Elastic DNN Inference with Unpredictable Exit in Edge Computing

Jiaming Huang, Yi Gao, and Wei Dong
College of Computer Science, Zhejiang University
Email: {huangjm, gaoy, dongw}@emnets.org

***Abstract*—Multi-exit neural networks have recently boomed in edge computing to maximize the computing power of different devices. However, many real-time tasks running on edge computing applications have encountered unpredictable exiting frequently due to system power outages, high-priority preemption, etc., which have been overlooked by multi-exit models until now. To tackle this issue, it is critical to decide at which branch the multi-exit model exits so that the unpredictable exit will always come with desirable results. In this paper, we propose EINet, a sample-wise planner of real-time multi-exit deep neural networks, which achieves efficient Elastic Inference with unpredictable exit while guaranteeing best-effort accuracy on different edge platforms. Therefore, a given trained deep neural network is first partitioned into multiple blocks with one exit each by EINet. Then EINet obtains the block-wise model profiles, including the block-wise accuracy and inference time. Using the model profiles, EINet is able to dynamically determine which exits to take during the inference task for each sample. We introduce Confidence Score Predictors to dynamically adapt the uniqueness of the input samples, and the Search Engine to efficiently find the near-optimal plan during the elastic inference. EINet is evaluated extensively using multiple DNNs and datasets with unpredictable exits. Results show that EINet can achieve the highest average accuracy compared with multiple baselines.**

***Index Terms*—Multi-exit, unpredictable exit, elastic inference, real-time DNN task, edge computing**

I. Introduction

In recent years, multi-exit neural networks (NNs) have flourished and emerged frequently in edge computing to better leverage and coordinate computing capabilities across devices, the edge, and the cloud [1]–[4]. They were first proposed in [5] which can produce intermediate outputs at early exit points to improve the efficiency of deep neural network (DNN) inference. Subsequently, this idea was widely applied to cloud-edge collaboration applications. More completely, [6] classifies dynamic inference on multi-exit NNs into instance-wise, spatial-wise, and temporal-wise. However, all existing work related to multi-exit models fails to take into account the issue of unpredictable exit.

In reality, multiple real-time DNN tasks running simultaneously in edge computing applications [7], [8] often encounter unpredictable exit due to system power outages, preemption by high-priority tasks (e.g., 5G vRAN [9]), specific user exit demands, etc. For example, Concordia [9] views the 5G vRAN tasks as high-priority tasks and allocates dedicated computation resources to them, while all other workloads will be preempted unpredictably. Surprisingly, the forced exit real-time inference tasks were overlooked for a long time.

In order to tackle the unpredictable exit issue encountered in practice, we want DNN tasks to always produce favorable results, no matter when being forced to exit. In the literature, many techniques have been proposed to improve the efficiency and accuracy of DNNs on edge platforms, such as model compression [10]–[13], lightweight model design [14]–[18], CPU/GPU scheduling [7], [8], [19]. More related to this work, instance-wise dynamic inference [5], [20]–[22] with multi-exit NNs is a potential technique to solve the unpredictable exit issue. The basic idea of this technique is to select an exit for each instance according to the predicted accuracy for each exit. However, these techniques are still facing the problem of forced exit before the inference finish. Orthogonal to the above techniques, we are the first to propose *Elastic Inference* based on multi-exit models. The elastic inference is time-insensitive which can make models generate desirable intermediate results until it is forced to exit unpredictably. To reach the above goal, it becomes critical for multi-exit NNs to decide when and at which branches to exit.

In this paper, we propose **EINet**, a sample-wise planner of real-time multi-exit NNs, which achieves efficient **E**lastic **I**nference instead of being forced to exit without any result while guaranteeing best-effort accuracy on different edge platforms. Unlike prior approaches, EINet treats the exit time as random and unpredictable and guides multi-exit NNs to dynamically select branches for different samples during the inference to achieve elastic inference. However, there are two challenges we need to address to realize EINet in practice.

First, how can our planner always guide the model to get desirable results before being stopped to meet real-time demand? To accommodate unpredictable exiting, multi-exit NNs are preferred to be fine-grained with more exits to cope with the possibility of interruption at any time. However, executing branches at each exit to preserve intermediate results may have a time overhead that prevents the inference from going deeper for better accuracy. To strike the balance between inference latency and accuracy, the planner needs to plan at which exits to execute branches based on known information. We present Search Engine in EINet which can find the near-optimal exit plan while inferring. The chosen plans will guide multi-exit NNs to skip (i.e. not execute) several exits in order to save time and achieve better accuracy.

Second, how can our planner be general to adapt to all models on various platforms for different input samples? To understand the characteristics of different models on various platforms, we present offline Block-wise Model Profiling to obtain inference time information. In addition to this, to better adapt to the features of the input samples, we propose to train Confidence Score Predictors (CS-Predictors) to enhance the interpretability of each round of inference. In short, CS-Predictors and model profiles will make EINet more general. Eventually, during the inference, general EINet will instantly update the exit plan according to different samples for the unpredictable exit.

Our main contributions can be summarized as follows:

- We present EINet, a sample-wise planner for efficient elastic inference. It can guide real-time AI tasks to run continuously and give desirable results when they are interrupted unpredictably instead of being killed by apps.
- We propose Block-wise Model Profiling to profile models on edge devices offline to understand their characteristics. Using model profiles, the CS-Predictors can be trained to adapt to the features of samples.
- We propose Search Engine to find the near-optimal exit plan online to balance the accuracy and latency. And with the help of trained CS-Predictors, EINet will continuously update the exit plans until being forced to exit.
- We implement EINet and conduct extensive experiments using MNIST, CIFAR-10, and CIFAR-100 datasets to examine the performance of elastic inference. Evaluation results show that for the same model on the same dataset, our framework can improve the overall accuracy compared to multiple baselines.

The rest of this paper is organized as follows. Section II presents the work related to the improve the efficiency of real-time DNN tasks in edge computing. Section III gives the overview of EINet. Section IV and Section V describe the design details of two stages in EINet. Section VI evaluates the performance of EINet and performs numerous validation experiments, and finally, Section VII concludes the paper.

II. Related Work

In this section, we will introduce existing works related to real-time AI tasks in edge computing scenarios. Many existing works in edge computing mainly focus on optimizing the inference time to make the task complete as quickly and accurately as possible to ensure real-time performance.

Model Compression. In order to get the inference result as soon as possible, model compression techniques are widely used to speed up model inference.

Many model compression techniques have been proposed. Large DNN models become lightweight by pruning [10] or quantization. Moreover, knowledge distillation [12], [13] can retrain a lightweight model from the original DNN model to achieve comparable accuracy. Instead of compressing large models, many lightweight models can also be designed directly, e.g., MobileNet [14], [15], ShuffleNet [16], [17] and CondenseNet [18]. While speeding up inference time, these models may suffer from a loss of accuracy.

Based on compressed models or designed lightweight models, many tasks can finish inference and output results shortly before the original inference time. However, there is still a large number of tasks that can not finish the inference and be forced to quit, these methods still cannot even output a result.

Model Partition and Scheduling. In real-time DNN tasks scenarios, to avoid the possible preemption problem of multiple task scenarios, many works [7], [8], [19], [23] performed model partitions on different levels and distribute them across multiple heterogeneous processors. Through reasonable model partition and scheduling, the efficiency of the entire system will be improved. However, reasonable and efficient scheduling does not work without knowing the actual inference time. There will still be some DNN tasks that cannot get the inference result caused by being forced to exit unpredictably.

Multi-exit NNs. Multi-exit models have been widely used in edge computing, which achieves dynamic inference to allow samples to exit early during the inference.

There are two main types of multi-exit NNs. The first type is to add branches to existing models. BranchyNet [5] was the first to propose adding branches to neural networks. HAPI [24] uses the trained model to search and insert branches in the search space through hardware information. FlexDNN [25] uses the tradeoff formulation to determine the insertion of branches and search for optimal branch structure by Neural Architecture Search (NAS). LEIME [3] divides the model into three blocks with one exit each and deploys them to the end device, edge server, and cloud respectively by constructing a joint optimization formulation. The other type is the hand-tuned multi-exit NNs. Multi-Scaled Dense Network (MSDNet) [22] builds on top of the DenseNet [26] architecture. It uses a two-dimensional array of horizontal and vertical layers, which decouples depth and feature coarseness. Later RANet [27] is proposed as the extension of MSDNet.

It seems that these models can ensure that at least an intermediate result can be output when the inference is forced to exit. For such unpredictable inference time, deciding on which branch to exit is critical.

Instance-wise dynamic inference [6] has been proposed to dynamically determine which branch to exit during the inference task for each sample, since most DNNs perform inference in a static manner, that is, both the computational graph and the network parameters are fixed once trained. There are already related studies on dynamic inference including adaptively skipping layers or blocks, or dynamically selecting channels during the inference.

The confidence-based exit plans [5], [20], [21] tune the confidence threshold without consuming extra computation during inference. The easy samples can be output at shallow exits without executing deeper layers. More advanced, the learned decision models [28], [29] determine the inference depth for different samples at the beginning of the inference. Instead of exiting directly, GaterNet [30] and BlockDrop [31] decide which block to drop based on the input samples. Moreover, DDI [32] achieves dynamic layers and channels, but comes with complex and difficult model training.

However, none of the works above achieve dynamic inference from the perspective of unpredictable exit. If the exited

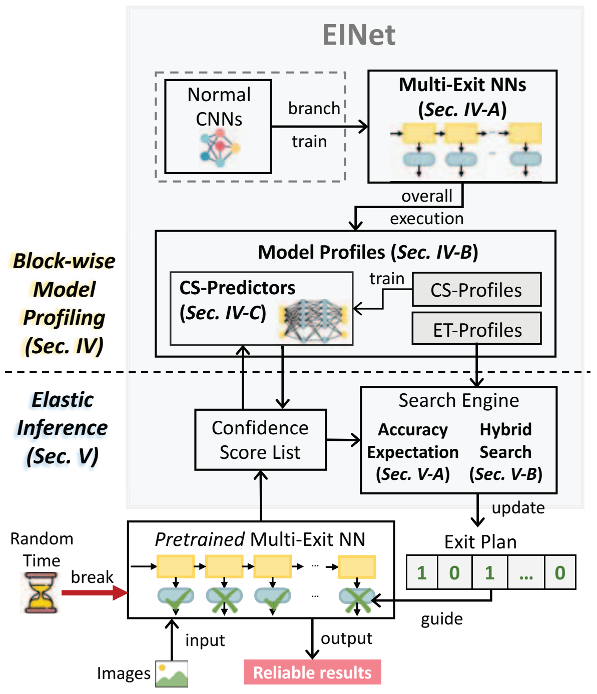


Fig. 1: An overview of EINet. EINet generates model profiles by executing multi-exit NNs during the offline Block-wise Model Profiling (Section IV) stage. And using profiles, it will continuously search and update plans by the Search Engine during the online Elastic Inference (Section V) stage.

branch is not chosen wisely, it will still result in no output.

To better select the exited branch, we propose a completely new branch-skipping-based planner. EINet is based on multi-exit NNs and orthogonal to the exit planners mentioned above. Instead of choosing one branch to exit or executing all branches, EINet will keep generating wise exit plans and guide the model to skip several branches during the inference. Thus, EINet brings a whole new plan generation solution for elastic inference under unpredictable exit scenarios.

III. Overview

In contrast to conventional systems or applications that terminate abruptly, real-time DNN tasks are designed to gracefully exit and provide more accurate results when unpredictable exit events occur. To address the challenge of obtaining accurate inference results in the presence of unpredictable exits, we introduce EINet. EINet serves as a sample-wise planner of real-time DNNs for elastic inference, which includes offline *Block-wise Model Profiling* (Section IV) and online *Elastic Inference* (Section V), as shown in Figure 1.

In the ***Block-wise Model Profiling*** stage, EINet carries out the execution of pre-trained multi-exit NNs on edge devices to obtain block-wise model profiles. In particular, for normal CNNs that lack multiple exits, EINet introduces additional branches to turn them into multi-exit NNs (Section IV-A). The generated model profiles consist of Confidence Score profiles (CS-profiles) and Execution Time profiles (ET-profiles) (Section IV-B). Among them, the **confidence score** refers to the softmax value of the output of the model at each exit. The CS-profiles are utilized to train CS-Predictors, which aims to adapt the input samples and enhance the interpretability of the inference process (Section IV-C). Simultaneously, ET-profiles generated during this stage are employed in the Search Engine of EINet, facilitating the search for an appropriate exit plan during the elastic inference process.

In the ***Elastic Inference*** stage, for situations where the inference will be forced to exit at an arbitrary time, EINet can guide the real-time tasks to submit an accurate result as quickly as possible. For a given input sample, when the model encounters a branching point, it generates an incomplete list of confidence scores, which is then input to the CS-Predictor. Subsequently, the CS-Predictor predicts the confidence scores for all subsequent exit points, resulting in a complete score list. To evaluate the performance of each exit plan, the score list, along with ET-profiles, is used to calculate the Accuracy Expectation (Section V-A). Given the vast search space for finding the near-optimal exit plan, the Hybrid Search is employed to explore plans with higher performance (Section V-B). These two algorithms collectively constitute the Search Engine module of EINet. The specific design details of the search and update processes will be further described in detail. Ultimately, the selected exit plan replaces the previous one and guides the model to execute the subsequent branch. EINet will execute such search and update process repeatedly until the inference is interrupted unpredictably.

IV. Block-wise Model Profiling

EINet can generate block-wise multi-exit model profiles by offline block-wise model profiling. In this section, we will introduce the design details of how EINet performs the branch insertion process that enables the transformation of conventional models into fine-grained multi-exit NNs. Our primary focus is on the convolutional neural network (CNN) architecture. Furthermore, we provide a detailed description of the recorded information in both profiles, as well as a comprehensive analysis of the inputs and outputs of the CS-Predictors.

A. Multi-exit Neural Networks

Since the time of exit is unpredictable, the inference may be interrupted and forced to exit at an arbitrary time. However, traditional neural networks with one exit cannot provide results immediately upon forced termination during inference. To deal with this, a preferable solution involves employing models with multiple exits. Thus, EINet is designed to incorporate pre-trained multi-exit models, enabling them to adapt to the concept of elastic inference.

Besides, EINet also has solutions for models that are not pre-trained or multi-exit. Since almost neural networks contain convolutional parts, we mainly discuss CNNs in this paper. For normal CNNs without multiple exits, EINet will perform branch insertion to turn them into multi-exit NNs. For all designed multi-exit NNs without training, EINet can also train them to get block-wise model profiles.

Design of branch insertion. For a normal CNN, there are many insertion points that can be inserted by branches. Some branch insertion plans of multi-exit NNs may not make good use of all computing resources. For example, if the inference is forced to quit just before the next exit, then the computing resources between the last exit and the present are wasted. As a result, the goal of this part is to design an efficient branch insertion plan for normal CNN. To improve computing resource utilization, it is feasible to reduce the time between two exits, which means building fine-grained models.

For normal single-exit CNNs, we just simply treat each convolutional layer and subsequent operations as a *conv part* and add a *branch* at the end of this part. One *conv part* and its *branch* are collectively called a *block*. That is, the fine-grained insertion plan is to add a branch to all convolutional parts. In particular, for neural networks with residuals, such as ResNet, etc., we treat one residual as the smallest unit to insert a branch. The training process backpropagates and updates the weights of models one by one.

For well-designed multi-exit model architectures [5], [22], [24], [25], we can also manually adjust their structures to make them more fine-grained. In this paper, we focus on MSDNet, the current state-of-the-art hand-tuned multi-exit model. It consists of multiple blocks with the same classifier each. The number of blocks and the structure of each block are both critical for elastic inference. More design details can be viewed in [22]. To avoid wasting computing resources, the MSDNet variations we choose has more blocks with fewer convolutional layer (*step* = 1,2), and its first block can contain an appropriate amount of convolutional layers (*base* = 2,4) and input channels (*channel* = 4,8,16) to tradeoff the inference accuracy and latency. The reasons for these choices are based on experimental results in Section VI-D1.

Design of branches. Due to the insertion of too many branches, the trained fine-grained multi-exit NNs will introduce latency overhead. In order to balance the inference accuracy and latency, the design of branches is also critical. The goal of this part is to design the appropriate structure of branches to make fine-grained multi-exit NNs more efficient.

The structure of a *branch* includes convolutional layers and fully connected layers. More convolutional or fully connected layers will inevitably lead to an increase in latency, but may not benefit accuracy. In order to balance accuracy and latency, the design of branches becomes critical. Based on the experimental results of one model in Section VI-D2, we finally choose the branch with one convolutional layer and two fully connected layers.

According to the above two design details of multi-exit NNs, Figure 2 takes VGG-16 as an example, demonstrating how EINet turns it into a fine-grain and efficient multi-exit model. EINet takes one convolutional layer and its subsequent operations as a *conv part* and takes one convolutional layer and two fully connected layers as a *branch* to form a *block*.

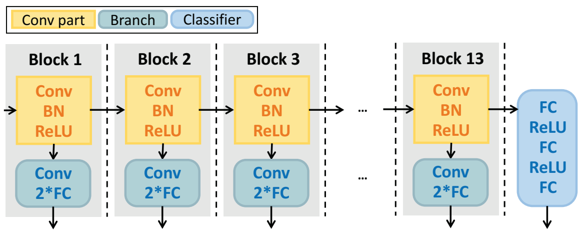


Fig. 2: Design of multi-exit VGG-16.

B. Block-wise Model Profiles

To better guide the online inference, EINet does the overall execution of specified models and records their block-wise profiles. In this section, we will introduce exactly what is documented in the two previously mentioned profiles.

1) ET-profiles: As we mentioned before, executing the branch will additionally take up the total inference time, and the model may not go deeper. To decide whether a branch is to be skipped or executed, we had better find out how long it takes to execute the model backbone and how long to execute each branch.

Since the time to execute each *block* (i.e. *conv part* and *branch*) is not very different for all samples, we tested MSDNet with 40 blocks to see the difference in execution time between the samples. Figure 3 shows the frequency distribution of execution time of 10,000 samples in each block, respectively. The latency difference of 90% samples is less than 0.07ms and the difference of 95% samples is less than 0.1ms. It is worth noting that for the particular structure set by MSDNet (shown in the upper right corner of the figure), the variation in inference time varies between different blocks.

Thus, these profiles record the average execution time of all testing samples inferring on the multi-exit NN for all blocks. For the same test samples and model, time may vary with the edge platform. Finally, ET-profiles record the average time to execute all *conv parts* T_c and all *branchs* T_b of a model on a specific edge platform, which will be used in Section V-A.

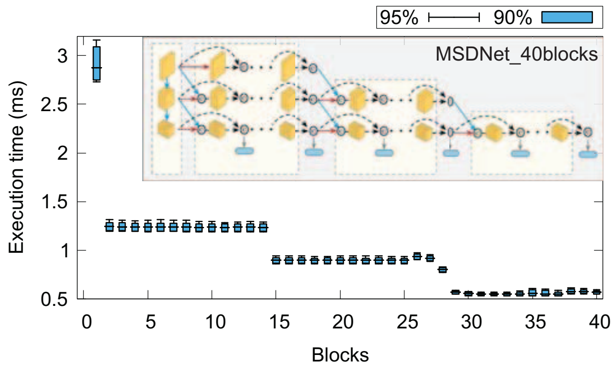
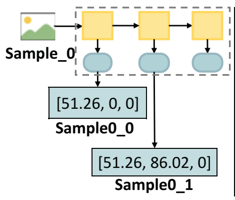


Fig. 3: The execution time of 10,000 samples running on MSDNet. The difference of 90% samples is less than 0.07ms and the difference of 95% samples is less than 0.1ms.

2) CS-profiles: Though all input samples are inferred by the same model, not all accurate inference results can be obtained at the same branch. That is, the average accuracy of all samples is coarse-grained and not a good reflection of how well each sample is inferred on that model. So to better understand the properties of a model in terms of inference accuracy of samples, instead of using the average accuracy of all samples, we applied the confidence score of each sample. The **confidence score** here refers to the softmax value generated by a sample at each branch during the model inference which is shown in the *Labels* column of the table in Figure 4.

Thus, these profiles record the confidence scores C of all testing samples on a specific model. Since the generation of confidence scores is one-time and does not change with the platform, there is no need to perform extra tests on the different edge devices. Once these CS-profiles have been generated, they are subsequently used to build training datasets for training block-wise CS-Predictors in Section IV-C.

With all these parameters recorded in ET-profiles and CS-profiles, the details of searching for and updating the near-optimal exit plan will be discussed in Section V.



ID	Training data	Labels
Sample0_0	[51.26, 0, 0]	[51.26, 86.02, 99.99]
Sample0_1	[51.26, 86.02, 0]	[51.26, 86.02, 99.99]
Sample1_0	[78.77, 0, 0]	[78.77, 99.99, 100.0]
Sample1_1	[78.77, 99.99, 0]	[78.77, 99.99, 100.0]
...	[... , ... , ...]	[... , ... , ...]

Fig. 4: Construction of training datasets for a three-exit NN.

C. *Confidence Score Predictors*

Since the output of the model varies with the input sample, block-wise CS-Predictors will be trained to adapt to the different inputs. In this section, we will introduce the construction of training sets using CS-profiles, as well as the design and training details of predictors.

Datasets construction. When a multi-exit NN has an inference result at any branch, CS-Predictor will be called to execute to predict the confidence score that the future branch may achieve during the inference. Why this prediction is made mainly for Search Engine to evaluate the performance of an exit plan, which will be introduced in Section V-A.

More specifically, training datasets include *Training data* and *Labels*, as shown in the table on the right of Figure 4. The *Labels* are the confidence scores of all exits, which are the lists stored in CS-profiles. The *Training data* are the confidence scores of a sample at the current and previous exits, which can be modified from the *Labels*. The scores at the subsequent unexecuted exits in the corresponding label list are set to 0. Taking a three-exit model as an example in Figure 4, it corresponds to two pieces of data for each input sample and both data pieces have the same label.

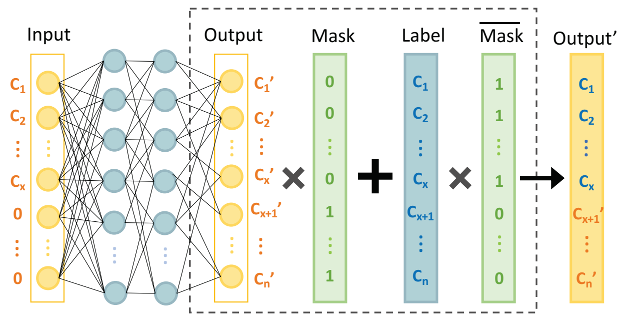


Fig. 5: Using masks for variable length outputs.

In summary, the CS-Predictors leverage the datasets derived from CS-profiles during their training phase, enabling them to effectively predict confidence scores.

Model design. To reduce the time overhead of predicting the confidence score that the future branch may achieve, CS-Predictors must be lightweight.

Since both training data and labels are one-dimensional, the structure of the predictor may not include convolutional layers. As a result, this model is designed to have only fully connected layers. The detailed structure is the model on the left in Figure 5. It should be noted that the hidden size of the first two fully connected layers is crucial for balancing accuracy and inference time. Our experimental results show that the hidden size can be selected as 2048, 1024 for models with large input sizes (over 30), and 256, 128 for smaller input sizes.

Loss function. In general, the output size of the current models is fixed and cannot be adjusted according to different inputs. But during the elastic inference, it is meaningless to predict confidence scores for those branches that have been executed. In order to predict the confidence scores of the following branches that haven't been executed, we introduce a mask to update the outputs to make the output size dynamic:

$$O' = OM + L\overline{M}, \tag{1}$$

where O is the inference output of the model, L is the ground truth of outputs, M is a binary mask list that sets confidence scores of exits that have output to be 0, and the rest of the exits to be 1, and $\overline{M}$ changed the value of 0 in M to 1, and the value of 1 to 0. Figure 5 shows the detail of the processing performed on the outputs. The mask is designed to replace some predicted values with already obtained confidence scores instead of taking the predicted value. The inference result will be covered by the confidence score that has been output, so as to realize the dynamic change of the output size.

Based on the updated outputs shown in Equation (1), we redefine the loss function (i.e. mean-square error, MSE) during the process of model training:

$$\mathcal{L} = \sum_{i=0}^{len(O')} (O'_i - L_i)^2. \tag{2}$$

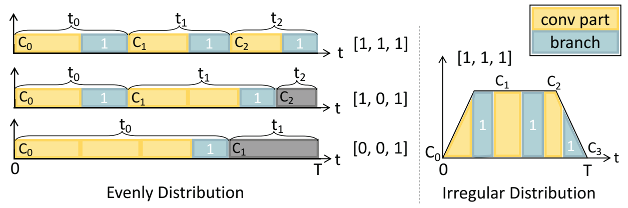


Fig. 6: Accuracy expectation algorithm with different time distributions. The left is uniform and the right is irregular.

Then bring Equation (1) into Equation (2) to get the simplified loss function:

$$\mathcal{L} = \sum_{i=x+1}^{len(O)} (O_i - L_i)^2, \tag{3}$$

where x indicates the model is executing at the x^{th} exit. Therefore, the first x values of M are 0, and Equation (3) can be easily derived. Based on the modified loss function above, the predictor that can predict the outputs of dynamic length will be trained for the corresponding multi-exit NN. It should be noted that the corresponding value in confidence lists will not be available if the branch is not executed. So we can only set the value of this exit to be the confidence score obtained by the nearest previous exit.

For more training details, we use *gradient clipping* to solve the possible gradient explosion during the backpropagation of training. And the learning rate needs to be reduced appropriately for predictors with smaller hidden sizes (e.g. 256) to ensure that the model training process can converge. During online elastic inference, the pre-trained CS-Predictor will be used to predict confidence scores for the Search Engine.

V. Elastic Inference

The previous section primarily introduced block-wise model profiling conducted prior to the online phase. In this section, we shift our focus to the Search Engine, which is considered the most important component of the online elastic inference stage. By leveraging ET-profiles and block-wise CS-Predictors, the Search Engine can evaluate each exit plan and dynamically select the near-optimal one in an expedited manner. Subsequently, the chosen plan replaces the previous one and directs the model inference path accordingly during the online inference. To effectively implement the Search Engine, we propose the accuracy expectation algorithm and the hybrid search algorithm which will be described in later sections.

A. Accuracy Expectation Algorithm

The accuracy expectation algorithm can evaluate the performance of exit plans. To better understand the exit plan, it can be seen as a binary list. Bit 0 means ignore the branch and bit 1 means execute the branch and get the inference result.

To evaluate the performance of such an exit plan, we propose an expectation calculation method based on probability. Taking the interrupted time evenly distributed as an example,

Algorithm 1 Accuracy Expectation Algorithm

```
Input:
  1)The statistical time of running conv parts: $T_c$;
  2)The statistical time of running branches: $T_b$;
  3)The confidence score of all exits: $C$;
  4)The $i^{th}$ exit plan of all exits: $P_i$.
Output:
  Performance expectation $E$.
1:  function CAL_EXP($T_c$, $T_b$, $C$, $P_i$)
2:      Initialize $E = 0$, $t_0 = 0$ and $t_1 = 0$
3:      $c_0 \leftarrow C_0$
4:      $T \leftarrow T_c + T_b$
5:      for $k \leftarrow 1$ to $len(P_i)$ do
6:          $t_1 \leftarrow t_1 + T_{ck}$
7:          if $P_{ik}$ then
8:              $t_1 \leftarrow t_1 + T_{bk}$
9:              $E \leftarrow E + c_0 \frac{t_1 - t_0}{T}$
10:             $c_0 \leftarrow C_k$
11:             $t_0 \leftarrow t_1$
12:         end if
13:     end for
14:     $E \leftarrow E + c_0 \frac{T - t_0}{T}$
15:     return $E$
16: end function
```

we divide the entire time of running all convolutional parts and branches into different intervals as shown on the left of Figure 6. Since the actual inference time is unpredictable, in which inference time interval it will fall is a probabilistic event.

Each time interval should have its corresponding score. For many samples, the overall average accuracy of each exit can be used as the score to calculate the performance. However, for a single sample, only the confidence scores can be obtained instead of the average accuracy. With the confidence scores and the ET-profiles generated in the block-wise model profiling stage, the algorithm will calculate the corresponding performance expectation for a specific exit plan. The calculation equation shows in (4):

$$E = \sum_{i=0}^{len(C)} \frac{C_i t_i}{T}, \tag{4}$$

where T is the total execution time, t_i is the i^{th} time interval between the i^{th} output and the $(i+1)^{th}$ output and C_i is the confidence score of the i^{th} output. We can see from the equation that the calculation between two exit plans which differ by only one bit cannot be simply converted. Because when the non-output in a plan is changed to output, the time interval and score for the current exit will change. More specifically, this change will cause nonlinear unpredictable changes in the following subsequent time intervals.

Algorithm 1 shows the details of the accuracy expectation algorithm under the uniform time distribution. The ET-profile is taken as input, which consists of T_c and T_b as mentioned in Section IV-B. The confidence score C is actually the O'

in equation (1) predicted by CS-Predictor during the elastic inference. For a specific exit plan P_i, the algorithm iterates over each bit to check whether the corresponding branch should execute or not. If the branch executes, record its current confidence score and the elapsed time since the last execution. Note that the elapsed time includes the time of exits having no outputs. Then the accuracy expectation can be calculated by Equation (4).

For situations involving irregular time distributions as depicted on the right of Figure 6, the area (i.e. weighted time) ratio is employed. Just replace the conventional subtraction of two times on lines 9 and 14 of Algorithm 1 with the integration of the area between these time points.

Therefore, each exit plan has its corresponding expected performance. In order to verify the effectiveness of this algorithm, we conducted an evaluation in Section VI-C1 using the overall average accuracy metric. The results of our evaluation indicate that the predicted accuracy closely approximates the ground truth value.

B. Hybrid Search Algorithm

During the searching phase, Search Engine employs a search algorithm to execute and evaluate the performance expectation of various exit plans multiple times. The objective is to find the near-optimal plan within a reduced timeframe. To expedite this search procedure, we propose the hybrid search algorithm to identify the most promising exit plan in this section.

Since the exit plans are binary lists, if the multi-exit NN has n exits, there will be 2^n plans. For models with fewer exits, this number is still considerable. Take the enumeration one by one, for a model with five exits, the enumeration search time is negligible (less than 1ms) and the optimum plan will be guaranteed. However, for models with a large number of exits, it is challenging to enumerate one with a higher expected performance and the search time will exponentially increase. To illustrate, in a model with 40 exits, the enumeration search time can extend up to approximately 40 days. Therefore, the enumeration method is optimal when there are few exit plans, but it is not suitable for all models.

To address the vast search space for models with more exits, an intuitive solution is to use the heuristic search. However, the accuracy expectation algorithm is nonlinear as mentioned in the previous section, which aggravates the difficulty in constructing valuation functions of the heuristic search. To explore this challenge, we implement the greedy algorithm by continuously exploring plans by incrementally increasing the number of outputs. It is to iterate through all unselected branches and select the best one with higher plan performance at a time, provided that all selected branches are known. After one selection, the same traversal is performed on a new base and selected until all branches have been selected. Thus, the search space, as well as time complexity, has changed from 2^n to n^2. Our experimental results on the model with 40 exits indicate that the enumeration of 15 selected branches still takes too long (over 3 days) while enumerating the 20 selected branches takes the maximum amount of time. However, the greedy search can always find all near-optimal plans in the case of more numbers of selected branches in a very desirable short period of time.

Algorithm 2 Hybrid Search Algorithm

```
Input:
  1)All exit plans of a model: $P$;
  2)The statistical time of running convolutional parts: $T_c$;
  3)The statistical time of running branches: $T_b$;
  4)The confidence score of all exits: $C$;
  5)The number of outputs for the enumeration search: $m$.
Output:
  A better exit plan $P'$.
1: $P_E \leftarrow \texttt{Enum}(P, m)$
2: $E_E \leftarrow \texttt{CAL}_{\texttt{EXP}}(T_c, T_b, C, P_E)$
3: $P_0 \leftarrow P_E$
4: $E_0 \leftarrow E_E$
5: for $i \leftarrow m+1$ to $len(C)$ do
6:     $P_G, E_G \leftarrow \texttt{Greedy}(P_0, i)$
7:     if $E_G > E_0$ then                    ▷ Update
8:         $E_0 \leftarrow E_G$
9:         $P' \leftarrow P_G$
10:    end if
11:    $P_0 \leftarrow P_G$
12: end for
13: return $P'$
```

Based on experimental results and our perceptions, the greedy search tends to fall into the local optimum in many cases, making the selection of exit plans non-optimal. To better take advantage of these two search methods in terms of search results and time, we propose the hybrid search algorithm, a two-stage search approach, which combines enumeration and greedy search. For the first few branches, we use enumeration. For the small number of selected branches including models with fewer exits, the search time is little and the optimal results can be guaranteed. However, for the later branches, we find the near-optimal exit plans using the greedy search to save the search time. Thus, based on this hybrid search, we can still guarantee to find the optimal plan in a very shorter time for the model with fewer exits and obtain the near-optimal solution very quickly for the model with more exits.

Algorithm 2 shows the detail of this algorithm. First, enumerate according to the number of branches to be enumerated (line 1). Then the expectation is calculated for the optimal solution obtained by the enumeration (line 2). This optimal plan and its expectation are used as the starting point of the greedy algorithm (lines 3-4). And the greedy traversal is performed until all branches have been selected (lines 5-12). Finally, we will get the near-optimal exit plan in less search time. And in Section VI-C, our evaluation results show the hybrid search can always find an exit plan with higher performance expectations under different time distributions.

Using the accuracy expectation algorithm and hybrid search algorithm, the Search Engine of EINet will dynamically search and update the exit plan once a branch is executed with an output result. And the newly chosen plan will guide the output of subsequent exits until the inference is finished or unpredictably interrupted during the online elastic inference.

VI. Evaluation

EINet is a novel sample-wise planner designed for scenarios involving unpredictable exits. Unlike traditional approaches that require selecting a single branch to exit or executing all branches, it continuously generates wise exit plans, effectively guiding the model to skip several branches during the inference. In this section, we will evaluate the overall performance improvement of EINet based on its above design.

A. Datasets and Setup

Our proposed planner, EINet, is implemented and rigorously validated primarily in the context of image classification tasks, utilizing well-established datasets commonly used in the field of computer vision.

Implementation. We implement EINet with *PyTorch*. For offline model training, we train all multi-exit NNs and their corresponding CS-Predictors on a server utilizing two NVIDIA GeForce RTX-3090 GPUs. We train each multi-exit NN for 300 epochs and its CS-Predictor for 3000 epochs. The training learning rates are all 0.001. For online end-to-end execution, we still perform validation on the same server mentioned above as the edge devices.

Algorithm	Py/C	Max (ms)	Avg (ms)	Min (ms)
Accuracy Expectation	Python	0.0610	0.0594	0.0584
	C	0.0003	0.0003	0.0003
Hybrid Search	Python	4.9145	4.6599	4.3861
	C	0.1292	0.1277	0.1267

TABLE I: Difference in Execution Time.

Since the Search Engine during the inference is time-critical, we implement this part in *C* to reduce the time overhead for better performance. Table I shows the difference in execution time of the accuracy expectation and hybrid search algorithms implemented in *Python* and *C*, respectively. It can be seen that using *C* can speed up nearly 100 times. Thus, we call the calculate expectation function and hybrid search function through the *ctypes* library during the elastic inference, which can achieve nearly 100 times faster than before. The rest of the implementation remains the same.

Datasets. The classic datasets in image classification we use are the MNIST, CIFAR-10, and CIFAR-100.

- The MNIST dataset contains 28×28 gray images, composed of 60,000 training and 10,000 testing images.
- The CIFAR-10 and CIFAR-100 datasets [33] contain 32×32 RGB images, composed of 50,000 training and 10,000 testing images, corresponding to 10 and 100 classes, respectively.

We use all training images to train multi-exit NNs and testing images to generate two profiles in the block-wise model profiling stage.

Evaluation Metrics. We hope that the inference results of the real-time task can be as accurate as possible in elastic inference. To simulate unpredictable scenarios, we set the unpredictable interruption time to be a distribution of the total execution time. To eliminate the effect of randomness on the results, we evaluate a large number of samples multiple times to get an overall accuracy average and regard it as our evaluation metric.

Baselines. We selected various baselines to evaluate the performance of EINet from two perspectives.

The first aspect is the comparison with different exit plans of a wide variety of multi-exit models. This comparison is to verify that EINet can achieve better overall performance regardless of the multi-exit model and exit plan.

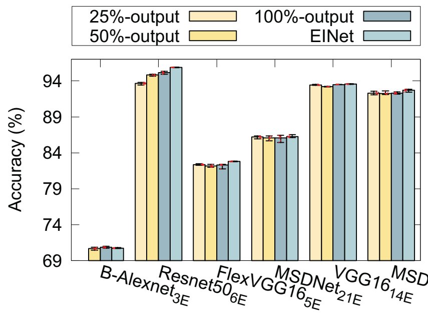
- For **models**, we choose B-AlexNet [5] with three exits, FlexVGG-16 [25] with five exits, fine-grained VGG-16 with 14 exits, fine-grained ResNet-50 with six exits, and MSDNet [22] with 21 and 40 blocks. Among them, the design of fine-grained models (i.e. VGG-16 and ResNet-50) is based on Section IV-A, and the selection of MSDNet variants will be introduced in Section VI-D1.
- For **exit plans**, we categorize them based on their behavior during inference into *static* and *dynamic* plans. *Static* plans include predetermined exit points at fixed percentages, such as 25%, 50%, and 100% of executed branches. On the other hand, *dynamic* plans involve confidence-based exit and EINet with random search.

The other aspect is the comparison with common neural networks. Since few works consider inference time to be unpredictable, this comparison is to get how much performance gain is achieved by EINet under the most common circumstances. We choose common CNN models with only one exit, compressed models, and normal multi-exit models without planners. Since the inference time of different models is different, to make the comparison fair, we apply MSDNet, the state-of-the-art, and use its adaptations as all baselines for experimental validation.

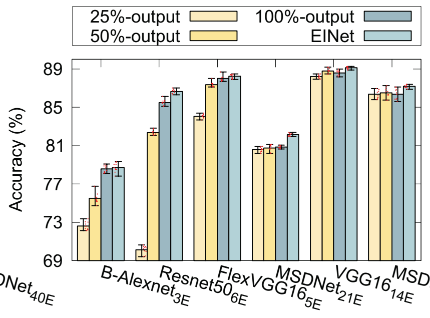
B. Overall Accuracy Improvement

In this section, we mainly conducted experiments to verify the performance improvement of EINet from the two aspects mentioned in Section VI-A.

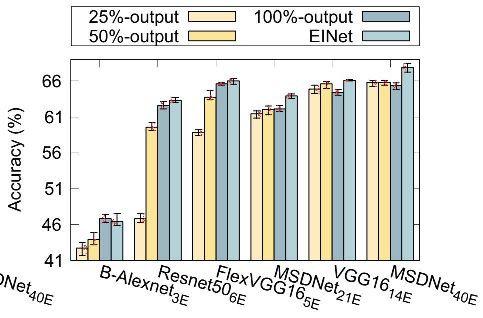
1) Static and dynamic exit plans: For **static** exit plans, Figure 7(a), 7(b) and 7(c) show the overall accuracy of the selected models on MNIST, CIFAR-10, and CIFAR-100 with all three static exit plans. Because not all inferences can be complete, the accuracy is lower than what the model can ultimately achieve. In general, EINet can achieve higher overall accuracy regardless of the multi-exit model and datasets. In addition, we noticed: (1) for models with fewer exits, the improvement is not obvious, because although the inference of CS-Predictors and the search of the Search Engine are fast, they still take time; (2) for the same model backbone, adding more branches can improve the overall accuracy in elastic inference; (3) for models with too many exits, a 100% exit



(a) MNIST



(b) CIFAR-10



(c) CIFAR-100

Fig. 7: Static exit plans on a wide variety of multi-exit NNs. EINet can achieve higher accuracy regardless of models. For the same model on the same dataset, EINet has about 0.13%-16.5% performance gain compared to the static exit plans.

Datasets	Models	Statis(%)	**Ours(%)**
CIFAR-10	B-AlexNet	78.43	78.71 (+0.28)
	ResNet-50	85.62	86.65 (**+1.03**)
	FlexVGG-16	88.10	88.23 (+0.13)
	MSDNet21	80.87	81.11 (+0.24)
	VGG-16	88.98	89.12 (+0.14)
	MSDNet40	86.38	86.60 (+0.22)
CIFAR-100	B-AlexNet	46.40	46.41 (+0.01)
	ResNet-50	62.73	63.29 (+0.56)
	FlexVGG-16	65.88	66.03 (+0.15)
	MSDNet21	62.25	63.92 (**+1.67**)
	VGG-16	65.63	66.08 (+0.45)
	MSDNet40	66.14	67.93 (**+1.79**)

TABLE II: Accuracy gain of EINet compared to statistics.

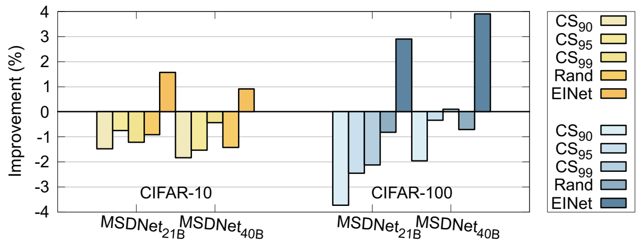


Fig. 8: EINet has about 0.79%-4.1% performance gain compared to other dynamic exit plans.

plan often results in performance loss due to the overhead of executing the branches.

Since the selected static plans above are so regular that they may not adapt to the characteristics of models on specific platforms, we generate a static optimal exit plan using the average time and accuracy in profiles. As there is no time constraint for searching this plan, we use enumeration to find the optimal. Table II shows the difference in accuracy between them. EINet can achieve up to 1.79% accuracy gain. Although for models with fewer exits, there are still small gains.

For **dynamic** exit plans, we compare other dynamic plans with confidence score threshold and EINet with random search methods with EINet. Figure 8 shows the relative improvement compared to the static plan without skipping (i.e. 100% output static plan). For two models on two datasets, EINet can guarantee performance improvement of about 1% to 4%, but confidence-based plans and EINet with random search all degrade performance. In addition, for confidence-based dynamic plans, raising the confidence threshold for early exit has a better effect on elastic inference, but it still can't achieve comparable improvement as EINet.

2) Common neural networks: The common models here include models with only one exit, compressed models also with only one exit, and multi-exit models without skipping any exits mentioned in Baselines (Section VI-A). Since the inference latency and final accuracy of different common models are not the same, we all use MSDNet adaptions to ensure the same total execution time for fairness.

Figure 9 shows the performance of EINet compared to common neural network techniques on four model variations. We conducted experiments 10 times and the result shows that EINet achieves a 40.4% to 61.5% improvement in accuracy compared to classic models; a 38.5% to 58.2% performance improvement compared to the compressed models, and a 0.8% to 1.5% improvement compared to the multi-exit models without any exit plan. In addition, comparing FlexVGG-16 and fine-grained VGG-16, MSDNet with 21 and 40 blocks, it can be found that the more fine-grained network has higher accuracy under the same dataset. Since the accuracy gap of the last exit is less than 0.5% between MSDNet with 21 and 40 blocks, the overall accuracy of the model with 40 blocks in elastic inference is improved by about 5%.

C. Evaluation of Search Engine

In this section, we mainly focus on the characteristics of the Search Engine component in the elastic inference stage.

1) Effect of Accuracy Expectation: The core of searching for the exit plan is the accuracy expectation. To verify the reasonableness of this algorithm, we compare the calculated expectation with the truth under different exit plans. Since different samples have an impact on the output and each

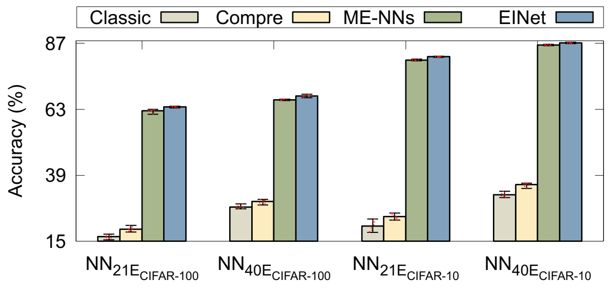


Fig. 9: EINet achieves over 50% accuracy improvement compared to common neural networks.

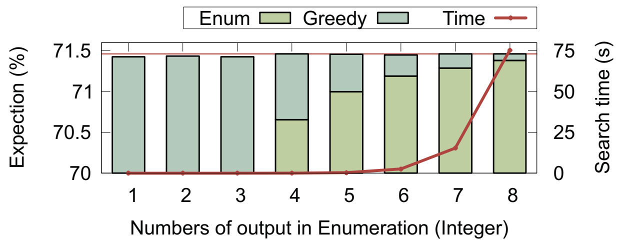


Fig. 11: Hybrid search performance. As the number of outputs in enumeration increases, the search time exponentially surges.

inference time is random and unpredictable, we run MSDNet with 40 blocks on CIFAR-100 five times and take the average as the overall accuracy truth.

Figure 10 shows the deference between calculated expectation and overall ground truth. The abscissa refers to the specified number of exits skipped uniformly by an exit plan. The expectation exhibits variations relative to the truth value, with fluctuations limited to within a 0.5% threshold. Based on this observation, it can be inferred that the accuracy expectation serves as a reasonable metric for assessing the efficacy of any given exit strategy to a certain extent.

Besides, the result also shows that executing all branches is not always optimal in elastic inference. For example, the plan to skip two exits uniformly is better than no skipping. In addition, to adapt to the characteristic of the input samples, exit plans should change based on various inputs to improve overall accuracy.

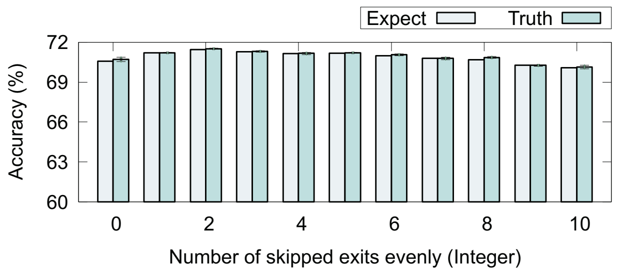


Fig. 10: The truth of accuracy and the calculated expectation of MSDNet with 40 blocks on CIFAR-100 are very close.

We did not do experiments related to confidence scores because it is difficult to find the corresponding ground truth. Through experiments of average accuracy, we can conclude that the accuracy expectation algorithm can measure the performance of an exit plan effectively.

2) Effect of Hybrid Search: Since the greedy search algorithm is not ideal in many cases, we use the enumeration method during the first half of the search and the greedy algorithm for the following half. We tested the hybrid search time and expectations for different levels of the enumeration (i.e. numbers of outputs for the enumeration search) for MSDNet with 40 blocks.

Figure 11 shows the search performance. The vertical coordinate represents the accuracy expectation, which is the combined effect of the two search algorithms. The horizontal coordinate represents the number of selected branches of the enumeration. As this number increases, the enumeration accuracy increases gradually, and the final search accuracy also increases slightly. From here we can see that directly using the greedy search for models with more exits can not find near-optimal plans. At the same time, the overall search time increases exponentially. Therefore, it is no need to enumerate more exits, four or five are enough. The search time is satisfactory and the results are near-optimal.

3) Different time distributions: In practice, the inference time of real-time tasks is unpredictable and random. To verify the time distributions of unpredictable exit on EINet, we conduct four search algorithms on MSDNet with 40 blocks and choose evenly as well as two Gaussian time distributions with the σ of 0.5 and 1, respectively. The average μ is taken as half of the total inference time. Among all baselines, the Baseline search refers to multi-exit model inference without any exit plan, while the Random search is to select the optimal strategy among 10,000 randomly selected plans.

Figure 12 shows the evaluation results. Different time distributions do have little impact on elastic inference results, but the hybrid search can always find a better exit plan. Although the results are expected, the actual search results are similar to the current expectation according to the effect of the accuracy expectation algorithm.

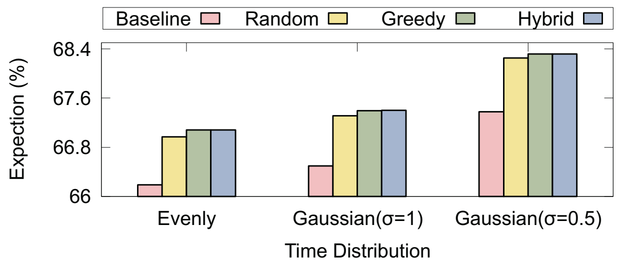
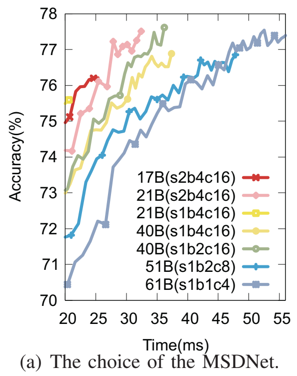
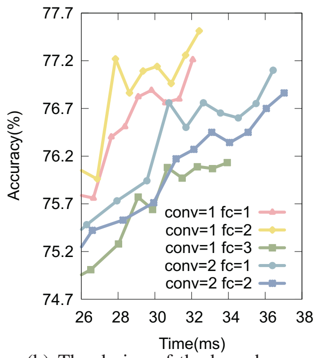


Fig. 12: Accuracy expectation of search methods at different time distributions. Hybrid can always find a better exit plan.



(a) The choice of the MSDNet.



(b) The design of the branch.

Fig. 13: The design of the branch structure and the multi-exit model is very critical in elastic inference.

D. *Impact of multi-exit NN Design*

1) Model structures: In this section, we will clarify why we choose MSDNet with 21 and 40 blocks. To evaluate the design of multi-exit NNs, we use different models with different numbers of blocks, steps, bases, and channels [22]. Results show in Figure 13(a). We hope that the model can achieve higher accuracy in a short inference time.

In the case of the same number of steps, bases, and channels, the more blocks are assigned, the longer the inference time will be. But more exits will make great use of computing resources. Therefore, the block of the model should not be too much or too little, 21 to 40 blocks are almost enough. The more steps are assigned, the longer the total inference time will be. Therefore, for a model with 40 or more blocks, the step is best set to 1. Similarly, the value of the base and channel is preferably smaller. Thus, we choose MSDNet with 21 and 40 blocks as our evaluation models.

2) Model branches: We use MSDNet to evaluate the design of branches with different numbers of combinations of convolutional layers and fully connected layers. The main structure of MSDNet includes 21 blocks, 2 steps, 4 bases, and 16 channels. The structure of the model classifier includes one convolutional layer and one fully connected layer, two convolutional layers and one fully connected layer, one convolutional layer, and two fully connected layers, etc.

Results show in Figure 13(b). Following [5], we also find that it is not necessary to add multiple convolutional layers to achieve better performance. The overall inference time increases while the accuracy decreases compared to other models with one convolutional layer at the same time. For the fully connected layers, increasing this layer can indeed increase the final accuracy of the model, but adding too many will lead to an increase in latency. Thus, for higher overall accuracy, we choose a combination of one convolutional layer and two fully connected layers as the design of the branches.

In conclusion, the design of branch structure and branch position of multi-exit NNs is critical in elastic inference.

VII. Conclusion

Multi-exit neural networks emerged frequently in edge computing to maximize the computing power of different devices. For the common and widespread unpredictable exit, which was overlooked by multi-exit models, we are the first to propose *Elastic Inference* in terms of time to tackle this issue. Elastic inference can output desirable results, no matter when being forced to exit, significantly improving the responsiveness of real-time applications.

In this paper, we propose EINet, a sample-wise planner of real-time multi-exit DNNs, which achieves efficient Elastic Inference instead of being killed while guaranteeing best-effort accuracy on different edge platforms. Unlike prior approaches, EINet treats the interrupted time as random and unpredictable. To get an anytime result, it takes advantage of fine-grained multi-exit NNs. To better guide the model to choose which branch to exit, EINet profiles the multi-exit NN and trains the CS-Predictors. With the help of model profiles and CS-Predictors, EINet will use Search Engine to evaluate the performance of each exit plan and update the found near-optimal exit plan dynamically to achieve sample-wise elastic inference with better accuracy.

Finally, we evaluate EINet and the result shows that the overall accuracy of EINet is improved by 0.13%-16.5% compared to the static plans, 0.79%-4.1% compared to other dynamic plans, and over 50% compared to common neural networks without multiple exits.

Acknowledgement

This work is supported by the National Natural Science Foundation of China under grant no. 62272407, the “Pioneer” and “Leading Goose” R&D Program of Zhejiang under grant No. 2023C01033, the National Natural Science Foundation of China under Grant No. 62072396, the National Youth Talent Support Program. We thank all the reviewers for their valuable comments and helpful suggestions. Yi Gao is the corresponding author.

References

[1] S. Teerapittayanon, B. McDanel, and H.-T. Kung, “Distributed deep neural networks over the cloud, the edge and end devices,” in *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE, 2017, pp. 328–339.
[2] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, “Spinn: Synergistic progressive inference of neural networks over device and cloud,” in *Proceedings of the 26th annual international conference on mobile computing and networking*, 2020, pp. 1–15.
[3] Z. Huang, F. Dong, D. Shen, J. Zhang, H. Wang, G. Cai, and Q. He, “Enabling low latency edge intelligence based on multi-exit dnns in the wild,” in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 729–739.
[4] M. Ebrahimi, A. d. S. Veith, M. Gabel, and E. de Lara, “Combining dnn partitioning and early exit,” in *Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking*, 2022, pp. 25–30.
[5] S. Teerapittayanon, B. McDanel, and H.-T. Kung, “Branchynet: Fast inference via early exiting from deep neural networks,” in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.
[6] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, “Dynamic neural networks: A survey,” *arXiv preprint arXiv:2102.04906*, 2021.

[7] Y. Xiang and H. Kim, "Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 392–405.

[8] N. Ling, K. Wang, Y. He, G. Xing, and D. Xie, "Rt-mdl: Supporting real-time mixed deep learning tasks on edge platforms," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, 2021, pp. 1–14.

[9] X. Foukas and B. Radunovic, "Concordia: teaching the 5g vran to share compute," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 580–596.

[10] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "Hrank: Filter pruning using high-rank feature map," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1529–1538.

[11] Y. Zhang, T. Gu, and X. Zhang, "Mdldroidlite: a release-and-inhibit control approach to resource-efficient deep neural networks on mobile devices," *IEEE Transactions on Mobile Computing*, 2021.

[12] G. Hinton, O. Vinyals, J. Dean *et al.*, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

[13] B. B. Sau and V. N. Balasubramanian, "Deep model compression: Distilling knowledge from noisy teachers," *arXiv preprint arXiv:1610.09650*, 2016.

[14] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[15] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[16] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[17] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 116–131.

[18] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2752–2761.

[19] W. Kang, K. Lee, J. Lee, I. Shin, and H. S. Chwa, "Lalarand: Flexible layer-by-layer cpu/gpu scheduling for real-time dnn tasks," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 329–341.

[20] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, "Adaptive neural networks for efficient inference," in *International Conference on Machine Learning*. PMLR, 2017, pp. 527–536.

[21] V. Bonato and C.-S. Bouganis, "Class-specific early exit design methodology for convolutional neural networks," *Applied Soft Computing*, vol. 107, p. 107316, 2021.

[22] G. Huang, D. Chen, T. Li, F. Wu, L. Van Der Maaten, and K. Q. Weinberger, "Multi-scale dense convolutional networks for efficient prediction," in *Proc. of ICLR*, 2018.

[23] S. Liu, S. Yao, X. Fu, R. Tabish, S. Yu, A. Bansal, H. Yun, L. Sha, and T. Abdelzaher, "On removing algorithmic priority inversion from mission-critical machine inference pipelines," in *2020 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2020, pp. 319–332.

[24] S. Laskaridis, S. I. Venieris, H. Kim, and N. D. Lane, "Hapi: Hardware-aware progressive inference," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[25] B. Fang, X. Zeng, F. Zhang, H. Xu, and M. Zhang, "Flexdnn: Input-adaptive on-device deep learning for efficient mobile vision," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2020, pp. 84–95.

[26] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, "Densenet: Implementing efficient convnet descriptor pyramids," *arXiv preprint arXiv:1404.1869*, 2014.

[27] L. Yang, Y. Han, X. Chen, S. Song, J. Dai, and G. Huang, "Resolution adaptive networks for efficient inference," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[28] X. Chen, H. Dai, Y. Li, X. Gao, and L. Song, "Learning to stop while learning to predict," in *International Conference on Machine Learning*. PMLR, 2020, pp. 1520–1530.

[29] X. Dai, X. Kong, and T. Guo, "Epnet: Learning to exit with flexible multi-branch network," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 235–244.

[30] Z. Chen, Y. Li, S. Bengio, and S. Si, "You look twice: Gaternet for dynamic filter selection in cnns," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9172–9180.

[31] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris, "Blockdrop: Dynamic inference paths in residual networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8817–8826.

[32] Y. Wang, J. Shen, T.-K. Hu, P. Xu, T. Nguyen, R. Baraniuk, Z. Wang, and Y. Lin, "Dual dynamic inference: Enabling more efficient, adaptive, and controllable deep inference," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 4, pp. 623–633, 2020.

[33] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Handbook of Systemic Autoimmune Diseases*, vol. 1, no. 4, 2009.