ConsistentEE: A Consistent and Hardness-Guided Early Exiting Method for Accelerating Language Models Inference

Ziqian Zeng[*1], **Yihuai Hong**[*1], **Hongliang Dai**[2], **Huiping Zhuang**[1], **Cen Chen**[1,3†]
[1]South China University of Technology, China
[2]Nanjing University of Aeronautics and Astronautics, China
[3]Pazhou Laboratory, China
{zqzeng, hpzhuang, chencen}@scut.edu.cn, yihuaihong@gmail.com, hongldai@nuaa.edu.cn

Abstract

Early Exiting is one of the most popular methods to achieve efficient inference. Current early exiting methods adopt the (weighted) sum of the cross entropy loss of all internal classifiers as the objective function during training, imposing all these classifiers to predict all instances correctly. However, during inference, as long as one internal classifier predicts an instance correctly, it can accelerate without losing accuracy. Thus, there is a notable gap between training and inference. We propose ConsistentEE, an early exiting method that is consistent in training and inference. ConsistentEE formulates the early exiting process as a reinforcement learning problem. A policy network is added to decide whether an instance should exit or continue. The training objective of ConsistentEE only requires each instance to be predicted correctly by one internal classifier. Additionally, we introduce the concept "Memorized Layer" to measure the hardness of an instance. We incorporate the memorized layer into reward function design, which allows "easy" instances to focus more on acceleration while "hard" instances to focus more on accuracy. Experimental results show that our method outperforms other baselines on various classification and generation tasks using PLMs and LLMs as backbones respectively.

Introduction

Recently, pre-trained language models (PLMs) (Devlin et al. 2019; Liu et al. 2019; Yang et al. 2019; Brown et al. 2020) and large language models (LLMs) (Ouyang et al. 2022) have become fundamental building blocks in the field of natural language processing (NLP). As the scales of these models continue to grow, their performance improves but their inference speed slows down. This hinders their application in resource-limited scenarios. To address this problem, many efforts have been made to achieve efficient inference. There are two lines of work: static and dynamic approaches (Zhou et al. 2020). Static approaches design lightweight architectures or compress the models while dynamic approaches aim to make adaptive inference for each instance. Static approaches include weights pruning (Michel, Levy, and Neubig 2019; Voita et al. 2019; Fan, Grave, and Joulin 2020), quantization (Kim et al. 2021; Yao et al. 2022; Xiao et al. 2023), and knowledge distillation (Sanh et al. 2019; Sun et al. 2019; Jiao et al. 2020). Dynamic approaches (Xu and McAuley 2023) include token skipping (Goyal et al. 2020; Kim and Cho 2021; Ye et al. 2021; Kim et al. 2022; Guan et al. 2022; Sun et al. 2022), and early exiting (Zhou et al. 2020; Xin et al. 2021; Zhang et al. 2022).

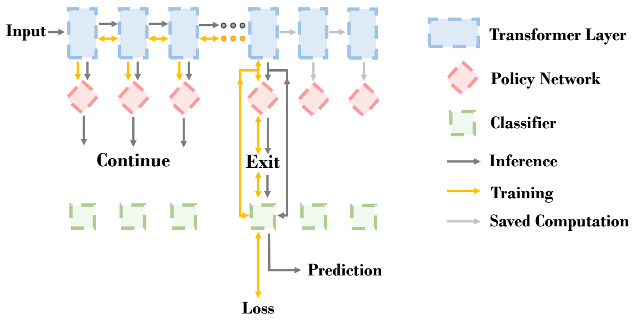

Figure 1: The training and inference procedure of ConsistentEE which formulates the early exiting process as a reinforcement learning problem. A policy network can make two possible actions, i.e., to exit, or to continue. If it exits, the corresponding internal classifier is required to predict the instance correctly, otherwise, no loss is incurred by the corresponding internal classifier.

As one of the most popular methods, early exiting adds an internal classifier to each intermediate layer, allowing instances to stop model inference in an early layer instead of going through the entire model, thus accelerating the inference time. Early Exiting can be useful, especially in the era of LLMs. Although the scale of LLMs continues to increase to deal with challenging tasks like reasoning, planning, etc., in the real-world scenario, they may not always encounter such challenging tasks (Schuster et al. 2022). A small sized LLM has been good at handling "easy" tasks such as paraphrasing, translation, etc. When LLMs encounter "easy" tasks, early exiting can avoid going through the entire model, thus saving inference time.

Current early exiting methods typically adopt the (weighted) sum of the cross entropy (CE) loss of all internal classifiers as the training loss, which imposes that all internal classifiers should predict all instances correctly. How-

[*]These authors contributed equally.
[†]Corresponding author.

ever, our experimental analysis (in Figure 2 left) shows that the classification accuracy of each internal classifier is not as satisfactory as expected. This observation raises a question: is it necessary to require all internal classifiers to predict each instance correctly? Actually, in the inference phase, as long as one internal classifier predicts an instance correctly, the inference can be accelerated without sacrificing accuracy. Given that there is no such strict requirement during inference, it is reasonable to abandon such a demanding objective during the training phase.

We propose ConsistentEE, an early exiting method that is consistent in training and inference. Specifically, ConsistentEE formulates the training process as a reinforcement learning (RL) problem. As shown in Figure 1, a policy network is introduced at each intermediate layer to determine whether to exit or not. If a policy network at an intermediate layer decides to exit, the corresponding internal classifier is expected to predict this instance correctly, otherwise no loss is incurred by this classifier. During training, an instance can exit at only one layer. Hence, only one internal classifier is required to predict it correctly. As depicted in Figure 2 right, unlike existing methods, the ConsistentEE objective enables each layer's classification accuracy to consistently remain at a high level.

For the reward function design, we consider both accuracy and acceleration. To ensure accuracy, we incorporate the loss of the internal classifier into the reward function, assigning higher reward values to instances with lower losses. Additionally, to consider acceleration, we involve the layer depth at which an instance exits into the reward function. Although one can place a trade-off coefficient to balance accuracy and acceleration in the reward function, we argue that instances of different hardness levels should put different weights on accuracy and acceleration. We observe that "easy" instances generally can be classified correctly at shallow layers. Those instances should exit as early as possible after they can be classified correctly. "Hard" instances typically can be classified correctly at deep layers. Those instances should focus on accuracy instead of acceleration at early layers. Hence, we also incorporate the hardness level of an instance into the reward function design.

However, the identification of "easy" and "hard" instances is itself a difficult problem and is extensively studied in the literature (Kumar, Packer, and Koller 2010; Arpit et al. 2017; Toneva et al. 2019). Inspired by the concept of unforgettable example (Toneva et al. 2019), we propose a new concept, *Memorized Layer*, to measure the hardness. The memorized layer is the layer where the instance is correctly classified and continuously correctly classified until the final layer. Experimental analysis reveals a high and medium correlation between the memorized layer and losses, and forgetting events respectively. Our reward function incorporates the "memorized layer" to encourage "easy instances" to pay more attention to acceleration while "hard instances" to focus on accuracy rather than acceleration.

The contributions are summarized as follows,

- We propose an early exiting method that can achieve consistency during training and inference by formulating the early exiting problem as a reinforcement learning problem.
- We propose a concept named *Memorized Layer* to measure the hardness of an instance. We incorporate it into the reward function to allow an instance to balance the accuracy and acceleration depending on individual hardness.
- The experimental results show that our method can outperform other baselines on natural language understanding and generation tasks.

Code: https://github.com/ZeroNLP/ConsistentEE.

Related Work

Early Exiting

Early exiting methods insert an internal classifier to each intermediate layer, allowing instances to exit at an early classifier rather than at the final classifier. According to the exiting criterion, early exiting methods can be categorized into three types: confidence-based or entropy-based, ensemble-based, and learning-based exiting.

Confidence-based early exiting methods utilize confidence, entropy, or (calibrated) max class probability to exit. In DeeBERT (Xin et al. 2020), FastBERT (Liu et al. 2020), RomeBERT (Geng et al. 2021) and Past-Future (Liao et al. 2021), the instance exits if the entropy is less than a predefined threshold. Right-Tool (Schwartz et al. 2020), SkipBERT (Wang et al. 2022), and CascadeBERT (Li et al. 2021) use (calibrated) max class probability as the exiting criterion.

Ensemble-based early exiting methods utilize predictions from multiple internal classifiers to make better decisions. In PABEE (Zhou et al. 2020), the instance exits when k consecutive internal classifiers make the same prediction. PCEE-BERT (Zhang et al. 2022) combined both ensemble-based and confidence-based exiting criteria. The instance exits if the confidence scores are greater than a predefined threshold for several consecutive exits.

Learning-based methods aim to learn a criterion for early exiting. Our method also falls into this category. BERxiT (Xin et al. 2021) trained a learning-to-exit (LTE) module to predict certainty level, indicating the extent to which the internal classifier can accurately predict the ground truth. When the output of the LTE is greater than 0.5, an instance can exit. CAT (Schuster et al. 2021) introduces a "meta consistency classifier" to predict the conformity level, i.e., whether the output of an internal classifier is consistent with the final classifier. An instance exits when the conformity level is greater than a threshold. The policy network in ConsistentEE is different from LTE in BERxit and the meta consistency classifier in CAT in two aspects. First, the purposes are different. LTE aims to predict certainty level, and meta consistency classifier aims to predict conformity level, while the policy network determines whether to exit. Second, the ground truth of certainty level and conformity level are available while the ground truth of the layer at which an instance should exit is unknown.

There are two types of training objectives in the above methods, i.e., the weighted sum of cross-entropy losses and the sum of cross-entropy losses. SkipBERT (Wang et al. 2022), PABEE (Zhou et al. 2020), Past-Future (Liao et al. 2021), PCEE-BERT (Zhang et al. 2022), and LeeBERT (Zhu 2021)) used weighted sum of cross entropy losses. DeeBERT (Xin et al. 2020), Right-Tool (Schwartz et al. 2020), BERxiT (Xin et al. 2021), and CAT (Schuster et al. 2021) used sum of cross entropy losses. Both objectives require all internal classifiers to predict all instances correctly.

The above methods are layer-wise early exiting methods where the entire input (i.e., all tokens) exits at the same layer. There are some token-wise early exiting methods such as HashEE (Sun et al. 2022) and TR-BERT (Ye et al. 2021) where different tokens can exit at different layers. When a token exits, the attention computation will not be performed on this token at later layers. Hence the acceleration comes from the reduced computation cost in the attention module. In this sense, token-wise early exiting methods belong to the family of token skipping methods (Goyal et al. 2020; Kim and Cho 2021; Kim et al. 2022; Guan et al. 2022).

The above methods are early exiting encoding methods to accelerate BERT inference. CALM (Schuster et al. 2022) and Free (Bae et al. 2023) focused on early-exiting for autoregressive models that allocates adaptive computation paths for each token generation. CALM proposed three confidence measures as exiting criteria including Softmax response, hidden-state saturation, and early exit classifier. A calibration procedure is proposed to find a shared exit threshold. Free (Bae et al. 2023) is concurrent with our work. FREE only allowed two exit points and replaced the state copying mechanism with synchronized parallel decoding to prevent performance degradation.

Methodology

Background and Motivation

The early exiting method adds an internal classifier to each intermediate layer, allowing instances to stop model inference at an early layer without going through the rest of the layers, thus accelerating the inference time. The traditional training objective function is a weighted sum of the cross-entropy loss of each layer.

$$\mathcal{L} = \sum_{(x,y)\in\mathcal{D}} \frac{\sum_{i=1}^{L} i \cdot l_i}{\sum_{i=1}^{L} i}, \tag{1}$$

$$l_i = H(y, P_i(x)), \tag{2}$$

where $(x, y) \in \mathcal{D}$ is the input-label pair in the dataset, l_i is the cross entropy loss of i-th internal classifier, $H(\cdot)$ is the cross entropy function, y is the ground truth distribution represented as a one-hot vector, $P_i(x)$ is the probability distribution produced by the i-th internal classifier. The earlier internal classifiers have less learning capacity, hence smaller weights are put on them (Kaya, Hong, and Dumitras 2019; Zhou et al. 2020). There are also some methods (Xin et al. 2020; Schwartz et al. 2020; Xin et al. 2021; Schuster et al. 2021) that do not consider varying learning capacities and assign an equal weight to each internal classifier.

During training, this loss (Eq. 2) imposes that all internal classifiers should predict all instances correctly. To investigate whether such an objective is suitable, we analyze the loss values and accuracy of each internal classifier on the RTE dataset. As shown in Figure 2 (left), with the traditional training loss, the internal classifiers do not perform as expected. The accuracy of shallow layers is unsatisfactory. Especially, the first three layers fall below the random guess baseline, i.e., 50%. We think that the traditional objective is too strict that the requirement may have exceeded the limits of the capabilities of internal classifiers, thus harming their performance.

The experimental analysis provokes a question: whether it is necessary to require all internal classifiers to predict each instance correctly? Actually, in the inference phase, as long as one internal classifier predicts an instance correctly, it can accelerate without sacrificing accuracy. We therefore propose ConsistentEE, an early exiting method that is consistent in training and inference. With ConsistentEE, for a training instance, only one internal classifier is required to predict it correctly and this instance will exit at the corresponding layer. ConsistentEE allows each classifier to focus on correctly classifying partial instances, reducing the burden of each layer. As shown in Figure 2, the training losses under ConsistentEE are consistently lower than those under the traditional training objective at every layer. Additionally, the accuracy of each layer in the training set remains at a significantly high level. The accuracy of layer 1 to 4 is not available becauase if the policy network chooses not to exit at these layers, then there is no loss and accuracy in these layers. RTE is a challenging dataset, and no samples exit at these layers.

ConsistentEE

The primary challenge in ConsistentEE is to determine the most appropriate layer for an instance to exit. As the ground truth exit layer is unavailable, ConsistentEE employs the reinforcement learning (RL) method to automatically learn the optimal layer for an instance to exit during training. As shown in Figure 1, a policy network is introduced to an intermediate layer besides an internal classifier. This policy network shares the same input with the internal classifier. It produces the probability distribution of two actions. One is exiting at the current layer, and the other one is continuing to the next layer. If the policy network takes an action to exit, then the internal classifier at the same layer is required to predict the instance correctly. Otherwise, if the policy network takes an action to continue, then there is no loss imposed on the corresponding internal classifier.

The training objective of ConsistentEE involves two parts. The first part aims to optimize the policy network so that it can make a good action, i.e., exit or continue. The second part aims to optimize the internal classifier so that it can classify the instance correctly. Only the internal classifier at the layer at which an instance decides to exit can incur a loss.

Early exiting can be regarded as a sparse reward process because rewards are only received when an instance decides to exit at a particular layer, with no feedback provided for intermediate actions. Considering this characteristic, we use the Policy Gradient technique to automatically discover the optimal exit layer for each instance. We introduce relevant notations in the following.

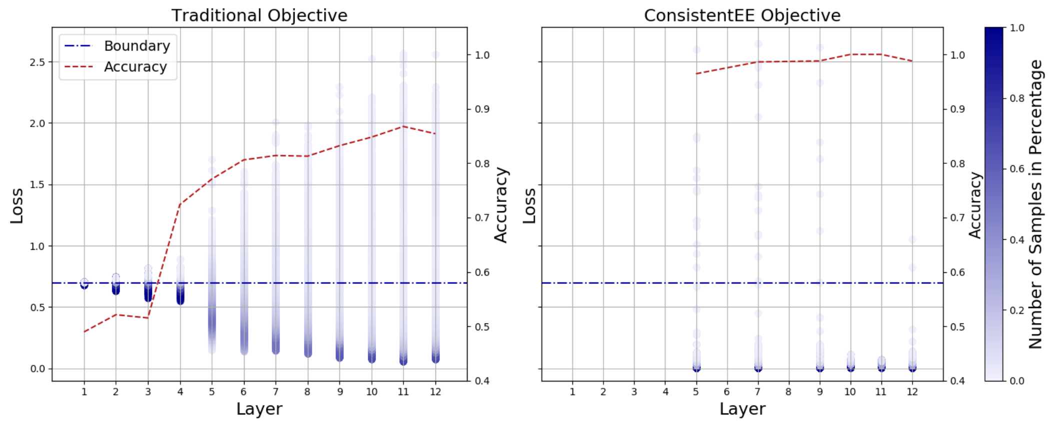

Figure 2: Loss values at different layers on the RTE dataset using the weighted sum objective and ConsistentEE objective respectively. The dashed dot line is the classification boundary. A loss above the boundary means misclassification. The dashed line is the classification accuracy of each layer. The darker the color, the more samples share the same loss value.

State s_t is the representation of the input at t-th layer. For the classification task with PLMs as backbones, s_t is the representation of `[CLS]` token at t-th layer. For the generation task with decoder-only LLMs as backbones, s_t is the hidden state of the last token of the sequence at t-th layer.

Action a_t is the action taken at the t-th layer. The action space is $\{\text{Exit, Continue}\}$. An instance can exit at the current layer or continue to the next layer.

Reward The reward function should consider accuracy and acceleration. For accuracy, we adopt the likelihood of predicting the ground truth as a part of the reward. A larger likelihood of predicting the ground truth leads to a larger reward. For acceleration, we involve the depth of the layer at which the instance exits. If an instance exits early (late), then it can gain a larger (smaller) reward. The vanilla version of reward R_{vanilla} is defined as:

$$R_{\text{vanilla}} = \begin{cases} -H(y, P_t(x)) - \alpha \cdot t & a_t = \text{Exit} \\ 0 & a_t = \text{Continue} \end{cases}, \tag{3}$$

where α is the trade-off coefficient.

In the vanilla version of the reward function, each instance has the same trade-off coefficient to balance the accuracy and acceleration. However, "easy" instances generally can be correctly classified at shallow layers. Such instances should exit as early as possible after they can be correctly classified. "Hard" instances typically can be classified correctly at deep layers. Such instances should focus on accuracy instead of acceleration at early layers. Hence, instances of different hardness levels should put different weights on accuracy and acceleration. Based on this idea, we improve the reward function by taking instance hardness into account:

$$R = \begin{cases} -H(y, P_t(x)) - \alpha \cdot (1 - \frac{\mathcal{M}}{L}) \cdot t & a_t = \text{Exit} \\ 0 & a_t = \text{Continue} \end{cases}, \tag{4}$$

where L is the total number of layers; $\mathcal{M}$ denotes *Memorized Layer*, a new concept we introduced to measure the hardness of an instance. A smaller (larger) value of $\mathcal{M}$ indicates an easier (harder) instance. More details about this concept will be introduced in the next section. With Eq. 4, for a hard (easy) instance, the trade-off coefficient $\alpha \cdot (1 - \frac{\mathcal{M}}{L})$ is expected to be smaller (larger), causing it to receive more (less) weight on accuracy compared to the vanilla version.

Policy Network $\pi(a_t|s_t;\theta)$ produces the probability of taking action a_t given the current state s_t. The Policy network is parameterized by θ.

Policy Objective Function We optimize the policy network to maximize the expected reward. The policy objective function is defined as:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi(a_t|s_t;\theta)} \left[R(\tau) \cdot \prod_{t=1}^{T} \pi(a_t|s_t;\theta) \right], \tag{5}$$

where τ are the trajectories $(s_1, a_1) \cdots (s_T, a_T)$, T is the number of states in the trajectories. Note that $T \leq L$. When $a_T = \text{Exit}$, trajectories terminate at step T. There is no further action in the remaining layers. The maximum length of the trajectory is the number of layers. We adopt repeated sampling and ϵ-greedy techniques to allow each instance to have multiple trajectories. Eq 5 is optimized on trajectories generated by all data in the dataset.

Task Objective Function We optimize the internal classifiers and the backbone to accomplish the task (e.g., classification, generation, etc.) The internal classifiers and the backbone are parameterized by ω. The task objective function is defined as:

$$J(\omega) = \sum_{(x,y)\in\mathcal{D}} \sum_{t=1}^{T} H(y, P_t(x)) \cdot \mathbb{1}(a_t = \text{Exit}), \tag{6}$$

where $\mathbb{1}(\cdot)$ is an indicator function that returns 1 if the condition is satisfied. Otherwise, it returns 0.

Memorized Layer and Hardness of Instance

Identifying easy and hard instances is the core problem in curriculum learning and has been extensively studied. (Kumar, Packer, and Koller 2010; Arpit et al. 2017) use losses

Spearman Correlation	Loss	Forgetting Events
RTE	0.76	0.59
MRPC	0.84	0.43
SST-2	0.66	0.32

Table 1: Spearman Correlation between memorized layer and loss, and between memorized layer and forgetting events on RTE, MRPC, and SST-2 datasets.

at some points during training to measure the hardness of instances. (Toneva et al. 2019) proposed a concept named unforgettable example which is predicted correctly at some point and is persistently correct until the end of training. Unforgettable examples are typically learned in the early stage of training, and may have a low loss during most of the training process. As a result, they may be considered as easy instances according to (Kumar, Packer, and Koller 2010; Arpit et al. 2017).

Inspired by the concept of unforgettable examples (Toneva et al. 2019), we propose a new concept named *Memorized Layer* to measure the hardness of the instance. If an instance is correctly classified at a certain layer and remains correctly classified until the final layer, we consider this instance to be successfully memorized at that layer. If an instance is memorized at an early (late) layer, we consider it easy (hard). We define the layer at which an instance starts to be memorized as *memorized layer*, denoted as $\mathcal{M}$.

$$\mathcal{M} = k : \forall i \geq k,\ \hat{y}_i = y, \tag{7}$$

where $\hat{y}_i$ is the prediction distribution of i-th internal classifier in a one-hot vector form. In the case that an instance is never predicted correctly, then $\mathcal{M} = L$, where L is the total number of layers of the backbone. Note that $1 \leq \mathcal{M} \leq L$, which means it is a bounded variable. The hardness of the instance can be quantified by the memorized layer.

Table 1 shows that Spearman's ρ correlation between memorized layer and loss (Arpit et al. 2017), memorized layer and forgetting events (Toneva et al. 2019) on three different datasets. An example undergoes a forgetting event when it was correctly classified at step $t-1$ but is misclassified at the current step t. We observe a high correlation between memorized layer and loss (Arpit et al. 2017) and a medium correlation between memorized layer and forgetting events, which indicates that memorized layer generally agrees with other measures of hardness.

Model Training and Inference

During training, we adopt the **iterative training** technique which iteratively improves the capacity of the policy network and the internal classifiers until convergence is reached. The training process is shown as follows:

1. Initialization. We adopt the weighted sum of CE losses as the objective function (Eq. 2) to obtain a good initialization on the internal classifiers and a good estimation on memorized layer.
2. Memorized Layer. We calculate memorized layer according to Eq. 7.

Dataset	Classes	Train/Dev/Test
RTE	2	2.5k/0.3k/3.0k
MRPC	2	3.7k/0.4k/1.7k
SST-2	2	67k/0.9k/1.8k
QNLI	2	105k/5.5k/5.5k
QQP	2	364k/40k/391k
MNLI	3	393k/9.8k/9.8k
Alpaca	-	52k/ - / -
Dolly	-	- /7.5k/7.5k
CNN/DM	-	287k/13.7k/11.5k

Table 2: Statistics of datasets.

3. Policy Network. We optimize the policy network by maximizing Eq. 5. The internal classifiers and the backbone are frozen.
4. Internal Classifiers and Backbone. We optimize the internal classifiers and backbone by minimizing Eq. 6. The policy network is frozen.
5. Iterative Training. Repeat steps 2, 3, and 4 util the convergence is reached.

During inference, if the probability of taking the action to exit at a particular layer is greater than 0.5, then the instance will exit at that layer, and we consider the prediction of the internal classifier at that layer as the final prediction.

Experiment

Experimental Settings

To evaluate acceleration capacities on the classification task with PLMs as backbones, we conduct experiments on six classification datasets of the GLUE benchmark (Wang et al. 2019).

Statistics of these datasets are listed in Table 2. We compared our method with various baselines including DeeBERT(Xin et al. 2020), PABEE (Zhou et al. 2020), BERxiT (Xin et al. 2021), Right-Tool (Schwartz et al. 2020), PCEE-BERT (Zhang et al. 2022), HashEE (Sun et al. 2022), and TR-BERT (Ye et al. 2021). For layer-wise exiting methods, we use saved layers to evaluate model acceleration. According to (Xin et al. 2020), saved layers are linearly proportional to actual runtime. While for token-wise exiting methods, we use saved runtime. Most results are reported on the development set, since the large number of evaluations are restricted by the GLUE evaluation server. Only BERT-Base (Devlin et al. 2019), three competitive baselines, and ConsistentEE are evaluated on the test set.

To evaluate acceleration capacities on the generation task with LLMs as backbones, we perform the supervised fine-tuning step using Alpaca (Taori et al. 2023) as training data and test on the Dolly dataset (Conover et al. 2023). We also evaluate our method on text summarization task on CN-N/DM dataset (Nallapati et al. 2016). The backbone LLMs are LLaMA-7b and LLaMA-13b (Touvron et al. 2023). Due to our limited computational resources, we train the LLM backbone with LoRA (Hu et al. 2022) in the initialization

Method	Averaged		RTE		MRPC		SST-2		QNLI		QQP		MNLI	
	Score	Layer	Acc	Layer	F1	Layer	Acc	Layer	Acc	Layer	F1	Layer	Acc	Layer
Dev Set														
BERT-Base	85.3	12	66.4	12	89.6	12	91.8	12	90.0	12	89.6	12	84.3	12
Layer-wise														
DeeBERT	83.9	-33%	65.7	-31%	88.2	-35%	89.6	-35%	89.1	-22%	87.5	-45%	83.3	-27%
PABEE	83.9	-28%	65.0	-18%	88.5	-31%	89.8	-33%	89.8	-18%	88.0	-35%	82.4	-23%
BERxiT	84.4	-35%	66.1	-31%	88.4	-40%	89.4	-38%	90.0	-23%	89.1	-43%	83.4	-35%
Right-Tool	84.0	-30%	65.3	-17%	88.2	-36%	90.1	-31%	89.4	-37%	87.9	-32%	82.9	-30%
PCEE-BERT	84.3	-30%	65.5	-26%	88.4	-17%	90.5	-33%	89.4	-38%	88.3	-33%	83.7	-32%
Token-wise														
HashEE ⏱	82.6	-10%	63.2	-9%	85.6	-10%	90.3	-4%	88.4	-10%	85.5	-6%	82.6	-19%
TR-BERT ⏱	83.6	-9%	65.5	-6%	87.7	-5%	90.6	-6%	88.6	-11%	86.3	-9%	83.1	-15%
ConsistentEE ⏱	**84.5**	**-36%**	65.7	-39%	88.6	-32%	90.7	-36%	89.3	-34%	89.2	-43%	83.6	-30%
ConsistentEE	**84.5**	**-53%**	**65.7**	**-55%**	**88.6**	**-53%**	**90.7**	**-58%**	**89.3**	**-57%**	**89.2**	**-55%**	**83.6**	**-39%**
ConsistentEE ⋆	86.6	-34%	**66.4**	**-44%**	89.9	-30%	92.0	-35%	**90.4**	**-42%**	90.2	-43%	90.9	-11%
Test Set														
BERT-Base	85.6	12	69.1	12	88.9	12	93.1	12	89.8	12	89.2	12	83.2	12
PABEE	84.2	-19%	67.6	-23%	88.0	-17%	90.5	-24%	89.1	-18%	88.4	-22%	81.4	-12%
BERxiT	84.4	-19%	67.9	-21%	87.8	-14%	90.3	-27%	89.4	-20%	88.7	-16%	82.4	-14%
DeeBERT	83.9	-17%	67.2	-22%	87.3	-11%	89.9	-26%	88.6	-20%	88.2	-13%	82.0	-9%
ConsistentEE	**85.5**	**-41%**	**69.0**	**-46%**	**89.0**	**-37%**	**92.9**	**-46%**	**89.9**	**-42%**	**89.0**	**-45%**	**83.4**	**-31%**

Table 3: Comparison among ConsistentEE and baselines on six datasets. The evaluation metrics for model quality are accuracy or F1 scores. The evaluation metrics for model acceleration are saved layers/runtime (w.r.t BERT-Base). Methods using saved runtimes as metrics are marked with ⏱. Methods with no accuracy loss are marked with ⋆. Results which exhibit better accuracy and saved layers than other methods are highlighted in bold.

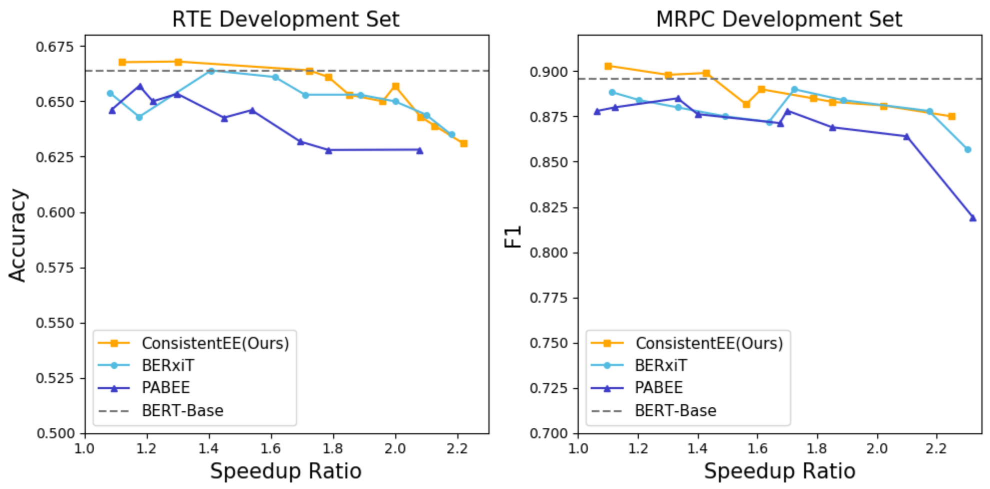

Figure 3: Accuracy-Speed curves on the BERT-Base model. The evaluation metric for speedup is saved layers.

step. Subsequently, we freeze the LLM backbone and only train the internal classifiers and policy networks. To save pre-processing time, we adopt the vanilla reward function rather than the hardness-guided reward function because the time complexity of computing the memorized layer is linear to the number of tokens in generated responses. We set the beam to 1, top-sampling to 0.75, top-k selection to 40 and temperature to 1. The baseline is CALM (Schuster et al. 2022). We use Rouge-L (Lin 2004) and BERT-F (Zhang et al. 2019) scores to measure the model quality and use save layers to measure the model acceleration. When we compute Rouge-L and BERT-F, we use the response without acceleration as the reference on Alpaca/Dolly dataset and use the gold summary as the reference on CNN/DM. Following

(Elbayad et al. 2020; Schuster et al. 2022), we use a state copying strategy to handle the computation of hidden states. Specifically, the computation of the input hidden state of token t at layer i relies on the output hidden states of the previous layer $i-1$ for all preceding tokens $\{1, \cdots, t-1\}$. In cases where some preceding tokens exit before the $i-1$ layer, we copy the output hidden state at the layer at which the token exits, and use it as the output hidden state in $i-1$ layer.

We use a two-layer Multi-Layer Perceptron (MLP) to implement the policy network.

Main Results on Classification

We evaluate the inference capabilities of ConsistentEE and baselines on the six datasets. The result of comparisons are shown in Table 3. On the development set, with no accuracy loss, the averaged saved layers of ConsistentEE is 34%, which shows that our method can achieve 1.52x acceleration without sacrificing accuracy. If a little loss (1% w.r.t the performance of BERT-Base) of accuracy is tolerated, the averaged saved layers of ConsistentEE is 53%, which outperforms the best baseline BERxiT (Xin et al. 2021) by 18%. ConsistentEE surpasses token-wise baselines averagely by 26% in terms of runtime performance. On the test set, ConsistentEE also outperforms BERxiT.

Figure 3 shows the accuracy-speed curves of ConsistentEE, PABEE, and BERxiT on two datasets. Notably, ConsistentEE mostly has higher accuracy than PABEE and BERxiT under different speed-up ratio, showing its superiority over the baselines. Furthermore, we have observed that when the speedup ratio is not overly demanding, the accuracy of ConsistentEE surpasses that of BERT-base. This finding suggests that ConsistentEE is a cautious method that prioritizes accuracy over acceleration. It ensures high accuracy while also offering the potential for speed improvements.

Ablation Study

As depicted in Figure 4, the performance of the vanilla reward and the hardness-guided reward are comparable when the speedup ratio is small. However, as the speedup ratio increases, the performance of the vanilla reward deteriorates significantly, while the hardness-guided reward maintains a satisfactory level of accuracy. The possible reason is that the vanilla reward treats each instance equally, causing hard instances struggle to maintain accuracy as they prioritize acceleration as easy instances do. In contrast, the hardness-guided reward encourages instances to assign different weights to accuracy and acceleration based on their individual hardness.

Main Results on Generation

As shown in Figure 5, on the Alpaca/Dolly dataset, ConsistentEE and CALM demonstrate similar performance when the speedup ratio is below 2x. However, as the speedup ratio increases, ConsistentEE outperforms CALM. On the CNN/DM dataset, ConsistentEE outperforms CALM consistently under different speedup ratios. Due to space limitation, we show responses generated from ConsistentEE under different speedup ratios in the Arxiv version[1].

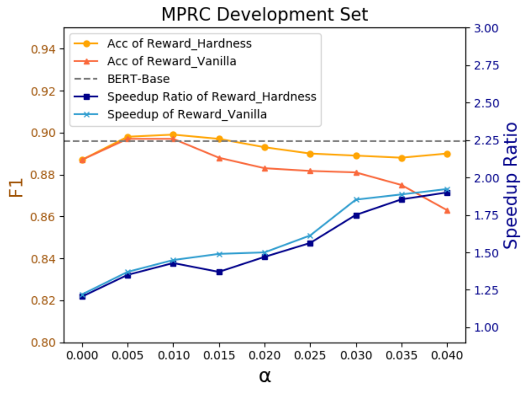

Figure 4: Accuracies and speedup ratios of different reward functions under varied α.

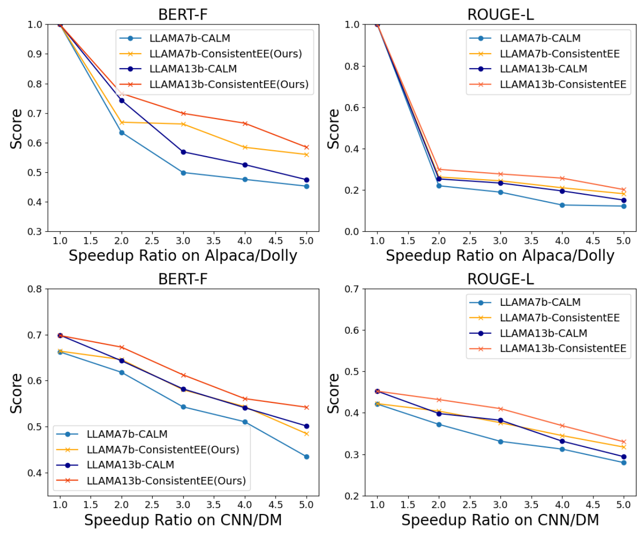

Figure 5: BERT-F and Rouge-L and speedup ratios of CALM and ConsistentEE.

Conclusion

We propose a reinforcement learning based approach to early exiting, so that at the training phase, only one internal classifier is required to predict the instance correctly. This makes the training phase consistent with the inference phase and can allow each layer to obtain better classification accuracy. For the reward function of the reinforcement learning framework, we propose the concept "memorized layer" to measure the hardness of each instance, and use it to dynamically balance accuracy and acceleration instead of using a fixed coefficient. Experimental results conducted on various datasets show that our approach is able to outperform the competitive baselines, demonstrating its effectiveness.

[1]https://arxiv.org/abs/2312.11882

Acknowledgements

This research was supported by the Guangzhou Basic and Applied Basic Research Foundation (Grant No. 2023A04J1687), National Natural Science Foundation of China (Grant No. 6230070401), South China University of Technology-TCL Technology Innovation Fund.

References

Arpit, D.; Jastrzebski, S.; Ballas, N.; Krueger, D.; Bengio, E.; Kanwal, M. S.; Maharaj, T.; Fischer, A.; Courville, A.; Bengio, Y.; et al. 2017. A Closer Look at Memorization in Deep Networks. *stat*, 1050: 1.

Bae, S.; Ko, J.; Song, H.; and Yun, S. 2023. Fast and Robust Early-Exiting Framework for Autoregressive Language Models with Synchronized Parallel Decoding. In *EMNLP*, 5910–5924.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D. M.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. In *NeurIPS*.

Conover, M.; Hayes, M.; Mathur, A.; Xie, J.; Wan, J.; Shah, S.; Ghodsi, A.; Wendell, P.; Zaharia, M.; and Xin, R. 2023. Free Dolly: Introducing the World's First Truly Open Instruction-Tuned LLM.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*, 4171–4186.

Elbayad, M.; Gu, J.; Grave, E.; and Auli, M. 2020. Depth-Adaptive Transformer. In *ICLR*.

Fan, A.; Grave, E.; and Joulin, A. 2020. Reducing Transformer Depth on Demand with Structured Dropout. In *ICLR*.

Geng, S.; Gao, P.; Fu, Z.; and Zhang, Y. 2021. RomeBERT: Robust Training of Multi-Exit BERT. *CoRR*, abs/2101.09755.

Goyal, S.; Choudhury, A. R.; Raje, S.; Chakaravarthy, V. T.; Sabharwal, Y.; and Verma, A. 2020. PoWER-BERT: Accelerating BERT Inference via Progressive Word-vector Elimination. In *ICML*, 3690–3699.

Guan, Y.; Li, Z.; Leng, J.; Lin, Z.; and Guo, M. 2022. Transkimmer: Transformer Learns to Layer-wise Skim. In Muresan, S.; Nakov, P.; and Villavicencio, A., eds., *ACL*, 7275–7286.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *ICLR*.

Jiao, X.; Yin, Y.; Shang, L.; Jiang, X.; Chen, X.; Li, L.; Wang, F.; and Liu, Q. 2020. TinyBERT: Distilling BERT for Natural Language Understanding. In *EMNLP*, Findings of EMNLP, 4163–4174.

Kaya, Y.; Hong, S.; and Dumitras, T. 2019. Shallow-deep networks: Understanding and mitigating network overthinking. In *ICML*, 3301–3310.

Kim, G.; and Cho, K. 2021. Length-Adaptive Transformer: Train Once with Length Drop, Use Anytime with Search. In *ACL-IJCNLP*, 6501–6511.

Kim, S.; Gholami, A.; Yao, Z.; Mahoney, M. W.; and Keutzer, K. 2021. I-BERT: Integer-only BERT Quantization. In *ICML*, 5506–5518.

Kim, S.; Shen, S.; Thorsley, D.; Gholami, A.; Kwon, W.; Hassoun, J.; and Keutzer, K. 2022. Learned Token Pruning for Transformers. In *SIGKDD*, 784–794.

Kumar, M.; Packer, B.; and Koller, D. 2010. Self-paced learning for latent variable models. In *NeurIPS*, 1189–1197.

Li, L.; Lin, Y.; Chen, D.; Ren, S.; Li, P.; Zhou, J.; and Sun, X. 2021. CascadeBERT: Accelerating Inference of Pre-trained Language Models via Calibrated Complete Models Cascade. In *EMNLP*, 475–486.

Liao, K.; Zhang, Y.; Ren, X.; Su, Q.; Sun, X.; and He, B. 2021. A Global Past-Future Early Exit Method for Accelerating Inference of Pre-trained Language Models. In *NAACL-HLT*, 2013–2023.

Lin, C.-Y. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*, 74–81.

Liu, W.; Zhou, P.; Wang, Z.; Zhao, Z.; Deng, H.; and Ju, Q. 2020. FastBERT: a Self-distilling BERT with Adaptive Inference Time. In *ACL*, 6035–6044.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, abs/1907.11692.

Michel, P.; Levy, O.; and Neubig, G. 2019. Are sixteen heads really better than one? In *NeurIPS*, 14014–14024.

Nallapati, R.; Zhou, B.; dos Santos, C. N.; Gül**c**cehre, **c**C.; and Xiang, B. 2016. Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond. In Goldberg, Y.; and Riezler, S., eds., *CoNLL*, 280–290.

Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C.; Mishkin, P.; Zhang, C.; Agarwal, S.; Slama, K.; Ray, A.; et al. 2022. Training language models to follow instructions with human feedback. In *NeurIPS*, volume 35, 27730–27744.

Sanh, V.; Debut, L.; Chaumond, J.; and Wolf, T. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Schuster, T.; Fisch, A.; Gupta, J.; Dehghani, M.; Bahri, D.; Tran, V.; Tay, Y.; and Metzler, D. 2022. Confident Adaptive Language Modeling. In *NeurIPS*.

Schuster, T.; Fisch, A.; Jaakkola, T. S.; and Barzilay, R. 2021. Consistent Accelerated Inference via Confident Adaptive Transformers. In *EMNLP*, 4962–4979.

Schwartz, R.; Stanovsky, G.; Swayamdipta, S.; Dodge, J.; and Smith, N. A. 2020. The right tool for the job: Matching model and instance complexities. In *ACL*, 6640–6651.

Sun, S.; Cheng, Y.; Gan, Z.; and Liu, J. 2019. Patient Knowledge Distillation for BERT Model Compression. In *EMNLP-IJCNLP*, 4322–4331.

Sun, T.; Liu, X.; Zhu, W.; Geng, Z.; Wu, L.; He, Y.; Ni, Y.; Xie, G.; Huang, X.; and Qiu, X. 2022. A simple hash-based early exiting approach for language understanding and generation. In *Findings of ACL*, 2409–2421.

Taori, R.; Gulrajani, I.; Zhang, T.; Dubois, Y.; Li, X.; Guestrin, C.; Liang, P.; and Hashimoto, T. B. 2023. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, 3(6): 7.

Toneva, M.; Sordoni, A.; Combes, R. T. d.; Trischler, A.; Bengio, Y.; and Gordon, G. J. 2019. An empirical study of example forgetting during deep neural network learning. In *ICLR*.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; Rodriguez, A.; Joulin, A.; Grave, E.; and Lample, G. 2023. LLaMA: Open and Efficient Foundation Language Models. *CoRR*, abs/2302.13971.

Voita, E.; Talbot, D.; Moiseev, F.; Sennrich, R.; and Titov, I. 2019. Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. In *ACL*, 5797–5808.

Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; and Bowman, S. R. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*.

Wang, J.; Chen, K.; Chen, G.; Shou, L.; and McAuley, J. 2022. Skipbert: Efficient inference with shallow layer skipping. In *ACL*, 7287–7301.

Xiao, G.; Lin, J.; Seznec, M.; Wu, H.; Demouth, J.; and Han, S. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *ICML*, 38087–38099.

Xin, J.; Tang, R.; Lee, J.; Yu, Y.; and Lin, J. 2020. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *ACL*, 2246–2251.

Xin, J.; Tang, R.; Yu, Y.; and Lin, J. 2021. BERxiT: Early Exiting for BERT with Better Fine-Tuning and Extension to Regression. In *EACL*, 91–104.

Xu, C.; and McAuley, J. J. 2023. A Survey on Dynamic Neural Networks for Natural Language Processing. In *Findings of EACL*, 2325–2336.

Yang, Z.; Dai, Z.; Yang, Y.; Carbonell, J. G.; Salakhutdinov, R.; and Le, Q. V. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *NeurIPS*, 5754–5764.

Yao, Z.; Yazdani Aminabadi, R.; Zhang, M.; Wu, X.; Li, C.; and He, Y. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In *NeurIPS*, 27168–27183.

Ye, D.; Lin, Y.; Huang, Y.; and Sun, M. 2021. TR-BERT: Dynamic Token Reduction for Accelerating BERT Inference. In *NAACL-HLT*, 5798–5809.

Zhang, T.; Kishore, V.; Wu, F.; Weinberger, K. Q.; and Artzi, Y. 2019. BERTScore: Evaluating Text Generation with BERT. In *ICLR*.

Zhang, Z.; Zhu, W.; Zhang, J.; Wang, P.; Jin, R.; and Chung, T.-S. 2022. Pcee-bert: Accelerating bert inference via patient and confident early exiting. In *Findings of NAACL*, 327–338.

Zhou, W.; Xu, C.; Ge, T.; McAuley, J.; Xu, K.; and Wei, F. 2020. Bert loses patience: Fast and robust inference with early exit. In *NeurIPS*, volume 33, 18330–18341.

Zhu, W. 2021. LeeBERT: Learned Early Exit for BERT with cross-level optimization. In *ACL-IJCNLP*, 2968–2980.