Portable Fast Platform-Aware Neural Architecture Search for Edge/Mobile Computing AI Applications

Kuo-Teng Ding*, Hui-Shan Chen*, Yi-Lun Pan*, Hung-Hsin Chen†, Yuan-Ching Lin‡ and Shih-Hao Hung*
*High Performance Computing Division, National Center for High-performance Computing
*Hsinchu, Taiwan
*email: {tony.ding,chwhs,serenapan,2003002}@narlabs.org.tw
†Department of Computer Science and Information Engineering, National Taiwan University
†Taipei, Taiwan
†email: r09922038@csie.ntu.edu.tw
‡Department of Computer Science, National Tsing Hua University
‡Hsinchu, Taiwan
‡email: s105062329@m105.nthu.edu.tw

***Abstract*—The recent rise and progress of neural network-based artificial intelligence are obvious, and we have settled up many traditional machine learning problems by deep learning. However, problems were encountered when deploying neural networks on diverse hardware platforms, which needs lots of computational capability and time to "try out" the best architecture tipping the balance of model accuracy and execution latency. The proposed Portable Fast Platform-Aware Neural Architecture Search (PFP-NAS) system allows users to use the trained neural network model easily without considering the hardware architecture of the edge/mobile computing on the client-side. The portable neural architecture search device shrinks the data center and converts it to allow users to utilize it on-demand and dynamically. This just-in-time, secure, and portable neural architecture search method is mainly based on the platform-aware client-side and applying the neural network model trained on the data center. Another primary thing to remember is that users use the expandable modules of this device-Performance Prediction Module and Client Requirement-oriented Module, i.e., Accuracy, Latency, Throughput floating point operations per second (FLOPs), mean Average Precision (mAP), Cost, etc. and then the device can detect hardware architectures, such as Development Kit/Tensor Processing Unit (TPU)/Graphics processing unit (GPU)/Field-programmable gate array (FPGA) on edge/mobile computing. When detected, the device will send signals to connect the data center and drive the trained model in the data center for the corresponding hardware architecture. The proposed technique has the following characteristics: a. Only a boot medium is needed to detect and determine the hardware and then get the most suitable neural network from the server; b. Provide performance prediction module and client requirement-oriented module; c. Automatically match the model and the corresponding hardware architecture; d. Designed with modular scalability, and there is no need to configure any settings on the client-side. Consequently, the proposed framework achieves a portable data center.**

***Keywords*—Portable; Neural Architecture Search; Performance Prediction.**

I. INTRODUCTION

Neural architecture search (NAS) issues are attracting more and more attention. Still, researchers try to cut into neural network search methods with its extreme computing cost and specific decentralized acceleration of neural network search.

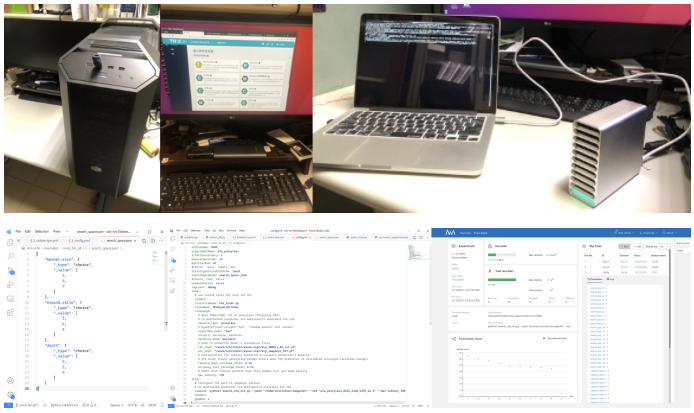

Fig. 1. The demo of PFP-NAS

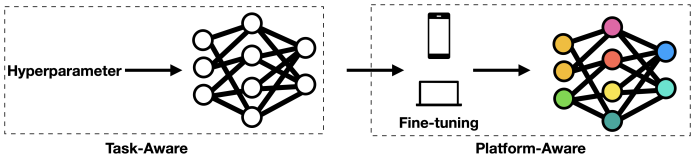

Fig. 2. Task-Aware Model Recommend and Platform-Aware Model Training

Acceleration methods from different aspects, such as Mathematical Model, Empirical Rule, and other methods have been proposed. However, these methods are often configured for special tasks and require fine-tuning and actual experiment feedback on the hardware platform. As we all know, it requires powerful computing capabilities and repeated training for the platform to get a good-quality model. This research places high hopes to solve the practical problems of insufficient computing power and data protection requirements.

Fig. 1 shows the simple demo of PFP-NAS. Users can install the NAS Agent program with a Universal Serial Bus (USB) device or download it from the internet and connect to its own target device. Users can also manipulate the PFP-NAS flow

 ISBN: 978-1-61208-894-5

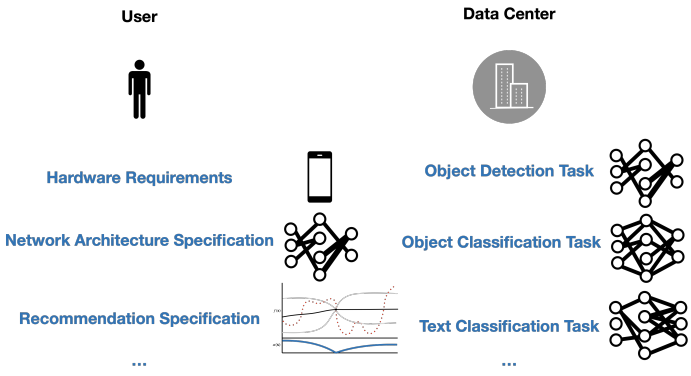

Fig. 3. Pre-trained Task-Aware Model Recommend for different tasks

with a modified Microsoft NNI user interface. The portable accelerated NAS service is proposed, which aims to split NAS into two important parts: the Task-Aware Model Recommend and the Platform-Aware Model Training, as shown in Fig. 2. The concept of pre-training is applied to model search in this architecture. The data center can learn in advance and discover the best structure of task-aware model. Moreover, the type of hardware architecture in the client-side can be paired with the recommended model in the data center to filter and select a suitable Platform-Aware model.

Fig. 3 shows that the data center will pre-train the model in advance and the pre-trained model will vary against different tasks. The pre-trained model can provide users with a hyperparameter recommendation set about a specific task. The user will need to provide platform-aware results (model accuracy and execution latency) to the data center, helping the whole hyperparameter search meet the requirements (shown in Fig. 4). In such a scenario, the client takes charge of network fine-tuning, and the data center recommends the hyperparameter. The communication process will take a few loops, and finally, the outcome is a slim and efficient model. Users can easily acquire the trained NAS model without considering their own hardware architecture on edge/mobile computing and without having to provide their own data sets. Through the designed portable accelerated neural network search device, the powerful computing power of the data center can be shrunk and converted into a dynamic and easy use for the end-users. Therefore, our design is the PFP-NAS framework to tackle the above problems.

To effectively select a suitable Platform-Aware model, the PFP-NAS can provide the Just-In-Time Performance module and the Platform-oriented module to the data center. These two modules can instantly evaluate the expected performance of the selected model and detect the user's hardware architecture. After detecting edge/mobile computing hardware architecture, such as TPU/GPU/FPGA, the user device will send relevant signals to contact the data center and drive the corresponding edge trained in the data center for mobile computing. Therefore, this research will design and provide the following three aspects of use scenarios and case solutions:

- Under the circumstance, when the client-side provides a personal dataset, the PFP-NAS only needs the module's description without the users' module. Then PFP-NAS can easily provide forecasting of the time-consuming and accuracy.
- When the client-side does not provide any personal dataset, PFP-NAS can provide a platform-aware recommendation service for protecting the client-side's privacy. The Just-In-Time Performance Predict Module and Platform-oriented module designed in this project can be used. The server side of the portable accelerated neural network search will be based on the existing information of the model and the user through the high-level description. The language interacts, and then the candidate models are screened out. The remaining part is to carry out portable training on the user device, allowing the user device to return the reward value or weight to adjust the recommendation logic.

The core concept of PFP-NAS is through portable accelerated NAS device services and NAS Agent to bring the powerful computing power of the data center to the user (Portable). Therefore, the following contributions are shown:

- Make computing resources portable: Through the designed PFP-NAS framework, the data center with powerful computing capability can be reduced and converted into a portable application service for users.
- Make dataset protective: It does not require users to upload datasets but allows users to interact and describe through high-level language to generate similar data sets, providing complete protection of user datasets.
- Make models speedup and optimized: The PFP-NAS framework proposed in this research will use powerful computing resources to automatically find and try every possible model in advance so that each model can be optimized to make full use of the back-end computing resources and provide them to the front-end good user experience.
- Make recommended models in real-time: With the benefit of the Just-In-Time Performance Predict Module, it can dynamically provide models that meet user needs.
- Make platform-as-a-service scalable: This research framework can bridge data centers (such as TWCC and AWS) to create tens of millions of portable computing power so that these data centers can provide accelerated NAS services.

The remaining of the paper is structured as follows: Section 2 presents backgrounds and related works. Section 3 presents the whole system design and system components. Section 4 indicates the experiement results around the system. Section 5 concludes the works and future works.

II. Related Works

This section will cover the related works ranging from automated machine learning and neural architecture search to hyperparameter tuning. The implementation of this work

 ISBN: 978-1-61208-894-5

is based on Microsoft NNI opensource project which will be mentioned in the automated machine learning part. The proposed framework is mainly rose from the concept of Once-for-All network [1] and will be described in the neural architecture search part.

A. Automated Machine Learning (AutoML)

Automated machine learning [2] provides machine learning technology that non-experts in this domain can also get started quickly, such as data preprocessing, feature engineering, hyperparameter optimization and model post-processing, because the complexity of these tasks often exceeds the knowledge of non-experts in the domain of machine learning. The development and research of traditional machine learning models requires a lot of resources and cost. Therefore, AutoML can greatly improve the efficiency of machine learning and promote machine learning research. The following introduces and analyzes three AutoML technology:

a) MLflow (Machine Learning Workflow): MLflow [3] is an open-source project, which is created by the Apache Spark technical team. It is a platform to manage machine learning lifecycle. It can support a lot of existing machine learning applications and libraries. The main components of MLflow platform are MLflow Tracking-log and compare parameters, code, and experimental data, MLflow Projects-package training models for reproducible runs using Conda and Docker, MLflow Models-share and deploy the same training model on different platforms, and MLflow Model Registry-manage the full life cycle of MLflow models. MLflow UI adopts Flask's Web application framework to show visualization experimental results. In addition, in terms of model training alone, MLflow supports a wide variety of tools, including scikit-learn, PyTorch, Spark, TensorFlow, R, etc.; nevertheless, MLflow cannot perfectly solve model incompatibility problems caused by using different tools.

b) Microsoft NNI (Neural Network Intelligence): Microsoft NNI [4] is a lightweight toolkit for AutoML, which is released by Microsoft, but it has powerful functions and is easy to operate. It is one of the prevalent Automatic Machine Learning (AutoML) open-source tools, which can effectively help users to automatically tune and optimize the neural network architecture of the machine learning model. Its features include hyperparameter tuning, neural network architecture search, model compression, feature engineering, and provides many general-purpose NAS frameworks.

However, hyperparameter tuning and optimization is the core and basic function of Microsoft NNI. It provides a lot of popular auto-tuning algorithms (Tuner) to adapt to the hyperparameter tuning of different machine learning and deep learning applications; it also provides early stop algorithms (Accessor). It is used to predict and evaluate that if each trial's performance is not as good as the expected value, the trial will be terminated early. In this study, the research team integrated the proposed algorithms to efficiently and robustly search hyperparameters and NAS models in AutoML framework.

Based on the above discussion, only Microsoft NNI provides complete hyperparameter tuning and NAS technologies and functions, therefore, Microsoft NNI is introduced as the underlying infrastructure design for further development in this research.

B. Neural Architecture Search (NAS)

Neural network training has evolved from the original handmade network to data extraction, and has developed into automated training and getting a suitable network structure with the least resources. Furthermore, Google published the paper - Neural Architecture Search with Reinforcement Learning. They used reinforcement learning for NAS and surpassed the previously hand-made network in image classification and language modeling tasks. At present, the effect of NAS has been comparable to the state-of-the-art model structure. Moreover, Nas-Bench-101 [5] emerged to evaluate the performance of NAS objectively in the fields of semantic segmentation, speech recognition, object detection, object classification, data enhancement, etc.

However, there are still some problems with NAS, such as the inability to find an objective method to compare NAS effects and reuse NAS results efficiently because various NAS methods have various different ways in search space and hyperparameter optimization. For example, for the hardware-aware NAS problem, the search space may be configured with different hardware specifications. Still, these found structures cannot be easily used for conversion learning on different tasks or on tasks with different datasets.

The traditional hand-crafted methods or NAS methods often require a lot of GPU resources. Han Cai1's lab published Once-for-All [1] and proposed the "Model Shrinking" method. It trains a large model while also training a small model, thereby reducing the cost of repeated training for the same type of neural network with similar architectures in order to select the best model. Model Shrinking will gradually reduce the scale of the trained neural network, and finally perform Fine-Tuning for each trained model. This paper proposes the neural network search strategy used in Model Shrinking, including Resolution, Kernel Size, Depth and Width in the convolutional neural network structure.

C. Hyperparameter Tuning

With the continuous expansion of the importance and application of machine learning tools, so does the demand for non-experts to use AutoML tools. Hyperparameter tuning is a common problem in many machine learning tasks, whether supervised or unsupervised. Some research has provided several different useful ways to solve these problems, such as cross-validation and excessive scoring functions. However, the scholars [6] discuss that some appropriate automated machine learning framework would include the automation of modeling and hyperparameters search which typically use black-box gradient-free optimization.

In addition, the other researchers [7], [8] focus on the information about the hyperparameter combination from the

 ISBN: 978-1-61208-894-5

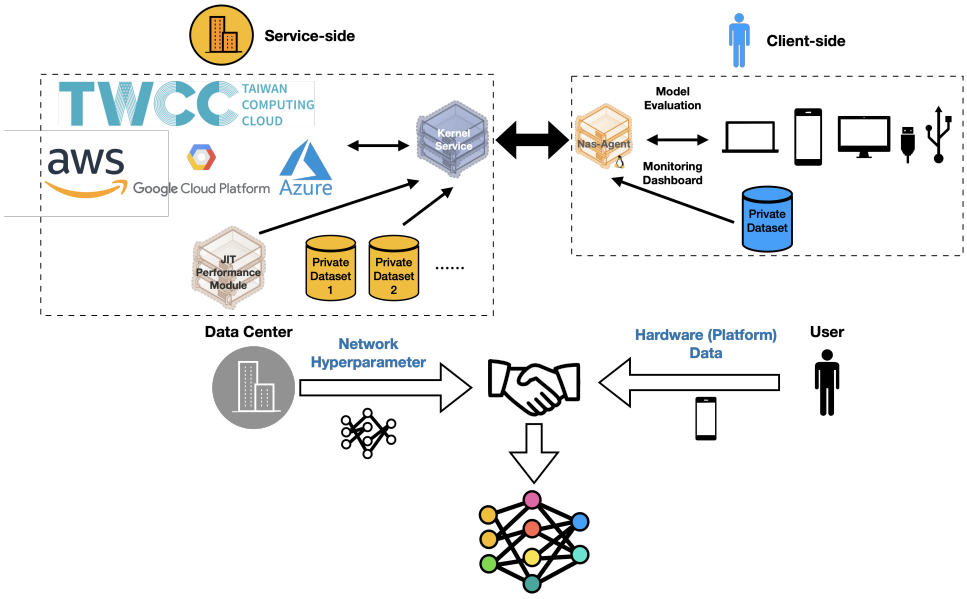

Fig. 4. Conceptual System Architecture Diagram

function f:X→R, and then output the data-generated model through the Bayesian optimization framework. The above method is also an effective method to search the hyperparameters. In AutoML setup, the f function may represent some hyperparameter definitions and the metric of the model that matches the given data, such as cross-validation residual, likelihood function, or risk function.

In addition, Bayesian optimization can be used to simultaneously tune multiple aspects of the machine learning model, such as data preprocessing and model hyperparameters [9]. Bayesian optimization can be utilized in various models, including Gaussian process (GP) [10], random forests (RF) [11], and tree-structured Parzen estimator (TPE); each of these models has its strengths. In this study, our team used two open-source hyperparameters tuning tools, Hyperopt [12] and Optunity [13], which are most widely used recently. The main purpose of applying the two above tools is to efficiently and robustly search hyperparameters in an AutoML framework.

III. System Architecture and Implementation

The architecture of PFP-NAS can be divided into several modules, which are the Just-in-Time Performance Prediction Module, Once-for-All Pre-training Module, Network Architecture Search Module, Network Architecture Searching Agent Module, and Kernel Service Module.

The Fig. 4 is the conceptual system architecture diagram. The main concept is to use the NAS Agent embedded on the client-side hardware platform. The NAS agent is a middleware between the datacenter and users, and is responsible for communicating with the kernel service module. The data center leverages its own powerful computing resources with the Once-for-All Pre-training Module; the Just-in-Time Performance Prediction Module will automatically suit the client-side hardware, enabling a customized NAS task host on the data center (server-side). A recommended model can be generated for the client-side hardware platform, such as USB/Tensor Processing Unit (TPU)/GPU/FPGA, but only needs a mere little amount of computing power. Users do not need to concern about dataset leaking.

When users actually use the PFP-NAS service, they only need to provide specific requirements and specifications. PFP-NAS will select the appropriate Just-in-Time performance prediction module and the appropriate pre-trained Once-for-All network to recommend the network structure.

Users use private data sets to test the recommended network on the hardware architecture of the client-side and then report the actual performance results to the server, as bottom left of Fig. 5 is shown. The server will revise the original network and further recommend the new network based on these results. Since there is already a pre-trained Once-for-All network, fine-tuning for different hardware is saved every time performing a platform-aware NAS for a specific large-scale network structure. It can save a lot of time because the user does not need to retrain the network on the client-side. The network scale of the Just-in-Time Performance Prediction Module is small, so the revision will be completed in a few seconds, and the calculation of the online model recommendation is fast and real-time. This research organizes the entire PFP-NAS training process and the interaction between the user and the server as shown in Fig. 9. Therefore, the procedures of the designed algorithm are shown as the following:

- Step 1. The user on the client-side submits the description file specifing the execution parameters and resouce requirements through web-based interface or Restful API. Kernel Service will perform data integration and create a new Python program execution task. During the execution

 ISBN: 978-1-61208-894-5

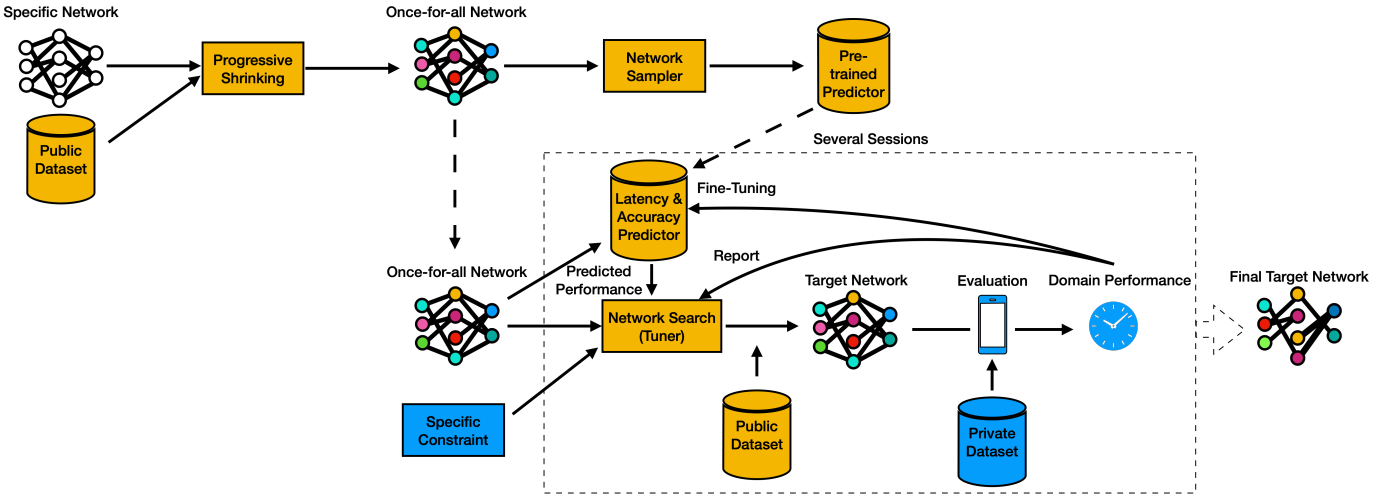

Fig. 5. The Interaction between the Once-for-All Pre-training Module, JIT Performance Predict Module and Client-side NAS Agent

of the Kernel Service, the related Input (Dataset, Criteria: Accuracy, Latency, Throughput, etc.) will be stored in MariaDB.

- Step 2. The python program and input parameters are sent to the NAS Module for execution, and the combinations of recommended hyperparameter of dedicated neural network are dynamically generated. The hyperparamether combinations will be stored in MariaDB performed by Kernel Service Module and visualized with a web-based interface which the client can browse on their device.
- Step 3. NAS Agent will gather performance metrics of hardware and the model inference result on the user platform, feedback to the server.
- Step 4. Kernel Service Module receives the user domain information sent by NAS Agent. The Just-in-Time Performance Prediction Module consists of the two three-layer multilayer perceptron neural networks and a joint score algorithm (shown in Alg. 1), and it will be automatically updated according to model inference latency and accuracy under user platform with private dataset. A new NAS task will be performed by Kernel Service following the prediction.
- Step 5: Repeat steps 2 to 4.
- Step 6. Generate an optimized candidate neural network model (NN) and send it back to the user.
- Step 7. (Optional) The client can effectively perform a fine-tuning with a recommended model with few epochs because the recommended model already has high-accuracy and low-latency.

We will dedicate each module of our PFP-NAS below.

A. Just-In-Time (JIT) Performance Prediction Module

JIT Performance Prediction Module comprises two 3-layer multilayer perceptron (MLP) prediction networks, divided into latency and accuracy prediction networks. PFP-NAS will train a set of specialized performances for each specific task's Once-for-All network and then predict the users' hardware status.

When each online mission of PFP-NAS is performed, the user will actually test the received recommended network structure on the user's hardware platform. Because there is no backward propagation process but only inference process, we can quickly get the actual performance results when the user tests the network with the private data set on the hardware platform. PFP-NAS will carry out the backward propagation of the prediction network based on the user domain results. Because of the networks in JIT Performance Prediction Module is relatively small, it only takes a few seconds to update the prediction network to obtain high accuracy during actual operation.

To acquire the base unit for latency and accuracy magnitude, we take both magnitudes into one base quantity - score (shown in Alg. 1; As we want to avoid a high latency model primarily, we will put a higher penalty coefficient to latency.

JIT Performance Predict Module will feed the score back to Kernel Service, and tune NAS modules.

B. Once-for-All Pre-training Module

The main purpose of the Once-for-All network is to deploy directly under different hardware restrictions without retraining and re-searching. Only the Lookup Table method is needed to determine the network architecture. It can flexibly deal with different depths, widths, kernel sizes, and resolutions, as shown in the Fig. 6 below (Once-for-All). In this way, continuous re-searching and continuous retraining can be successfully avoided. The problem that the NAS algorithm cannot handle the diverse hardware environment in the past can also be solved. Furthermore, the core service will interact with JIT Performance Prediction Module. When the Once-for-All module generates a new subnet architecture, it will be one-hot encoded and fed to the JIT Performance Prediction Module to obtain the delay and accuracy. Then NAS module uses this

 ISBN: 978-1-61208-894-5

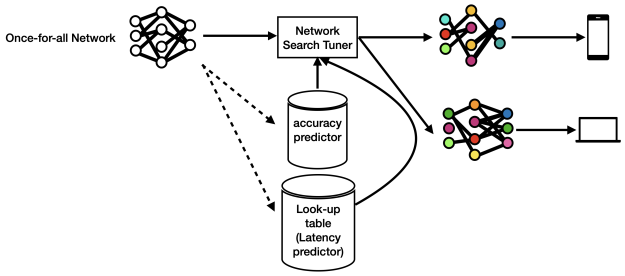

Fig. 6. The Original Once-for-All Training Flow

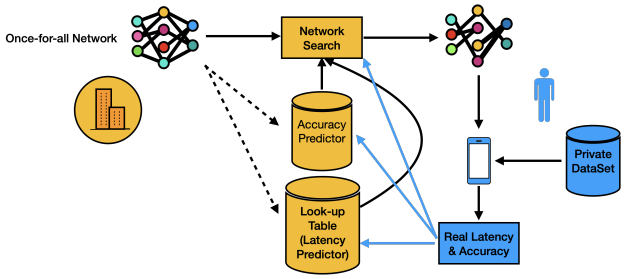

Fig. 7. The Proposed Once-for-All Module Flow

set of results to perform parameter recommendation and then hand it over to the Once-for-All module for model training and search of the next round. The implementation of the Once-for-All module is based on MobileNet V2, which mainly implements 5 large blocks and 1 small block, and the depth of each large block is 2-4 layers, as shown in the following Fig. 7. The user-side only needs to evaluate (not fine-tuning) the network and feedback the performance measure.

After integrating the Once-for-All Module into the system, the process architecture provided is shown in the right top of Fig. 5. Above all, we use many open data sets to train the Once-for-All network for a specific model architecture in the data center and pre-train a network of JIT Performance Prediction Module suitable for different hardware platforms. Considering the size of the open data set and the resources required to train a Once-for-All network, such a process requires a lot of time and computing resources. We will distribute these tasks to the computing center for preprocessing.

C. Neural Architecture Search (NAS) Module

This module can be easily replaced with any hardware-measurement-based NAS, such as ProxylessNAS, ENAS, and reinforce-learning-based NAS. The whole process of model searching will automatically adapt the target according to model performance (accuracy) and latency on hardware. In this article, we use ProxylessNAS for the default network searching algorithm, the ProxylessNAS adopts a method for making accuracy and latency magnitude differentiable, which is the loss function of a network model, and thereby use gradient descent to optimize the network searching model for finding a better network architecture.

D. Neural Architecture Search (NAS) Agent Module

NAS Agent Module is a client daemon for preliminary and simplified detection of hardware information on the client-side. At the same time, it collects the non-confidential performance metrics of testing results and feeds them back to the Server for prediction and optimization in the next cycle. The main purpose of NAS Agent is to allow users to communicate with the server easily. Users acquire the NAS model trained by the Server without considering their own hardware architecture on edge/mobile computing and without providing their own data sets. The entire data center is transformed to allow users to use it dynamically and easily through a portable fast NAS device. The device (NAS Agent) of the client-side receives the candidate models generated by the Kernel Service (Portable Fast Platform-Aware NAS) of the server-site, and these models can be tested on the client-side. Then, the following steps are performed on the client-side:

- Step1. Run the candidate model generated by the kernel service (Portable Fast Platform-Aware NAS) of the Server Site , and process the models of AI Framework in .prototext and .caffemodel formats;
- Step2. Execute Model Optimizer to output .xml and .bin formats;
- Step3. Inference Engine interacts with User Application and collects the information of hardware architecture;
- Step4. Automatically detect the hardware architecture running the inference job and send it back to the server-site.

E. Kernel Service Module

The Kernel Service Module provides the HTTP service with Restful API, the web-based interface, and task scheduling. The components of Kernel Service are introduced below.

1) Dispatcher: Dispatcher is responsible for the task scheduling of Tuner and Assessor, which are parts of NAS Module, and the generation of configuration files for each trial, including hyperparameter combinations and specific neural network architecture. During the training process, it analyzes the intermediate result value of each trial, and evaluate whether it should be terminated early.

2) NNI Manager: NNI Manager is responsible for each training experiment. With interaction with NAS Module, it can find the best hyperparameter combination and the best neural network architecture for the training model through experiments.

3) DBMS: The record of Once-for-All Pre-training Module and user domain information of each feedback with trail ID will be stored in MariaDB.

4) Training Service: The Training Service will gather and synchronize parameters along with related programs with the dispatcher, and perform the training task. The NAS Module mentioned above will automatically incorporate the Training Service as the computing resource for training.

Users can use built-in training services provided by Microsoft NNI to run trial jobs on the local machine, remote

 ISBN: 978-1-61208-894-5

Algorithm 1 The designed joint latency and accuracy score

Input: latency and accuracy
Output: score

```
if latency > LATENCY_THRES then
    score ← accuracy - (latency / LATENCY_THRES ) ∗ PENALTY_COEFF
else
    score ← accuracy
end if
```

Fig. 8. The designed joint latency and accuracy score

machines, and on clusters like PAI, Kubeflow, AdaptDL, FrameworkController, DLTS, and AML.

IV. Experiments

Our experiments mainly focus on the effectiveness of AI training with PFP-NAS on multiple hardware requirements. In this section, we test the effectiveness of PFP-NAS on various hardware requirements with parallel multiple computing nodes. Our experiments apply PFP-NAS with Imagenet [14] as public dataset on five workstations with Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz and test 6, 12, 24 NVidia GeForce GTX 1080 GPUs. The client task is Cifar-10 [15], and the pre-trained once-for-all network is based on mobilenet V2.

To prove the effectiveness of designed architecture, we test the PFP-NAS on the local workstation (the NAS agent and kernel service modules are hosted on the single workstation) and remote execution. We fix the hyperparameter search algorithm on annealing and set the same hardware requirements in this experiment. The results (Table I) show the top-1 and top-5 accuracy do not have much difference.

To verify the effectiveness of PFP-NAS under different hardware requirements, we set different latency, trial times, number of dispatched process in treatment of our experiments. We can see that the model performance will be better under the some treatment settings in Table II, and the same results (Table III) can be seen in treatment of hyperparameter search method.

We compared different hyperparameter tuning methodologies in PFP-NAS, such as Anneal, TPE, and Random search, and found that the result does not significantly differ; possible reasons might be the experiment network size are too small to have a large search space. Although the effectiveness of PFP-NAS is proven, the combination relationship with tuning methodologies still remains unclear, and the experiments of larger-scale neural network are needed.

V. Conclusions

In the era of artificial intelligence, while gaining convenience from advanced machine learning, deep learning, and

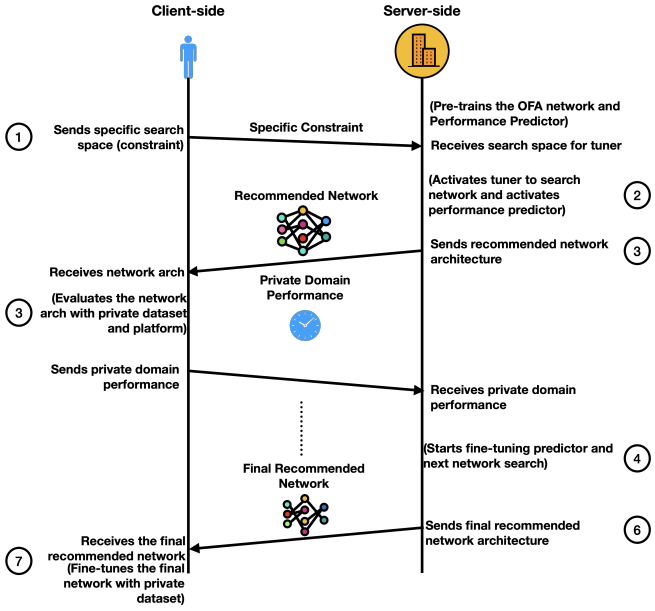

Fig. 9. The Model Recommendation Procedures

neural network search technology, it also faces data privacy issues and the risk of data leakage. When the user wants to keep its own private data set and does not provide it to the server, how the server can automatically determine the hardware architecture of the client-side and successfully recommend the NAS model becomes a tall order. The mechanism designed in this research solves this pain point. When the client-side data set and hardware architecture are confidential information and cannot be leaked, the kernel service of the server-side recommends a corresponding neural network model based on the preliminary information provided by the client-side. Then, the user actually runs the neural network model on its own hardware and feeds back the performance measurements to the server to update information and generate a new model.

The experiments around the different hardware settings show the effectiveness of our proposed PFP-NAS in the lightweight network, and further investigation about other network types and scales is ongoing. Also, through the designed PFP-NAS, the time-consuming training workout that originally required a lot of time is reduced to a few seconds for recommended models. For privacy, the PFP-NAS leverages the computing capability from the data center to help users create their own model but deals with the dataset privacy issue: users without powerful computing resources can utilize the cloud-based computing service and have no worries about the data leaking.

Next stage, we will leverage our proposed method into diversity experiments mainly to extend the Once-for-All module with other popular networks (not only the MobileNet V2), and apply the PFP-NAS to other application scenarios.

 ISBN: 978-1-61208-894-5

TABLE I
PFP-NAS PERFORMANCE IN SINGLE MACHINE AND MULTIPLE MACHINES

machine platform	dispatched process	tuner algorithm	psuedo trial	latency mape	avg_corr	max_latency	top-1 (%)	top-5(%)
local	2	anneal	0	0	1	85	94.22	94.16
local	4						94.39	94.18
controller	12						94.49	94.41

TABLE II
PFP-NAS PERFORMANCE IN DIFFERENT TUNER TRIAL REQUIREMENTS

dispatched process	tuner algorithm	psuedo trial	latency MAPE	avg_corr	max_latency	top-1(%)	top-5 (%)
12	anneal	0	0	1	85	94.49	94.41
		10	0.05	0.92		94.24	94.154
		150				94.47	94.418
		300				94.44	94.24
		1000				94.11	94.088
24		300				94.49	94.358
6					125	94.01	93.822
					165	93.96	93.886
					205	94.18	93.89
					245	94.08	94.036
					285	94.14	93.918

TABLE III
PFP-NAS PERFORMANCE IN DIFFERENCT HYPERPARAMETER SEARCHING METHODS.

dispatched process	tuner algorithm	psuedo trial	latency MAPE	avg_corr	max_latency	top-1 (%)	top-5 (%)
6	tpe	300	0.05	0.92	85	94.08	94.026
	random_search					93.99	93.762
	anneal					94.24	94.16

REFERENCES

[1] H. Cai, C. Gan, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," *CoRR*, vol. abs/1908.09791, 2019. [Online]. Available: http://arxiv.org/abs/1908.09791

[2] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015.

[3] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar, "Accelerating the machine learning lifecycle with mlflow," *IEEE Data Eng. Bull.*, vol. 41, pp. 39–45, 2018.

[4] Microsoft Corporation, "Neural network intelligence," https://github.com/microsoft/nni , accessed: 2021-07-19.

[5] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-bench-101: Towards reproducible neural architecture search," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 7105–7114. [Online]. Available: http://proceedings.mlr.press/v97/ying19a.html

[6] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms," *CoRR*, vol. abs/1208.3719, 2012. [Online]. Available: http://arxiv.org/abs/1208.3719

[7] W. Burgard, O. Brock, and C. Stachniss, *Active Policy Learning for Robot Planning and Exploration under Uncertainty*, 2008, pp. 321–328.

[8] E. Brochu, T. Brochu, and N. de Freitas, "A bayesian interactive optimization approach to procedural animation design," in *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '10. Goslar, DEU: Eurographics Association, 2010, p. 103–112.

[9] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 215–223. [Online]. Available: http://proceedings.mlr.press/v15/coates11a.html

[10] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012.

[11] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization*, C. A. C. Coello, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 507–523.

[12] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 1. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 115–123. [Online]. Available: http://proceedings.mlr.press/v28/bergstra13.html

[13] M. Claesen, J. Simm, D. Popovic, Y. Moreau, and B. D. Moor, "Easy hyperparameter search using optunity," 2014.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[15] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

 ISBN: 978-1-61208-894-5