN model input state b_i . K is the covariance matrix derived from the kernel function, illustrating the correlation between different input states b_i and b_j . σ^2 represents the variance of observational noise, reflecting the random fluctuations or errors during the measurement of computation delay t_i . I is the identity matrix utilized to adjust the covariance matrix's structure to account for observational noise's impact.

(4) Model performance verification. We randomly select states from B^* as input states b_* and predict the mean and variance:

$$m(b_*) = k_*^T (K + \sigma^2 I)^{-1} t \quad (5)$$

$$\sigma^2(b_*) = K(b_i, b_*) - K_*^T (K + \sigma^2 I)^{-1} K_* \quad (6)$$

where K_* is the covariance vector between training data and new data calculated by the kernel function, reflecting the correlation between the input states b_i of the training samples and the new input state b_* . Here, the predictive mean $m(b_*)$ represents the estimated profiling discrepancy at state b_* . The predictive variance $\sigma^2(b_*)$ reflects the uncertainty of the GPR estimation. ProfEdge uses the predicted discrepancy $m(b_*)$ as the primary sampling criterion, because it directly indicates where the current profiling model is likely to have large error. The predictive variance is used only as auxiliary information when multiple regions have similar predicted discrepancy.

(5) Adaptive fine-grained training set construction. For data points with poor profiling, i.e., those with large GPR-predicted discrepancy, collect nearby fine-grained data adaptively. The adaptive sampling data has two parts: the front and back of the current data point. For example, if the current data point is the i th, the previous is the i'_- th (actual interval $|i - i'_-|$), and the next is the i'_+ th (actual interval $|i - i'_+|$). The new sampling intervals are $\left\lceil \frac{|i - i'_-|}{u} \right\rceil$ and $\left\lceil \frac{|i - i'_+|}{u} \right\rceil$, where u is the fine-grained degree, defaulting to 4. Add the newly collected data points to the training set B^* .

(6) Repeat steps (3)–(5) until the evaluation accuracy reaches the threshold $1 - P$, which is set to 90% by default.

By combining random forest for evaluation and Gaussian process regression for sampling guidance, this approach integrates coarse-grained sampling, model training, profiling evaluation, adaptive fine-grained sampling, and model updating. It aims to achieve high evaluation accuracy with minimal sampling data.

The core of the above steps is to perform fine-grained sampling around the new sampling center point b_i . Thus, we propose an adaptive sampling guidance algorithm based on the Gaussian process regression theory to identify new sampling center points. The algorithm focuses on finding the point with the worst performance in the current sampling set as the sampling center. More specifically, the algorithm selects the state with the largest GPR-predicted discrepancy as the new sampling center, instead of selecting the state with the largest predictive variance only.

As shown in Algorithm 1, given the current sampling set B^* and T^* , the full candidate state set B , and the current latency profiling model M_P , the output is the new sampling center b_i . Step 1: Assess and predict states in B^* using M_P to get evaluation set Y . Step 2: Calculate the profiling discrepancy E between the predicted latency Y and the real measured latency T^* on the sampled states. Steps 3 and 4: Initialize a Gaussian model G , and train it with B^* and E to learn the relationship between batch-size states and profiling discrepancy. Step 5: Predict the discrepancy $\hat{E}$ and the uncertainty Σ using G . Step 6: Return the point with the maximum predicted discrepancy as the new sampling point.

4.2 Parameter optimization based on dynamic Bayesian optimization

During the training of M_P , hyperparameters affect its performance. We use Bayesian optimization to find the optimal hyperparameter combination. It assesses the objective function to gauge model performance under given

Algorithm 1 Adaptive sampling guidance algorithm**Input:** Current sampling set B^* , T^* , candidate state set B of batch-size states, and latency profiling model M_P .**Output:** New sampling center point b_i .

- 1: $Y = M_P(B^*)$ //Assess and predict states in B^* using M_P
- 2: $E = \left| \frac{Y - T^*}{T^*} \right|$ //Calculate profiling discrepancy on sampled states
- 3: $G = \text{GaussianProcessRegressor}()$
- 4: $G.\text{train}(B^*, E)$ //Train GPR to predict profiling discrepancy
- 5: $\hat{E}, \Sigma = G.\text{predict}(B)$ //Predict discrepancy and uncertainty over candidate states
- 6: $b_i = \text{np.argmax}(\hat{E})$ //Select the point with the largest predicted discrepancy
- 7: **return** b_i

hyperparameters and guides the next hyperparameter choice, enhancing search efficiency in the hyperparameter space. Traditional Bayesian optimization iteratively seeks suitable hyperparameters on a fixed dataset. Extensive sampling on edge devices is time-consuming. We modify the Bayesian optimization method by combining coarse-grained initial sampling and fine-grained targeted sampling. Each sampling expands the dataset, retrains the model, and iteratively optimizes hyperparameters via Bayesian optimization. This controls sampling costs and boosts training efficiency and performance. The main components of the designed Bayesian optimization scheme of ProfEdge are as follows.

Surrogate Model. Bayesian optimization employs a probabilistic surrogate model to imitate the objective function, offering estimates and uncertainty quantifications for unobserved points. In our case, the surrogate model is built from sampling data on edge devices and updates as the dataset grows. The Gaussian process profiling distribution is given by, $f_* | \mathbf{x}_*, \mathcal{D} \sim \mathcal{N}(\mu(\mathbf{x}_*), \sigma^2(\mathbf{x}_*))$, where $\mu(\mathbf{x}_*)$ represents the mean, and $\sigma^2(\mathbf{x}_*)$ the variance. In edge-device scenarios, these parameters are initialized with coarse-grained initial sampling data and optimized during fine-grained sampling.

Acquisition Function. The acquisition function determines the next hyperparameters. It is formulated to measure the expected improvement of a new trial over the current best observation:

$$\text{EI}(\mathbf{x}) = \mathbb{E}[\max(f_b - f(\mathbf{x}), 0)] \quad (7)$$

where f_b is the current best observed value. The acquisition function calculates the potential value of each candidate hyperparameter combination based on the predicted objective function results and selects the point with the highest value for the next evaluation. In the early stages of optimization, the acquisition function prioritizes exploration, broadly testing different regions to avoid missing potential optimal solutions. In later stages, it gradually shifts to exploitation, fine-tuning the current optimal region to accelerate convergence.

Optimization. The next hyperparameters are selected by optimizing the acquisition function, which involves global optimization to find the most promising combination for model performance. In edge devices, the optimization considers sampling costs to prevent frequent sampling. We employ gradient-descent-based algorithms, leveraging the surrogate model's profilings and uncertainty information to efficiently search the hyperparameter space. The parameter update rule for the optimization algorithm is:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \eta \nabla \text{EI}(\mathbf{x}_t) \quad (8)$$

where $\mathbf{x}_t$ is the hyperparameters at the current iteration, η is the learning rate, and $\nabla \text{EI}(\mathbf{x}_t)$ is the gradient of the acquisition function at $\mathbf{x}_t$.

Evaluation. The chosen hyperparameter set is evaluated using the objective function, and the surrogate model is updated. We select the mean squared error (MSE) as the objective function: $\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$. y_i is the true value, $\hat{y}_i$ is the model-predicted value, and n is the sample size. In edge device scenarios, after each sampling, the model

performance is evaluated using MSE, and the surrogate model is updated to inform the next hyperparameter selection.

By using the above-mentioned methods, we optimize hyperparameter combinations while controlling sampling costs, thereby enhancing model training efficiency and performance.

4.3 Model performance evaluation migration module.

In this section, to further reduce the sampling cost of the latency profiling model, we propose a latency profiling model migration mechanism. Instead of reconstructing a complete latency profiling model on each target device, ProfEdge migrates an existing profiling model from a source device to the target device and only performs lightweight calibration with a small number of stratified target-device measurements. By migrating and adapting existing latency profiling models to target devices, the full construction of new latency profiling models is avoided, thereby reducing sampling costs.

We establish a mapping relationship between the configuration information of Device 1 and the target migration device to migrate and adapt the existing latency profiling model. For example, if the target migration device is Device 2, the performance mapping relationship between Device 1 and Device 2, along with the evaluation results of the test DNN model from the latency profiling model on Device 1, is used to quickly infer the performance of the DNN model on Device 2. This approach enables the reuse of the existing latency profiling model without full reconstruction on the target device. Different from complete target-device profiling, only a small number of stratified anchor measurements are collected on the target device to calibrate the migrated prediction.

Training assumption of the migration model. ProfEdge does not assume a large offline pre-trained mapping model over many device pairs. For each source-target migration task, only one existing source-device latency profiling model is required. This source-device profiling model is assumed to be already available from previous profiling on the source device. When migrating to a new target device, ProfEdge collects a small number of stratified target-device anchor measurements and uses them to construct and calibrate a lightweight source-target mapping model. Therefore, one directed source-target device pair is sufficient for performing the basic migration process. Additional historical device pairs can be used to further improve the mapping model, but they are not required by the basic design. More target-device anchor measurements generally improve the calibration accuracy, as validated by our experiments with different calibration ratios.

The random forest model possesses robust relational mapping capabilities, and therefore, we adopt the Random Forest model architecture as the underlying framework for the device performance mapping mechanism. Each decision tree recursively partitions the hardware-model-batch feature space. Since model performance is a continuous value, the relational mapping constitutes a fitting problem. Given the source-device profiling result $L_1(b)$ at batch state b , the device configuration features C_1 and C_2 , and the model architecture feature $\phi(M)$, the mapping model estimates the cross-device performance scaling factor. Our objective is to minimize the mapping error to achieve accurate performance mapping, which is calculated using the following formula:

$$S(R) = \arg \min_j \frac{1}{|R|} \left(\sum_{r \in \{R_1, R_2\}} \sum_{i \in r} \left(\frac{L_2^{(i)}}{L_1^{(i)}} - \frac{1}{|r|} \sum_{k \in r} \frac{L_2^{(k)}}{L_1^{(k)}} \right)^2 \right) \quad (9)$$

where j represents the index of the mapping features, including device configuration parameters, model architecture features, batch state, and the source-device profiling result. R_1 and R_2 denote the partitioned intervals of the configuration parameters. $S = \frac{L_2^{(i)}}{L_1^{(i)}}$ represents the performance scaling factor for the i -th training sample, which is the ratio of the DNN model inference latency between Device 1 and Device 2. This design ensures that the mapping model effectively captures the nonlinear impact of device configuration differences on performance scaling.

The evaluation results of the DNN model's performance on Device 2 are influenced by two factors. On one hand, they are influenced by the evaluation results from Device 1, and on the other hand, they are derived from the performance mapping relationship between Device 1 and Device 2. Therefore, ProfEdge first obtains an initial target-device latency prediction through RF-based performance scaling:

$$\hat{L}_2^0(b) = L_1(b) \cdot \frac{1}{T} \sum_{t=1}^T f_t(C_1, C_2, \phi(M), b, L_1(b)). \quad (10)$$

where $C_m = [c_m^{(1)}, c_m^{(2)}, \dots, c_m^{(n)}]$ represents the relevant configuration information of device $m \in \{1, 2\}$. T represents the number of underlying device performance-related parameters. $f_t(C_1, C_2, \phi(M), b, L_1(b))$ represents the profiling result of the t -th related mapping component based on the source and target device configurations, model features, batch state, and source-device profiling result.

To further correct the residual mismatch between the source-device profiling model and the target-device latency behavior, we collect a small set of stratified target-device anchor measurements $\mathcal{A} = \{b_i\}_{i=1}^K$. For each anchor batch state b_i , the residual correction is calculated as:

$$\delta(b_i) = \log \frac{L_2(b_i)}{\hat{L}_2^0(b_i)}. \quad (11)$$

The residual values are then interpolated over the entire profiled batch space to obtain $\hat{\delta}(b)$. The final migrated latency prediction on Device 2 is calculated as:

$$\hat{L}_2(b) = \hat{L}_2^0(b) \cdot \exp(\hat{\delta}(b)). \quad (12)$$

In this way, the original fixed empirical correction factor is replaced by a data-driven residual calibration term estimated from a small number of real target-device measurements. These anchor samples are much fewer than those required to reconstruct a complete latency profiling model on the target device.

To accurately construct the mapping model, we design the following loss function calculation formula to improve the accuracy of the mapping relationship profiling:

$$Loss = \frac{1}{N} \sum_{i=1}^N \left(\frac{L_2^{(i)}}{L_1^{(i)}} - \frac{1}{T} \sum_{t=1}^T f_t(C_1^{(i)}, C_2^{(i)}, \phi(M^{(i)}), b^{(i)}, L_1^{(i)}) \right)^2. \quad (13)$$

where N represents the number of training samples used to construct the mapping relationship. Each sample corresponds to a specific source-target device pair, DNN model, batch state, and profiling result. To quantify the reliability of the profiling, the variance $V(S)$ and the confidence interval $L_2(S)$ of the performance scaling factor S are defined as follows:

$$V(S) = \frac{1}{T} \sum_{t=1}^T (f_t(C_1, C_2, \phi(M), b, L_1(b)) - S)^2 \quad (14)$$

$$L_2(S) \in [L_1(S - z\sqrt{V(S)}), L_1(S + z\sqrt{V(S)})] \quad (15)$$

where z is the quantile of the standard normal distribution, with a default value of $z = 1.96$ corresponding to a 95% confidence level. The variance $V(S)$ reflects the sensitivity of the model to differences in device configurations. A larger variance indicates higher profiling uncertainty.

Scope and boundary conditions. The proposed migration mechanism mainly targets stable deployment scenarios, where the software stack, execution backend, and model implementation are consistent across devices. It implicitly assumes that the latency curves of the same DNN model on different devices preserve partially similar shape characteristics, so that the target-device latency can be inferred through scaling-based transfer and

lightweight residual calibration. This assumption may become weaker when the source and target devices have substantially different accelerator support, when the workload shifts between memory-bandwidth-bound and compute-bound execution regimes, or when different execution backends introduce different operator fusion, scheduling, or kernel selection behaviors. In such cases, more target-device anchor measurements may be required to maintain migration accuracy.

In summary, the performance mapping mechanism is capable of reducing the sampling cost for DNN model performance while maintaining the profiling accuracy of DNN model performance evaluation. Rather than eliminating all target-device measurements, ProfEdge avoids full target-device profiling and uses only lightweight stratified calibration to correct the migrated prediction. It provides a practical and quantifiable profiling tool for reducing target-device sampling cost under stable cross-device deployment scenarios, with lightweight calibration improving robustness under moderate device/runtime discrepancies.

5 Performance Evaluation

The experiments use four edge devices as shown in Table 1, with the following configurations: (1) Device 1 (j1): Jetson Xavier NX. (2) Device 2 (j2): Jetson AGX Xavier. (3) Device 3 (s1): Raspberry Pi 4B. (4) Device 4 (p1): x86 PC edge device. The default testing device is device 1 (j1) Jetson Xavier NX. j1j2 means transferring the original prediction model of device j1 to device j2 and correcting it. The initial sampling ratio is 20%, i.e., the length of B is 50, and g is 10. To ensure other algorithms meet the default accuracy requirements, the sampling ratios for other algorithms are set as follows: Random Forest (RF) and Linear sampling scale are 40%-60%, Layerwise sampling scale is 50%. The default parameters of RF are as follows: `n_estimators` is 50, `max_depth` is none, `min_samples_split` is 2. The default size of B is set to 50, corresponding to batch-size states from 1 to 50. This setting keeps the sampling space consistent across different models and devices, making the comparison fair and effective. The value of 50 is only an experimental setting rather than a limitation of ProfEdge, and it can be adjusted according to the supported batch-size range, model workload, and device capability.

Baseline. Our work is architectural and orthogonal to existing solutions. We focus on comparing four classical model evaluation approaches. Because the existing algorithms mainly consist of these six types and their variants. We compare ProfEdge with the following baselines.

- (1) *Linear*. These methods model [14] data's underlying patterns via feature linear combinations to predict DNN models' unknown states. They fit linear relationships in data by optimizing parameters.
- (2) *Random Forest (RF)*. These methods [41] use multiple decision trees to capture and balance complex nonlinear relationships in data, enabling accurate profiling of DNN models' unknown states. They train numerous diverse subtrees in parallel (e.g., via random feature subset sampling) and combine voting or averaging mechanisms. This approach effectively suppresses overfitting and enhances the model's generalization stability.
- (3) *Layerwise*. The first two methods build latency profiling models by sampling the entire model. These methods [51] use RF-based sampling and modeling for each layer's operators of the model.
- (4) *Flops*. These methods [17, 43] use RF as the underlying model. They leverage the FLOPs of DNN models and related architectural features to quantify modeling computational performance, thus profiling inference efficiency of different DNN models on target hardware.
- (5) *ConvMeter*. This method [2] predicts ConvNet runtime using lightweight model-level features, such as FLOPs, input size, activations, and outputs. It builds a simple hardware-aware regression model to profile inference efficiency under different hardware and batch-size settings.
- (6) *nn-Meter*. This method [52] is a kernel-level latency predictor for edge devices. It detects backend-specific execution kernels, including fused operators, and estimates end-to-end DNN latency by summing the predicted latency of these kernels.

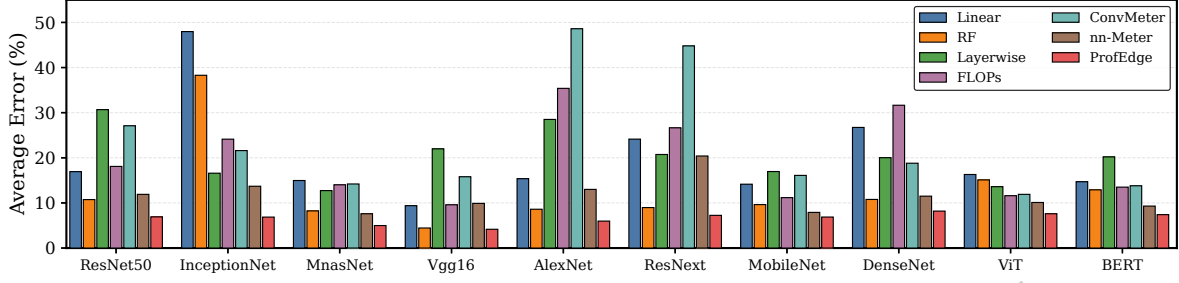
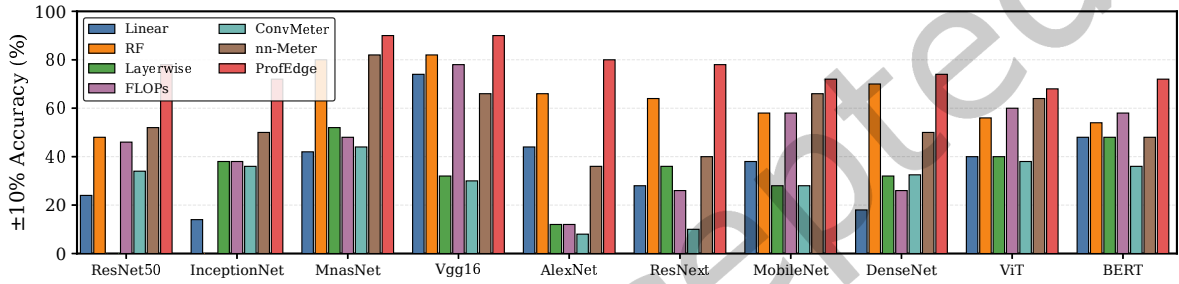


Fig. 5. Average profiling error across different DNN models.

Fig. 6. Profiling accuracy when the relative profiling error is within $\pm 10\%$.

Benchmark. We utilize common ML tasks in daily-life workloads as our benchmark, such as image classification, semantic segmentation, object detection, and natural language processing. We evaluate our work comprehensively with 10 models [37]: (1) ResNet50, (2) InceptionNet, (3) MnasNet, (4) Vgg16, (5) AlexNet, (6) ResNext, (7) MobileNet, (8) DenseNet, (9) ViT, and (10) BERT.

Evaluation criteria. Since existing works mainly focus on the accuracy of profiling results [51, 52], we concentrate on the following three metrics: (a) The average error ratio of profiling results, which is calculated by formula 1. (b) The correctness of profiling results with an error within 10%, indicating the proportion of cases where the profiling error is less than or equal to 10% across all states of the DNN model. (c) The construction time of the latency profiling model, which includes the states B of the DNN model M_D sampling time and latency profiling model M_P training time.

5.1 Main results

As shown in Figures 5 and 6, ProfEdge achieves lower profiling error and higher accuracy within 10% relative error than the compared methods across different DNN models. The evaluated workloads include both CNN-based models and transformer-based models, such as ViT and BERT, showing that ProfEdge can maintain stable profiling performance across heterogeneous model architectures. The improvement mainly comes from the adaptive sampling strategy of ProfEdge. Existing methods usually rely on fixed sampling strategies or predefined model/operator-level features, which may not sufficiently capture the state-dependent latency variation under different batch sizes. In contrast, ProfEdge first uses coarse-grained sampling to capture the global latency trend and then performs fine-grained sampling around high-error regions. Therefore, ProfEdge can reduce the average profiling error while increasing the proportion of predictions whose relative error is below 10%.

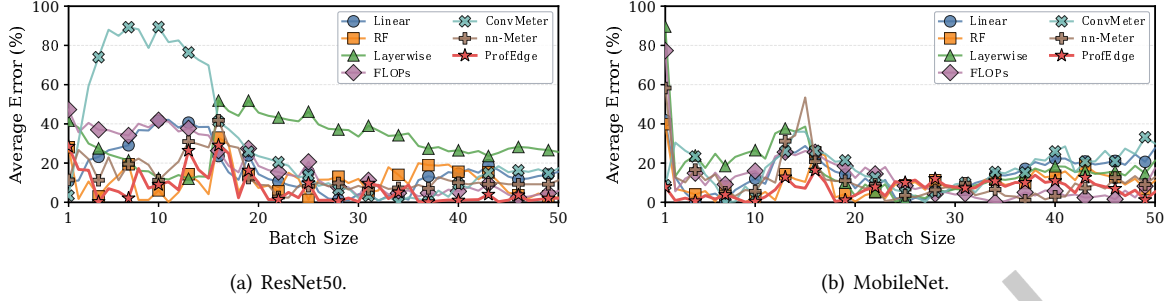


Fig. 7. Profiling error across different batch-size states for different DNN models.

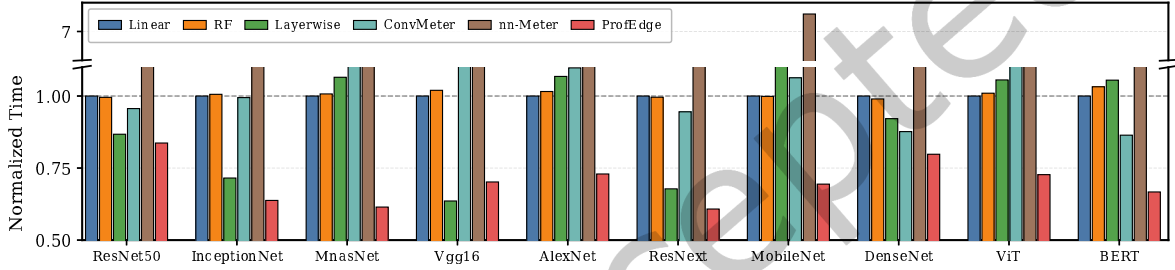


Fig. 8. Normalized construction time cost of latency profiling models for different algorithms.

As shown in Figure 7, we further compare the profiling error over different batch-size states for different DNN models. ProfEdge generally achieves lower and more stable error across most states. Although ProfEdge is not always the best method at every individual state, it maintains better overall profiling accuracy because it explicitly refines high-error regions through adaptive fine-grained sampling. This result also explains why ProfEdge achieves higher average accuracy in Figures 5 and 6. Instead of optimizing only a fixed set of sampled points, ProfEdge uses the error feedback from the current profiling model to locate states with large prediction errors and adds local samples around them. As a result, most batch-size states can be controlled within the acceptable profiling error range.

As shown in Figure 8, ProfEdge constructs the latency profiling model with lower normalized time cost than most compared methods. This is because ProfEdge does not rely on exhaustive or fixed large-scale sampling. Instead, it starts from a small coarse-grained sampling set and only adds fine-grained measurements in regions where the current profiling model is likely to produce large errors. Compared with methods that require more extensive sampling or more expensive model construction, ProfEdge reduces unnecessary latency measurements while maintaining competitive profiling accuracy. This confirms that the adaptive sampling strategy can effectively balance profiling accuracy and construction cost across different DNN models, including both CNN and transformer workloads.

As shown in Figures 9 and 10, we evaluate the effectiveness of ProfEdge’s cross-device migration module under different stratified target-device calibration ratios. The “Original” baseline denotes the latency profiling model normally constructed on the target device. The 5%, 10%, and 15% settings correspond to the cross-device migration module of ProfEdge, calibrated with different proportions of stratified target-device anchor measurements.

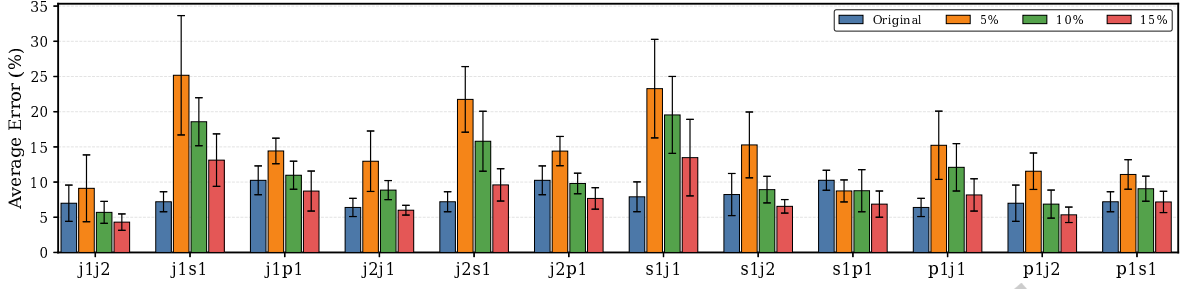


Fig. 9. Profiling errors of ProfEdge after cross-device migration under different stratified calibration ratios.

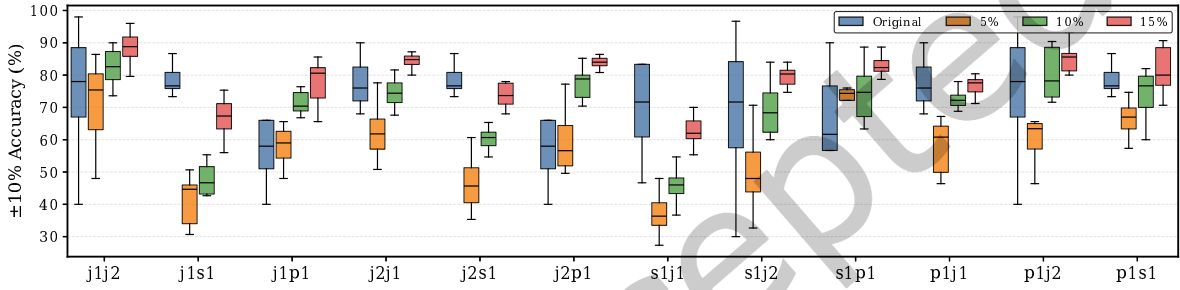


Fig. 10. The proportion of ProfEdge's migrated profiling results with relative error below 10% under different stratified calibration ratios.

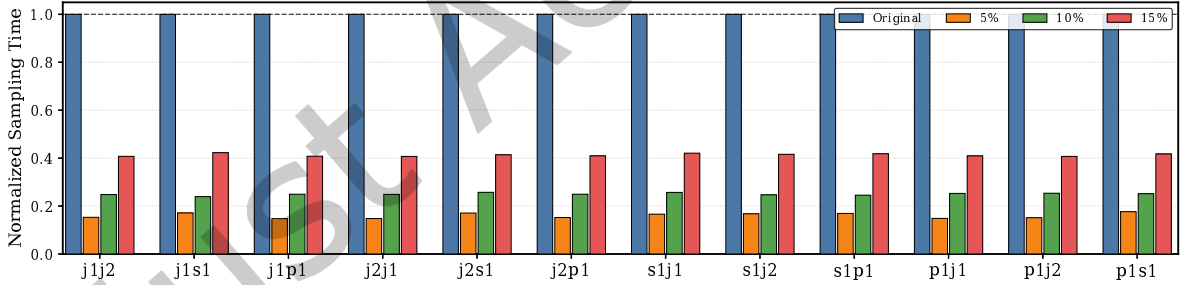


Fig. 11. Normalized sampling cost of ProfEdge's cross-device migration under different stratified calibration ratios.

Compared with using very few calibration samples, increasing the calibration ratio consistently reduces the migrated profiling error and improves the proportion of predictions whose relative error is below 10%. This demonstrates that ProfEdge can effectively use a small number of stratified target-device measurements to correct the residual mismatch between the source-device profiling model and the target-device latency behavior. Figure 11 further reports the normalized sampling cost of ProfEdge under different calibration settings. The “Original” baseline represents the normal sampling cost required to construct a latency profiling model on the target device, and its cost is normalized to 1. The results show that ProfEdge’s migration-based calibration strategies require only a small fraction of target-device samples, while achieving profiling accuracy comparable to the normally

Table 2. Fixed hyperparameter settings

Method	ResNet50			MobileNet			Vgg16		
	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3
P1	50	5	0.1	50	5	0.1	50	5	0.2
P2	250	10	0.7	250	10	0.5	250	10	0.7

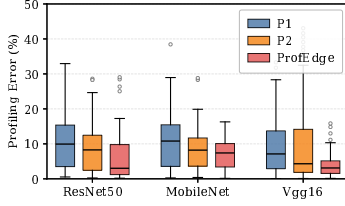


Fig. 12. The profiling error of the evaluated model for different hyperparameters.

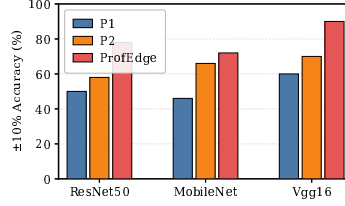
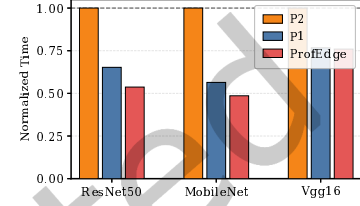
Fig. 13. The accuracy with different hyperparameters when the profiling error is within $\pm 10\%$.

Fig. 14. Construction time of the latency profiling model under different hyperparameter settings.

constructed target-device profiling model. This verifies that ProfEdge can avoid full target-device profiling and provides an effective trade-off between profiling accuracy and sampling cost during cross-device migration.

5.2 Ablation evaluation Results

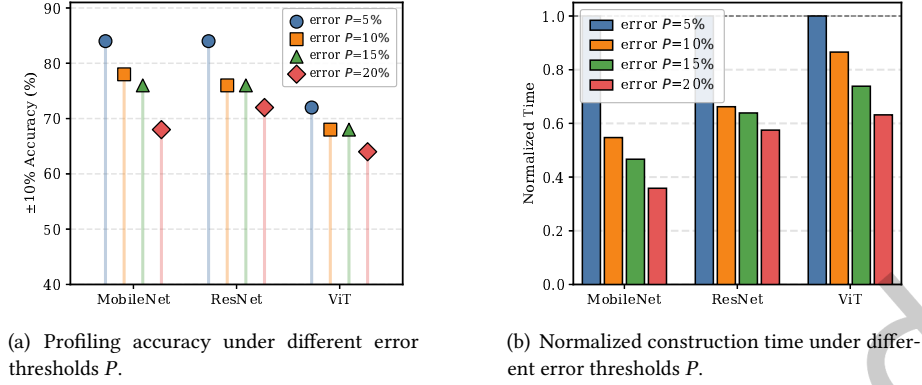
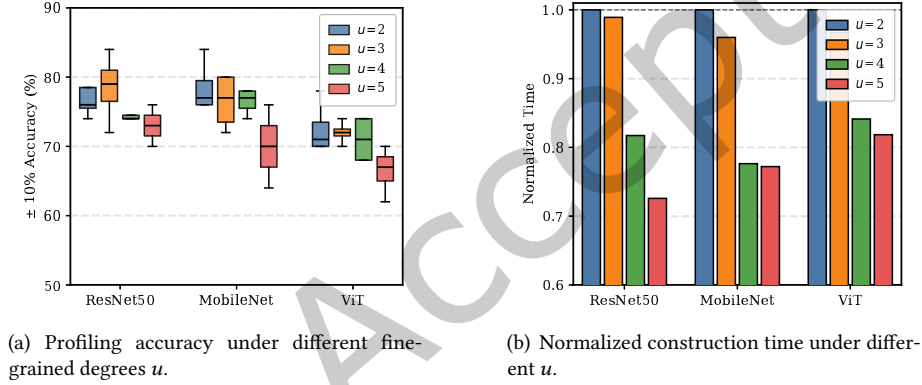
Since the latency profiling model of ProfEdge is based on a random forest model, which requires the design of various hyperparameters, we use ProfEdge with different fixed hyperparameters as ablation experiments and compare them with our designed ProfEdge. Different fixed hyperparameter settings are shown in Table 2. n_1 is $n_estimators$, n_2 is max_depth , and n_3 is $in_samples_split$.

As shown in Figures 12-14, different hyperparameters significantly impact the sampling cost and evaluation accuracy of M_p . For any DNN model and DNN state b , ProfEdge outperforms other fixed hyperparameter ProfEdge variants. This is partly because our use of Bayesian optimization enables us to find a relatively optimal set of parameters. Additionally, our dynamic updating strategy enhances the efficiency of exploring the optimal parameter set and reduces the exploration cost. As seen in Figure 12, although ProfEdge is not always the best-performing, its profiling error remains below 10%. This is due to both the influence of different hyperparameters and DNN model characteristics on the latency profiling model's performance and ProfEdge's goal of controlling the error within 10% while minimizing sampling costs.

5.3 Sensitivity Study

In this section, we further analyze the sensitivity of two key hyperparameters in ProfEdge: the acceptable profiling error threshold P and the fine-grained degree u . The former determines when the adaptive sampling process can stop, while the latter controls the granularity of additional sampling around high-error regions. By varying these two parameters, we evaluate how ProfEdge balances profiling accuracy and construction cost under different settings.

As shown in Figure 15, we conduct a sensitivity study on the acceptable profiling error threshold P by varying it from 5% to 20%. Figures 15(a) and 15(b) report the corresponding profiling accuracy and normalized construction

Fig. 15. Sensitivity study of the error threshold P .Fig. 16. Sensitivity study of the fine-grained degree u .

time on MobileNet, ResNet, and ViT, respectively. As shown in Figure 15(a), a stricter error threshold generally leads to higher profiling accuracy. When $P = 5\%$, ProfEdge achieves the highest $\pm 10\%$ accuracy on all three models. As P increases from 5% to 20%, the accuracy gradually decreases, since ProfEdge allows a larger acceptable profiling error and therefore performs fewer fine-grained sampling operations. As shown in Figure 15(b), increasing P significantly reduces the normalized construction time. This is because a larger error threshold allows ProfEdge to terminate adaptive sampling earlier and avoid unnecessary latency measurements. For MobileNet and ResNet, the construction time decreases consistently as P increases. For ViT, the construction time also decreases overall, although a slight fluctuation appears when $P = 15\%$, which is mainly caused by the different high-error regions selected during adaptive sampling. Overall, Figure 15 directly demonstrates the accuracy–cost tradeoff controlled by P . A smaller P improves profiling accuracy at the cost of more sampling, while a larger P reduces sampling cost with moderate accuracy degradation. Therefore, we set $P = 10\%$ as the default value in ProfEdge, which provides a practical balance between profiling accuracy and construction cost.

As shown in Figure 16, we conduct a sensitivity study on the fine-grained degree u by varying it from 2 to 5. Figures 16(a) and 16(b) report the corresponding $\pm 10\%$ profiling accuracy and normalized construction

Table 3. Time breakdown across different devices. BO and GPR represent the profiling overhead beyond latency sampling.

Model	Raspberry Pi			Jetson			PC		
	Sampling	BO	GPR	Sampling	BO	GPR	Sampling	BO	GPR
MobileNet	88.62%	11.37%	0.02%	91.18%	8.81%	0.01%	80.44%	19.55%	0.01%
ResNet	90.57%	9.42%	0.01%	89.41%	10.59%	0.01%	85.35%	14.64%	0.01%
ViT	96.23%	3.76%	<0.01%	97.76%	2.24%	<0.01%	92.33%	7.66%	<0.01%

time on ResNet50, MobileNet, and ViT, respectively. As shown in Figure 16(a), different values of u lead to different profiling accuracy across DNN models. A smaller u usually enables denser fine-grained sampling around high-error regions, which can improve profiling accuracy but also increases the construction cost. In contrast, a larger u reduces the number of additional fine-grained samples, but may miss local latency variations and lead to lower profiling accuracy. As shown in Figure 16(b), $u = 4$ achieves relatively low construction time while maintaining competitive profiling accuracy across the tested models. Although $u = 2$ or $u = 3$ can achieve slightly higher accuracy in some cases, they require a higher construction cost. When $u = 5$, the construction time can be further reduced, but the profiling accuracy decreases more noticeably. Therefore, we set $u = 4$ as the default value in ProfEdge, since it provides a practical balance between profiling accuracy and sampling cost.

5.4 overhead

To evaluate whether the optimization modules introduce non-negligible overhead, we further measure the runtime breakdown of ProfEdge on different devices. As shown in Table 3, latency sampling dominates the total profiling cost across all devices and models. For example, the sampling cost accounts for 88.62%–96.23% on Raspberry Pi, 89.41%–97.76% on Jetson, and 80.44%–92.33% on PC. In contrast, the overhead of GPR training is almost negligible, remaining below 0.02% in all cases. The overhead of BO is also much smaller than the sampling cost, and its proportion further decreases as the model becomes larger. These results show that the additional computation introduced by GPR and BO does not offset the benefit of reducing latency sampling cost.

6 Related Work

The inference latency of deep neural network models on edge devices is mainly constrained by two key factors: model architecture characteristics and hardware configuration differences. Many existing profiling algorithms focus on model-level analysis and fail to adequately consider the heterogeneity of edge devices, resulting in significant profiling deviations. Next, we comprehensively review the two most representative model performance evaluation methods.

Model-level profiling. Existing research mainly focuses on profiling the inference latency of neural network architectures. Gujarati et al. [15] propose a profiling method that models the DNN inference process as a sequence of deterministic mathematical operations. Adolf et al. [1] develop an analytical framework for profiling model inference latency based on the assumption that similar structures have similar performance. Some researchers utilize simpler metrics, such as Liu et al. [27], who directly use FLOPs to estimate the inference cost of DNNs in the Darf framework. Additionally, some related studies [17, 38, 43] use FLOPs and MAC directly as delay substitutes or as feature inputs to regression models. These methods share a common limitation: most adopt fixed sampling strategies, which often require high sampling costs to ensure profiling accuracy.

Operator-level profiling. Some studies use machine learning to learn operator delays from low-level implementations. These methods mainly use code features with simple regression models for delay profiling [42, 51, 52]. For example, Stamoulis et al. [42] proposed a gradient-based predictor to estimate delay via hierarchical computation unfolding ratios. Justus et al. [21] treated single network layers as atomic operations to predict overall

model inference latency. Zhang et al. [52] proposed and developed nn-Meter, an efficient system for accurately profiling DNN model inference latency across different edge devices. It samples and predicts the performance of various operators, summing up results for an overall profiling. Yang et al. [51] pointed out that key operators like convolutional layers dominate DNN performance. They precisely predicted these operators' performance to map overall model performance, reducing sampling overhead. Although these methods are good at assessing the performance of unknown models, their latency profiling accuracy in scenarios requiring high assessment precision needs further improvement.

7 Conclusion

This paper addresses the high cost and inefficiency of profiling DNN inference latency on edge devices by proposing ProfEdge, which is based on Gaussian process regression and Bayesian optimization. Existing approaches often rely on large-scale latency measurements or internal model structures, making it difficult to balance profiling accuracy and construction cost, especially in black-box deployment scenarios. ProfEdge meets these challenges through threefold innovations. First, it combines Gaussian process regression with a coarse-grained sampling strategy to dynamically identify high-error regions and perform local fine-grained sampling, which significantly reduces data collection overhead. Second, it introduces Bayesian optimization to dynamically adjust the latency profiling model parameters, further enhancing profiling accuracy. Third, it designs a cross-device transfer mechanism based on device configuration mapping, which avoids the full reconstruction of latency profiling models on new devices by using lightweight stratified calibration samples. Experiments on multiple edge devices and different DNN models with diverse architectures validate the effectiveness of ProfEdge. The results show that ProfEdge reduces the average profiling error by up to 80% and the model construction time by over 70% compared to baseline methods. Additionally, ProfEdge enables efficient cross-device migration of latency profiling models. With only a small number of stratified target-device calibration samples, ProfEdge achieves profiling accuracy close to that of fully constructed target-device profiling models, while avoiding full target-device profiling.

8 Acknowledgments

This work is partially supported by the Smart Grid-National Science and Technology Major Project (Grants No. 2026ZD0812104), National Natural Science Foundation of China (Grants No. 62302096, 62272098, and U24B20152), Central University Basic Research Fund of China (Grants No. 2242024K40019), and the Major Project of Fundamental Research on Frontier Leading Technology of Jiangsu Province (Grants No. BK20222006).

References

- [1] Robert Adolf, Saketh Rama, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2016. Fathom: Reference workloads for modern deep learning methods. In *2016 IEEE International Symposium on Workload Characterization*. IEEE, 1–10.
- [2] Tim Beringer, Jakob Stock, Arya Mazaheri, and Felix Wolf. 2024. Dissecting Convolutional Neural Networks for Runtime and Scalability Prediction. In *Proceedings of the 53rd International Conference on Parallel Processing*. 168–178.
- [3] Gérard Biau. 2012. Analysis of a random forests model. *The Journal of Machine Learning Research* 13 (2012), 1063–1095.
- [4] Claudio Casetti, Carla Fabiana Chiasserini, Falko Dressler, Agon Memedi, Diego Gasco, and Elad Michael Schiller. 2024. AI/ML-based services and applications for 6G-connected and autonomous vehicles. *Computer Networks* 255 (2024), 110854.
- [5] Zhuoqing Chang, Shubo Liu, Xingxing Xiong, Zhaohui Cai, and Guoqing Tu. 2021. A survey of recent advances in edge-computing-powered artificial intelligence of things. *IEEE Internet of Things Journal* 8, 18 (2021), 13849–13875.
- [6] Suresh Chavhan, Deepak Gupta, Sarada Prasad Gochhayat, Chandana B N, Ashish Khanna, K Shankar, and Joel JPC Rodrigues. 2022. Edge computing AI-IoT integrated energy-efficient intelligent transportation system for smart cities. *ACM transactions on internet technology* 22, 4 (2022), 1–18.
- [7] Yanming Chen, Tong Luo, Weiwei Fang, and Neal N Xiong. 2024. Edgeci: Distributed workload assignment and model partitioning for cnn inference on edge clusters. *ACM Transactions on Internet Technology* 24, 2 (2024), 1–24.
- [8] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. 2024. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

- [9] Long Cheng, Yan Gu, Qingzhi Liu, Lei Yang, Cheng Liu, and Ying Wang. 2024. Advancements in accelerating deep neural network inference on AIoT devices: A survey. *IEEE Transactions on Sustainable Computing* 9, 6 (2024), 830–847.
- [10] Haolin Chu, Xiaolong Zheng, Liang Liu, and Huadong Ma. 2023. nnPerf: Demystifying DNN runtime inference latency on mobile platforms. In *Proceedings of the 21st ACM Conference on Embedded Networked Sensor Systems*. 125–137.
- [11] Shi Dong, Junxiao Tang, Khushnood Abbas, Ruizhe Hou, Joarder Kamruzzaman, Leszek Rutkowski, and Rajkumar Buyya. 2024. Task offloading strategies for mobile edge computing: A survey. *Computer Networks* (2024), 110791.
- [12] Lukasz Dudziak, Thomas Chau, Mohamed Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas Lane. 2020. Brp-nas: Prediction-based nas using gcns. *Advances in neural information processing systems* 33 (2020), 10480–10490.
- [13] Peter I Frazier. 2018. Bayesian optimization. In *Recent advances in optimization and modeling of contemporary problems*. Informa, 255–278.
- [14] Jingzhi Gong and Tao Chen. 2024. Deep configuration performance learning: A systematic survey and taxonomy. *ACM Transactions on Software Engineering and Methodology* 34, 1 (2024), 1–62.
- [15] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving {DNNs} like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 443–462.
- [16] Jianwei Hao, Piyush Subedi, Lakshmish Ramaswamy, and In Kee Kim. 2023. Reaching for the sky: Maximizing deep learning inference throughput on edge devices with ai multi-tenancy. *ACM Transactions on Internet Technology* 23, 1 (2023), 1–33.
- [17] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*. 784–800.
- [18] Haochen Hua, Yutong Li, Tonghe Wang, Nanqing Dong, Wei Li, and Junwei Cao. 2023. Edge computing with artificial intelligence: A machine learning perspective. *Comput. Surveys* 55, 9 (2023), 1–35.
- [19] Kaifeng Hua, Shengchao Su, and Yiwang Wang. 2025. Intelligent service migration for the internet of vehicles in edge computing: A mobility-aware deep reinforcement learning framework. *Computer Networks* 257 (2025), 111021.
- [20] Sekione Reward Jeremiah, Laurence Tianruo Yang, and Jong Hyuk Park. 2024. Digital twin-assisted resource allocation framework based on edge collaboration for vehicular edge computing. *Future Generation Computer Systems* 150 (2024), 243–254.
- [21] Daniel Justus, John Brennan, Stephen Bonner, and Andrew Stephen McGough. 2018. Predicting the computational cost of deep learning models. In *2018 IEEE international conference on big data (Big Data)*. IEEE, 3873–3882.
- [22] Xiangjie Kong, Yuhua Wu, Hui Wang, and Feng Xia. 2022. Edge computing for internet of everything: A survey. *IEEE Internet of Things Journal* 9, 23 (2022), 23472–23485.
- [23] Borui Li, Tiange Xia, and Shuai Wang. 2025. InfScaler: Enabling Efficient ML Inference Serving on Multi-Accelerator Edge Devices via Asymmetric Auto-Scaling. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–7.
- [24] Borui Li, Tiange Xia, Weilong Wang, Jingyuan Zhang, Shuai Wang, Chenhong Cao, and Zheng Dong. 2025. FluidEdge: Expediting Serverless Machine Learning Inference via Bottleneck-aware Auto-scaling on Edge SoCs. *IEEE Transactions on Mobile Computing* (2025).
- [25] Wendi Li, Hao Wu, Bing Gao, Bing Mao, Fengyuan Xu, and Sheng Zhong. 2026. Diverse Human Driving Vehicle Simulation in Background Traffic for Autonomous Driving Tests. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 40. 36307–36315.
- [26] Ying Li, Yifan Sun, and Adwait Jog. 2023. Path forward beyond simulators: Fast and accurate gpu execution time prediction for dnn workloads. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 380–394.
- [27] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).
- [28] Jiawei Liu, Bing Dong, Weilong Wang, Muhan Yuan, Borui Li, Zhao-Dong Xu, and Shuai Wang. 2025. nnWeb: Towards efficient WebGPU-based DNN inference via automatic collaborative offloading. *Computer Networks* 270 (2025), 111489.
- [29] Tianen Liu, Shuai Wang, Zheng Dong, Borui Li, and Tian He. 2025. From perception to computation: Revisiting delay optimization for connected autonomous vehicles. *Comput. Surveys* 57, 8 (2025), 1–45.
- [30] Tianen Liu, Shuai Wang, Borui Li, Zheng Dong, Guang Wang, Wei Gong, and Tian He. 2025. Real-batch: Real-time adaptive batch processing for accurate object detection in autonomous driving. *IEEE Transactions on Mobile Computing* (2025).
- [31] Weihong Liu, Jiawei Geng, Zongwei Zhu, Yang Zhao, Cheng Ji, Changlong Li, Zirui Lian, and Xuehai Zhou. 2023. Ace-Sniper: cloud-edge collaborative scheduling framework with dnn inference latency modeling on heterogeneous devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 2 (2023), 534–547.
- [32] Songtao Lu, Weilong Wang, Borui Li, Shuai Wang, Xiaolei Zhou, and Zhao-Dong Xu. 2024. EdgeMeter: Towards Efficient and Accurate Latency Prediction of Neural Network Model Inference on Edge Devices. In *China Conference on Wireless Sensor Networks*. Springer, 118–129.
- [33] Lianbo Ma, Nan Li, Guo Yu, Xiaoyu Geng, Shi Cheng, Xingwei Wang, Min Huang, and Yaochu Jin. 2023. Pareto-wise ranking classifier for multiobjective evolutionary neural architecture search. *IEEE Transactions on Evolutionary Computation* 28, 3 (2023), 570–581.
- [34] Saeed Nair, Saad Abbasi, Alexander Wong, and Mohammad Javad Shafiee. 2022. Maple-edge: A runtime latency predictor for edge devices. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 3660–3668.

- [35] Marco Palena, Tania Cerquitelli, and Carla Fabiana Chiasserini. 2024. Edge-device collaborative computing for multi-view classification. *Computer Networks* 254 (2024), 110823.
- [36] Paniz Parastar, Giuseppe Caso, Jesus Alberto Omaña Iglesias, Andra Lutu, and Ozgu Alay. 2024. Rethinking the mobile edge for vehicular services. *Computer Networks* 253 (2024), 110687.
- [37] PyTorch. 2025. PyTorch Hub. <https://pytorch.org/hub/>
- [38] Stefan Reif, Benedict Herzog, Judith Hemp, Timo Hönig, and Wolfgang Schröder-Preikschat. 2020. Precious: Resource-demand estimation for embedded neural network accelerators. In *First International Workshop on Benchmarking Machine Learning Workloads on Emerging Hardware*, Vol. 2.
- [39] Megha Sharma, Abhinav Tomar, and Abhishek Hazra. 2024. Edge computing for industry 5.0: Fundamental, applications and research challenges. *IEEE Internet of Things Journal* (2024).
- [40] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE internet of things journal* 3, 5 (2016), 637–646.
- [41] Md Maruf Hossain Shuvo, Syed Kamrul Islam, Jianlin Cheng, and Bashir I Morshed. 2022. Efficient acceleration of deep learning inference on resource-constrained edge devices: A review. *Proc. IEEE* 111, 1 (2022), 42–91.
- [42] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. 2019. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 481–497.
- [43] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2820–2828.
- [44] Pavan Kumar Anasosalu Vasu, James Gabriel, Jeff Zhu, Oncel Tuzel, and Anurag Ranjan. 2023. Mobileone: An improved one millisecond mobile backbone. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 7907–7917.
- [45] Weilong Wang, Borui Li, Hai Wang, Wei Xi, Shuai Wang, and Tian He. 2026. InfSquad: SLO-aware Serverless Machine Learning Inference with Wasm-assisted Hybrid Functions. *IEEE Transactions on Services Computing* (2026).
- [46] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10734–10742.
- [47] Lars Wulfert, Johannes Kühnel, Lukas Krupp, Justus Viga, Christian Wiede, Pierre Gembaczka, and Anton Grabmaier. 2024. AlfES: A next-generation edge AI framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46, 6 (2024), 4519–4533.
- [48] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. 2019. A first look at deep learning apps on smartphones. In *The World Wide Web Conference*. 2125–2136.
- [49] Xiaowei Xu, Yukun Ding, Sharon Xiaobo Hu, Michael Niemier, Jason Cong, Yu Hu, and Yiyu Shi. 2018. Scaling for edge inference of deep neural networks. *Nature Electronics* 1, 4 (2018), 216–222.
- [50] Guozhi Yan, Kai Liu, Chunhui Liu, and Jie Zhang. 2024. Edge intelligence for internet of vehicles: A survey. *IEEE Transactions on Consumer Electronics* (2024).
- [51] Yanan Yang, Laiping Zhao, Yiming Li, Huanyu Zhang, Jie Li, Mingyang Zhao, Xingzhen Chen, and Keqiu Li. 2022. INFless: a native serverless system for low-latency, high-throughput inference. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 768–781.
- [52] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. 2021. 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*. 81–93.
- [53] Mingjin Zhang, Xiaoming Shen, Jiannong Cao, Zeyang Cui, and Shan Jiang. 2024. Edgeshard: Efficient llm inference via collaborative edge computing. *IEEE Internet of Things Journal* (2024).
- [54] Jie Zhao, Feng Lyu, Hao Wu, Fan Wu, and Shucheng Li. 2025. Game in Motion: Heterogeneous Task Offloading in Dynamic Vehicular Edge Computing. *IEEE Transactions on Networking* 34 (2025), 2332–2346.

Received 5 December 2025; revised 14 June 2026; accepted 29 August 2026