Elastic DNN Inference with Unpredictable Exit in Edge Computing

Jiaming Huang, Yi Gao *Member, IEEE*, Wei Dong *Member, IEEE*
College of Computer Science, Zhejiang University
Email: {huangjm, gaoyi, dongw}@zju.edu.cn

***Abstract*—Multi-exit neural networks have gained popularity in edge computing to leverage the computing power of diverse devices. However, real-time tasks in edge applications often face frequent unpredictable exits caused by power outages or high-priority preemptions, which have been largely overlooked by multi-exit models. To address this challenge, it is crucial to determine the appropriate exit point in the multi-exit model to ensure desirable results during unpredictable exits. In this paper, we propose EINet, a sample-wise planner for real-time multi-exit deep neural networks. EINet enables efficient Elastic Inference with unpredictable exits while ensuring best-effort accuracy on various edge platforms. Our approach involves partitioning a trained deep neural network into multiple blocks, each with its exit. Furthermore, EINet utilizes block-wise model profiles, which include accuracy and inference time information for each block. By leveraging these profiles, EINet dynamically determines the optimal exit plan for each sample during the inference process. We introduce Confidence Score Predictors to adapt to the unique characteristics of input samples and employ the Search Engine to efficiently find near-optimal plans for elastic inference. Extensive evaluations of EINet using multiple deep neural networks and datasets with unpredictable exits demonstrate its superior performance. EINet exhibits significant accuracy improvements: 0.13%-16.5% compared to static plans, 0.79%-4.1% compared to other dynamic plans, and over 50% compared to predictable inference in typical scenarios.**

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

I. Introduction

IN recent years, the field of edge computing has witnessed a proliferation of multi-exit neural networks (NNs), which aim to synchronize computing capabilities across devices, the edge, and the cloud [1]–[4]. Initially introduced in [5], multi-exit NNs offer the ability to generate intermediate outputs at early exit points, thereby enhancing the efficiency of deep neural network (DNN) inference. Subsequently, this concept has found widespread application in cloud-edge collaboration scenarios. More completely, [6] categorizes dynamic inference on multi-exit NNs into instance-wise, spatial-wise, and temporal-wise approaches. However, existing research on multi-exit NNs fails to take into account the issue of unpredictable exit.

In real-world scenarios, edge computing applications often involve the simultaneous execution of multiple real-time DNN tasks [7], [8]. However, these tasks frequently encounter unpredictable exits caused by various factors such as system power outages, preemption by high-priority tasks such as 5G vRAN [9], or specific user exit requests. Take, for example, the task preemption in which, in the context of Concordia [9], 5G vRAN tasks are considered high-priority and are allocated dedicated computation resources, leading to the unpredictable preemption of all other workloads. Figure 1 shows previous methods stop inference with no results facing such unpredictable exits. Surprisingly, the challenges posed by forced exit real-time inference tasks have long been overlooked.

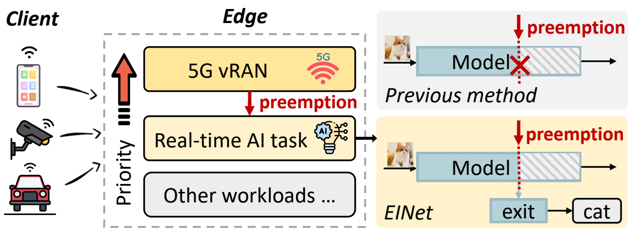


Fig. 1: When high-priority tasks (e.g., 5G vRAN) preempt AI tasks, elastic DNN inference enables early exit with results, unlike previous methods that stop inference with no results.

To tackle the challenge of unpredictable exits encountered in practice, our objective is to ensure that DNN tasks consistently yield favorable results. The literature offers numerous techniques aimed at enhancing 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]. Of particular relevance to our work, instance-wise dynamic inference [5], [20]–[22] with multi-exit NNs has shown promise in addressing the issue of unpredictable exits. These techniques involve selecting an appropriate exit for each instance, but they still face the problem of forced exit before the inference finishes. Orthogonal to the aforementioned techniques, we are the first to propose *Elastic Inference* utilizing multi-exit models. The elastic inference enables models to generate desirable intermediate results that remain unaffected by time constraints, persisting until an unpredictable forced exit occurs. To achieve this objective, it becomes critical for multi-exit NNs to decide when and at which branches to exit.

In this paper, we present **EINet**, a novel sample-wise planner for real-time multi-exit NNs. It enables efficient Elastic Inference instead of abruptly being terminated without any result while maintaining best-effort accuracy on various edge platforms. Unlike existing approaches, EINet acknowledges the unpredictable nature of exit times and dynamically guides multi-exit NNs to select branches adaptively for different samples during the inference. However, the practical implementation of EINet entails addressing two key challenges.

First, how can our planner consistently guide the model to produce desirable results before being stopped to meet real-time demand? To accommodate the possibility of interruptions at any moment, we employ fine-grained multi-exit NNs which are equipped with a higher number of exits. However, executing branches at each exit to preserve intermediate results may introduce time overhead, limiting the depth of inference and potentially compromising accuracy.To strike a balance between inference latency and accuracy, our planner utilizes the Search Engine to dynamically determine which exits to execute branches based on the available information. By selecting near-optimal exit plans, the multi-exit NNs can skip (i.e., not execute) certain exits, saving time while still achieving improved accuracy.

Second, how can our planner be general across different models and platforms for diverse input samples? To gather the information specific to different models and platforms, we present offline Block-wise Model Profiling to obtain model profiles. Additionally, to better adapt to the unique features of input samples, we propose training Confidence Score Predictors (CS-Predictors) to enhance the interpretability of each round of inference. In summary, the combination of CS-Predictors and model profiles enhances the generality of EINet. During the inference, it dynamically updates the exit plan for each sample, considering the unpredictable nature of exits.

Our main contributions can be summarized as follows:

- We introduce EINet, a sample-wise planner for efficient elastic inference that enables real-time AI tasks to continuously generate desirable results even when interrupted unpredictably.
- We propose Block-wise Model Profiling, which allows offline profiling of models on edge devices to understand their characteristics. The model profiles facilitate the training of CS-Predictors that adapt to sample features.
- We present the Search Engine, an online mechanism that finds near-optimal exit plans to balance accuracy and latency. Combined with trained CS-Predictors, EINet dynamically updates exit plans until a forced exit occurs.
- We implement EINet and conduct extensive experiments using MNIST, CIFAR-10, and CIFAR-100 datasets to evaluate the performance of elastic inference. The results demonstrate that for the same model on the same dataset, our framework can improve the overall accuracy compared to multiple baselines.

The rest of the paper is organized as follows: Section II discusses related work on improving real-time DNN efficiency in edge computing. Section III provides an overview of EINet. The design details of the two stages in EINet are presented in Sections IV and V. The performance evaluation is covered in Section VI. Finally, Section VII concludes the paper.

II. Related Work

In this section, we will introduce existing efforts related to AI model inference on the edge side. These works mainly focus on optimizing the inference procedure to make the task complete more quickly and pursue resource efficiency.

At the **resource level**, several works [7], [8], [19], [23] have focused on model partitioning and distribution across multiple heterogeneous processors. By carefully scheduling the models, the overall system efficiency can be enhanced.

However, achieving efficient scheduling requires accurate knowledge of the actual inference time. Without this information, there may still be DNN tasks that cannot obtain the desired results due to unpredictable exits.

At the **model level**, many *model compression* techniques have been proposed and widely used to improve inference efficiency. 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] and CondenseNet [18]. While speeding up inference time, these models may suffer from a loss of accuracy.

Several recent approaches are focused on designing *Multi-exit NNs* to allow samples to exit early during the inference. There are two main types. The first type is to add branches to existing models. BranchyNet [5] was the first to propose adding branches to NNs. Then many excellent works [3], [24]–[26] have been proposed to enhance the model inference performance in edge scenarios. The other type is the hand-tuned multi-exit NNs. Multi-Scaled Dense Network (MSDNet) [22] builds on top of the DenseNet [27] architecture. It uses a two-dimensional array of horizontal and vertical layers, which decouples depth and feature coarseness. Later RANet [28] is proposed as the extension of MSDNet.

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. It seems that multi-exit models can ensure that at least an intermediate result can be output when the inference is forced to exit. However, for such unpredictable exit, making efficient use of computing resources is critical.

At the **framework level**, *instance-wise dynamic inference* [6] has been proposed to dynamically determine the inference path for each sample during the inference task. This is in contrast to static inference, where both the computational graph and model parameters remain fixed after training. The confidence-based exit plans [5], [20], [21] tune the confidence threshold without consuming extra computation during inference. Samples have the flexibility to exit at shallow points without executing deeper layers. Furthermore, the learned decision models [29], [30] determine the inference depth for different samples right from the start. Instead of exiting directly, approaches like GaterNet [31] and BlockDrop [32] dynamically select specific blocks to drop based on the input samples. Moreover, DDI [33] achieves dynamic layers and channels, but involves complex model training processes.

However, none of the works above achieve dynamic inference from the perspective of unpredictable exit. If the exited branch is not chosen wisely, it will still result in no output.

In conclusion, to enhance the selection of the exit branch, we propose a novel branch-skipping-based planner. EINet operates on multi-exit NNs but diverges from the aforementioned exit planners. Instead of choosing a single branch to exit or

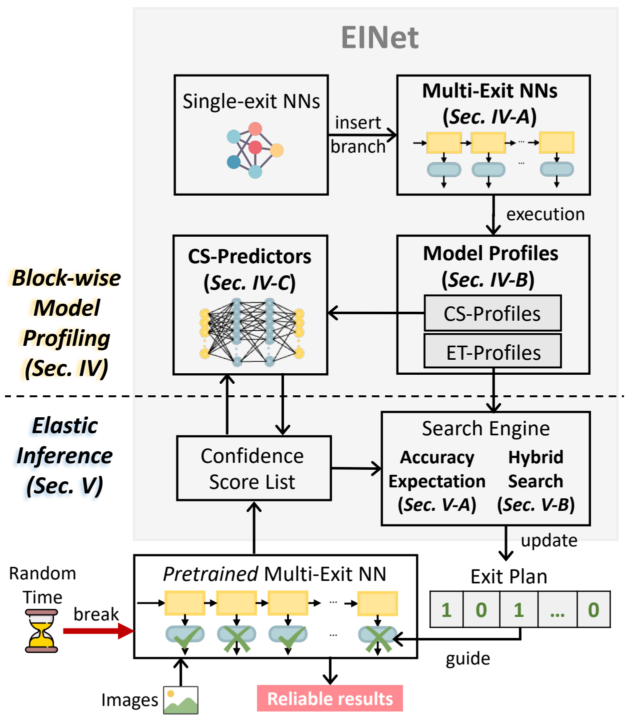


Fig. 2: An overview of EINet. EINet generates model profiles by executing multi-exit NNs on various platforms in offline Block-wise Model Profiling (Section IV) stage. Using profiles, it will continuously search and update plans by Search Engine in the online Elastic Inference (Section V) stage.

executing all branches, EINet continuously generates wise exit plans that facilitate the skipping of specific branches during the inference process. Consequently, EINet presents a distinct and comprehensive solution for generating optimal plans in elastic inference scenarios characterized by unpredictable exits.

III. Overview

Real-time tasks with multi-exit DNNs can exit with results when faced with unpredictable exits forced by the platform system or other applications. To achieve guaranteed performance of these tasks, we propose EINet to get more accurate results in unpredictable and limited inference time. Figure 2 shows the overview of EINet, including the offline *Block-wise Model Profiling* (Section IV) stage and the online *Elastic Inference* (Section V) stage.

In the ***Block-wise Model Profiling*** stage, EINet will transfer single-exit NNs without branches into multi-exit NNs (Section IV-A). To get offline block-wise model profiles, EINet will execute pre-trained multi-exit NNs on a certain edge device. The generated model profiles consist of Confidence Score profiles (CS-profiles) and Execution Time profiles (ET-profiles) (Section IV-B). Note that the **confidence score** refers to the maximum softmax value for class with the highest probability generated at each branch. Using CS-profiles, the CS-Predictor will be trained to predict the following confidence score of each sample according to the current confidence score list (Section IV-C). ET-profile generated at the same time will be used in the Search Engine to search and update exit plans (i.e. direct the multi-exit NN whether to skip or execute at each exit to get the result) during the online inference.

In the ***Elastic Inference*** stage, for situations where real-time tasks with multi-exit NNs will be forced to quit at a random time, EINet can always guide them to output a reliable result according to near-optimal exit plans. Take one input sample as an example, when the model executes a branch, it generates an incomplete confidence score list, which is subsequently fed into the CS-Predictor. Later, the CS-Predictor predicts the confidence scores for all remaining exits, enabling the creation of a complete score list. Leveraging this list in conjunction with the ET-profiles, each exit plan has its performance calculated by the Accuracy Expectation algorithm (Section V-A). To navigate the vast search space and identify the near-optimal exit plan, the Hybrid Search algorithm (Section V-B) is employed to explore the plan with higher performance. Collectively, these two algorithms constitute the Search Engine of EINet for better exit plan. Finally, the selected exit plan will supplant the previous one, guiding the model to execute the subsequent branches. EINet iteratively repeats this search and update process until the inference is interrupted unpredictably.

IV. Block-wise model profiling

In this stage, EINet can generate block-wise model profiles by executing pre-trained multi-exit NNs. Subsequently, we delve into the branch insertion process of EINet, which facilitates the conversion of conventional CNNs into fine-grained multi-exit NNs. Moreover, we offer an elaborate account of the captured information within both profiles, namely CS-profiles and ET-profiles, alongside comprehensive details regarding the design of CS-Predictors.

A. Multi-exit Neural Networks

Traditional NNs with one exit at the end cannot even provide results when the inference is interrupted and forced to exit unpredictably. To deal with this, a preferable solution involves employing models with multiple exits. We mainly take CNNs as the base NN backbones in this paper. Converting such a given single-exit CNN into a multi-exit model includes the main processes of selecting insertion points, branch structure design, and branch insertion.

1) Insertion points: There are many potential insertion points of classic model backbones. However, some of them may not effectively leverage all available computing resources. Concretely, if the inference terminates shortly before reaching the subsequent exit, the computing resources allocated between the last exit and the current point remain underutilized. To improve both computing resource utilization and task performance, it is feasible to minimize the time between two exits, necessitating the construction of fine-grained models. As a result, the objective of branch insertion is to devise fine-grained branch insertion plans. EINet has fine-grained insertion solutions for both normal single-exit CNNs and well-designed multi-exit models. As a result, the objective is to find fine-grained insertion points. EINet has solutions for both normal single-exit CNNs and well-designed multi-exit models.

For **normal single-exit CNNs**, EINet just treats each convolution and subsequent operators as a *conv part* and adds a *branch* at the end of this part. One *conv part* and its *branch* are collectively called a *block*. The fine-grained insertion points are set at the end of each convolutional part. In particular, for NNs with residuals, such as ResNet, etc., we treat each residual unit as *conv part* to insert a branch at the end of it.

For **well-designed multi-exit models** [5], [22], [24], [25], EINet will specifically fine-tune 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. The structural design of MSDNets consists of three main parameters $step$, $base$ and $channel$, and more design details can be viewed in [22]. To avoid wasting computing resources, the MSDNet variations we choose have more blocks with fewer convolutional layers (i.e. $step = 1, 2$), and their first block can contain an appropriate amount of convolutional layers (i.e. $base = 2, 4$) and input channels (i.e. $channel = 4, 8, 16$) to tradeoff the inference accuracy and latency. Experimental results presented in Section VI-D1 illustrate the rationality of the above fine-tuning settings.

2) Branch Structures: Since multi-exit networks should be fine-grained, the structure of branches that need to be inserted becomes critical. If inserting too many branches, the fine-grained multi-exit NNs will introduce latency overhead executing branches. To balance the inference performance and latency, the goal is to design the appropriate structure of branches to make fine-grained multi-exit NNs more efficient.

For consistency with the backbone, 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. Based on the experimental results in Section VI-D2, we decide on the branch with one convolutional layer and two fully connected layers. It is justified to insert respective optimal branches at each insertion point, but the structures are varied and difficult to exhaust. Therefore, in this paper, we only consider the same structures to better illustrate the ability of EINet.

3) Branch insertion: Figure 3 demonstrates how EINet turns VGG-16 into a fine-grain and efficient multi-exit VGG-16. EINet takes each convolution and its subsequent operations as a *conv part* and inserts a *branch* with one convolutional layer and two fully connected layers to form a *block*. Integrating the designed branches into the base model at insertion points involves connecting the output of the chosen *conv part* to the input of the new branch. As for training the above fine-grained multi-exit models, model backbones are not frozen and the training process updates the weights of models and branches from back to front while backpropagating. Although this approach produces a gradient accumulation that may interfere with the preceding inference, it will have a stronger presentation than freezing the parameters of the backbone.

B. Block-wise Model Profiles

To better understand the characteristics of multi-exit models for guidance during the online inference, EINet does the overall execution of these models and records their block-wise profiles. In this section, we will introduce exactly what is documented in the two previously mentioned profiles.

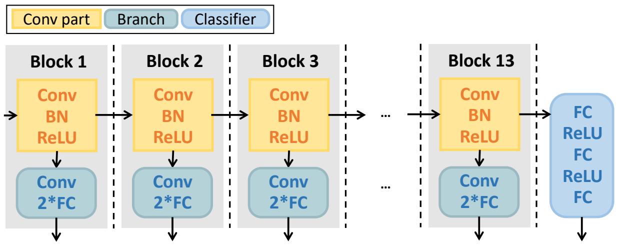


Fig. 3: Example of turning the normal single-exit VGG-16 into the fine-grained multi-exit VGG-16.

1) ET-profiles: It takes up the total inference time to execute inserted branches, which prevents the model from going deeper. To decide whether a branch is to be executed or skipped, we had better find out how long it takes to execute the model backbone and how long to execute each branch. Thus, the execution time recorded in ET-profiles includes the time to execute each *conv part* and inserted *branch*.

To assess the impact of block variations on execution time, we conducted an initial test of MSDNet using 40 blocks. Figure 4 presents the frequency distribution of execution time for 10,000 samples within each block. Notably, the time difference for 90% of the samples was found to be less than 0.07ms, while for 95% of the samples, the difference was less than 0.1ms. It is worth highlighting that the inference time variation across different blocks varied within the specific structure set by MSDNet, as illustrated in the upper right corner of Figure 4.

Since the time to execute each *block* (i.e. *conv part* and *branch*) of each sample is not widely different, EINet records the average execution time of all testing samples inferring on the multi-exit NN. The execution time is additionally affected by the platform on which it is running. Thus, EINet regenerates ET-profiles for each edge platform even with the same test samples and multi-exit models.

In conclusion, ET-profiles record the average time to execute all *conv parts* T_c and all *branchs* T_b of a multi-exit model on a specific edge platform, which will be used in Section V-A.

2) CS-profiles: Another important metric to be considered is the task inference performance in finite time. To better understand the properties of a model in terms of inference accuracy, we can test the average accuracy of all samples. However, though all input samples are inferred by the same model, they are represented differently during inference. That is, the average accuracy metric for the task is coarse-grained for each independent sample.

Thus, we applied the **confidence score** list for each sample at the branch instead of average accuracy, as the example shown in the *Labels* column of the table in Figure 5. The **confidence score** here refers to the maximum softmax value for class with the highest probability generated at each branch during the inference. For each model, we can generate such a series of confidence lists for all samples, thereby forming CS-profiles. Since the generation of confidence scores is

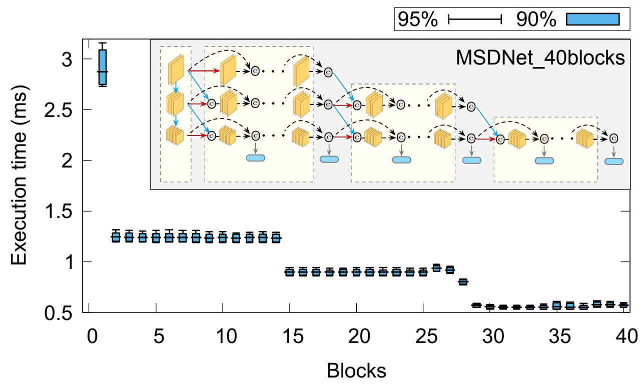
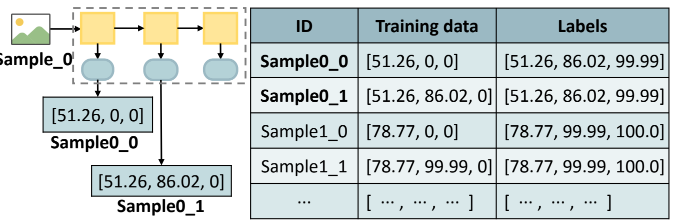


Fig. 4: The execution time of 10,000 samples running on MSDNet. The gap between 90% samples is less than 0.07ms and the gap between 95% samples is less than 0.1ms.



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. 5: Construction of training datasets using CS-profiles for the CS-Predictor of a three-exit NN.

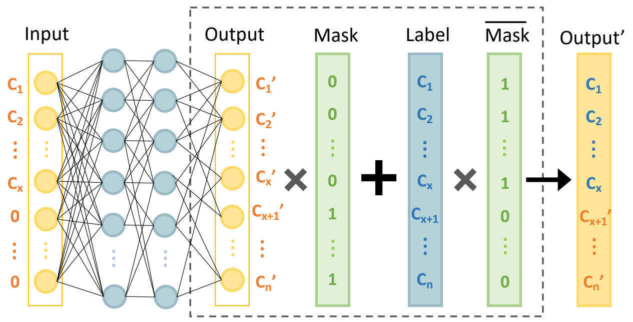


Fig. 6: Using masks for variable length of inference outputs.

platform-independent, CS-profiles can be directly generated by specifying the model and input samples and not change with platforms. When confidence score lists have been generated, they can be subsequently used to build training datasets for training block-wise CS-Predictors in Section IV-C.

Having captured all pertinent information within ET-profiles and CS-profiles, the process of searching for and updating the near-optimal exit plan will be elaborated upon in Section V.

C. Confidence Score Predictors

To better understand the features of input samples and track the representation in the inference process, we propose to train block-wise CS-Predictors. When a multi-exit NN obtains an inference result at any branch, the corresponding CS-Predictor will be called to execute predicting confidence scores of all the following branches. Such a prediction informs the decision of EINet on which the following branch to task as an exit, which will be detailed in Section V-A. In this section, we will introduce the construction of training sets using CS-profiles, as well as the design and training details of CS-Predictors.

1) Datasets construction: To train CS-Predictors, training datasets including data and labels are indispensable. What we need is exactly the confidence score information recorded in CS-profiles. The right table in Figure 5 showcases the *Labels* representing the confidence scores of all exits. The *Training data* consists of confidence scores at the current and previous exits for a given sample, which can be derived from the *Labels*. In the corresponding label list, the scores at subsequent unexecuted exits are set to 0. For instance, considering a three-exit model depicted in Figure 5, each input sample corresponds to two data pieces, both sharing the same label.

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

2) Model structure: Since CS-Predictors will be called multiple times during inference, they must be lightweight to reduce the additional time overhead of prediction.

Since both training data and labels are one-dimensional and their length is the number of exit branches, CS-Predictors can be built from lightweight fully connected layers without convolutions. The detailed prediction structure includes the input layer, one hidden layer, and the output layer, which is shown as the model on the left in Figure 6. Since the scale of fully connected layers is crucial for balancing prediction accuracy and time, we assign the input and output size of the hidden layer to be 2,048 or 1,024 for models with around 30 branches, and 256 or 128 for models with fewer branches. Moreover, we assign ReLU and dropout layers following the input and hidden layers to improve the model robustness.

During the inference of ME-NNs, the length of output results prefers to be variable. Because only the confidence scores of the following unexcuted branches are worth predicting. To achieve variable length outputs, we introduce a binary mask outside the model to update the outputs:

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

where O is the inference output of the model, L is the confidence scores generated by past exits, M is a binary mask list that sets past exits be 0, and the rest exits to be 1, and $\overline{M}$ changed the value of 0 in M to 1, and the value of 1 to 0. Figure 6 shows the detail of the $Mask$ and $\overline{Mask}$ performed on the outputs. The $Mask$ is designed to extract the predicted confidence scores of the unexecuted branches and the $\overline{Mask}$ is to keep the already generated scores. Thus, they collaboratively form the predicted results, which realizes the dynamic change of the inference output length. It should be noted that the corresponding value in $Label$ lists may not be available if the branch is not executed. To avoid compromising the evaluation of the exit plan (explained in Section V-A), we set the value of these exits to be the confidence score obtained by the nearest previous exit.

3) Model training: To accommodate dynamically changing outputs, we modified the loss function during model training and added the Activation Cache mechanism during inference.

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

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

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

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

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 confidence scores of the following unexecuted branches will be trained for the corresponding multi-exit NN.

4) Inference optimization: During online elastic inference, the pre-trained CS-Predictor will be used to predict confidence scores. However, we observe that during the inference process of a sample, the prediction of the confidence score is incremental. Taking the input layer as an example, the transformation can be represented as:

$$a_1 = f_1(W_1 X + B_1), \tag{4}$$

where X is the input vector, W_1 is the weight matrix of the input layer, B_1 is the bias vector, and $f_1()$ is the activation function. As the input vector sparsity decreases (i.e., the number of zeros becomes less), the calculation of Equation III becomes redundant and repeated for the confidence scores that have been generated. To predict confidence scores incrementally, we propose Activation Cache to cache the generated activation further reducing the computing and latency overhead.

Whenever a new variable is added to the input vector, we construct the sparse vector by setting all other variables to zero to optimize computing efficiency. To keep the effect of the activation function, we cached the activation before each activation function layer. Whenever a new computation is generated, it is added to the cached activation and then passed through the activation function. Such an Activation Cache-based incremental prediction process reduces the computational overhead while decreasing the inference time.

In conclusion, EINet converts single-exit CNNs into fine-grained multi-exit models and generates block-wise model profiles for them. These profiles will aid the elastic inference, including: adapting to the properties of different models on different platforms or even training CS-Predictors to adapt to the inference representation properties of different samples.

V. Elastic Inference

In this section, we shift our focus to online elastic inference stage. Search Engine is considered the most important component in this stage. Utilizing offline ET-profiles and block-wise CS-Predictors, it can efficiently assess each exit plan and dynamically select the near-optimal one in an expedited manner. Subsequently, the selected plan supersedes the previous one and directs the model inference path accordingly. To fulfill its effectiveness, we propose the accuracy expectation algorithm and the hybrid search algorithm.

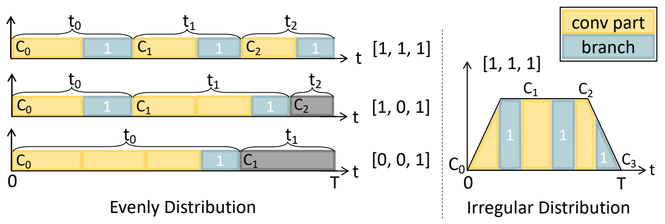


Fig. 7: Accuracy expectation algorithm with different time distributions. The left is uniform and the right is irregular.

Algorithm 1 Accuracy Expectation Algorithm

Input:
1) The profiled time to run conv parts: T_c;
2) The profiled time to run 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:      $c_0 \leftarrow C_0$, $T \leftarrow T_c + T_b$
4:      for $k \leftarrow 1$ to $len(P_i)$ do
5:          $t_1 \leftarrow t_1 + T_{ck}$
6:          if $P_{ik}$ then
7:              $t_1 \leftarrow t_1 + T_{bk}$
8:              $E \leftarrow E + c_0 \frac{t_1 - t_0}{T}$
9:              $c_0 \leftarrow C_k$, $t_0 \leftarrow t_1$
10:         end if
11:     end for
12:     $E \leftarrow E + c_0 \frac{T - t_0}{T}$
13:     return $E$
14: end function
```

A. Accuracy Expectation Algorithm

The accuracy expectation algorithm can evaluate the performance of exit plans. Exit plans can be seen as a binary list for better understanding. Bit 0 means ignore the branch and bit 1 means execute the branch and get the inference result.

To evaluate the performance of an exit plan under unpredictable exit scenarios, we propose to calculate the expectation based on exit probability. Since the actual inference time is unpredictable, in which inference time interval it will fall is a probabilistic event. Taking the uniform distribution of inference time as an example, we divide the entire profiled time of running each convolutional part and branch into different colored intervals as shown on the left of Figure 7. The time of running one convolutional part and branch is labeled t_i. The confidence score during this period is the output of the previous branch, marked as C_i. Once the executing branch produces a new result, it moves on to the next time interval t_{i+1} with C_{i+1}. The following three-bit lists correspond to three different exit plans, in which 0 means do not exit. The equation of calculating expectation is as follows:

$$E = \sum_{i=0}^{len(C)} \frac{C_i t_i}{T}, \tag{5}$$

where T is the total execution time, t_i is the i^{th} time interval between the i^{th} and $(i+1)^{th}$ output and C_i is the confidence score of the i^{th} output. In [34], real-world cases demonstrate that the preemption can be modeled using arbitrary curves. For situations involving irregular time distributions as depicted on the right of Figure 7, the area (i.e. weighted time) ratio is employed. Just replace the ratio of the time interval t_i and total execution time T with the ratio of the integrated area over the time interval and the total execution time.

Algorithm 1 shows the accuracy expectation algorithm under the uniform time distribution. The ET-profile is taken as input including T_c and T_b. The confidence score C is actually the O' in equation (1) predicted by CS-Predictor. For a specific exit plan P_i, the algorithm iterates over each bit to check whether the corresponding branch should execute or not (line 4). If the branch executes, the expectation can be calculated (line 8) and the current confidence score and the elapsed time should be recorded (line 9). For irregular time distributions, just modify Eqution 5 as mentioned before. The evaluation results in Section VI-C1 indicate that the performance expectation closely approximates the ground truth.

B. Hybrid Search Algorithm

During the searching phase in Search Engine, the hybrid search algorithm is to find the near-optimal plan quickly.

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. For a model with five exits, the enumeration search time can be less than 1 ms. However, for models with a large number of exits, it is challenging to enumerate all plans because of the exponentially increased search time. To illustrate, for a model with 40 exits, the enumeration search time can extend up to approximately 40 days. Therefore, the enumeration may be 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 the heuristic search. However, the accuracy expectation algorithm is nonlinear, which aggravates the difficulty in constructing valuation functions of the heuristic search. Figure 7 illustrates that even a one-bit difference in exit plans leads to completely different computations and the conversion between them is non-linear and complicated. Because when the non-output branch in a plan is changed to output, the time interval and score for the current exit will change. To explore this challenge, we implement the greedy algorithm by continuously exploring plans by incrementally increasing the number of outputs. Concretely, the Search Engine iterates through all non-output branches and greedily selects one to form the local optimum exit plan with the already selected branches. This traversal and selection will be performed until all branches are selected. Thus, the search space, as well as time complexity, has changed from 2^n to n^2.

Algorithm 2 Hybrid Search Algorithm

Input:
1) All exit plans of a model: P;
2) The runtime statistics of convolutions and branches: T_c, T_b;
3) The confidence score of all exits: C;
4) The number of outputs for the enumeration search: m.

Output:
A better exit plan P'.

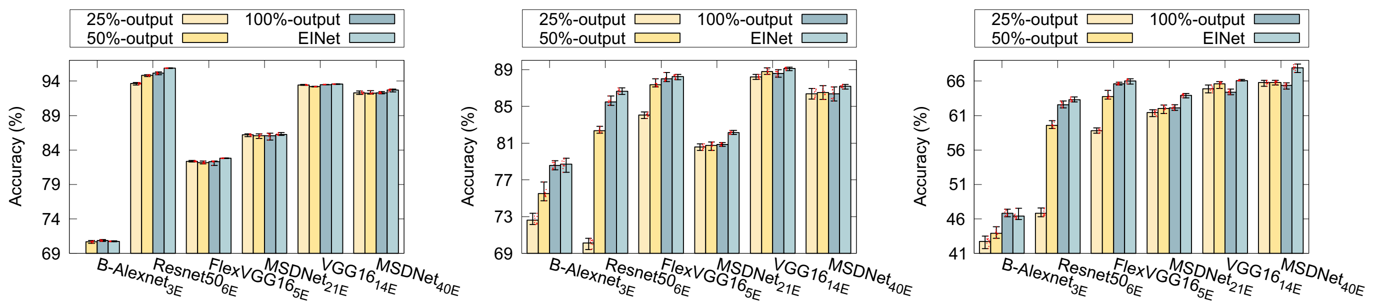
```
1:  $P_E \leftarrow \texttt{Enum}(P, m)$
2:  $E_E \leftarrow \texttt{CAL}_{\texttt{EXP}}(T_c, T_b, C, P_E)$
3:  $P_0 \leftarrow P_E$
4:  $E_0 \leftarrow E_E$
5:  for $i \leftarrow m+1$ to $len(C)$ do
6:      $P_G, E_G \leftarrow \texttt{Greedy}(P_0, i)$
7:      if $E_G > E_0$ then                    ▷ Update
8:          $E_0 \leftarrow E_G$, $P' \leftarrow P_G$
9:      end if
10:     $P_0 \leftarrow P_G$
11: end for
12: return $P'$
```

Though greedy search can find near-optimal plans in a short time, it tends to fall into the local optimum in many cases.

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. When the number of branches to be searched is a handful, the enumeration search time is considerable and the optimal results can be guaranteed. Meanwhile, for the later branches, we find the near-optimal exit plans employing the greedy search to save the search time. Thus, based on this hybrid search, EINet can be guaranteed to find the optimal plan for the model with fewer exits and obtain the near-optimal solution in a very short time for the model with more exits.

Algorithm 2 shows the details of the hybrid search algorithm. First, the enumeration search is performed according to the number of branches specified for enumeration (line 1). Then the performance expectation is calculated for the optimal exit plan 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). The greedy search performs traversal and selection until all branches have been selected (lines 5-11). Finally, we will get the near-optimal exit plan in less search time. 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.

In conclusion, through the utilization of the accuracy expectation algorithm and hybrid search algorithm, the Search Engine of EINet can dynamically explore and update the exit plan as branches are executed and produce outputs. The newly chosen plan will guide the subsequent execution of branches until the completion of inference or unpredictably interruptions during online elastic inference.



(a) MNIST (b) CIFAR-10 (c) CIFAR-100

Fig. 8: Static exit plans on a wide variety of multi-exit NNs. EINet can achieve higher accuracy regardless of the models. For the same model on the same dataset, EINet has about 0.13%-16.5% performance gain compared to the static exit plans.

TABLE I: Difference in Execution Time.

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

VI. Evaluation

In this section, we will evaluate the performance of EINet based on its above design in two stages.

A. Datasets and Setup

Implementation. We implement EINet with *PyTorch*. For offline model training and profiling, all multi-exit NNs and their corresponding CS-Predictors are trained and executed on a server utilizing two NVIDIA GeForce RTX-3090 GPUs with a Core i7-10400 CPU. For online elastic inference, we still perform validation on the same server mentioned above. Besides, due to the time-critical nature of the Search Engine during inference, we implement this part in *C* programming language to minimize time overhead and enhance performance. Table I shows the disparity in execution time of the accuracy expectation and hybrid search algorithms implemented in *Python* and *C*, respectively. Utilizing *C* yields a significant speed improvement of nearly 100 times. Thus, we invoke the calculate expectation algorithm and hybrid search algorithm through the *ctypes* library during the elastic inference, achieving nearly 100 times faster than before. The rest of the implementation remains unchanged.

For training details, we train each multi-exit NN for 300 epochs and its CS-Predictor for 3000 epochs. We use SGD with 0.9 momentum as the optimizer and the training learning rates are all 0.001. For CS-Predictors, we employ *gradient clipping* and *dropout* layer 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 training process of predictors can converge.

To simulate unpredictable exit, we randomly set the inference time for each sample within the total profiled execution time. This randomization follows a uniform distribution.

Datasets. EINet is implemented and rigorously validated primarily in the context of image classification tasks. The well-established and commonly used datasets 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 [35] 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. Among them, the CS-profiles can be used to form the trainsets for training CS-Predictors.

Evaluation Metrics. Accuracy is frequently used as a metric for recognition tasks. Unlike common scenarios, the inference time is random and unpredictable in our scenarios. To eliminate the effect of randomness on the results, we evaluate a large number of samples multiple times to get the overall average accuracy and regard it as the evaluation metric.

Baselines. We carefully selected diverse baselines to assess the performance of EINet focusing on two key aspects:

(1) Comparative evaluation of a wide variety of multi-exit models with different exit plans. This comparison aims to validate the superior overall performance consistently achieved by EINet, regardless of specific multi-exit NNs and exit plans.

- For **models**, we include 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. The design of the fine-grained models (VGG-16 and ResNet-50) follows the guidelines outlined in Section IV-A1, while the selection of MSDNet variants will be introduced in Section VI-D1.
- For **exit plans**, we classify them into two categories based on their inference behavior: *static* plans and *dynamic* plans. *Static* plans entail predetermined exit points at fixed percentages, such as 25%, 50%, and 100% of executed branches. Conversely, *dynamic* plans incorporate confidence-based exit and EINet with random search.

TABLE II: EINet achieves an accuracy gain of up to 1.79% compared to the theoretically optimal 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**)

(2) Comparative analysis with neural networks in typical scenarios. Given the limited consideration of unpredictable exits in existing works, this comparison aims to quantify the performance improvement achieved by EINet under prevalent circumstances. We choose single-exit CNN models with only one exit at the end, 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

Unlike common scenarios that select a branch to exit early or even don't exit early, EINet, a novel sample-wise planner, can continuously generate wise exit plans, guiding the model to skip several branches for unpredictable exits. In this section, we mainly conducted experiments to verify the performance improvement of EINet from the following perspectives.

1) Static exit plans: Figure 8 (a-c) show the average accuracy of the preselected models on MNIST, CIFAR-10, and CIFAR-100. EINet was compared with the three static strategies respectively. Due to the inherent limitations of unpredictable exit, the exit accuracy falls below the ultimate accuracy of models. In summary, EINet can achieve 0.13-16.5% higher performance regardless of the types of multi-exit models and datasets. It demonstrates that EINet can dynamically customize exit plans for each input sample, wisely selecting branches to execute for better performance. By avoiding unnecessary branch execution, it saves overhead and enables deeper model inference. In addition, we noticed that increasing the number of branches for the same model backbone can lead to improved overall accuracy.

To address the limitation of the selected regular static plans, we generate a static optimal exit plan based on average time and accuracy profiles. As there is no time constraint for searching this plan, we employ enumeration to find the optimal one. The accuracy differences between the selected plans are presented in Table II, with EINet demonstrating an accuracy gain of up to 1.79%. Even for models with fewer exits, there are still minor improvements observed.

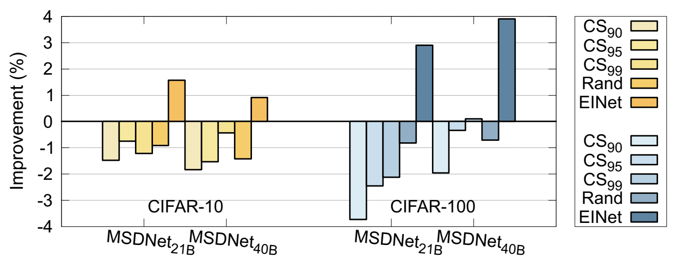


Fig. 9: EINet has about 0.79%-4.1% performance gain compared to other dynamic exit plans.

2) Dynamic exit plans: To evaluate the effectiveness of dynamic exit plans, we compare them with confidence score threshold-based plans and EINet utilizing random search methods. Figure 9 illustrates the improvement compared to the static plan without skipping (i.e. 100% output static plan). For two models across two datasets, EINet achieves performance improvements ranging from approximately 1% to 4%. In contrast to randomly selecting an exit plan, EINet leverages the confidence characteristics of each sample to make intelligent decisions. Compared with the confidence-based dynamic strategy, EINet uses CS-Predictor to reduce the overhead of executing branches, thereby ensuring task performance. In addition, increasing the confidence threshold for early exit in confidence-based dynamic plans yields better results, but they still fall short of the improvements achieved by EINet.

3) Common neural networks: The common models here include *Classic* models with only one exit, *Compre*ssed models also with only one exit, and multi-exit NNs (*ME-NNs*) without skipping any exits. Since the inference latency and final accuracy vary with the basic models,we employ MSDNet adaptions to achieve the same total execution time for fairness.

Figure 10 shows the performance comparison of EINet against various common neural network techniques in four model variations. The experiments were conducted 10 times, and the result shows that EINet achieves remarkable improvements in accuracy compared to the classic model, ranging from 40.4% to 61.5%. Because classic models with a single exit face a significant reduction in task performance when unpredictably preempted, as they fail to produce any results. Furthermore, compared to compressed models where inference time is optimized and completion is expedited, EINet still exhibits performance enhancements ranging from 38.5% to 58.2%. In comparison to a multi-exit model without any exit plan (100% exit), EINet achieves a performance improvement of 0.8% to 1.5%. In addition, when comparing FlexVGG-16, fine-grained VGG-16, as well as MSDNet with 21 and 40 blocks, it is observed that the more fine-grained network achieves higher accuracy on 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 approximately 5%.

C. Evalution of Search Engine

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

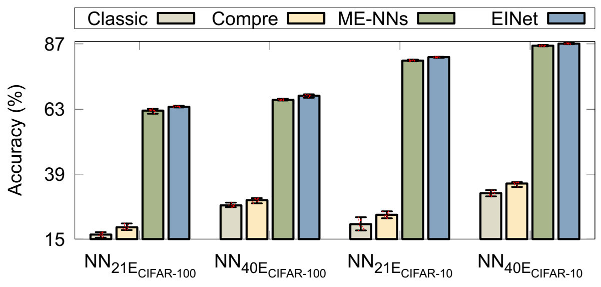


Fig. 10: EINet achieves over 50% accuracy improvement compared to common neural networks.

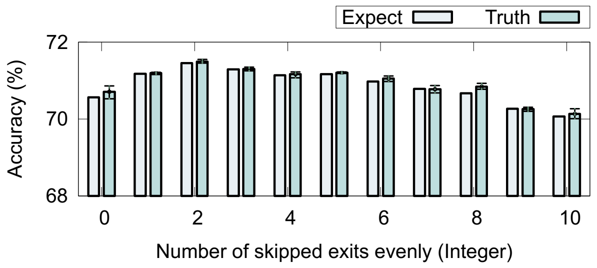


Fig. 11: The truth of accuracy and the calculated expectation of MSDNet with 40 blocks on CIFAR-100 are very close.

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.

Figure 11 shows the gap between calculated expectation and overall ground truth. The abscissa refers to the specified number of exits skipped uniformly by an exit plan. Since the truth of a single sample confidence score is hard to evaluate and each 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. The expectation value fluctuates above and below the truth, with overall variation less than 0.5%. Based on above 11 plans, it can be seen that the accuracy expectation serves as a reasonable metric for assessing the performance of any given exit plan. 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 characteristics of the input samples, exit plans better change based on various inputs to improve overall accuracy.

In conclusion, 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: We tested the hybrid search time and expectations for different numbers of outputs for the enumeration search on MSDNet with 40 blocks.

Figure 12 shows the performance of the hybrid search. The vertical coordinate represents the accuracy expectation, which is the combined results of the two search algorithms. The horizontal coordinate represents the number of selected branches for enumeration. As it increases, the enumeration accuracy increases gradually, and the final search accuracy also increases slightly. But the searching time has risen expo-

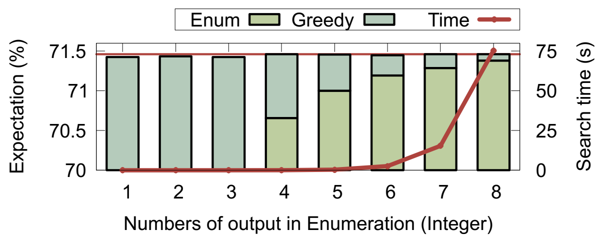


Fig. 12: Enumerating only a handful of branches and then performing a greedy search on the remaining branches gives the near-optimal result in an acceptable time.

nentially. Therefore, there is no need to enumerate more exits, four or five are enough. The search time is satisfactory and the results are near-optimal. Besides, using the greedy search directly for models with more exits (the first column in Figure 12) may fall into the local optimum.

In conclusion, enumerating only a handful of branches and then performing a greedy search on the remaining branches gives the near-optimal result in an acceptable time.

3) Different time distributions: In practice, the inference time distribution in unpredictable exit scenarios is irregular. To better evaluate the performance of EINet in such realistic scenarios, we choose uniform time distribution and two Gaussian time distributions with the average μ taken as half of the total inference time, the σ of 0.5 and 1, respectively. We conduct experiments of four search algorithms on MSDNet with 40 blocks. Among them, the *Baseline* search refers to inference without exits, while the *Random* search is to find the optimal exit plan among 10,000 randomly selected plans.

Figure 13 shows the evaluation results. Different time distributions have minimal impact on the elastic inference outcomes. Moreover, the hybrid search method consistently manages to find a superior exit plan. Though the performance of *Random* search seems comparable, it takes around 20 times longer to execute. Enumeration was excluded also due to their long search time. Although the results are expected, the actual search results are similar according to Section VI-C1.

4) Activation cache: To optimize the inference efficiency of CS-predictors during the elastic inference, caching mechanisms are used to trade memory for inference time.

We record in Table III the inference time reduced and the extra memory space occupied by the corresponding predictors for different models. For small-scale predictors, it can improve inference speed up to 3.52-4% by taking up only 1-2 KB of storage. For large-scale predictors, it is still possible to improve inference speed by 3.08-3.77%, only taking up more memory spaces. It takes up only a few dozen KB at most, which is considered acceptable.

D. Impact of multi-exit NN Design

1) Model structures: In this section, we aim to provide clarity on our selection of MSDNet with 21 and 40 blocks. To evaluate the design of multi-exit NNs, we consider models with varying numbers of *blocks*, *steps*, *bases*, and *channels*, as outlined in [22]. Results are shown in Figure 14 (a).

TABLE III: Activation cache trades off a 3.08-4% inference speedup for only a small amount of memory space.

Models	Reduced time	Memory
B-AlexNet	3.84%	1036B
ResNet-50	3.52%	2072B
FlexVGG-16	4.00%	2068B
MSDNet21	3.41%	8276B
VGG-16	3.77%	8248B
MSDNet40	3.08%	16544B

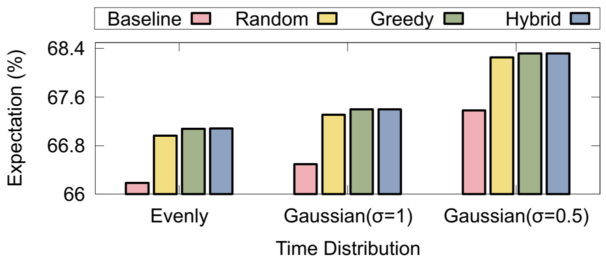


Fig. 13: Accuracy expectation of search methods at different time distributions. Hybrid can always find a better exit plan.

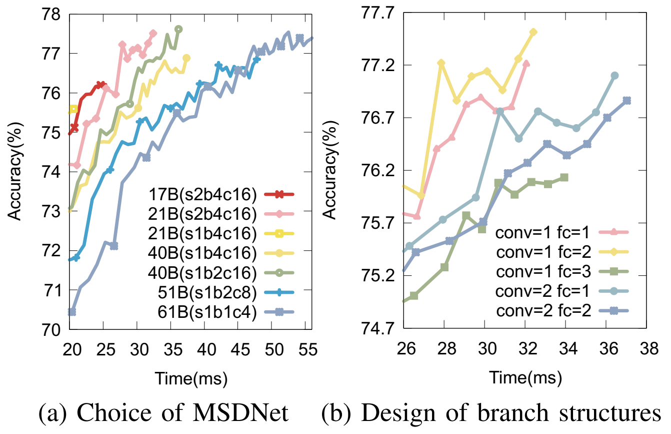


Fig. 14: The design of the branch structure and the multi-exit model plays a pivotal role in elastic inference.

Our objective is to achieve higher accuracy within a shorter inference time. In the case of the same number of *steps*, *bases*, and *channels*, increasing the number of blocks results in longer inference times. However, having more exits enables more efficient utilization of computing resources. Consequently, the choice of the number of blocks should strike a balance, neither too few nor too many. In our evaluation, we found that the range of 21 to 40 blocks is nearly optimal. Since assigning more steps leads to increased total inference time, for models with 40 or more blocks, it is best to set the step value to 1, ensuring faster inference. Likewise, smaller values for the base and channel parameters are preferable. Thus, we selected MSDNet models with 21 and 40 blocks as our evaluation models, as they align with these considerations.

2) Model branches: To assess the design of branch structures, we conducted evaluations with varying combinations of convolutions and fully connected layers, such as one convolution and one fully connected layer, two convolutions and one fully connected layer, and so on. The main structure of MSDNet includes 21 blocks, 2 steps, 4 bases, and 16 channels.

The results, as depicted in Figure 14 (b), align with the findings in [5], indicating that it is not necessary to add multiple convolutional layers to achieve better performance. In fact, increasing the number of convolutional layers leads to longer inference times and may decrease accuracy. On the other hand, adding more fully connected layers can enhance the final accuracy, but also leads to an increase in latency. Considering these factors, we choose a combination of one convolutional layer and two fully connected layers that offers a balance between inference accuracy and latency.

Discussion. While our primary focus in this paper has been on CNN, EINet holds potential for applicability to other model structures like RNN, Transformers, and GANs. The key consideration lies in converting a single-exit model into a multi-exit model. For instance, in the case of a Transformer-based model, the placement of exit branches between blocks enables it to be a multi-exit model. However, the specific placement and design of these branches require further exploration and consideration in future research.

VII. Conclusion

In this paper, we propose EINet, a sample-wise planner for real-time multi-exit DNNs, which enables efficient Elastic Inference instead of being killed while guaranteeing best-effort accuracy. To better guide model inference, EINet profiles multi-exit NNs and trains CS-Predictors. Using profiles and CS-Predictors, EINet employs Search Engine to update the near-optimal exit plan dynamically. Finally, the evaluation result shows that the overall accuracy is improved by 0.13-16.5% compared to static plans, 0.79-4.1% compared to dynamic plans, and over 50% compared to predictable inference in typical scenarios. These results highlight the reliable elastic inference with unpredictable exits achieved by EINet.

Acknowledgments

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. 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. Dong, H. Wang, D. Shen, Z. Huang, Q. He, J. Zhang, L. Wen, and T. Zhang, "Multi-exit dnn inference acceleration based on multi-dimensional optimization for edge intelligence," *IEEE Transactions on Mobile Computing*, 2022.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

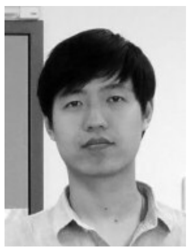
[32] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris, "Blockdrop: Dynamic inference paths in residual networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8817–8826.

[33] 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.

[34] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, vol. 130, 2015.

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

Jiaming Huang received the B.S. degree from the College of Internet of Things, Hohai University. She is currently working toward the Ph.D. degree in the College of Computer Science, Zhejiang University. Her current research interests include Internet of Things and edge computing.



Yi Gao (M'15) received the B.S. and Ph.D. degrees from the College of Computer Science at Zhejiang University, China, in 2009 and 2014, respectively. From 2015 to 2016, he visited McGill University as a Visiting Scholar. He is currently a full professor in the College of Computer Science at Zhejiang University. His current research interests include Internet of Things and intelligent edge computing. He is a member of the IEEE and the ACM.

Wei Dong (S'08–M'12) received the B.S. and Ph.D. degrees from the College of Computer Science at Zhejiang University in 2005 and 2011, respectively. He is currently a full professor in the College of Computer Science at Zhejiang University. He leads the Emerging Networked Systems (EmNets) research group in Zhejiang University. His research interests include AIoT, wireless and mobile computing, and IoT security. He is a member of the IEEE and the ACM.