Sniper: Cloud-Edge Collaborative Inference Scheduling with Neural Network Similarity Modeling

Weihong Liu♯, Jiawei Geng♯, Zongwei Zhu*, Jing Cao, Zirui Lian
School of Software Engineering, Suzhou institute for Advanced Research,
University of Science and Technology of China, Jiangsu, China
{lwh2017,gjw1998}@mail.ustc.edu.cn,zzw1988@ustc.edu.cn,{congjia,ustclzr}@mail.ustc.edu.cn

ABSTRACT

The cloud-edge collaborative inference demands scheduling the artificial intelligence (AI) tasks efficiently to the appropriate edge smart device. However, the continuously iterative deep neural networks (DNNs) and heterogeneous devices pose great challenges for inference tasks scheduling. In this paper, we propose a self-update cloud-edge collaborative inference scheduling system (Sniper) with time awareness. At first, considering that similar networks exhibit similar behaviors, we develop a non-invasive performance characterization network (PCN) based on neural network similarity (NNS) to accurately predict the inference time of DNNs. Moreover, PCN and time-based scheduling algorithms can be flexibly combined into the scheduling module of Sniper. Experimental results show that the average relative error of network inference time prediction is about 8.06%. Compared with the traditional method without time awareness, Sniper can reduce the waiting time by 52% on average while achieving a stable increase in throughput.

KEYWORDS

AI system, cloud-edge collaborative inference, neural network similarity, scheduling, heterogeneous computing

1 INTRODUCTION

The cloud-edge collaborative architecture [1] (e.g., EdgeXFoundry [2], Sedna [3]) has attracted increasing attention with the widespread use of smart edge devices. As shown in Figure 1, there are multiple heterogeneous devices and a constant stream of heterogeneous inference tasks within a certain domain (e.g., industrial internet, smart city). These inference tasks are requested to be handled with deep neural network (DNN) models in the cloud and smart edge devices in the domain. However, the communication and computational performance of different edge servers within the same domain is highly heterogeneous (e.g., 4G/5G edge computing base stations of different service providers in smart cities and edge gateways of various standards in the industrial internet). Moreover, due to the diversity of tasks and the fact that DNN models are continuously self-updating and iterating, the models also exhibit heterogeneity. Consequently, it poses great challenges for the cloud to schedule each task to the most appropriate device and leads to a significant bottleneck in quality of service (QoS).

To improve the QoS, various methods have been proposed in [4–9], which brings excellent efficiency to cloud-edge collaborative inference. Among them, when deploying models to edge devices, both the model-parallel approach in [4] and [5] and the data-parallel approach in [6] and [7] are inseparable from task scheduling. These two mechanisms mainly adopt the following two methods in scheduling. a) Non-time-aware classical scheduling methods including *Earliest Deadline First* (*EDF*) [8] and *Polling* [9] etc. b) time-aware scheduling methods that use inference time as the key basis. For the reason that traditional non-time-aware approaches are prone to throughput bottlenecks and load disparity, current works prefer time-aware scheduling methods. However, most of the studies [4–7] actually run DNNs to obtain inference times. This method is difficult and time-consuming to implement in a cloud-edge collaborative inference with rapid model iterations and diverse devices.

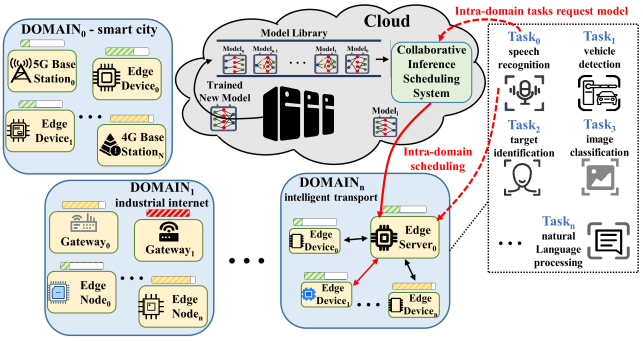


Figure 1: Illustration of Cloud-Edge Collaborative Inference.

Therefore, DNN inference time prediction is gaining popularity. Some studies [10–12] conduct intrusive model profiling to obtain DNN inference performance in advance by evaluating the different computation complexity of each layer in a network and a device. However, such intrusive methods require expensive calculation and open-source code, making them impractical to be applied in scheduling for cloud-edge collaborative inference scenarios where the extra overhead cannot be tolerated. A recent work [13] proposes a non-intrusive approach to analyzing the memory usage of each DNN layer. However, the analysis of DNN inference time is more complex than memory analysis due to the strong correlation between network performance and the acceleration methods of smart devices.

A recent work [14] defines the similarity of neural network representations via computing statistics on two different data embeddings that capture appropriate geometrical or statistical properties. Furthermore, our thorough research suggests that similar DNN structural features can result in similar DNN performance. Inspired by this, we find that the DNN inference time can be theoretically predicted by a performance characterization method, which is useful for establishing the most effective scheduling system. Therefore,

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530474

♯ Co-first authors.
* Corresponding authors.

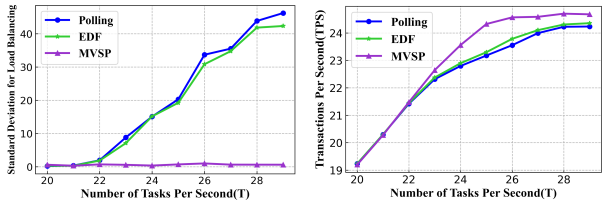
this work focuses on time-aware of DNN inference tasks and proposes a non-intrusive inference system (***Sniper***). The primary aims of ***Sniper*** are (a) to predict the DNN inference time by the similarity of DNN performance characterization, and (b) to sufficiently leverage edge computing power while improving QoS. To achieve these aims, we make the following contributions.

- We analyze the *Neural Network Similarity* (***NNS***) and discuss the feasibility of its usage.
- We propose a DNN *Performance Characterization Network* (***PCN***) based on the NNS modeling, which predicts inference time without expensive computation.
- We propose a cloud-edge collaborative inference scheduling system (***Sniper***) based on non-intrusive ***PCN*** to obtain optimal scheduling performances.

Experiments based on a real implementation show that the prediction errors of the DNN inference time are 8.06% or less. Moreover, under system load limit conditions, ***Sniper*** outperforms the state-of-the-art methods without time awareness, by up to 52% shorter average waiting time while achieving a stable increase in throughput. In addition, ***Sniper*** has a broader range of applications and takes less time per scheduling than the *MVSP* [6] based on measurements.

2 MOTIVATION

According to some works [15, 16], numerous new DNN models have emerged with different inference times on diverse devices. For such NP-hard problems as scheduling of multiple inference tasks on different devices, we perform experiments on ten computing nodes consisting of three heterogeneous devices (NVIDIA AGX, TX2, and NX) for nine types of inference tasks to explore the performance of different algorithms in heterogeneous systems, including the non-time-aware algorithms (*Polling* [9], *EDF* [8]) and the time-aware algorithm (*MVSP* [6]). This experiment specifies two metrics for evaluating the algorithm performance: *transactions per second (TPS)* and *standard deviation for load balancing* [17]. The *TPS* can be used as a quantitative indicator of the throughput, and the *standard deviation for load balancing* measures the load balancing effect of the computing platform.



(a) Load Balancing (b) Transactions Per Second

Figure 2: Scheduling Performance at Different T

As shown in Figure 2(a), as the number of tasks per second (T) increases, the *EDF* and *Polling* algorithms, which lack time awareness, experience load imbalance and become increasingly severe when multiple tasks arrive ($T > 22$). However, *MVSP* based on obtaining inference time with real measurements significantly outperforms the *EDF* and *Polling* algorithms in load balancing. In addition, as shown in Figure 2(b), the *TPS* of *EDF* and *Poling* is also lower than that of *MVSP*. Consequently, a summative conclusion can be drawn as follows.

- Algorithms without time awareness generate load imbalance, which is exacerbated on heterogeneous edge devices.
- The imbalanced load causes the computing power of the platform to be underutilized, further resulting in inferior throughput (*TPS*).

More seriously, due to the extensive number of fast iterative DNN models in cloud-edge collaborative architecture, the measurement-based time awareness methods introduce additional waiting delay. As a result, owing to intolerable waiting delays and the lack of accurate inference time prediction methods, many scheduling works [4, 6, 18] that claim to reduce response times fall short of fully exploiting computing power when scheduling multiple tasks.

In summary, to get the optimal inference scheduling performance, the inference time of the model has to be obtained in advance. However, it is not feasible for the measurement-based time awareness due to the fact that DNN models are continuously self-updating and iterating. Therefore, there is a need to propose a method that can rapidly predict the inference time of DNNs on heterogeneous devices.

3 SYSTEM DESIGN

This section describes an inference scheduling system (***Sniper***) based on DNN feature modeling. Section 3.1 shows the overall design of ***Sniper***, and then the rest of the section describes the three main modules in turn.

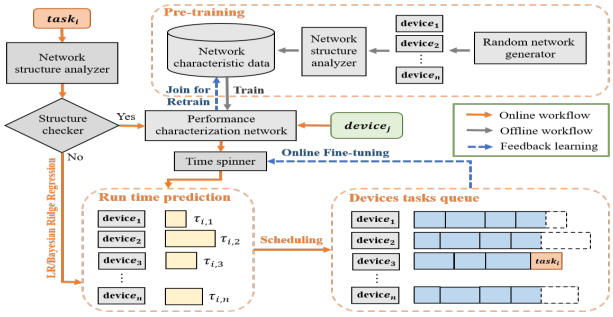


Figure 3: An Overview of *Sniper* System

3.1 Overall Design

Figure 3 shows the overall design of ***Sniper*** and the data flow in the scheduling process. Prior to using the system, it is necessary to train the performance characterization network (***PCN***) in the pre-train structure. The training data comes from the random network generator running on heterogeneous edge devices in the computing platform. During application, the data passes through the three modules of ***Sniper*** in turn, which are the **DNN feature extraction module** in Section 3.2, the **performance characterization module** in Section 3.3, and the **scheduling module** in Section 3.4. In the DNN feature extraction module, we use the Network Structure Analyzer to analyze the DNN network structure

in the cloud, which is constantly iterative and self-updating. Then, according to the assessment of the Structure Checker, the DNNs that require performance prediction will enter the performance characterization module, while the rest will enter the scheduling module directly. In order to correct the prediction error, we put up the Time Spinner to fine-tune the prediction findings. Finally, each task is correctly scheduled in the scheduling module based on the predicted inference time to optimize QoS and maximum *TPS*.

3.2 DNN Feature Extraction Module

Network Structure Analyzer. The Network Structure Analyzer performs offline analysis of the neural network and extracts network structure features relevant to performance prediction.

We need a feature extraction method for the constantly iterative and self-updating models, which can provide sufficient inferential behavioral features for network performance prediction. It is mentioned in [19] that several types of smart acceleration chips accelerate fundamental network layer structures in various fashions. Notably, the inference behavior of fundamental network layers is similar across the same type of devices. Based on this finding, we can evaluate the network behavior features in five dimensions and present the retrieved features in Table 1.

Table 1: DNN feature extraction based on five dimensions

Dimensions	Feature
Network Structure	maximum number of branches; number of layers; number of critical path layers; proportion of critical path parameters, layers and computation; average computation of network layers
Calculation Times	number of computation; number of critical path; computation; number of convolutional computation; number of fully connected computation
Number of Parameters	number of parameters; number of critical path parameters; number of convolutional parameters; number of fully connected parameters
Proportion of Network Layer	number and proportion of normalized layers, convolutional layers(1x1, 3x3, 5x5, 7x7), atrous convolutional layers(3x3), fully connected network layers, concat layers, add layers, pooling layers(average, maximum)
Dependency	Determined by actual task performance

To extract critical performance-related features from neural networks, we use the random network generator on the devices (NVIDIA AGX, TX2, and NX) to build a vast number of DNN datasets. Then, we used the *Pearson coefficients* of each feature and the measured runtime to determine their correlation. The linear residual fitting method is utilized to continue the critical feature extraction based on the calculated *Pearson coefficients*. The procedure of extracting the relevant features is illustrated in the Algorithm 1.

Through the multi-stage feature processing above, we extract 6 key features from Table 1, including *total computation, number of atrous convolution layers(3x3), number of convolution layers(1x1, 5x5), number of critical path network parameters,* and *proportion of critical path network parameters.* Based on the above features, we predict the network performances using a traditional fitting algorithm (Bayesian Ridge Regression) on a single device. On the NVIDIA AGX, TX2, or NX, their average prediction errors of the training and test sets are within **0.6ms** and **3ms**, respectively. These results demonstrate that the above features can achieve performance feature fitting of most networks on a single device.

Algorithm 1 Key Feature Extraction

Input: F_M: features matrix, T: time, K: max length of K_F
Output: K_F: key features
1: **function** KFE(F_M, T)
2: Load F_M and T, Initialize K_F by NULL
3: **while** length of K_F less than K **do**
4: Calculate Pearson coefficient of each feature in F_M with respect to T
5: Select the feature with the largest Pearson coefficient into K_F
6: Delete the selected feature from F_M
7: Calculate the residual error (E) of each feature in K_F after linear fitting
8: $T \leftarrow E$
9: **end while**
10: **end function**

Structure Checker. The Structure Checker is primarily responsible for choosing a strategy of predicting based on the network structure features gathered in the Network Structure Analyzer. The Structure Checker uses the overall computational volume to determine the size of the network. The medium or large networks will flow to the performance characterization module for prediction. For small networks, we directly use the prediction results of traditional prediction models as the basis for scheduling.

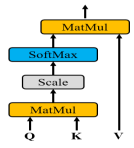
3.3 Performance Characterization Module

Network Characteristic Database. Due to the extensive number of DNN models and the self-updating iterations of them, the initial training set cannot fully consider all networks and devices. Therefore, we form a feedback data flow by maintaining the Network Characteristic Database, preserving historical network structures, and inference times. Based on the feedback data flow, we design a feedback learning algorithm for ***Sniper***, including an online fine-tuning mechanism and an offline retraining mechanism.

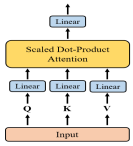
Performance Characterization Network (PCN). ***PCN*** is primarily used to predict the performance of networks on heterogeneous devices based on network features. We describe the concept of *Neural Network Similarity* (***NNS***) in this part and then use the ***PCN*** with the self-attention [20] module as a carrier network for ***NNS*** to predict the performance of DNNs.

To analyze the differences in performance characterization of each network on edge devices, this section defines the concept of ***NNS***, which is the similarity of the performance characterization of DNNs on a single device based on its structure, behavior, and other characteristics. The proof of ***NNS*** for a single device are shown in Section 3.2. In heterogeneous devices, the key features affecting performance vary with hardware parameters. Since the same network has different acceleration results in different smart chips, it is necessary to introduce hardware parameter analysis for heterogeneous scaling of ***NNS***. Therefore, we designed the ***PCN*** with a self-attention module to characterize the ***NNS***, which takes network features and hardware key parameters as input, and takes inference time as output, as shown in Figure 4(c).

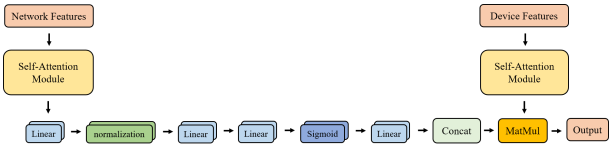
To increase the universality of the neural network features in Table 1, the self-attention module is utilized to automate the combination of network features to achieve network key feature extraction, as illustrated in Figure 4(a). Automatic feature derivation is accomplished in the self-attention module via the point product combination of network features, which generates the queue, key, and value in the self-attention module. Finally, as shown in Figure 4(b), the self-attention module is constructed by connecting



(a) Scaling Point Product Attention



(b) Self-Attention Module



(c) Performance Characterization Network (***PCN***)

Figure 4: Network Structure of PCN

the fully linked layer to the scaled point product attention. The self-attention of x can be calculated through the following equation.

$$Attention(x) = softmax(\frac{Q(x) \times K(x)^T}{\sqrt{d_k}}) \times V(x) \quad (1)$$

where $Q(x)$, $K(x)$ and $V(x)$ are the results for the query, keys and values of x, which are all vectors; $\frac{1}{\sqrt{d_k}}$ is the scaling factor.

The overall network structure of ***PCN*** is shown in Figure 4(c). The neural network features consist of the features listed in Table 1. Then, the output features from the self-attention module are connected to a multilayer fully connected network. Next, a sigmoid layer is added to map features into [-1, 1] to multiply with the output critical features of the device by the self-attention module. Finally, the inference time is obtained. The network structure of ***PCN*** is shown in Figure 4(c). The calculation formulas are as follows:

$$\mathcal{F}(x) = Concat(multi\_connect(x)) \quad (2)$$

$$Output = \mathcal{F}(Attention(x_n)) \times Attention(x_d) \quad (3)$$

where $multi\_connect(x)$ is the multi-layer fully connected network through which the Self-Attention output passes; $concat(m_x)$ is a concatenation of the output (m_x) of a multi-headed network; x_n, x_d are the input for network feature and device feature.

We use the single-device prediction model discussed in Section 3.2 and the random network generator to generate a large number of datasets for the pre-training of ***PCN***.

Time Trimmer. The Time Trimmer is mainly utilized in online fine-tuning. It can automatically select the network characteristics that best fit the prediction residuals and make the prediction results more accurate by parameter fine-tuning. To achieve online fine-tuning of the sniper, we calculate the *Pearson coefficients* for each feature using the predicted residuals as indicators and fit the residuals using the top ones with the highest correlation.

3.4 Scheduling Module

The scheduling module is mainly responsible for generating scheduling decisions based on the scheduling matrix to minimize task waiting time with a guaranteed *TPS*. Finding the optimal scheduling decisions for multiple DNN tasks on multiple heterogeneous devices proved to be an NP-Hard problem. However, in the ***Sniper***, this scheduling problem can be obtained by the relaxation model P based on the generated scheduling matrix. Current algorithms to solve such problems include *Least Laxity First* (*LLF*) [21], heuristic algorithm [22], etc. To quickly obtain the optimal scheduling strategy, ***Sniper*** schedules the tasks using the *LLF* with extra limits, which ensures the maximum *TPS* while reducing waiting time.

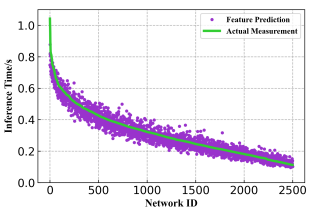
4 EXPERIMENT AND EVALUATION

4.1 Experiment Setup

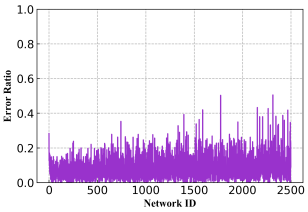
This experimental environment is based on a heterogeneous inference computing platform consisting of 10 nodes, including NVIDIA AGX, TX2, and NX. In the above devices, we uniformly installed some necessary runtime environments, such as Ubuntu 16.04, CUDA 10.2, CUDNN 7.5, PyTorch 1.7, etc., and executed DNN tasks with PyTorch as the base framework. The task set includes a variety of classification tasks and target detection tasks, where the backbone uses nine mainstream neural networks listed in Table. 2.

4.2 DNN Performance Prediction

This section focuses on testing the performance prediction error of ***PCN***. As described in Section 3.3, we use ***PCN*** to predict the model inference time on three heterogeneous devices.



(a) Comparison of predicted results



(b) Error ratio of prediction

Figure 5: Prediction results of *PCN* for random networks.

Figure 5(a) shows the prediction accuracy on a custom random network. Then, we sort the random networks according to their running times and assign network IDs in order. As shown in Figure 5(b), most predictions have a small relative error, with an average relative error of only 8.5%. For the prediction of small networks, most of the error values do not exceed 0.1s. Overall, the average error of the test set on multiple devices is kept at 0.024s.

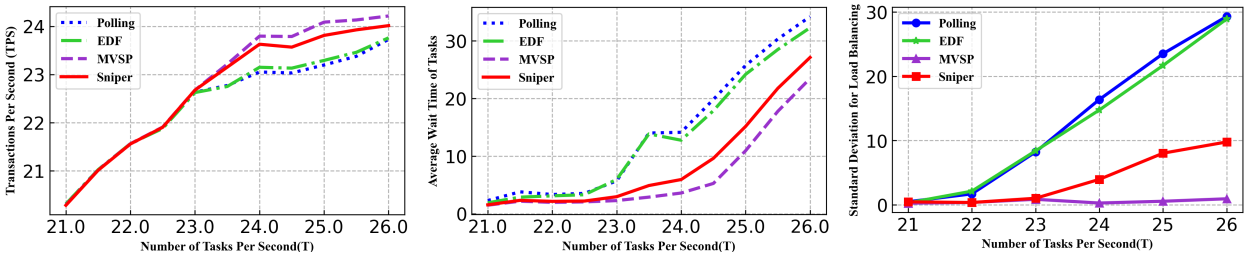
Immediately after, we apply ***PCN*** to actual DNN structures. The feasibility of the DNN performance characterization module is verified by empirically testing different structural networks. As shown in Table 2, the average relative error of actual network prediction is about 8.06%, which is in line with our expectations.

4.3 Schedulability Experiments

In the scheduling experiments, we evaluate our proposed scheduling system based on the computing platform described in Section 4.1. In the comparison experiment setup, we choose *Polling* and *EDF* mentioned in motivation as the benchmark for comparison, and reproduce the scheduling experiment (*MVSP*) in [6] as the performance ceiling for comparison, which uses the measured task

Table 2: Prediction results of common networks with multiple devices

MODEL	NVIDIA AGX			NVIDIA TX2			NVIDIA NX		
	Predict/s	Test/s	Error/%	Predict/s	Test/s	Error/%	Predict/s	Test/s	Error/%
DenseNet121	0.24711	0.22287	10.872	0.28707	0.27512	4.346	0.26656	0.27514	0.03119
DenseNet169	0.31186	0.28044	11.201	0.35396	0.34487	2.637	0.33236	0.34461	3.555
DenseNet201	0.36108	0.37569	3.888	0.40694	0.45189	9.947	0.38341	0.45133	15.049
ResNet152	0.61750	0.58672	5.246	0.71304	0.71271	0.046	0.66405	0.71203	6.739
ResNet101	0.40799	0.41183	0.933	0.47122	0.49633	5.061	0.43879	0.49633	11.594
ResNet50	0.23611	0.24487	3.575	0.28147	0.29500	4.586	0.25820	0.29480	12.416
VGG16	0.35507	0.40665	12.684	0.49143	0.44764	9.782	0.42152	0.44754	5.813
VGG19	0.44508	0.48078	7.426	0.61601	0.53073	16.067	0.52838	0.53335	0.931
InceptionV3	0.37680	0.32652	15.399	0.52150	0.59952	13.014	0.44732	0.36740	21.753
Average Error			7.914			7.276			8.997



(a) Transactions Per Second of Task (b) Average Waiting of Task (c) Standard Deviations of Nodes Load

Figure 6: The scheduling results of Polling, EDF, MVSP and *Sniper*.

runtime as the performance perception. (**without considering the real measured cost of the DNN tasks**)

This experiment sets the *TPS* and the *Average Wait Time* (*AWT*) as the observed quantities for evaluating the scheduling performance in the different number of tasks per second (T). The *AWT* can be used as a quantitative indicator of the QoS. The scheduling results of ***Sniper*** and three comparison algorithms are shown in Figure 6.

Obviously, Figure 6(a) and 6(b) demonstrate that compared to *Polling* and *EDF*, ***Sniper*** resulted in an AWT decrease by more than 52% and a TPS increase by more than 2.2% ($T = 24$).

Compared with *MVSP* that uses the real measurement data, the *TPS* of ***Sniper*** is the same at $T \leq 23$, which are both 22.67. Even in the case of the most enormous *TPS* gap (T=25), it only reaches 0.27, which is still 0.6 higher than the other algorithms. As for the analysis of the *AWT* of tasks, the ***Sniper*** significantly outperforms the *Polling* and the *EDF* scheduling and does not differ much from the *MVSP* in load range ($T \leq 23$). Even under near-load conditions (T=24), the *AWT* of ***Sniper*** is only 46% of *EDF* scheduling.

Next, we analyzed the load balancing between the algorithms for the different T. As shown in Figure 6(c), ***Sniper*** achieves the same performance as MVSP in terms of load balancing in the load range ($T \leq 23$); although the load of ***Sniper*** increases significantly with the increase of T, it is still better than other scheduling algorithms.

4.4 System Cost Analysis

The ***Sniper*** requires PCN pretraining before being used, which is consisted of the training set generating and ***PCN*** training. As shown in Table 3, the main cost comes from the training set generating, which can be linearly scaled down by increasing devices. In system application, the Time Spinner takes 10s to adjust parameters in batches for tasks accumulated, and the ***PCN*** takes 60s for retraining. Both the pre-training and Feedback learning are executed when device is free and do not increase the scheduling latency.

Table 3: Time cost of sniper

Phase		AGX	TX2	NX
Pre-training	**Training set generating**	0.9hr	1.7hr	1.3hr
	PCN training	120s		
Feedback learning	**PCN retraining**	60s		
	Spinner training	10s		
Per schedule		0.05s		

The system time cost of the scheduling phase is divided into three main parts, including inference time acquisition, scheduler computation, data transfer (DNNs and dataset). Among them, the difference in cost between the two methods is mainly caused by the different ways of obtaining inference time because the remaining are nearly identical. In the cloud-edge collaborative inference scenario, the cost of the time-acquisition method in ***Sniper*** is significantly lower than that in *MVSP* due to the following reasons.

- Larger DNNs will bring a more significant time cost to *MVSP*, while the time cost of ***Sniper*** is always a small constant. For example, in the third column of Table 2, a real measurement of *DenseNet121* takes 0.223s, while the larger *ResNet152* takes 0.587s. However, the time cost of ***Sniper*** is stable at around 0.05s for any DNNs.
- When a great number of DNNs are updated and iterated simultaneously, *MVSP* has to send each DNN to the edge devices to measure the inference time. Due to resource constraints, most edge devices must run these DNNs serially,

while ***Sniper*** can quickly obtain the inference time of all DNNs in parallel with the cloud computing power.

- In the scenario of multiple heterogeneous devices, *MVSP* cannot make scheduling decisions until the DNN inference time for all the devices is obtained. Therefore, the cost of *MVSP* depends on the straggler with the worst performance. In contrast, the cost of ***Sniper*** is constant.

5 RELATED WORKS

Performance Characterization for DNN. Recent research [23] has stressed the necessity of performance characterization for DNN training and inference acceleration. The work in [11] conducts a quantitative analysis of the performance of various scheduling techniques. Additionally, the cost model in [10] contains both static indications (e.g., kernel launches and FLOPs) and dynamic metrics that require specific hardware measurements. Further, several existing studies [4, 5] report on the performance of DNN by seeing the results on a test set, which results in an unnecessary waste of computing resources and accuracy. Unfortunately, accurate prediction of performance is tricky because of the absence of a standard DNN characterization approach in the preceding work.

Cloud-Edge Collaborative Inference. The standard methods [24, 25] for cloud-edge collaborative inference is to divide the DNN into two parts that operate in the cloud and at the edge. Another method [26] uses the intermediate layer's confidence level to determine whether the remaining layers should be executed in the cloud or at the edge. In terms of scheduling, a great number of studies optimize scheduling performance by utilizing real running inference time[6, 7]. Recently, inference scheduling in the cloud has been extensively investigated in [4]. However, the same enough computer resources are not available for edge devices [18].

From all these previous works, the unique contribution of ***Sniper*** is to explore the performance characterization similarity of DNNs through introducing the ***NNS*** concept, which allows the system to determine the inference time intelligently.

6 CONCLUSION

This paper presented ***Sniper***, a cloud-edge collaborative inference scheduling system that supports rapid iterative DNNs. In ***Sniper***, we introduced the concept of ***NNS***, the non-invasive ***PCN***, and the complete network training, retraining, and fine-tuning mechanism. The experimental results demonstrated that the proposed system predicts the network inference time with an average relative error of 8.06%. It successfully reduced the waiting time in DNN scheduling by 52% and improved a stable *TPS* increase. Overall, this works had significant improvement for the QoS and efficiency of cloud-edge collaborative inference. In future work, we will consider combinations with more scheduling algorithms and hardware resource modeling methods.

7 ACKNOWLEDGMENT

This work was partially supported by the National Natural Science Foundation of China (No. 62102390, 62102179), the National Science Youth Fund of Jiangsu Province (No. BK20190224, BK20200462) and the OPPO Research Fund.

REFERENCES

[1] Min Li, Yu Li, and et al. Appealnet: An efficient and highly-accurate edge/cloud collaborative architecture for DNN inference. In *58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco*, pages 409–414. IEEE, 2021.
[2] Linux Foundation. Edgexfoundry. Website, 2018. https://www.edgexfoundry.org.
[3] The KubeEdge SIG AI. Sedna. Website, 2020. https://github.com/kubeedge/sedna.
[4] Yecheng Xiang and Hyoseung Kim. Pipelined data-parallel CPU/GPU scheduling for multi-dnn real-time inference. In *IEEE Real-Time Systems Symposium, RTSS 2019, Hong Kong, SAR, China, December 3-6, 2019*, pages 392–405. IEEE, 2019.
[5] Liekang Zeng, Xu Chen, Zhi Zhou, Lei Yang, and Junshan Zhang. Coedge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices. *IEEE/ACM Trans. Netw.*, 29(2):595–608, 2021.
[6] Mounir Bensalem, Jasenka Dizdarevic, and Admela Jukan. Modeling of deep neural network (DNN) placement and inference in edge computing. In *IEEE International Conference on Communications Workshops*, pages 1–6. IEEE, 2020.
[7] Xiaorui Wu, Hong Xu, and Yi Wang. Irina: Accelerating DNN inference with efficient online scheduling. In *APNet '20: 4th Asia-Pacific Workshop on Networking, Seoul, Korea, 3-4 August, 2020*, pages 36–43. ACM, 2020.
[8] Daniel Casini, Alessandro Biondi, and Giorgio Carlo Buttazzo. Task splitting and load balancing of dynamic real-time workloads for semi-partitioned EDF. *IEEE Trans. Computers*, 70(12):2168–2181, 2021.
[9] Shuai Liu, Zidong Wang, Guoliang Wei, and Maozhen Li. Distributed set-membership filtering for multirate systems under the round-robin scheduling over sensor networks. *IEEE Trans. Cybern.*, 50(5):1910–1920, 2020.
[10] Zhihao Jia, James Thomas, and et al. Optimizing dnn computation with relaxed graph substitutions. In *Proceedings of Machine Learning and Systems*, volume 1, pages 27–39, 2019.
[11] Zhihao Jia and Sina aand et al. Lin. Exploring hidden dimensions in accelerating convolutional neural networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 2274–2283, 10–15 Jul 2018.
[12] Li Lyna Zhang, Shihao Han, and et al. nn-meter: towards accurate latency prediction of deep-learning model inference on diverse edge devices. In *MobiSys '21: The 19th Annual International Conference on Mobile Systems, Applications, and Services, Virtual Event, Wisconsin, USA*, pages 81–93. ACM, 2021.
[13] Qing Qin, Jie Ren, and et al. To compress, or not to compress: Characterizing deep learning model compression for embedded inference. In *2018 IEEE Intl Conf on Parallel Distributed Processing with Applications*, pages 729–736, 2018.
[14] Adrián Csiszárik, Péter Korösi-Szabó, and et al. Similarity and matching of neural network representations. *CoRR*, abs/2110.14633, 2021.
[15] Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. A survey of the usages of deep learning for natural language processing. *IEEE Trans. Neural Networks Learn. Syst.*, 32(2):604–624, 2021.
[16] Atefeh Shahroudnejad. A survey on understanding, visualizations, and explanation of deep neural networks. *CoRR*, abs/2102.01792, 2021.
[17] Jia Zhao, Kun Yang, and et al. A heuristic clustering-based task deployment approach for load balancing using bayes theorem in cloud environment. *IEEE Trans. Parallel Distributed Syst.*, 27(2):305–316, 2016.
[18] Shaojun Zhang and et al. Dybatch: Efficient batching and fair scheduling for deep learning inference on time-sharing devices. In *20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing*, pages 609–618, 2020.
[19] Lei Yang, Zheyu Yan, and et al. Co-exploration of neural architectures and heterogeneous ASIC accelerator designs targeting multiple tasks. In *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*, pages 1–6. IEEE, 2020.
[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, and et al. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, pages 5998–6008, 2017.
[21] M. Dominic and Bijendra N. Jain. Conditions for on-line scheduling of hard real-time tasks on multiprocessors. *J. Parallel Distributed Comput.*, 55(1):121–137, 1998.
[22] Najme Mansouri, Behnam Mohammad Hasani Zade, and Mohammad Masoud Javidi. Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory. *Comput. Ind. Eng.*, 130:597–633, 2019.
[23] Ananda Samajdar, Jan Moritz Joseph, and et al. A systematic methodology for characterizing scalability of DNN accelerators using scale-sim. In *IEEE ISPASS*, pages 58–68. IEEE, 2020.
[24] Marion Sbai, Muhamad Risqi U. Saputra, and et al. Cut, distil and encode (CDE): split cloud-edge deep inference. In *18th Annual IEEE International Conference on Sensing, Communication, and Networking, SECON*, pages 1–9. IEEE, 2021.
[25] Babak Zamirai, Salar Latifi, Pedram Zamirai, and Scott A. Mahlke. SIEVE: speculative inference on the edge with versatile exportation. In *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA*, pages 1–6. IEEE, 2020.
[26] Roberto Gonçalves Pacheco, Rodrigo S. Couto, and et al. Calibration-aided edge inference offloading via adaptive model partitioning of deep neural networks. In *ICC 2021 - IEEE International Conference on Communications, Montreal*, pages 1–6. IEEE, 2021.