A Simple Hash-Based Early Exiting Approach For Language Understanding and Generation

Anonymous ACL submission

Abstract

Early exiting allows instances to exit at different layers according to the estimation of difficulty. Previous works usually adopt heuristic metrics such as the entropy of internal outputs to measure instance difficulty, which suffers from generalization and threshold-tuning. In contrast, learning to exit, or learning to predict instance difficulty is a more appealing way. Though some effort has been devoted to employing such "learn-to-exit" modules, it is still unknown whether and how well the instance difficulty can be learned. As a response, we first conduct experiments on the learnability of instance difficulty, which demonstrates that modern neural models perform poorly on predicting instance difficulty. Based on this observation, we propose a simple-yet-effective Hash-based Early Exiting approach (HASHEE) that replaces the learn-to-exit modules with hash functions to assign each token to a fixed exiting layer. Different from previous methods, HASHEE requires no internal classifiers nor extra parameters, and therefore is more efficient. Experimental results on classification, regression, and generation tasks demonstrate that HASHEE can achieve higher performance with fewer FLOPs and inference time compared with previous state-of-the-art early exiting methods.

1 Introduction

Early exiting is a widely used technique to accelerate inference of deep neural networks. With the rising of pre-trained language models (PLMs) (Devlin et al., 2019; Yang et al., 2019; Lan et al., 2020; Raffel et al., 2020; Qiu et al., 2020), early exiting is drawing increasing attention in the NLP community. At its core, early exiting allows simple instances to exit early while allowing hard instances to exit late. Thus, how to measure instance difficulty is a crucial problem.

Most existing early exiting methods attach multiple internal classifiers to the PLM and adopt some heuristic metrics, such as entropy (Xin et al., 2020; Liu et al., 2020a) or maximum softmax score (Schwartz et al., 2020) of internal outputs, to measure instance difficulty. However, these methods can not easily generalize to new tasks. On the one hand, these metrics are not accessible on some tasks such as regression. On the other hand, In order for these methods to perform well, one usually needs to fine-tune the threshold, which varies widely across different tasks and datasets.

Another way to measure instance difficulty is to directly learn it. Recent studies (Elbayad et al., 2020; Xin et al., 2021) that use the idea of "learn-to-exit" have achieved promising results. They jointly train a neural model to predict for each instance the exiting layer. At their core, the learn-to-exit module is to estimate the difficulty for each instance. Compared with previous heuristically designed metrics for difficulty, learn-to-exit is task-agnostic and does not require threshold-tuning, therefore is a more promising way.

Despite their success, it is still unknown whether or how well the instance difficulty can be learned. As a response, in this work, we construct datasets for two kinds of instance difficulty: (a) Human-defined difficulty, and (b) Model-defined difficulty. The dataset for human-defined difficulty has two labels, 0 for instances that can be annotated by human and 1 for instances that cannot. For model-defined difficulty, we train a multi-exit BERT (Devlin et al., 2019), which is attached with an internal classifier at each layer, on a sentence-level classification task, SNLI (Bowman et al., 2015), and a token-level classification task, OntoNotes NER (Hovy et al., 2006). The trained multi-exit BERTs are then used to annotate for each development instance whether it can be correctly predicted by each internal classifier. Thus, our constructed sentence-level and token-level model-defined difficulty datasets are multi-label classification datasets. Experimental results demonstrate that, modern neural networks perform poorly on predicting instance difficulty. This observation is consistent with previous work (Laverghetta et al., 2020) on estimating instance difficulty for curriculum learning.

Given that instance difficulty is hard to be predicted, then what works in the learn-to-exit modules? We hypothesis that the consistency between training and inference may play an important role. That is, for a training instance x_i that is predicted to exit at layer l, an inference instance x_j that is similar with x_i should be predicted to exit at layer l, too. Since neural networks are usually smooth functions (Ziegel, 2003), this consistency can be easily satisfied by neural learn-to-exit modules. If this hypothesis holds, we can replace the learn-to-exit module with a simple hash function. In particular, we use hash functions to assign each token to a fixed exiting layer. This hash-based early exiting method is named HASHEE.

Compared with previous methods that use heuristic metrics for difficulty or jointly learn to exit, HASHEE offers several advantages: **(a)** HASHEE requires no internal classifiers nor extra parameters, which are necessary in previous work. **(b)** HASHEE can perform token-level early exiting without supervision, therefore can be widely used on various tasks including language understanding and generation. **(c)** The speed-up ratio can be easily tuned by modifying the hash function. **(d)** HASHEE can significantly accelerate model inference on a per-batch basis instead of per-instance basis as in previous work (Xin et al., 2020; Liu et al., 2020a; Zhou et al., 2020).

We conduct experiments on classification, regression, and generation tasks. Experimental results on ELUE (Liu et al., 2021a) demonstrate that HASHEE, despite its simplicity, can achieve higher performance with fewer FLOPs and inference time than previous state-of-the-art methods on various tasks. Besides, our experiments on several text summarization datasets show that HASHEE can reduce ∼50% FLOPs of BART (Lewis et al., 2020) and CPT (Shao et al., 2021) while maintaining 97% ROUGE-1 score.

2 Can Instance Difficulty Be Learned?

In this section, we examine whether or to what extent instance difficulty can be learned. In particular, we manage to evaluate how well a neural network that trained on some data with difficulty annotation can generalize to unseen data. Here we consider two kinds of difficulty: human-defined difficulty and model-defined difficulty.

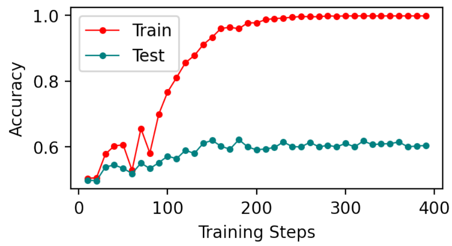

Figure 1: Training a BERT model to predict human-defined difficulty.

2.1 Human-defined Difficulty

Dataset Construction Human-defined difficulty of an instance measures how difficult for human to judge its label. To construct such a dataset, we use the SNLI dataset (Bowman et al., 2015), which is a collection of 570k human-written English sentence pairs that are manually labeled with the inference relation between the two sentences: entailment, contradiction, or neutral. The labels in SNLI are determined by the majority of the crowd-sourced annotators. If there is no majority for an instance, its label would be "Unknown". We collect 1,119 unknown instances from SNLI dataset as our difficult instances, and collect 1,119 labeled instances from the instances of three classes (i.e., entailment, contradiction, and neutral) in equal proportion as our simple instances, obtaining a balanced binary classification (difficult or simple) dataset with 2,238 instances. We randomly sample 1,238 instances with balanced labels as training set and use the remaining 1,000 instances as test set.

Learning Human-defined Difficulty We then train a BERT model (Devlin et al., 2019) with a linear classifier on the top on our constructed training set, and evaluate on the test set to see if it can predict whether an unseen instance is simple or difficult. As shown in Figure 1, the BERT model that fits well on the training set can only achieve ∼60% accuracy on the test set, demonstrating that neural models (even BERT) can not easily learn to estimate human-defined difficulty.

2.2 Model-defined Difficulty

However, model can have a different view of instance difficulty from human. For example, an instance can be defined as a difficult one if it can not be correctly predicted by a well-trained model.

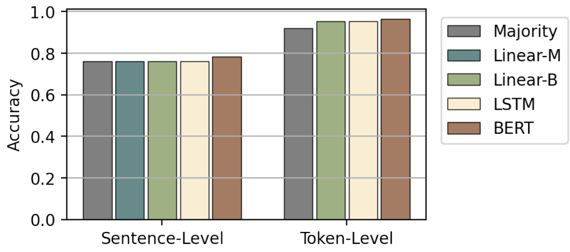

Figure 2: The best accuracy achieved by different models on our constructed datasets for model-defined difficulty. The trained neural networks perform on par with the simple majority model.

Model	Precision	Recall	F1 Score
Sentence-Level Difficulty			
Majority	60.5	36.7	45.7
Linear-M	54.8	42.1	47.6
Linear-B	52.9	45.3	48.8
BiLSTM	54.5	45.2	49.4
BERT	61.1	49.9	54.9
Token-Level Difficulty			
Majority*	-	-	-
Linear-B	56.6	38.7	46.0
BiLSTM	46.8	39.9	43.0
BERT	65.6	44.6	53.1

Table 1: Experimental results on our constructed model-defined difficulty datasets. We report micro-averaged precision, recall and F1 score over the negative label. *: The majority model for the token-level task would always predict positive class for all the labels, and therefore the F1 score is not applicable.

Thus, we also construct datasets to characterize model-defined difficulty for each instance, which is more realistic in the context of early exiting. In particular, we construct two datasets labeled with model-defined difficulty at sentence-level and token-level, respectively.

Sentence-level Difficulty Estimating model-defined difficulty of a sentence (or sentence pairs) is helpful to language understanding tasks such as text classification and natural language inference (Xin et al., 2021). To obtain the sentence-level difficulty, we train a multi-exit BERT that is attached with an internal classifier at each layer on SNLI training set. Once the multi-exit BERT is trained, it can serve as an annotator to label each instance in the SNLI development set whether it can be correctly predicted by each internal classifier. In our experiments, we use $\text{BERT}_{\text{BASE}}$ that has 12 layers, and therefore for each instance in the SNLI development set we have 12 labels, each takes values of 0 or 1 to indicate whether or not the corresponding internal classifier correctly predict its label. By this, we label the 9,842 SNLI development instances to construct a multi-label classification dataset, from which we randomly sample 8,000 instances as training set and use the remaining 1,842 instances as test set.

Token-level Difficulty We also construct a dataset for estimating model-defined difficulty of each token, which can be used in language generation tasks (Elbayad et al., 2020) and sequence labeling tasks (Li et al., 2021b). Similarly, we train a multi-exit BERT on OntoNotes NER (Hovy et al., 2006) training set, and use it to annotate each token in the OntoNotes development instances whether it can be correctly predicted by each internal classifier. By this, we obtain a token-level multi-label classification datasets consisting of 13,900 instances, from which we randomly sample 10,000 instances to construct a training set and use the remaining 3,900 instances as test set.

Learning Model-defined Difficulty For each constructed model-defined difficulty dataset, we evaluate several models: (1) **Majority** model always predicts the majority class for each label, with class priors learned from the training data. (2) **Linear-M** is a multi-classification linear layer that takes as input the average pooled word embeddings and outputs the exiting layer. This model corresponds to the multinomial variants of Elbayad et al. (2020). Since the inputs of Linear-M is non-contextualized, we did not apply it to estimate token-level difficulty. (3) **Linear-B** is a binary classification linear layer that takes as input the hidden states at each BERT layer and outputs whether or not the instance (or token) is correctly predicted. This model corresponds to the geometric variants of Elbayad et al. (2020) and the learn-to-exit module in BERxiT (Xin et al., 2021). (4) We also train and evaluate a bidirectional **LSTM** model (Hochreiter and Schmidhuber, 1997) with one layer and hidden size of 256. It takes as input the instance and outputs the exiting layer. (5) **BERT** model (Devlin et al., 2019) is also considered for this task. For these models, except for Linear-B, we use the binary cross entropy loss to handle the multi-label classification. Since most development instances are correctly predicted, our constructed datasets are label-imbalanced. To alleviate this issue, we adopt over-sampling for classes with fewer instances.

Our experimental results are shown in Figure 2, from which we find that: (1) For the task of estimating sentence-level difficulty, the shallow neural models perform as well as simple majority model. Only the BERT model can slightly outperform the majority model. (2) For token-level difficulty, these neural models perform slightly better than the majority model. The insignificant improvement over the majority model demonstrate that, the performance of the neural models mainly come from the learning of prior distribution of label instead of extracting difficulty-related features from instances. In the case of label imbalance, the accuracy can not well measure model performance. Besides, in the context of early exiting, we are more interested in cases that the model performs a false exit for an unsolved instance. Thus, we also report the precision, recall, and F1 score on the negative class. As shown in Table 1, all the evaluated models perform poorly on recognizing the incorrectly predicted instances and tokens.

Though, it can not be concluded that the instance difficulty can not be learned since there are still a variety of machine learning models and training techniques that are under explored. Our preliminary experiments demonstrate that, at least, instance difficulty, whether human-defined or model-defined, is hard to learn for modern neural networks. In fact, our evaluated learn-to-exit models are upper baselines than that used in previous work because: (1) we also adopt more powerful deep models instead of simple linear models in previous methods (Elbayad et al., 2020; Xin et al., 2021), and (2) Different from our method that trains learn-to-exit module on development set, previous methods jointly train their learn-to-exit module on the training set where few instances are incorrectly predicted, leading to more serious label imbalance. To facilitate future research, our constructed difficulty datasets will be publicly available.

3 HASHEE: Hash Early Exiting

3.1 What is Unnecessary and What Works?

On the one hand, previous methods (Elbayad et al., 2020; Xin et al., 2021) that use learn-to-exit modules have achieved competitive results, which implies that something works in the learn-to-exit modules. On the other hand, our preliminary experiments show that instance difficulty is hard to be predicted in advance, which indicates that learning can be unnecessary to achieve a good performance.

To find what works, we formally describe the prediction of an early exiting model as $P(y|x) = \sum_{d \in \mathcal{D}} P(y|x,d)P(d|x)$, where d is the difficulty (e.g., the exiting layer) for x. Note that in practice, $P(\mathcal{D}|x)$ is an one-hot distribution, so when d is predicted, the exiting layer, i.e., the model architecture is determined. Therefore, the difficulty d actually corresponds to an architecture.[1] Now given that the mapping from instance x to its difficulty d, i.e., the best architecture, is hard to be learned, a natural idea to make $P(y|x)$ performs well is to keep $P(d|x)$ consistent: *if a training instance x_i is predicted to exit at layer l, then an inference instance x_j that is similar with x_i should exit at layer l, too.* By this, the activated architecture can well-handle the instance x_j during inference because it is well-trained on similar instances such as x_i. Note that this consistency between training and inference can be easily satisfied by previous learn-to-exit modules due to the smoothness of neural models (Ziegel, 2003). Based on this hypothesis, we manage to remove the learning process and only stick to the consistency. In particular, we replace the neural learn-to-exit module $P(d|x)$ with a simple hash function.

3.2 Method

Without loss of generality, we first consider sequence classification tasks. A straightforward idea is to design a hash function to map semantically similar instances into the same bucket, and therefore the hash function should be some powerful sequence encoder such as Sentence-BERT (Reimers and Gurevych, 2019), which is cumbersome in computation. In addition, a high-quality sequence encoder as a hash function usually maps instances with the same label into the same bucket (i.e. the same exiting layer), which makes the internal classifier at that layer suffer from label imbalance. Due to the difficulty of holding consistency at sentence-level, we rather propose to hold the consistency at token-level. By assigning each token into a fixed bucket, the token-level consistency between training and inference is easily satisfied.

An overview of our method is illustrated in Figure 3. We adopt a simple and efficient hash function to map each token into a fixed bucket in advance, where each bucket corresponds to an exiting layer.

[1] Note that this formulation is similar to some differentiable Neural Architecture Search (NAS) and Mixture-of-Expert (MoE) works, which also encountered similar difficulties in learning architectures (Wang et al., 2021; Roller et al., 2021).

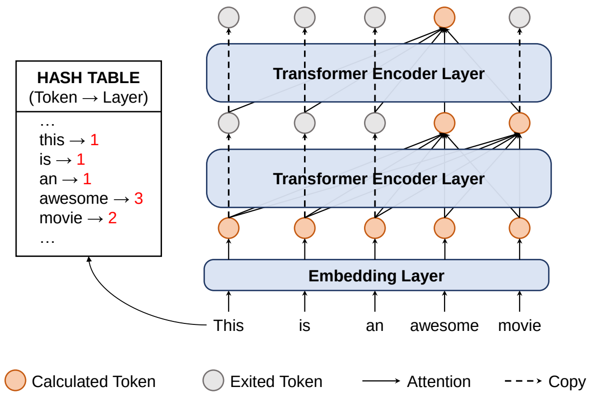

Figure 3: Overview of the Hash-based Early Exiting (HASHEE). Tokens are assigned to fixed exiting layers using a hash function.

We use pre-trained Transformers (Vaswani et al., 2017) as our backbones. During model's forward pass, the representation of exited tokens will not be updated through self-attention, and its hidden states of the upper layers are directly copied from the hidden states of the exiting layer. By this token-level early exiting, the computation in self-attention and the following feed-forward network is reduced.

3.3 Hash Functions

To hold the token-level consistency between training and inference, HASHEE employs hash functions to compute in advance the exiting layer for each token. During training and inference, each token exits at a fixed layer according to the precomputed hash lookup table. The hash functions can take a variety of forms. Here we consider several hash functions as possible alternatives.

Random Hash Random hash is a lower baseline, wherein we assign each token to a fixed, random exiting layer at initialization. To examine our hypothesis, we also consider to use two different random hash functions for training and inference respectively, in which case the consistency does not hold. We denote these two random hash functions as **Rand-cons** and **Rand-incons**.

Frequency Hash To achieve higher speed-up, a natural way is to assign frequent tokens to lower layers to exit. Intuitively, frequent tokens are usually well-trained during pre-training and therefore do not require too much refinement by looking at their contexts. Thus we can design a hash function that assigns tokens into exiting layers by frequency. In particular, the tokens are sorted by frequency and then divided equally into B buckets.

MI Hash Further, we also consider a task-specific hash function that is based on the mutual information (MI) between each token and the corresponding label, which, as an instance of HASHEE, is also adopted in Liu et al. (2021b). Tokens are sorted by their MI values between the task label, and then divided equally into B buckets. Tokens with higher MI values are assigned to lower layers.

Clustered Hash It is also intuitive that similar tokens should be assigned to the same layer to exit, and therefore we also experiment with a clustered hash function. The clusters are obtained by performing k-means clustering using token embeddings from BERT$_{\text{BASE}}$ embedding layer. The clustered tokens are then sorted by norm, which often relates to token frequency (Schakel and Wilson, 2015) and difficulty (Liu et al., 2020b). The clustered tokens with small average norms are assigned to lower layers.

4 Experiments

4.1 Tasks and Datasets

Since HASHEE requires no supervision, it can be applied to a variety of tasks and architectures. In our work, we conduct experiments on natural language understanding tasks including sentiment analysis, natural language inference, similarity regression, and a language generation task, text summarization. Statistics of our used datasets are shown in Appendix A.1.

Understanding Tasks For the convenience of comparison with other efficient models, we evaluate our proposed HASHEE on the ELUE benchmark (Liu et al., 2021a), which is comprised of SST-2 (Socher et al., 2013), IMDb (Maas et al., 2011), SNLI (Bowman et al., 2015), SciTail (Khot et al., 2018), MRPC (Dolan and Brockett, 2005), and STS-B (Cer et al., 2017)). Note that STS-B is a regression task.

Generation Tasks For language generation, we evaluate HASHEE on two English summarization datasets, CNN/DailyMail (Hermann et al., 2015) and Reddit (Kim et al., 2019), and two Chinese summarization datasets: TTNews (Hua et al., 2017) and CSL (Xu et al., 2020b).

4.2 Experimental Setup

Baselines We compare HASHEE with the following competitive baseline models: **(1) Pre-Trained**

Models	**SST-2** (8.5k)	**IMDb** (20.0k)	**SNLI** (549.4k)	**SciTail** (23.6k)	**MRPC** (3.7k)	**STS-B** (5.7k)	**ELUE Score**
Pre-Trained Language Models							
BERT-3L	79.3 (4.0×)	88.4 (4.0×)	87.1 (4.0×)	84.3 (4.0×)	76.0 (4.0×)	75.8 (4.0×)	-3.70
ALBERT-3L	82.4 (3.6×)	90.7 (3.9×)	87.8 (3.7×)	87.5 (3.9×)	80.0 (3.6×)	79.1 (3.9×)	-1.59
RoBERTa-3L	81.8 (4.1×)	90.7 (4.2×)	88.0 (3.8×)	84.9 (3.9×)	75.6 (3.9×)	67.5 (3.9×)	-2.17
ElasticBERT-3L	84.1 (4.0×)	91.8 (4.0×)	89.3 (4.0×)	91.9 (4.0×)	83.1 (4.0×)	83.5 (4.0×)	0.00
Static Models							
DistilBERT	84.8 (2.0×)	92.0 (2.0×)	89.2 (2.0×)	89.7 (2.0×)	83.8 (2.0×)	81.7 (2.0×)	-2.55
TinyBERT	85.3 (2.0×)	89.0 (2.0×)	89.3 (2.0×)	90.0 (2.0×)	**84.7** (2.0×)	**85.0** (2.0×)	-2.20
HeadPrune	84.8 (1.3×)	84.7 (1.5×)	87.8 (1.5×)	88.3 (1.5×)	77.8 (1.5×)	74.8 (1.5×)	-6.85
BERT-of-Theseus	84.4 (2.0×)	90.7 (2.0×)	89.4 (2.0×)	92.1 (2.0×)	82.4 (2.0×)	**85.0** (2.0×)	-2.55
Dynamic Models							
DeeBERT	78.9 (3.4×)	79.5 (4.1×)	48.1 (3.6×)	71.9 (3.4×)	79.1 (3.5×)	-	-
FastBERT	82.7 (3.7×)	**92.5** (3.5×)	88.8 (3.5×)	89.0 (3.6×)	80.3 (4.2×)	-	-
PABEE	83.1 (2.9×)	91.6 (3.4×)	88.7 (3.1×)	90.7 (3.3×)	75.2 (3.5×)	80.1 (3.2×)	-1.31
CascadeBERT	82.4 (3.8×)	91.8 (3.7×)	89.0 (3.6×)	91.7 (3.8×)	78.8 (3.8×)	-	-
BERxiT w/ BERT	71.8 (2.2×)	85.0 (2.8×)	88.4 (3.6×)	80.3 (3.4×)	74.9 (4.0×)	57.8 (4.0×)	-6.12
BERxiT w/ ElasticBERT	72.6 (4.4×)	91.2 (4.0×)	84.7 (3.9×)	91.0 (4.0×)	78.6 (4.3×)	81.5 (4.0×)	-3.90
Ours							
HASHEE	**85.5** (**4.8**×)	92.4 (**6.2**×)	**89.6** (**4.4**×)	**92.3** (**5.1**×)	84.0 (**4.8**×)	84.3 (**4.6**×)	**1.20**

Table 2: Main results on the ELUE benchmark (Liu et al., 2021a). We report for each model on each task the performance and the corresponding speed-up ratio, which is calculated as the FLOPs reduction relative to BERT$_{\text{BASE}}$. For MRPC, we report the mean of accuracy and F1. For STS-B, we report Pearson and Spearman correlation. For all other tasks we report accuracy. "-" indicates that the method is not applicable on that task.

Language Models. We directly fine-tune the first layers of pre-trained language models including BERT (Devlin et al., 2019), ALBERT (Lan et al., 2020), RoBERTa (Liu et al., 2019), and ElasticBERT (Liu et al., 2021a) with a MLP classifier on the top. **(2) Static Models.** We compare with several static approaches to accelerate language model inference, including DistilBERT (Sanh et al., 2019), TinyBERT (Jiao et al., 2020), HeadPrune (Michel et al., 2019), and BERT-of-Theseus (Xu et al., 2020a). **(3) Dynamic models.** We compare with DeeBERT (Xin et al., 2020), FastBERT (Liu et al., 2020a), PABEE (Zhou et al., 2020), BERxiT (Xin et al., 2021), and CascadeBERT (Li et al., 2021a).

Training For most NLU experiments we adopt the ElasticBERT$_{\text{BASE}}$ model (Liu et al., 2021a) as our backbone model, which is a pre-trained multi-exit Transformer encoder. For small datasets (i.e., SST-2, MRPC, and STS-B) we report the mean performance and the standard deviation (in Table 3 and 9) over 5 runs with different random seeds. For text summarization datasets we adopt BART$_{\text{BASE}}$ (Lewis et al., 2020) and CPT$_{\text{BASE}}$ (Shao et al., 2021) as our backbone models and use the frequency hash to assign tokens to the encoder layers. All of the experiments are conducted on GeForce RTX 3090 GPUs. More experimental details are given in Appendix A.2.

Hash Functions	**Speed -up**	**SST-2** (8.5k)	**SNLI** (549.4k)	**MRPC** (3.7k)
Backbone: ElasticBERT-6L				
Rand-incons	3.0×	85.5 (±0.53)	89.7	85.0 (±0.22)
Rand-cons	3.0×	85.7 (±0.45)	90.1	86.3 (±0.67)
Frequency	4.9×	85.5 (±0.41)	89.6	84.0 (±0.27)
MI	3.3×	85.5 (±0.49)	90.0	86.0 (±0.23)
Clustered	3.0×	85.7 (±0.50)	90.2	86.3 (±0.47)
Backbone: ElasticBERT-12L				
Rand-incons	1.6×	85.7 (±0.38)	89.6	86.6 (±0.45)
Rand-cons	1.5×	86.5 (±0.37)	90.2	87.4 (±0.34)
Frequency	2.8×	85.6 (±0.37)	89.8	84.4 (±0.17)
MI	1.8×	86.6 (±0.17)	90.1	87.2 (±0.66)
Clustered	1.5×	87.0 (±0.54)	90.1	87.3 (±0.48)

Table 3: Comparison of different hash functions. The speed-up ratios are calculated by FLOPs reduction relative to BERT$_{\text{BASE}}$ and averaged over the three tasks. The ELUE score is averaged over the three tasks. For small datasets, i.e., SST-2 and MRPC, we report the mean and standard deviation over five runs.

4.3 Results and Analysis

Results on ELUE We first show our main comparison results on ELUE test sets in Table 2. Us-

Model	Speed-up			English		Chinese	
	Enc.	Dec.	Total	Reddit	CNN/DM	CSL	TTNews
BART	1.0×	1.0×	1.0×	29.71/9.91/23.43	44.16/21.28/40.90	64.49/52.48/61.81	53.84/38.09/49.85
DAT	1.0×	0.5×	0.8×	27.02/**8.89**/**22.68**	40.30/17.77/37.53	-	-
BART-6L	2.0×	**1.4**×	**1.8**×	26.22/6.82/21.05	40.02/16.60/36.82		-
HashEE w/ BART	**3.3**×	1.0×	**1.8**×	**28.77**/8.52/21.97	**41.04/18.41/37.65**	-	-
CPT	1.0×	1.0×	1.0×	-	-	65.49/53.82/62.96	53.48/37.59/49.82
CPT-6L	2.0×	1.8×	2.0×	-	-	52.29/39.35/50.06	50.89/33.75/45.42
HashEE w/ CPT	**2.3**×	**1.5**×	**2.3**×	-	-	**62.42/49.96/59.15**	**52.67/35.31/46.97**

Table 4: Experimental results on two English and two Chinese summarization datasets. We report ROUGE-1, ROUGE-2, and ROUGE-L for each dataset. The speedup ratios for English and Chinese models are calculated by the FLOPs reduction relative to $\text{BART}_{\text{BASE}}$ and CPT_{BASE}, respectively, and averaged over the performed datasets. Here we re-implement the confidence thresholding variant of DAT (Elbayad et al., 2020).

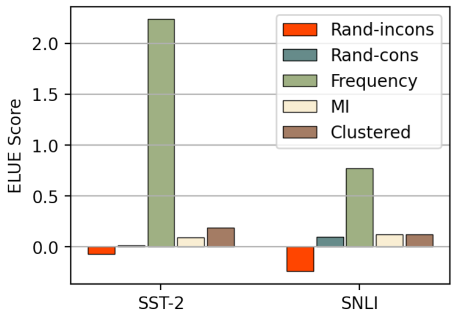

Figure 4: Comparison of the ELUE scores achieved by HashEE with different hash functions.

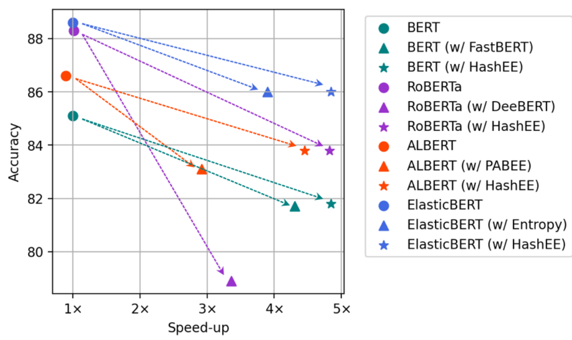

Figure 6: Comparison of different backbone models.

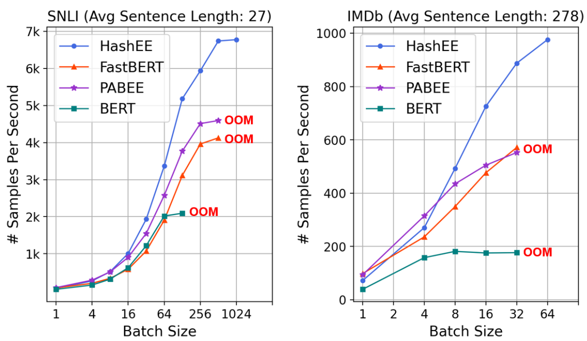

Figure 5: Comparison of actual inference time.

ing the frequency hash that assigns tokens to the first 6 layers of $\text{ElasticBERT}_{\text{BASE}}$, HashEE can outperform most considered baselines with fewer FLOPs. To fairly compare with baselines of various speedup ratios, we also report the ELUE score (Liu et al., 2021a), which is a two-dimensional (performance and FLOPs) metric for efficient NLP models, measuring how much a model oversteps ElasticBERT. Table 2 shows that HashEE achieved a new state-of-the-art ELUE score. To fairly compare with the learn-to-exit baseline we also implement BERxiT (Xin et al., 2021) with $\text{ElasticBERT}_{\text{BASE}}$.

Comparison of Different Hash Functions We then evaluate HashEE with different hash functions detailed in Section 3.3. For all these hash functions, we assign tokens to the 6 and 12 layers of ElasticBERT-6L and ElasticBERT-12L, respectively. Experimental results on SST-2, SNLI, and MRPC are given in Table 3. Among the hash functions, the frequency hash achieves the highest speedup while maintaining a considerable performance. With the backbone of ElasticBERT-12L, these hash functions, except for the frequency hash, cannot achieve considerable speedup. Besides, we find that ElasticBERT-12L did not significantly outperform ElasticBERT-6L with HashEE. We conjecture that higher layers are not good at querying information from hidden states of tokens that exit too early. In this work, we are more interested in the case of high acceleration ratio, so we adopt ElasticBERT-6L as our main backbone. To make a more intuitive comparison of these hash functions with different speedup ratios, we also show in Figure 4 the ELUE scores on SST-2 and SNLI with ElasticBERT-6L as backbone. We find that the frequency hash outperforms other hash functions by a large margin, and therefore in the following experiments we mainly use the frequency hash. Besides, only the Rand-incons hash obtains negative ELUE score, demonstrating the benefit of maintaining consistency between training and inference.

Comparison of Actual Inference Time Because most of the operations in the Transformer architecture are well optimized by modern deep learning frameworks and parallel processing hardwares such as GPU and TPU, FLOPs may not precisely reflect the actual inference time. To that end, here we also evaluate actual inference time on a single GeForce RTX 3090 GPU. Note that the speedup ratio of previous early exiting methods are usually tested on a per-instance basis, i.e. the batch size is set to 1. However, batch inference is often more favorable in both offline scenarios and low-latency scenarios (Zhang et al., 2019). Here we compare HashEE with two baselines that have similar performance, i.e., FastBERT and PABEE. Our experiments are conducted on two datasets with very different average sentence length, i.e., SNLI and IMDb. Results are given in Table 5 and Figure 5. We find HashEE has an advantage in processing speed when the batch size exceeds 8. Besides, HashEE can perform larger batch inference due to its memory-efficiency.

	SNLI (Avg Len: 27)			IMDb (Avg Len: 278)		
	Acc	FLOPs	# samples/sec	Acc	FLOPs	# samples/sec
BERT	90.4	1.0×	2093 (1.0×)	93.0	1.0×	177 (1.0×)
FastBERT	88.8	3.5×	4128 (2.0×)	92.5	3.5×	553 (3.1×)
PABEE	88.7	3.1×	4596 (2.2×)	91.6	3.4×	571 (3.2×)
HashEE	89.6	4.4×	6779 (3.2×)	92.4	6.2×	976 (5.5×)

Table 5: Maximal number of processing samples per second on a single RTX 3090 GPU with 24GB memory.

Comparison of Different Backbones To evaluate the versatility of HashEE, we also conduct experiments with other backbone models, i.e., BERT, ALBERT, and RoBERTa. As shown in Figure 6, HashEE outperforms other baselines with the same backbone.

Accelerating Seq2Seq Models Since HashEE requires no supervision, it can also be applied to seq2seq models for generation tasks. We first evaluate HashEE with $\text{BART}_{\text{BASE}}$ as our backbone on two English summarization tasks. As shown in Table 4, HashEE can achieve significant speedup for BART encoder while maintaining considerable ROUGE scores. Besides, we find that previous early exiting methods that measure the uncertainty of internal outputs would rather slow down decoder inference due to the heavy computation of prediction over large vocabulary. In addition, to further explore the speedup potential of HashEE, we also experiment with CPT (Shao et al., 2021), which has a deep encoder and a shallow decoder. Results on CSL and TTNews depict that HashEE can achieve 2.3× speedup relative to CPT while maintaining 97% ROUGE-1. We also report results of the 6-layer versions of BART (with 3 encoder layers and 3 decoder layers) and CPT (with 5 encoder layers and 1 decoder layer).

5 Related Work

Large-scale pre-trained language models (PLMs) have achieved great success in recent years. Despite their power, the inference is time-consuming, which hinders their deployment in low-latency scenarios. To accelerate PLM inference, there are currently two streams of work: (1) Compressing a cumbersome PLM through knowledge distillation (Sanh et al., 2019; Sun et al., 2019; Jiao et al., 2020), model pruning (Gordon et al., 2020; Michel et al., 2019), quantization (Shen et al., 2020), module replacing (Xu et al., 2020a), etc. (2) Selectively activating parts of the model conditioned on the input, such as Universal Transformer (Dehghani et al., 2019), FastBERT (Liu et al., 2020a), DeeBERT (Xin et al., 2020), PABEE (Zhou et al., 2020), LeeBERT (Zhu, 2021), CascadeBERT (Li et al., 2021a), ElasticBERT (Liu et al., 2021a) and other similar methods (Elbayad et al., 2020; Schwartz et al., 2020; Liao et al., 2021; Xin et al., 2021; Sun et al., 2021). Different from these methods, our proposed HashEE requires no internal classifiers (which imply extra parameters) and supervision, and therefore can be widely used in a variety of tasks and model architectures.

6 Conclusion

We first empirically study the learnability of instance difficulty, which is a crucial problem in early exiting. Based on the observation that modern neural models perform poorly on estimating instance difficulty, we propose a hash-based early exiting approach, named HashEE, that removes the learning process and only sticks to the consistency between training and inference. Our experiments on classification, regression, and generation tasks show that HashEE can achieve state-of-the-art performance with fewer computation and inference time.

References

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 632–642. The Association for Computational Linguistics.

Daniel M. Cer, Mona T. Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity - multilingual and cross-lingual focused evaluation. *CoRR*, abs/1708.00055.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing, IWP@IJCNLP 2005, Jeju Island, Korea, October 2005, 2005*. Asian Federation of Natural Language Processing.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020. Depth-adaptive transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Mitchell A. Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing BERT: studying the effects of weight pruning on transfer learning. In *Proceedings of the 5th Workshop on Representation Learning for NLP, RepL4NLP@ACL 2020, Online, July 9, 2020*, pages 143–155. Association for Computational Linguistics.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1693–1701.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Eduard H. Hovy, Mitchell P. Marcus, Martha Palmer, Lance A. Ramshaw, and Ralph M. Weischedel. 2006. Ontonotes: The 90% solution. In *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 4-9, 2006, New York, New York, USA*. The Association for Computational Linguistics.

Lifeng Hua, Xiaojun Wan, and Lei Li. 2017. Overview of the nlpcc 2017 shared task: Single document summarization. In *National CCF Conference on Natural Language Processing and Chinese Computing*, pages 942–947. Springer.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. Tinybert: Distilling BERT for natural language understanding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP 2020, Online Event, 16-20 November 2020*, pages 4163–4174. Association for Computational Linguistics.

Tushar Khot, Ashish Sabharwal, and Peter Clark. 2018. Scitail: A textual entailment dataset from science question answering. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 5189–5197. AAAI Press.

Byeongchang Kim, Hyunwoo Kim, and Gunhee Kim. 2019. Abstractive summarization of reddit posts with multi-level memory networks. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 2519–2531. Association for Computational Linguistics.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Antonio Laverghetta, Jamshidbek Mirzakhalov, and John Licato. 2020. Towards a task-agnostic model of difficulty estimation for supervised learning tasks. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: Student Research Workshop, AACL/IJCNLP 2021, Suzhou, China, December 4-7, 2020*, pages 16–23. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020.

BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7871–7880. Association for Computational Linguistics.

Lei Li, Yankai Lin, Deli Chen, Shuhuai Ren, Peng Li, Jie Zhou, and Xu Sun. 2021a. Cascadebert: Accelerating inference of pre-trained language models via calibrated complete models cascade. In *Findings of EMNLP*.

Xiaonan Li, Yunfan Shao, Tianxiang Sun, Hang Yan, Xipeng Qiu, and Xuanjing Huang. 2021b. Accelerating BERT inference for sequence labeling via early-exit. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 189–199. Association for Computational Linguistics.

Kaiyuan Liao, Yi Zhang, Xuancheng Ren, Qi Su, Xu Sun, and Bin He. 2021. A global past-future early exit method for accelerating inference of pre-trained language models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 2013–2023. Association for Computational Linguistics.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020a. Fastbert: a self-distilling BERT with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 6035–6044. Association for Computational Linguistics.

Xiangyang Liu, Tianxiang Sun, Junliang He, Lingling Wu, Xinyu Zhang, Hao Jiang, Zhao Cao, Xuanjing Huang, and Xipeng Qiu. 2021a. Towards efficient NLP: A standard evaluation and A strong baseline. *CoRR*, abs/2110.07038.

Xuebo Liu, Houtim Lai, Derek F. Wong, and Lidia S. Chao. 2020b. Norm-based curriculum learning for neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 427–436. Association for Computational Linguistics.

Yijin Liu, Fandong Meng, Jie Zhou, Yufeng Chen, and Jinan Xu. 2021b. Faster depth-adaptive transformers. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 13424–13432. AAAI Press.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 142–150. The Association for Computer Linguistics.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14014–14024.

Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. 2020. Pre-trained models for natural language processing: A survey. *SCIENCE CHINA Technological Sciences*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3980–3990. Association for Computational Linguistics.

Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason Weston. 2021. Hash layers for large sparse models.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Adriaan M. J. Schakel and Benjamin J. Wilson. 2015. Measuring word significance using distributed representations of words. *CoRR*, abs/1508.02297.

Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A. Smith. 2020. The right tool for the job: Matching model and instance complexities. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 6640–6651. Association for Computational Linguistics.

Yunfan Shao, Zhichao Geng, Yitao Liu, Junqi Dai, Fei Yang, Li Zhe, Hujun Bao, and Xipeng Qiu. 2021. CPT: A pre-trained unbalanced transformer for both chinese language understanding and generation. *CoRR*, abs/2109.05729.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-BERT: hessian based ultra low precision quantization of BERT. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8815–8821. AAAI Press.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1631–1642. ACL.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for BERT model compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 4322–4331. Association for Computational Linguistics.

Tianxiang Sun, Yunhua Zhou, Xiangyang Liu, Xinyu Zhang, Hao Jiang, Zhao Cao, Xuanjing Huang, and Xipeng Qiu. 2021. Early exiting with ensemble internal classifiers. *CoRR*, abs/2105.13792.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. 2021. Rethinking architecture selection in differentiable NAS. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. Deebert: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 2246–2251. Association for Computational Linguistics.

Ji Xin, Raphael Tang, Yaoliang Yu, and Jimmy Lin. 2021. Berxit: Early exiting for BERT with better fine-tuning and extension to regression. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pages 91–104. Association for Computational Linguistics.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020a. Bert-of-theseus: Compressing BERT by progressive module replacing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 7859–7869. Association for Computational Linguistics.

Liang Xu, Hai Hu, Xuanwei Zhang, Lu Li, Chenjie Cao, Yudong Li, Yechen Xu, Kai Sun, Dian Yu, Cong Yu, Yin Tian, Qianqian Dong, Weitang Liu, Bo Shi, Yiming Cui, Junyi Li, Jun Zeng, Rongzhao Wang, Weijian Xie, Yanting Li, Yina Patterson, Zuoyu Tian, Yiwen Zhang, He Zhou, Shaoweihua Liu, Zhe Zhao, Qipeng Zhao, Cong Yue, Xinrui Zhang, Zhengliang Yang, Kyle Richardson, and Zhenzhong Lan. 2020b. CLUE: A chinese language understanding evaluation benchmark. In *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pages 4762–4772. International Committee on Computational Linguistics.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5754–5764.

Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving. In *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*, pages 1049–1062. USENIX Association.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian J. McAuley, Ke Xu, and Furu Wei. 2020. BERT loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Wei Zhu. 2021. Leebert: Learned early exit for BERT with cross-level optimization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 2968–2980. Association for Computational Linguistics.

Eric R. Ziegel. 2003. The elements of statistical learning. *Technometrics*, 45(3):267–268.

A Appendix

A.1 Dataset Statistics

Here we list the statistics of our used language understanding and generation datasets in Table 6 and Table 7.

Tasks	Datasets	|Train|	|Dev|	|Test|
Sentiment Analysis	SST-2	8,544	1,101	2,208
	IMDb	20,000	5,000	25,000
Natural Language Inference	SNLI	549,367	9,842	9,824
	SciTail	23,596	1,304	2,126
Similarity and Paraphrase	MRPC	3,668	408	1,725
	STS-B	5,749	1,500	1,379

Table 6: Statistics of our used language understanding datasets.

Datasets	Source	# Pairs		
		Train	Dev	Test
Reddit	Social Media	41,675	645	645
CNN/DM	News	287,084	13,367	11,489
TTNews	News	50,000	-	2,000
CSL	Academic	20,000	3,000	3,000

Table 7: Statistics of our used text summarization datasets.

Tasks	LR	BSZ	Epoch	WSR	WD
SST-2	5e-5	16	3	0.1	0.1
IMDb	5e-5	32	3	0.1	0.01
SNLI	5e-5	32	3	0.1	0.01
SciTail	5e-5	32	3	0.1	0.01
MRPC	5e-5	32	4	0.1	0.01
STS-B	5e-5	16	5	0	0.1

Table 8: Best-performed hyperparameters on ELUE tasks. LR: Learning Rate. BSZ: Batch Size. WSR: Warmup Step Ratio. WD: Weight Decay.

# L	# B	Speed-up	SST-2 (8.5k)	SNLI (549.4k)	MRPC (3.7k)
12	12	2.8×	85.6 (±0.37)	89.8	84.4 (±0.17)
	6	2.9×	84.9 (±0.69)	89.7	83.7 (±0.26)
	4	3.0×	85.2 (±0.43)	89.6	83.7 (±0.15)
	3	3.0×	85.3 (±0.37)	89.7	82.9 (±0.29)
	2	3.1×	85.2 (±0.19)	89.7	82.8 (±0.40)
6	6	4.9×	85.5 (±0.41)	89.6	84.0 (±0.27)
	3	5.0×	85.2 (±0.42)	89.5	83.5 (±0.54)
	2	5.1×	85.4 (±0.33)	89.6	83.6 (±0.19)

Table 9: Comparison of different numbers of model layers and buckets with frequency hash function. "# L" and "# B" mean number of layers and number of buckets. For small datasets, i.e., SST-2 and MRPC, we report the mean and standard deviation over five runs with different random seeds.

A.2 Experimental Details

For small datasets in ELUE, i.e. SST-2, MRPC, and STS-B, we conduct grid search over batch sizes of {16, 32}, learning rates of {2e-5, 3e-5, 5e-5}, number of epochs of {3, 4, 5}, warmup step ratios of {0.1, 0.01}, and weight decays of {0.1, 0.01} with an AdamW optimizer. We select the hyperparameters that achieved the best performance on the development sets, and perform 5 runs with different random seeds to obtain the mean performance and standard deviation. For SNLI, SciTail, and IMDb, we use the same hyperparameters. All of the hyperparameters used in our language understanding experiments are given in Table 8.

For English summarization tasks, i.e., CNN/DailyMail and Reddit, we use the same hyperparameters as BART. For Chinese summarization tasks, i.e., TTNews and CSL, we use the same hyperparameters as CPT.

A.3 Additional Experimental Results

In previous experiments we assign tokens to the same number of buckets as the number of layers. Here we also explore other configurations. For each configuration, we assign tokens to B buckets, corresponding to exiting layers $\{1 + 12b/B\}_{b=0}^{B-1}$. For instance, if we have 12 layers and 3 buckets, the 3 buckets correspond to the {1, 5, 9} layers. Overall results are given in Table 9, where we show results of 8 configurations with the frequency hash. Similar with Table 3, we find that 6-layer models perform well while achieving higher acceleration ratios. In addition, the number of buckets has no significant effect on acceleration ratio. Configurations that the number of layers equals to the number of buckets perform slightly better than other configurations.

A.4 Details on FLOPs Calculation

Here we take a closer look at the HASHEE model forward process, and see which FLOPs are saved during inference.

Given the hidden states at layer l as $\mathbf{H}^l \in \mathbb{R}^{n\times d}$ and the hidden states of remaining tokens are denoted as $\mathbf{h}^l \in \mathbb{R}^{m\times d}$, where n is the original sequence length and m is the number of remaining tokens at layer l, the calculation of one Transformer encoder layer with HASHEE can be formally described as

$$\mathbf{q}_i, \mathbf{K}_i, \mathbf{V}_i = \mathbf{h}^l \mathbf{W}_i^Q, \mathbf{H}^l \mathbf{W}_i^K, \mathbf{H}^l \mathbf{W}_i^V, \quad (1)$$

$$\mathbf{x}_i = \mathrm{Softmax}(\frac{\mathbf{q}_i \mathbf{K}_i^\top}{\sqrt{d_k}}) \mathbf{V}_i, \quad (2)$$

$$\mathbf{x} = \mathrm{Concat}(\mathbf{x}_1, \cdots, \mathbf{x}_h) \mathbf{W}^O, \quad (3)$$

$$\mathbf{h}^{l+1} = \mathrm{ReLU}(\mathbf{x}\mathbf{W}_1)\mathbf{W}_2, \quad (4)$$

$$\mathbf{H}^{l+1} = \mathrm{Copy}(\mathbf{H}^l, \mathbf{h}^{l+1}), \quad (5)$$

where we lowercase the representations with reduced shape, i.e., $\mathbf{q}_i, \mathbf{x}_i \in \mathbb{R}^{m \times d_k}$, $\mathbf{x}, \mathbf{h} \in \mathbb{R}^{m \times d}$. d_k is the dimension of each attention head. $\mathrm{Copy}(\mathbf{H}^l, \mathbf{h}^{l+1})$ is to copy the hidden states of the exited tokens from $\mathbf{H}^l$ and concatenate with the updated hidden states $\mathbf{h}^{l+1}$. By this token-level early exiting, the computation in self-attention and the following feed-forward network is reduced.

In particular, we show in Table 10 the saved MACs (Multiply–Accumulate Operations) in each module of one Transformer encoder layer. We estimate FLOPs with twice the MACs.

Module		Saved MACs
SelfAttn	LinearProj	$(n-m)d^2$
	MultiHeadAttn	$2n(n-m)(h+d)$
	OutProj	$(n-m)d^2$
	LayerNorm	$2(n-m)d$
FFN	FFN	$2(n-m)dd_{ff}$
	LayerNorm	$2(n-m)d$

Table 10: Saved MACs in one Transformer encoder layer. Here we assume $hd_k = d$. d_{ff} is the hidden size of the Feed-Forward Network (FFN) sublayer.