CascadeBERT: Accelerating Inference of Pre-trained Language Models via Calibrated Complete Models Cascade

Lei Li[†], **Yankai Lin**[§], **Deli Chen**[†§], **Shuhuai Ren**[†], **Peng Li**[§], **Jie Zhou**[§], **Xu Sun**[†]
[†]MOE Key Laboratory of Computational Linguistics, School of EECS, Peking University
[§]Pattern Recognition Center, WeChat AI, Tencent Inc., China
{lilei, shuhuai_ren}@stu.pku.edu.cn
{chendeli, xusun}@pku.edu.cn
{yankailin, patrickpli, withtomzhou}@tecent.com

Abstract

Dynamic early exiting aims to accelerate the inference of pre-trained language models (PLMs) by emitting predictions in internal layers without passing through the entire model. In this paper, we empirically analyze the working mechanism of dynamic early exiting and find that it faces a performance bottleneck under high speed-up ratios. On one hand, the PLMs' representations in shallow layers lack high-level semantic information and thus are not sufficient for accurate predictions. On the other hand, the exiting decisions made by internal classifiers are unreliable, leading to wrongly emitted early predictions. We instead propose a new framework for accelerating the inference of PLMs, CascadeBERT, which dynamically selects proper-sized and complete models in a cascading manner, providing comprehensive representations for predictions. We further devise a difficulty-aware objective, encouraging the model to output the class probability that reflects the real difficulty of each instance for a more reliable cascading mechanism. Experimental results show that CascadeBERT can achieve an overall 15% improvement under 4× speed-up compared with existing dynamic early exiting methods on six classification tasks, yielding more calibrated and accurate predictions.[1]

1 Introduction

Large-scale pre-trained language models (PLMs), e.g., BERT and RoBERTa, have demonstrated superior performance on various natural language understanding tasks (Devlin et al., 2019; Liu et al., 2019). While the increased model size brings more promising results, the long inference time hinders the deployment of PLMs in real-time applications. Researchers have recently exploited various kinds of approaches for accelerating the inference of PLMs,

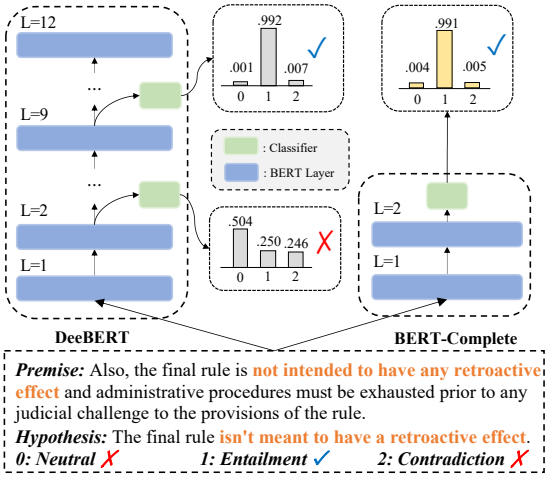


Figure 1: An easy instance with a large word overlap (colored in orange) between the premise and the hypothesis from the MNLI dataset. The classifiers in shallow layers of a dynamic early exiting model cannot predict correctly, while BERT-Complete (Turc et al., 2019), a small BERT pre-trained from scratch with the same size can make a correct and confident prediction.

which can be categorized into model-level compression and instance-level speed-up. The former aims at obtaining a compact model via quantization (Zafrir et al., 2019; Shen et al., 2020; Zhang et al., 2020), pruning (Voita et al., 2019; Michel et al., 2019) or knowledge distillation (KD) (Sanh et al., 2019; Sun et al., 2019; Jiao et al., 2020), while the latter adapts the amount of computation to the complexity of each instance (Graves, 2016). A mainstream method for instance-level speed-up is dynamic early exiting, which emits predictions based on intermediate classifiers (or off-ramps) of internal layers when the predictions are confident enough (Xin et al., 2020b; Liu et al., 2020; Schwartz et al., 2020; Li et al., 2021).

In this paper, we focus on dynamic early exiting, as it can be utilized to accelerate inference and reduce the potential risk of the overthinking problem (Kaya et al., 2019). Such a paradigm is

[1]Our code is available at https://github.com/lancopku/CascadeBERT

Findings of the Association for Computational Linguistics: EMNLP 2021, pages 475–486
November 7–11, 2021.

intuitive and simple, while faces a performance bottleneck under high speed-up ratios, i.e., the task performance is poor when most examples are exited in early layers. We conduct probing experiments to investigate the mechanism of dynamic exiting, and find that the poor performance is due to the following two reasons: (1) The shallow representations lack high-level semantic information, and are thus not sufficient for accurate predictions. As PLMs like BERT exhibit a hierarchy of representations, e.g., shallow layers extract low-level features like lexical/syntactic information while deep layers capture semantic-level relations (Tenney et al., 2019; Jawahar et al., 2019), we argue that the high-level semantic inference ability is usually required even for easy instances. As shown in Figure 1, the classifier of the second layer in a representative early exiting model DeeBERT (Xin et al., 2020b) cannot predict correctly even for an easy instance with a large word overlap. On the contrary, BERT-Complete, a shallow 2-layer model pre-trained from scratch (Turc et al., 2019) that is thus capable of extracting semantic-level features, can make confident and correct predictions like that in deep layers of DeeBERT. (2) The intermediate classifiers in the early exiting models cannot provide reliable exiting decisions. We design a metric to examine the ability of models to distinguish difficult instances from easy ones, which can reflect the quality of exiting decisions. We find that the predictions of internal classifiers cannot faithfully reflect the instance difficulty, resulting in wrongly emitted results and thus hindering the efficiency of early exiting.

To remedy those drawbacks, we instead extend the dynamic early exiting idea to a model cascade, and propose **CascadeBERT**, which conducts inference based on a series of complete models in a cascading manner with a dynamic stopping mechanism. Specifically, given an instance for inference, instead of directly exiting in the middle layers of a single model, the framework progressively checks if the instance can be solved by the current PLM from the smallest to the largest one, and emits the prediction once the PLM is confident about the prediction. Furthermore, we propose a difficulty-aware regularization to calibrate the PLMs' predictions according to the instance difficulty, making them reflect the real difficulty of each instance. Therefore, the predictions can be utilized as a good indicator for the early stopping in inference. Experimental results on six classification tasks in the GLUE benchmark demonstrate that our model can obtain a much better task performance than previous dynamic early exiting baselines under high speed-up ratios. Further analysis demonstrates that the proposed difficulty-aware objective can calibrate the model predictions, and proves the effectiveness and the generalizability of CascadeBERT.

2 Investigations into Early Exiting

Dynamic early exiting aims to speed-up the inference of PLMs by emitting predictions based on internal classifiers. For each instance, if the internal classifier's prediction based on the current layer representation of the instance is confident enough, e.g., the maximum class probability exceeds a threshold (Schwartz et al., 2020), then the prediction is emitted without passing through the entire model. However, whether the internal representations could provide sufficient information for accurate predictions and whether the intermediate classifiers can be utilized for making accurate exiting decisions still remain unclear. In this section, we investigate the working mechanism of dynamic early exiting by exploring these two questions.

2.1 Are Shallow Features Sufficient?

As discussed by Tenney et al. (2019), PLMs like BERT learn a hierarchy of representations. We assume that the high-level semantics is usually required even for easy instances, and therefore the predictions based on shallow representations are insufficient for accurate predictions. To examine this, we evaluate the model performance based on outputs of different layers, as the representation contains adequate information is necessary for a decent task performance. Specifically, we compare the following models:

DeeBERT (Xin et al., 2020b), which is a representative of early exiting methods. The internal classifiers are appended after each layer in the original BERT for emitting early predictions.

BERT-*k*L, which only utilizes the first k layers in the original BERT model for prediction. A classifier is added directly after the first k layers. The parameters of the first k layers and the classifier are fine-tuned on the training dataset. It could be seen as a static early exiting method.

BERT-Complete (Turc et al., 2019), which is a light version of the original BERT model pre-trained from scratch using the masked language

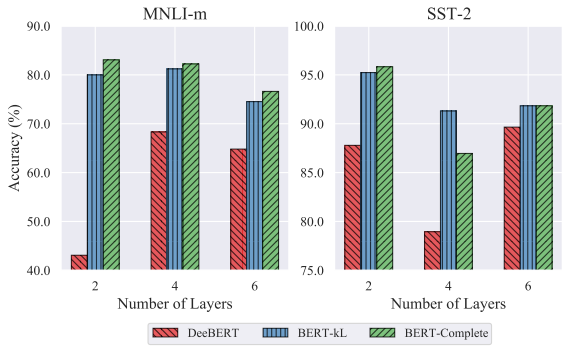


Figure 2: Performance comparison utilizing different models with the same number of layers on MNLI-m and SST-2. Complete models capable of extracting semantic-level information clearly outperform models like DeeBERT which overlooks the high-level semantic features.

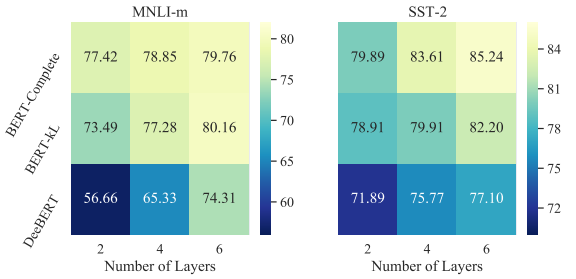


Figure 3: DIS (%, higher is better, see Eq. 3 in Section 2.2) heatmap of different models on the development set of MNLI and SST-2. The DIS of internal off-ramps in the DeeBERT of shallow layers is lower than that of BERT-kl and BERT-Complete, which leads to more wrongly emitted instances. The exiting decisions in shallow layers of DeeBERT thus can be unreliable.

modeling (MLM) objective. We assume the representations of this model contain high-level semantic information, as MLM requires a deep understanding of the language.

For a fair comparison, models are evaluated on a subset of instances which DeeBERT chooses to emit at different layers. We report prediction accuracy using different number of layers on MNLI (Williams et al., 2018) and SST-2 (Socher et al., 2013). Figure 2 shows the results on the development sets, and we can see that:

(1) BERT-Complete clearly outperforms DeeBERT, especially when the predictions are made based on shallow layers. It indicates that the high-level semantics is vital for handling tasks like sentence-level classification.

(2) BERT-kL also outperforms DeeBERT. We attribute it to that the last serveral layers can learn task-specific information during fine-tuning to obtain a decent performance. A similar phenomenon is also observed by Merchant et al. (2020). However, since the internal layer representation in DeeBERT are restricted by the layer relative position in the whole model, this adaption effect cannot be fully exploited, resulting in the poor performance in shallow layers.

These findings verify our assumption that the semantic-level features are vital, motivating us to exploit complete models for predictions. Besides, DeeBERT performs poorly on the selected instances which it decides to emit at different layers, triggering our further explorations on the quality of exiting decisions.

2.2 Are Internal Classifiers Reliable?

We further probe whether the early exiting decisions made by internal classifiers are reliable, by first introducing two key concepts:

- *Instance Difficulty* $d(x)$, which indicates whether an instance x can be handled by a specific model. We define instances that the model cannot predict correctly as difficult instances, i.e., $d(x) = 1$, and those can be handled well as easy ones, i.e., $d(x) = 0$.

- *Model Confidence* $c(x)$, which denotes how confident the model is about its prediction for a specific instance x. For each instance, we utilize the maximum class probability of the output distribution as the confidence score.

Intuitively, a difficult instance should be predicted with less confidence than that of an easy one, such that the output distribution can be utilized as an indicator for early exiting decisions. However, the model confidence can be inconsistent with the instance difficulty due to the overconfident problem. To measure this consistency, we propose *Difficulty Inversion Score* (DIS). Specifically, we first define a difficult inversion indicator function for instance pair (x_i, x_j) measuring the inconsistency between model confidence and instance difficulty as:

$$\mathrm{DI}\,(x_i, x_j) = \begin{cases} 1, & \text{if } d(x_i) > d(x_j) \text{ and } c(x_i) < c(x_j) \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

The instances are then sorted by their confidence scores in an ascending order, i.e., $c(x_i) \leq c(x_j)$ for any $i < j$. We compute the sum of difficulty

inversion pair as:

$$\text{DI-Sum} = \sum_{i=1}^{N} \sum_{j=1}^{i-1} \text{DI}(x_i, x_j), \tag{2}$$

where N is the number of instance. The final DIS is a normalized DI-Sum:

$$\text{DIS} = 1 - \frac{1}{K} \text{DISum}, \tag{3}$$

where K is a normalizing factor calculated as the product of the number of easy instances and the number of difficult instances, to re-scale DIS to the range from 0 to 1. According to the definition, the DIS measures the proportion of instance pairs that are correctly ranked by the classifier. Classifiers with lower DIS achieve lower consistency between the model confidence and instance difficulty, thus making more unreliable exiting decisions. The DIS thus can be utilized as a proxy for evaluating the quality of exiting decisions. We compute the DIS on the development sets of MNLI-m and SST-2 for internal classifiers of different models discussed in Section 2.1, and the results are illustrated in Figure 3. We find that:

(1) The DIS of internal classifiers in shallow layers of DeeBERT falls far behind BERT-kL and BERT-Complete. This indicates that the exiting decisions in the shallow layers of DeeBERT are unreliable, and the task performance thus can be poor when most instances are wrongly emitted in early layers.

(2) The ability to distinguish difficult examples from easy ones is enhanced as the layer number increases. Since the deep layer representations with semantic information can boost the task performance, it is reasonable to expect that the off-ramps in deep layers can provide more comprehensive early exiting decisions.

Our analysis demonstrates that current dynamic early exiting predictions made by internal classifiers in shallow layers are not reliable, motivating us to inform the model of the instance difficulty for more robust exiting decisions.

3 Methodology

To remedy the drawbacks of conducting dynamic exiting in a single model, we extend the idea to a model cascade, by proposing CascadeBERT, that utilizes a suite of complete PLMs with different number of layers for acceleration in a cascading manner, and further devise a difficulty-aware calibration regularization to inform the model of instance difficulty.

3.1 Cascade Exiting

Formally, given n complete pre-trained language models $\{M_1, \ldots, M_n\}$ fine-tuned on the downstream classification dataset, which are sorted in an ascending order by their corresponding number of layers $\{L_1, \ldots, L_n\}$, our goal is to conduct inference with the minimal computational cost for each input instance x while maintaining the model performance. Our preliminary exploration shows that it is relatively hard to directly selecting a proper model for each instance according to the instance difficulty. Therefore, we formulate it as a cascade exiting problem, i.e., execute the model prediction sequentially for each input example from the smallest M_1 to the largest M_n, and check whether the prediction of the input instance x can be emitted. Specifically, we use the confidence score $c(x)$, i.e., the maximum class probability, as a metric to determine whether the predictions are confident enough for emitting:

$$c(x) = \max_{y \in Y} (\Pr(y \mid x)), \tag{4}$$

where Y is the label set of the task and $\Pr(y \mid x)$ is the class probability distribution outputted by the current model. The predicted result is emitted once the confidence score exceeds a preset threshold τ. By varying the threshold τ, we can obtain different speed-up ratios based on the application requirements. A smaller τ indicates that more examples are outputted using the current model, making the inference faster, while a bigger τ will make more examples go through larger models for better results. The cascaded exiting framework is summarized in Algorithm 1. Since every model in our cascading framework is a complete model, predictions are more accurate with instance representations that contain both low-level and high-level features, even when only the smallest model is executed.

3.2 Difficulty-Aware Regularization

To further make the cascade exiting based on confidence score more reliable, we design a difficulty-aware regularization (DAR) objective based on instance difficulty, to regularize the model classifiers produce lower confidence for more difficult instances. To measure the instance difficulty, we first split the training dataset $\mathcal{D}$ into K folds

Algorithm 1: Cascade Exiting

Input: Models $\{M_1, \ldots, M_n\}$, threshold τ
Data: Input x
Result: Class probability distribution $\Pr(y \mid x)$
for $i \leftarrow 1$ **to** n **do**
  // calculate class distribution
  $\Pr(y|x) = M_i(x)$
  // compute confidence score
  $c(x) = \max_y(\Pr(y \mid x))$
  if $c(x) > \tau$ **then**
    Early exit $\Pr(y \mid x)$
return $\Pr(y \mid x)$

$\{\tilde{\mathcal{D}}_i \mid i = 1, \ldots, K\}$. For each complete model in our cascade, we train K models with the same architecture using the leave-one-out method, e.g., model M_n^i is trained on the $\mathcal{D}_{-\tilde{\mathcal{D}}_i} = \bigcup_j^{j \neq i} \tilde{\mathcal{D}}_j$. We utilize M_n^i to evaluate the difficulty of the examples in $\tilde{\mathcal{D}}_i$ for model M_n. Specifically, the samples are marked as easy, i.e., $d = 0$, if they can be correctly classified by the model. Otherwise, they are marked as difficult, i.e., $d = 1$. To eliminate the impact of randomness, we group the predictions of 5 seeds and strictly label the examples that can be correctly predicted in all seeds as easy examples, while the others as difficult ones.

With the instance difficulty for each instance in the training dataset, we add a difficulty-based margin objective for each instance pair:

$$\mathcal{L}(x_i, x_j) = \max\left\{0, -g\left(x_i, x_j\right)\left(c\left(x_i\right) - c\left(x_j\right)\right) + \epsilon\right\}, \tag{5}$$

where ϵ is a confidence margin. We design $g\left(x_i, x_j\right)$ as below:

$$g\left(x_i, x_j\right) = \begin{cases} 1, & \text{if } d(x_i) > d(x_j) \\ 0, & \text{if } d(x_i) = d(x_j) \\ -1, & \text{otherwise.} \end{cases} \tag{6}$$

This objective is added to the original task-specific loss with a weight factor λ to adjust its impact. By optimizing the above objective function, the confidence scores of difficult instances are adjusted to be lower than those of easy instances, thus making the confidence-based emitting decisions more reliable.

4 Experiments

We evaluate our method on the classification tasks in the GLUE benchmark (Wang et al., 2018) with BERT (Devlin et al., 2019). We first give a brief introduction of the dataset used and the experimental setting, followed by the description of baseline models for comprehensive evaluation. The results and analysis of the experiments are presented last.

Dataset	# Train	# Dev	# Test	Metric	ϵ
MNLI	393k	20k	20k	Accuracy	0.3
MRPC	3.7k	0.4k	1.7k	F1-score	0.5
QNLI	105k	5.5k	5.5k	Accuracy	0.3
QQP	364k	40k	391k	F1-score	0.3
RTE	2.5k	0.3k	3k	Accuracy	0.5
SST-2	67k	0.9k	1.8k	Accuracy	0.5

Table 1: Statistics of six classification datasets in GLUE benchmark. The selected difficulty margins ϵ of each datasets are provided in the last column.

4.1 Experimental Settings

We use six classification tasks in GLUE benchmark, including MNLI (Williams et al., 2018), MRPC (Dolan and Brockett, 2005), QNLI (Rajpurkar et al., 2016), QQP,[2] RTE (Bentivogli et al., 2009) and SST-2 (Socher et al., 2013). The metrics for evaluation are F1-score for QQP and MRPC, and accuracy for the rest tasks. Our implementation is based on the Huggingface Transformers library (Wolf et al., 2020). We use two models for selection with 2 and 12 layers, respectively, since they can provide a wide range for acceleration. The difficulty score is thus evaluated based on the 2-layer model. The effect of incorporating more models in our cascade framework is explored in the later section. We utilize the weights provided by Turc et al. (2019) to initialize the models in our suite. The split number K for difficulty labeling is set to 8. We use AdamW (Loshchilov and Hutter, 2018) with a learning rate 2e-5 to train model for 10 epochs, since we find that small models need more time to converge. We set DAR weight λ as 0.5, and perform grid search over ϵ in $\{0.1, 0.3, 0.5, 0.7\}$. The best model is selected based on the validation performance. The statistics of datasets and the selected ϵ are provided in Table 1.

The inference speed-up ratio is estimated as the ratio of number of the original model and layers actually executed in forward propagation in our cascade. Compared to performing dynamic exiting in a single model, the overhead of CascadeBERT consists of two parts. The former is the extra embedding operations, which is nearly 0.3M FLOPs and is negligible compared with the 1809.9M FLOPs of each layer (Liu et al., 2020). The latter is brought by instances that run forward propagation multiple times, which is counted in the speed-up ratio calculation. For example, for an instance which is first

[2] https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs

	Method	MNLI-m/mm	MRPC	QNLI	QQP	RTE	SST-2	AVG
	BERT-base†	84.6 (1.00×) / 83.4 (1.00×)	88.9 (1.00×)	90.5 (1.00×)	71.2 (1.00×)	66.4 (1.00×)	93.5 (1.00×)	82.6
≳ 2×	BERT-6L‡	79.9 (2.00×) / 79.2 (2.00×)	85.1 (2.00×)	86.2 (2.00×)	68.9 (2.00×)	**65.0** (2.00×)	90.9 (2.00×)	79.3
	DeeBERT†	74.4 (1.87×) / 73.1 (1.88×)	84.4 (2.07×)	85.6 (2.09×)	70.4 (2.13×)	64.3 (1.95×)	90.2 (2.00×)	77.5
	PABEE†	79.8 (2.07×) / 78.7 (2.08×)	84.4 (2.01×)	88.0 (1.87×)	70.4 (2.09×)	64.0 (1.81×)	89.3 (1.95×)	79.2
	CascadeBERT	**83.0** (2.01×) / **81.6** (2.01×)	**85.9** (2.01×)	**89.4** (2.01×)	**71.2** (2.01×)	64.6 (2.03×)	**91.7** (2.08×)	**81.1**
	BERT-4L‡	75.8 (3.00×) / 75.1 (3.00×)	82.7 (3.00×)	84.7 (3.00×)	66.5 (3.00×)	63.0 (3.00×)	87.5 (3.00×)	76.5
≳ 3×	DeeBERT‡	63.2 (2.98×) / 61.3 (3.03×)	83.5 (3.00×)	82.4 (2.99×)	67.0 (2.97×)	59.9 (3.00×)	88.8 (2.97×)	72.3
	PABEE‡	75.9 (2.70×) / 75.3 (2.71×)	82.6 (2.72×)	82.6 (3.04×)	69.5 (2.57×)	60.5 (2.38×)	85.2 (3.15×)	75.9
	CascadeBERT	**81.2** (3.00×) / **79.5** (3.00×)	**84.0** (3.00×)	**88.5** (2.99×)	**71.0** (3.02×)	**63.8** (3.03×)	**90.9** (2.99×)	**79.8**
	BERT-3L‡	74.8 (4.00×) / 74.3 (4.00×)	80.5 (4.00×)	83.1 (4.00×)	65.8 (4.00×)	55.2 (4.00×)	86.4 (4.00×)	74.3
≳ 4×	DeeBERT‡	55.8 (4.01×) / 54.2 (3.99×)	**82.9** (3.99×)	75.9 (4.00×)	62.9 (4.01×)	57.4 (4.00×)	85.4 (4.00×)	67.8
	PABEE‡	62.3 (4.32×) / 63.0 (4.30×)	79.9 (4.00×)	-	68.0 (3.45×)	56.0 (3.62×)	-	-
	CascadeBERT	**79.3** (4.03×) / **77.9** (3.99×)	82.6 (4.00×)	**86.5** (3.99×)	**70.0** (4.04×)	**61.6** (4.02×)	**90.3** (4.01×)	**78.3**

Table 2: Test results from the GLUE server. We report F1-score for QQP and MRPC and accuracy for other tasks, with the corresponding speed-up ratios shown in parentheses. For baseline methods, † denotes results taken from the original paper and ‡ denotes results based on our implementation. The - denotes that results are not available by tuning the threshold of PABEE. Best results are shown in **bold**.

fed into a 2-layer model and then goes through a 4-layer model to obtain the final prediction result, the number of layers actually executed is therefore 6 and the corresponding speed-up ratio is 2× compared to the original 12-layer full model.

4.2 Baselines

We implement two kinds of baselines, including:

Early Exiting, including BERT-kL, where the first k layers with a fine-tuned classifier are used for outputting the final classification results. We take $k = 6$, $k = 4$ and $k = 3$ to obtain a statically compressed model with speed-up ratios of 2×, 3× and 4×, respectively; DeeBERT (Xin et al., 2020b), which makes dynamic early predictions based on the internal classifiers; PABEE (Zhou et al., 2020), an enhanced variant by emitting the result until several layers produce a consistent prediction. [3]

Knowledge Distillation methods that do not require external data, including DistilBERT (Sanh et al., 2019), which distills knowledge from the teacher model to the student during pre-training via logit distillation; BERT-PKD (Sun et al., 2019), which distills internal states of the teacher model to the student model; BERT-of-Theseus (Xu et al., 2020), which gradually replaces the module in the original model; BERT-PD (Turc et al., 2019), which directly pre-trains a compact model from scratch and conducts distillation on the task dataset.

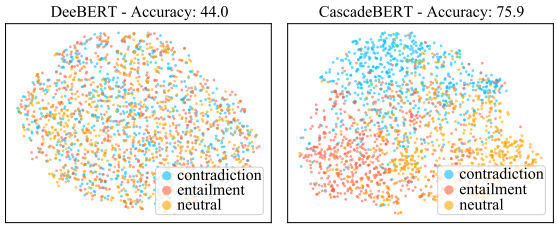
[3]PABEE provides limited speed-ratios since the threshold for tuning speed-up ratios can only be set to integers.

Method	MNLI-m/mm	QNLI	QQP	SST-2	AVG
DistilBERT	78.9 / 78.0	85.2	68.5	**91.4**	80.4
BERT-PKD	79.9 / 79.3	85.1	70.2	89.4	80.8
BERT-Theseus	78.6 / 77.4	85.5	68.3	89.7	79.9
BERT-PD	79.3 / 78.3	87.0	69.8	89.8	80.8
CascadeBERT	**81.2 / 79.5**	**88.5**	**71.0**	90.9	**82.2**

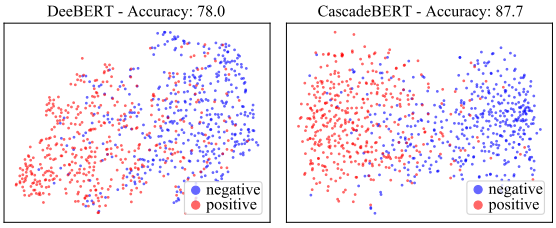
Table 3: Test result comparison with static knowledge distillation methods under speed-up ratio 3×.

4.3 Overall Results

The performance comparison with early exiting methods are presented in Table 2. We observe that CacadeBERT outperforms all the baseline methods under different speed-up ratios, validating the effectiveness of our proposal. Furthermore, the performance gap becomes clearer as the acceleration ratio increases. For example, CascadeBERT outperforms DeeBERT by a big margin with a relative 15.5% improvement (10.5 points on average) under speed-up ratio 4×. This phenomenon demonstrates that CascadeBERT can break the performance bottleneck by utilizing comprehensive representations from complete models. Interestingly, we find CascadeBERT performs closely with DeeBERT on MRPC. We attribute it to that this paraphrase identification task requires less high-level semantic information, thus only utilizing low-level features at specific layers can sometimes become beneficial. Different from DeeBERT and PABEE, FastBERT (Liu et al., 2020) enhances the internal classifiers with a self-attention mechanism to use all the hidden states for predictions, resulting in



(a) Instance representations t-SNE projection on MNLI-m.



(b) Instance representations t-SNE projection on SST-2.

Figure 4: t-SNE visualization of instance representations of different class in DeeBERT and our CascadeBERT at the second layer. The instance representations of our CascadeBERT exhibit a more distinct boundary between different classes, helping the following classifier to make accurate predictions. Best viewed in color.

a different magnitude of computational overhead. Comparison results with FastBERT are provided in Appendix A. CascadeBERT can still outperform FastBERT, especially on the tasks requiring semantic reasoning ability.

Besides, our proposal also achieves superior performance over strong knowledge distillation methods like BERT-PKD and BERT-of-Theseus, as shown in Table 3. Distillation methods can implicitly learn the semantic reasoning ability by forcing student models to mimic the behaviors of the teacher model. However, it is still relatively hard to obtain a single compressed model to handle all instances well, as different instances may require the reasoning ability of different granularities. Our cascading mechanism instead provides flexible options for instances with different complexities, thus achieving better results.

5 Analysis

In this section, we investigate how the proposed CascadeBERT makes accurate predictions under high speed-up ratios, and analyze the effects of the proposed difficulty-aware regularization and incorporating more models to the cascade. We finally examine the generalizability by applying it to RoBERTa. The experiments are conducted on MNLI, QNLI, QQP and SST-2 for stable results.

	Method	MNLI-m/mm	QNLI	QQP	SST-2	AVG
~3×	CascadeBERT	**81.2 / 79.5**	**88.5**	**71.0**	**90.9**	**82.2**
	- w/o DAR	80.0 / 79.3	87.8	**71.0**	90.3	81.7
~4×	CascadeBERT	**79.3** / 77.9	86.5	**70.0**	**90.3**	**80.8**
	- w/o DAR	78.9 / **78.1**	**86.6**	69.8	89.6	80.6

Table 4: Ablated results of the proposed difficulty-aware regularization under different speed-up ratios.

5.1 Visualization of Instance Representations

To investigate how the representations with sufficient information benefit accurate predictions, we visualize the instance representations after 2 layers using t-SNE projection (Maaten and Hinton, 2008). The results and the corresponding classifier accuracy are shown in Figure 4. We observe that the boundary of instances belonging to different classes of our CascadeBERT is much clearer than that of DeeBERT. Since the representations contain sufficient information for predictions, our model can thus obtain more accurate results. Interestingly, the shallow representations in DeeBERT of SST-2 are already separable to some extent, which indicates that the task is somewhat easy. It is consistent with our main results that the performance degradation of different methods is negligible on SST-2.

5.2 Effects of Difficulty-Aware Regularization

We show the performance of an ablated version of our proposal, CascadeBERT w/o DAR in Table 4. The results indicate that the DAR can improve the overall performance of our framework. Note that the improvement is very challenging to achieve, as the original model cascade already outperforms strong baseline models like PABEE. Furthermore, we explore whether the performance boost comes from an enhanced ability of the model to distinguish difficult instances from easy ones. Specifically, we compute the DIS and the task accuracy (Acc) of the smallest model in our cascade. The results are listed in Table 5. We find that the DAR can effectively improve the DIS while slightly harms the task performance, indicating that DAR boosts the overall performance by helping model make more reliable emitting decisions. The exceptional decrease of DIS on QQP is attributed to the fact that the original DIS score is relatively high, which makes further improvements very challenging. Besides, the DAR can lower the prediction

Dataset	Method	DIS (↑)	Acc(↑)	ECE (↓)
MNLI-m	CascadeBERT	**78.00**	75.97	**7.90**
	- w/o DAR	76.73	**76.02**	11.07
QNLI	CascadeBERT	**78.89**	84.53	**3.41**
	- w/o DAR	77.79	**84.73**	8.79
QQP	CascadeBERT	84.39	87.21	**3.37**
	- w/o DAR	**85.77**	**88.71**	4.99
SST-2	CascadeBERT	**82.02**	87.70	**5.61**
	- w/o DAR	79.30	**87.95**	8.73

Table 5: The ECE (%), Acc (%) and DIS (%) scores on different datasets. ↑ denotes higher is better, while ↓ means lower is better. The proposed DAR can boost the performance by giving hints of instance difficulty and calibrate the model predictions.

confidence of difficult instances, which improves the consistency between the predicted probability and how likely the model is to be correct for an instance. We quantitively measure this calibration effect of DAR, by utilizing the expected calibration error (ECE) (Guo et al., 2017).[4] As shown in Table 5, the DAR not only improves the DIS score, but also calibrates the model predictions, achieving lower expected calibration error.

5.3 Impacts of More Models in Cascade

We further consider to incorporate more models into the CascadeBERT framework. Theoretically, we prove that adding more models in cascade can boost the task performance under mild assumptions. Besides, the benefits will become marginal as the number of model increases. The detailed proof is provided in the Appendix C. We empirically verify this by adding a medium-sized model with 6 layers which satisfies our assumptions into the cascade. The performance under different speed-up ratios of a 2-12 cascade consists of a 2L model and a 12L model and the above mentioned 2-6-12 cascade are illustrated in Figure 5. Overall, we find that adding a model with a moderate size can slightly improve the performance, while the gain becomes marginal when the speed-up ratio is higher, since most instances are emitted from the smallest model.

5.4 Fine-tuned Models as an Alternative

To verify the generalizability of our cascading framework, we propose to apply our method to RoBERTa (Liu et al., 2019). However, small versions of RoBERTa pre-trained from scratch are currently not available. We notice that BERT-kL

[4]Refer to Appendix B for the details of the ECE score.

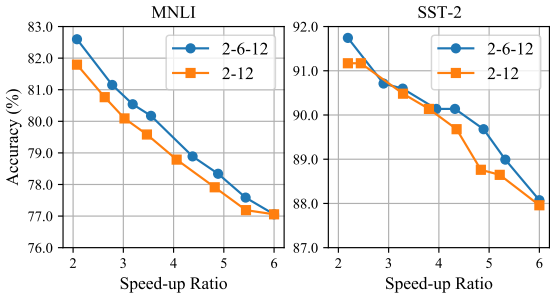


Figure 5: Task performance on the validation set and speed-up ratio trade-off curve comparison of a 2-model cascade (orange square) and a 3-model cascade (blue circle) on SST-2 and MNLI-m.

Method	MNLI-m/mm	QNLI	QQP	SST-2	AVG
RoBERTa-base	87.0 / 86.3	92.4	71.8	94.3	86.4
RoBERTa-4L	**80.3 / 79.2**	86.2	69.8	90.8	81.2
DeeBERT	53.9 / 55.4	77.2	67.6	88.6	68.5
PABEE	74.0 / 74.2	-	-	87.5	-
CascadeRoBERTa	78.9 / 78.1	86.8	70.5	90.8	81.0
+ Vanilla KD	79.7 / 78.8	**86.9**	**70.8**	**91.4**	**81.5**

Table 6: Test results from the GLUE server with RoBERTa models in our cascade framework. The speed-up ratio is approximately 3× (±4%). The - denotes unavailable results of PABEE.

model can achieve comparable performance via fine-tuning, as discussed in Section 2. Therefore, we propose to leverage a fine-tuned RoBERTa-2L with the vanilla KD (Hinton et al., 2015) incorporated for enhancing its semantic reasoning ability, as an alternative of the original complete model. The results around 3× speed-up are listed in Table 6. Our framework still outperforms dynamic early exiting baselines by a clear margin, validating that our framework is universal and can be combined with knowledge distillation techniques to further boost the performance.

6 Related Work

Model-level compression includes knowledge distillation (KD), pruning and quantization. KD focuses on transferring the knowledge from a large teacher model to a compact student model (Hinton et al., 2015). Sanh et al. (2019) propose DistilBERT and Sun et al. (2019) enhance KD by aligning the internal representations of the student and the teacher model. Besides, Jiao et al. (2020) propose TinyBERT via a two-stage KD on augmented data. Pruning methods deactivate the unimportant structures in the model like attention heads (Voita

et al., 2019; Michel et al., 2019) and layers (Fan et al., 2019). Quantization methods target at using fewer physical bits to efficiently represent the model (Zafrir et al., 2019; Shen et al., 2020; Zhang et al., 2020). We do not compare pruning and quantization methods since these techniques are orthogonal to our framework.

Instance-level speed-up accelerates the inference via adapting the computation according to the instance complexity (Graves, 2016). A representative framework is dynamic early exiting, which has been verified in natural language understanding (Xin et al., 2020b; Schwartz et al., 2020; Liu et al., 2020; Zhou et al., 2020; Liao et al., 2021; Sun et al., 2021), sequence labeling (Li et al., 2021), question answering (Soldaini and Moschitti, 2020) and document ranking (Xin et al., 2020a). In this paper, we probes the work mechanism of dynamic early exiting, and find that it faces a serious performance bottleneck under high speed-up ratios. To remedy this, we generalize the idea to a model cascade and prove it is effectiveness even under high speed-up ratios for various natural language understanding tasks. Concurrently with our work, Enomoto and Eda (2021) adopt the similar idea and achieve better inference efficiency on image classification tasks.

7 Conclusion

In this paper, we point out that current dynamic early exiting framework faces a performance bottleneck under high speed-up ratios, due to insufficient shallow layer representations and poor exiting decisions of the internal classifiers. To remedy this, we propose CascadeBERT, a model cascade framework with difficulty-aware regularization for accelerating the inference of PLMs. Experimental results demonstrate that our proposal achieves substantial improvements over previous dynamic exiting methods. Further analysis validates that the framework is generalizable and produces more calibrated results.

Acknowledgements

We thank all the anonymous reviewers for their constructive comments, Xuancheng Ren and Hua Zheng for their valuable suggestions in preparing the manuscript, and Wenkai Yang for providing the theoretical analysis. This work was supported by a Tencent Research Grant. Xu Sun is the corresponding author of this paper.

References

Luisa Bentivogli, Ido Kalman Dagan, Dang Hoa, Danilo Giampiccolo, and Bernardo Magnini. 2009. The fifth pascal recognizing textual entailment challenge. In *TAC Workshop*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP)*.

Shohei Enomoto and Takeharu Eda. 2021. Learning to cascade: Confidence calibration for improving the accuracy and computational cost of cascade inference systems. In *AAAI*, pages 7331–7339.

Angela Fan, Edouard Grave, and Armand Joulin. 2019. Reducing transformer depth on demand with structured dropout. In *ICLR*.

Alex Graves. 2016. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *ICML*, pages 1321–1330.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *ACL*, pages 3651–3657.

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 *Findings of EMNLP*, pages 4163–4174.

Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. 2019. Shallow-deep networks: Understanding and mitigating network overthinking. In *ICML*, pages 3301–3310.

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 *ACL*, pages 7871–7880.

Xiaonan Li, Yunfan Shao, Tianxiang Sun, Hang Yan, Xipeng Qiu, and Xuanjing Huang. 2021. Accelerating BERT inference for sequence labeling via early-exit. In *ACL*, pages 189–199.

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 *NAACL-HLT*, pages 2013–2023.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020. FastBERT: A self-distilling BERT with adaptive inference time. In *ACL*, pages 6035–6044.

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. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. In *ICLR*.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *JMLR*, 9:2579–2605.

Amil Merchant, Elahe Rahimtoroghi, Ellie Pavlick, and Ian Tenney. 2020. What happens to BERT embeddings during fine-tuning? In *BlackboxNLP Workshop*, pages 33–44.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *NeurIPS*, pages 14014–14024.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *EMNLP*, pages 2383–2392.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *NeurIPS Workshop on Energy Efficient Machine Learning and Cognitive Computing*.

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 *ACL*, pages 6640–6651.

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 *AAAI*, pages 8815–8821.

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 *EMNLP*, pages 1631–1642.

Luca Soldaini and Alessandro Moschitti. 2020. The cascade transformer: an application for efficient answer sentence selection. In *ACL*, pages 5697–5708.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for BERT model compression. In *EMNLP-IJCNLP*, pages 4323–4332.

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

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *ACL*, pages 4593–4601.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: The impact of student initialization on knowledge distillation. *arXiv preprint arXiv:1908.08962*.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *ACL*, pages 5797–5808.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *EMNLP Workshop on BlackboxNLP*, pages 353–355.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL-HLT*, pages 1112–1122.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *System Demonstrations, EMNLP*, pages 38–45.

Ji Xin, Rodrigo Nogueira, Yaoliang Yu, and Jimmy Lin. 2020a. Early exiting BERT for efficient document ranking. In *SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 83–88.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020b. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *ACL*, pages 2246–2251.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020. BERT-of-Theseus: Compressing BERT by progressive module replacing. In *EMNLP*, pages 7859–7869.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: Quantized 8bit BERT. In *NeurIPS Workshop on Energy Efficient Machine Learning and Cognitive Computing*.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. TernaryBERT: Distillation-aware ultra-low bit BERT. In *EMNLP*, pages 509–521.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. BERT loses patience: Fast and robust inference with early exit. In *NeurIPS*, pages 18339–18350.

A Comparison with FastBERT

Performance comparison of our CascadeBERT with FastBERT is shown in Table 7 under $3.00\times$ and $4.00\times$ speed-up ratios. Note that FastBERT utilizes a complicate internal classifier with self-attention mechanism and take the hidden states of all the sequence tokens for making predictions, while we only adopt the original linear-based classifier. Our method can still outperform the FastBERT under high speed-up ratios, especially on tasks that require high-level semantic reasoning ability like MNLI.

	Method	MNLI-m / -mm	QNLI	QQP	SST-2	Average
3×	FastBERT	79.8 / 78.9	88.2	71.5	92.1	82.1
3×	CascadeBERT	81.2 / 79.5	88.5	71.0	90.9	82.2
4×	FastBERT	76.1 / 75.2	86.4	70.5	90.7	79.8
4×	CascadeBERT	79.3 / 77.9	86.5	70.0	90.3	80.8

Table 7: Performance comparison with FastBERT.

B Expected Calibration Error

Calibration measures the consistency between predictions' confidence and accuracy. A well calibrated model can be more reliable, e.g., it can give us a hint that it knows what it does not know, and thus it is easier for deployments in real-world applications. It is formally expressed as a joint distribution $P(Q, Y)$ over confidences $Q \in R$ and labels $Y \in L$. When $P(Y = y \mid Q = q) = q$, the model is perfectly calibrated. For example, if the average confidence score of 100 instances is 0.8, there should be 80 instances that are correctly predicted. This probability can be approximated by grouping predictions into k disjoint and equally-sized bins, where each bin consists of b_k predictions. The expected calibration error is defined as a weighted average of difference between each bin's accuracy (acc$(\cdot)$) and prediction confidence (conf$(\cdot)$):

$$\text{ECE} = \sum_k \frac{b_k}{n} |\text{acc}(k) - \text{conf}(k)| \tag{7}$$

where n is the number of total instances. A lower ECE denotes the model is better calibrated. In this paper, we set $k = 10$ for calculating the ECE score.

C Analysis for More Models in Cascade

Suppose there are n models $\{M_1, \ldots, M_n\}$ sorted from the smallest to largest according to number of layers in our cascade, with corresponding number of layers $\{L_1, \ldots, L_n\}$ and the task performance, e.g., classification accuracy $\{a_1, \cdots, a_n\}$, we want to explore whether incorporating another complete model into the original cascade can further improve the task performance and speed-up trade-off. In more detail, we propose to evaluate the difference of classification accuracy between the original cascade and the new cascade, under the same speed-up ratio. We denote the new added model as M^* with classification accuracy a^* consisting of L^* layers, $L_i < L^* < L_{i+1}$ for a specific i. Considering the instance emitting distribution, we denote the number of instances exiting after model M_j $(j = 1, \ldots, n)$ as s_j in the original n models cascade. For the new $n+1$ models cascade, the number of samples exiting after model M_j $(j = 1, \ldots, n)$ is $\hat{s}_j$ and there will be $\hat{s}^*$ instances emitting from M^*. Besides, we assume that the accuracy a_i of M_i is the same for any subsets of the original dataset. The performance difference thus can be written as:

$$T = \frac{1}{N}\left(\sum_{k=1}^{n} a_k \hat{s}_k + a^* \hat{s}^* - \sum_{k=1}^{n} a_k s_k\right) \tag{8}$$

under the conditions of

$$\begin{cases} \sum\limits_{k=1}^{n} s_k = \sum\limits_{k=1}^{n} \hat{s}^k + \hat{s}^* = N \\ \sum\limits_{k=1}^{n} s_k L^k = \sum\limits_{k=1}^{i} \hat{s}_k L^k + \hat{s}^*(L^i + L^*) + \\ \qquad\qquad \sum\limits_{k=i+1}^{n} \hat{s}_k (L^k + L^*) \end{cases} \tag{9}$$

where $L^k = \sum\limits_{i=1}^{k} L_i$ is the actual layer cost with the computation overhead and N is the number of test instances. The first condition indicates the total number of test instances is the same, and the second one guarantees that the total layer cost is same thus the speed-up ratio is identical.

There are infinite solutions for the above system of equations, as we can adjust the exiting thresholds of different models to achieve the same speed-up ratio. We propose to simplify this by making a assumption that we only adjust the thresholds of M_i, M_{i+1} and M^* to achieve the same speed-up ratio, thus the following equation holds:

$$\hat{s}_k = s_k, \quad k = 1, 2, \cdots, i-1, i+2, \cdots, n \tag{10}$$

Conditions in Eq. (9) can thus be re-written as

$$\begin{cases} s_i + s_{i+1} = \hat{s}_i + \hat{s}_{i+1} + \hat{s}^* \\ s_i L^i + s_{i+1} L^{i+1} = \hat{s}_i L^i + \hat{s}^*(L^i + L^*) \\ \qquad\qquad + \hat{s}_{i+1}(L^{i+1} + L^*) \end{cases} \tag{11}$$

Then we can further calculate s_i and s_{i+1} as

$$\begin{cases} s_i = \hat{s}_i + \hat{s}^* - \frac{L^*}{L_{i+1}}(\hat{s}_{i+1} + \hat{s}^*) \\ s_{i+1} = \hat{s}_{i+1} + \frac{L^*}{L_{i+1}}(\hat{s}_{i+1} + \hat{s}^*) \end{cases} \tag{12}$$

By plugging the equations in Eq. 12 into the Eq. 8, and use the assumption in Eq. 10 we get

$$\begin{aligned} T &= \frac{1}{N}\left[a_i(\hat{s}_i - s_i) + a_{i+1}(\hat{s}_{i+1} - s_{i+1}) + a^*\hat{s}^*\right] \\ &= \frac{1}{N}\left[a^*\hat{s}^* + a_i(\frac{L^*}{L_{i+1}}(\hat{s}_{i+1} + \hat{s}^*) - \hat{s}^*) \right. \\ &\qquad \left. -a_{i+1}\frac{L^*}{L_{i+1}}(\hat{s}_{i+1} + \hat{s}^*)\right] \\ &= \frac{1}{N}\left[\hat{s}^*(a^* - a_i) - \frac{L^*}{L_{i+1}}(\hat{s}_{i+1} + \hat{s}^*)(a_{i+1} - a_i)\right] \end{aligned}$$

The final expected performance difference is thus:

$$T(\hat{s}^*, L^*) = \frac{1}{N}\left[\hat{s}^*(a^* - a_i) - \frac{L^*}{L_{i+1}}(\hat{s}_{i+1} + \hat{s}^*)(a_{i+1} - a_i)\right] \tag{13}$$

where the index i satisfies that $L_i < L^* < L_{i+1}$. Note that the model accuracy a^* is related to the size L^* of model, as a larger model with more layers tends to achieve a better task performance. It indicates that the performance difference depends on the number of samples exits at model M^* ($\hat{s}^*$), and the layers of M^* (L^*). If we fix the index i when we add the new model M^*, since we have $a_i \le a^* \le a_{i+1}$, from Eq (13) we can get

$$\begin{aligned} &T(\hat{s}^*, L^*) \\ &\le \frac{1}{N}\left[\hat{s}^*(a^* - a_i) - \frac{L^*}{L_{i+1}}(\hat{s}_{i+1} + \hat{s}^*)(a^* - a_i)\right] \\ &\le \frac{1}{N}(a^* - a_i)\left[\hat{s}^* - \frac{L^*}{L_{i+1}}(\hat{s}_{i+1} + \hat{s}^*)\right] \end{aligned}$$

On the one hand, as $L^* \to L_{i+1}$, $(a^* - a_i)$ will increase to $(a^{i+1} - a_i)$, but $\hat{s}^* - \frac{L^*}{L_{i+1}}(\hat{s}_{i+1} + \hat{s}^*)$ will decrease to $-\hat{s}_{i+1}$; On the other hand, when L^* gets close to L^i, $a^* - a^i \to 0$. This trade-off indicates that the layer size of M^* should be carefully chosen to achieve performance improvements. Otherwise, the overall gain could be negative. Besides, the upper bound of maximum gain also depends on the number of samples exit at M^*. Thus, adjusting thresholds properly is also important. Additionally, we can further scale the upper bound as:

$$\begin{aligned} &T(\hat{s}^*, L^*) \\ &\le \frac{1}{N}(a_{i+1} - a_i)\left[\hat{s}^* - \frac{L_i}{L_{i+1}}(\hat{s}_{i+1} + \hat{s}^*)\right] \\ &\le \frac{s_i + s_{i+1}}{N}(a_{i+1} - a_i)\left(1 - \frac{L_i}{L_{i+1}}\right) \end{aligned}$$

which indicates that

$$\begin{aligned} &\max_{s^*, L^*}\{T(\hat{s}^*, L^*)\} \\ &\le \frac{s_i + s_{i+1}}{N}\left(\max_i\{a_{i+1} - a_i\}\right)\left(1 - \min_i\left\{\frac{L_i}{L_{i+1}}\right\}\right). \end{aligned}$$

Note that

$$\max_i\{a_{i+1} - a_i \mid M_i, \cdots, M_n\} \tag{14}$$

and

$$\min_i\left\{\frac{L_i}{L_{i+1}} \mid M_i, \cdots, M_n\right\} \tag{15}$$

are non-increasing as n gets larger. It means the maximum expected performance gain of adding another model can be marginal as the number of models in the original cascade becomes larger.

In all, our analysis shows that we should carefully select model M^* with layers L^*, and tune the exiting threshold to adjust number of samples exit after M^*, to guarantee that the target in Eq. 13 is positive, in order to gain improvements by incorporating more models into the original cascade.