# BERxiT: Early Exiting for BERT with Better Fine-Tuning and Extension to Regression

**Ji Xin, Raphael Tang, Yaoliang Yu,** and **Jimmy Lin**

David R. Cheriton School of Computer Science, University of Waterloo
Vector Institute for Artificial Intelligence

{ji.xin,r33tang,yaoliang.yu,jimmylin}@uwaterloo.ca

## Abstract

The slow speed of BERT has motivated much research on accelerating its inference, and the early exiting idea has been proposed to make trade-offs between model quality and efficiency. This paper aims to address two weaknesses of previous work: (1) existing fine-tuning strategies for early exiting models fail to take full advantage of BERT; (2) methods to make exiting decisions are limited to classification tasks. We propose a more advanced fine-tuning strategy and a learning-to-exit module that extends early exiting to tasks other than classification. Experiments demonstrate improved early exiting for BERT, with better trade-offs obtained by the proposed fine-tuning strategy, successful application to regression tasks, and the possibility to combine it with other acceleration methods. Source code can be found at https://github.com/castorini/berxit.

## 1 Introduction

Large-scale pre-trained language models such as BERT (Devlin et al., 2019) have brought the natural language processing (NLP) community large performance gain but at the cost of heavy computational burden. While pre-trained models are available online and fine-tuning is typically done without a strict time budget, inference poses a much lower latency tolerance, and the slow inference speed of these models can impede easy deployment. It becomes even more difficult when inference has to be done on edge devices due to limited network capabilities or privacy concerns.

Early exiting (Schwartz et al., 2020; Xin et al., 2020a; Liu et al., 2020) has been proposed to accelerate the inference of BERT and models with similar architecture, i.e., those comprising multiple transformer layers (Vaswani et al., 2017) with a classifier at the top. Instead of using only one classifier, additional classifiers are attached to each transformer layer (see Figure 1), and the entire model is fine-tuned together. At inference time, the sample can perform early exiting through one of the intermediate classifiers.

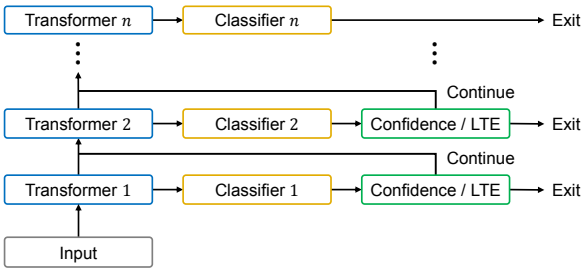


Figure 1: Multi-output structure of early exiting BERT.

While existing early exiting papers provide promising quality–efficiency trade-offs, improvements are necessary for two important components: fine-tuning strategies and exiting decision making. In these papers, fine-tuning strategies are relatively simple and fail to take full advantage of the pre-trained model's effectiveness; we propose a novel fine-tuning strategy, *Alternating*, for this multi-output model. Moreover, previous work makes exiting decisions based on the confidence of output probability distributions, and is hence only applicable to classification tasks; we extend it to other tasks by proposing the *learning-to-exit* idea. With carefully designed fine-tuning strategies and methods for making exiting decisions, the model can achieve better quality–efficiency trade-offs and can be extended to regression tasks.

We refer to our proposed ideas collectively as BERxiT (BERT+exit), and apply it to Muppets[1] including BERT, RoBERTa (Liu et al., 2019), and ALBERT (Lan et al., 2020); we also apply it on top of another BERT acceleration method, DistilBERT (Sanh et al., 2019). We conduct experiments

[1] BERT and his friends.



*Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics*, pages 91–104
April 19 - 23, 2021. 

on datasets including classification and regression tasks, and show that our method can save up to 70% of inference time with minimal quality degradation.

Our contributions include the following: (1) an effective fine-tuning method *Alternating*; (2) the *learning-to-exit* idea that extends early exiting to tasks other than classification; (3) extensive experiments that show the effectiveness of our ideas and the successful combination of early exiting with other BERT acceleration methods; (4) additional experiments that provide insight into the inner mechanism of pre-trained models.

## 2 Related Work

BERT (Devlin et al., 2019) is a pre-trained multi-layer transformer (Vaswani et al., 2017) model. RoBERTa (Liu et al., 2019) and ALBERT (Lan et al., 2020) are variants of BERT with almost identical model architectures but different training methods and parameter sharing strategies. In our paper, we refer to these transformer-based models as Muppets, and apply our method on them.

In the general deep learning context, there are a number of well-explored methods to accelerate model inference. Pruning (Han et al., 2015; Fan et al., 2020; Gordon et al., 2020) removes unimportant parts of the neural model, from individual weights to layers and blocks. Quantization (Lin et al., 2016; Shen et al., 2020) reduces the number of bits needed to operate a neural model and to store its weights. Distillation (Hinton et al., 2015; Jiao et al., 2020) transfers knowledge from large teacher models to small student models. These methods typically require pre-training Muppets from scratch[2] and produce only one small model with a predetermined target size. Early exiting requires only fine-tuning and also produces a series of small models, from which the user can choose flexibly. It extends the idea of Adaptive Computation (Graves, 2016) for recurrent neural networks, and is also closely related to BranchyNet (Teerapittayanon et al., 2016), Multi-Scaled DenseNet (Huang et al., 2018), and Slimmable Network (Yu et al., 2019).

Early exiting for Muppets has been explored by RTJ[3] (Schwartz et al., 2020), DeeBERT (Xin et al., 2020a,b), and FastBERT (Liu et al., 2020). Despite their promising results, there is still room for improvement regarding the fine-tuning strategies of RTJ and DeeBERT. FastBERT, on the other hand, uses self-distillation (Zhang et al., 2019; Phuong and Lampert, 2019) for fine-tuning, which works well for small Muppets such as $\text{BERT}_{\text{BASE}}$. However, our preliminary experiments[4] show that self-distillation is unstable on larger Muppets such as $\text{BERT}_{\text{LARGE}}$, suggesting that future work is necessary for fully understanding and robustly applying self-distillation on Muppets.

All these three methods make early exiting decisions based on confidence (or its variants) of the predicted probability distribution, and are therefore limited to classification tasks. Runtime Neural Pruning (Lin et al., 2017), SkipNet (Wang et al., 2018b), and BlockDrop (Wu et al., 2018) use reinforcement learning (RL) to decide whether to execute a network module. Universal Transformer (Dehghani et al., 2019) and Depth-Adaptive Transformer (DAT, Elbayad et al., 2020) use learned decisions for early exiting in sequence-to-sequence tasks. Concurrently, PABEE (Zhou et al., 2020) proposes patience-based early exiting which is applicable to regression, but it relies on inter-layer prediction consistency and is therefore not very efficient for exiting at early layers. Inspired by them, we propose a method to extend early exiting for Muppets to regression tasks, using only layer-specific information as in classification. Moreover, our method requires neither RL nor complicated distribution fitting as in DAT, but uses a straightforward layer-wise certainty estimation, and achieves performance comparable with confidence-based early exiting on classification tasks.

## 3 Model Structure and Fine-Tuning

We start from a pre-trained Muppet model (the *backbone* model), attach additional classifiers to it, fine-tune the model, and use it for accelerated inference by early exiting.

**Backbone model** The backbone model is an $n$-layer pre-trained Muppet model. We denote the $i^{th}$ layer hidden state corresponding to the `[CLS]` token as $\mathbf{h}_i$:

$$\mathbf{h}_i = f_i(x; \theta_1, \cdots, \theta_i), \tag{1}$$

where $x$ is the input sequence, $\theta_i$ is the parameters of the $i^{th}$ transformer layer, and $f_i$ is the mapping from input to the $i^{th}$ layer hidden state.

[2] In distillation, there is typically a general distillation that uses the large-scale pre-training corpus and is time consuming (Jiao et al., 2020).

[3] Short for *Right Tool for the Job*—the paper does not provide a concise name for the method.

[4] See Appendix A.

**Classifiers** In the original BERT paper (Devlin et al., 2019), the way to fine-tune is to attach a classifier to the *final* transformer layer, and then to jointly update both the backbone model and the classifier. The classifier is a one-layer fully-connected network. It takes as input the final layer hidden state $\mathbf{h}_n$ and outputs a prediction. Its output is a probability distribution over all classes for classification tasks and a scalar for regression[5] tasks.

To enable early exiting, we instead attach a classifier to *every* transformer layer, i.e., there are $n$ classifiers in total. Each classifier can make its own prediction, and therefore the model can accelerate inference by exiting earlier.

**Fine-tuning strategies** We discuss how to fine-tune this multi-output network. The loss function for the $i^{th}$ layer classifier is

$$L_i(x, y) = H(y, g_i(\mathbf{h}_i; w_i)), \tag{2}$$

where $x$ and $y$ are the input sequence and corresponding label, $g_i$ the $i^{th}$ layer's classifier, $w_i$ the parameters of $g_i$, and $H$ the task-specific loss function, e.g., cross-entropy for classification tasks and mean squared error (MSE) for regression tasks.

The most straightforward fine-tuning strategy is perhaps minimizing the sum of all classifiers' loss functions and jointly updating all parameters in the process. We refer to this strategy as *Joint*, and it is also used in RTJ (Schwartz et al., 2020):

$$\min_{\substack{\theta_1,\cdots,\theta_n \\ w_1,\cdots,w_n}} \sum_{i=1}^{n} L_i. \tag{3}$$

If we hope to preserve the best model quality for the final layer, the desired fine-tuning strategy is *Two-stage*, which is also used in DeeBERT (Xin et al., 2020a). The first stage is identical to vanilla BERT fine-tuning: updating the backbone model and only the final classifier. In the second stage, we freeze all parameters updated in the first stage, and fine-tune the remaining classifiers. Objectives in the two stages are as follows.

$$\text{Stage 1:} \quad \min_{\substack{\theta_1,\cdots,\theta_n \\ w_n}} \quad L_n \tag{4}$$

$$\text{Stage 2:} \quad \min_{w_1,\cdots,w_{n-1}} \quad \sum_{i=1}^{n-1} L_i \tag{5}$$

[5]In this case, we still refer to this one-layer network as *classifier* for naming consistency.

These two fine-tuning strategies are not ideal. Intuitively, in this multi-output network, loss functions of different classifiers interfere with each other in a *negative* way. Transformer layers have to provide hidden states for two competing purposes: immediate inference at the adjacent classifier and gradual feature extraction for future classifiers. Therefore, achieving a balance between the classifiers is critical. *Two-stage* produces final classifiers with optimal quality at the price of earlier layers, since most parameters are solely optimized for the final classifier. *Joint* treats all classifiers equally, and therefore its final classifier is less effective than that of *Two-stage*. To combine the advantages, we propose a novel fine-tuning strategy, *Alternating*. It alternates between two objectives (taken from Equation 3 and 4) for odd-numbered and even-numbered iterations.

$$\text{Odd:} \quad \min_{\substack{\theta_1,\cdots,\theta_n \\ w_n}} \quad L_n \tag{6}$$

$$\text{Even:} \quad \min_{\substack{\theta_1,\cdots,\theta_n \\ w_1,\cdots,w_n}} \quad \sum_{i=1}^{n} L_i \tag{7}$$

Combining objectives from *Joint* and *Two-stage*, *Alternating* has the potential to find the most preferable region in the parameter space: the intersection between optimal regions for different layers.

## 4 Exiting Decision Making

After the entire model (including the backbone and all classifiers) is fine-tuned, it can perform early exiting for an inference sample. In this section we discuss two methods to make exiting decisions.

### 4.1 Confidence Threshold

When the model is "certain" enough of its prediction at an intermediate layer, the forward inference can be terminated.

For classification tasks, a straightforward measurement of the prediction certainty is the maximum probability of the output prediction, which is referred to as *confidence* in previous work (Schwartz et al., 2020; Liu et al., 2020). Similarly, Xin et al. (2020a) use entropy as the metric, which is also closely related to confidence. Before inference starts, a confidence threshold is chosen. In forward propagation, the confidence of the output at each layer is compared with the threshold; if it is larger than the threshold at a certain layer, the sample exits and future layers are skipped.

### 4.2 Learning to Exit

While using a confidence or entropy threshold is straightforward and effective, it is exploiting the fact that the classifier's output is a probability distribution in classification tasks. This is generally not the case for other tasks such as regression. To address the gap, we propose *learning-to-exit* (LTE) as a substitute when the distribution is unavailable.

The $i^{th}$ layer hidden state $\mathbf{h}_i$ is a vector in the embedding space. Intuitively, different regions of the embedding space have different certainty levels. For instance, in binary classification tasks, regions closer to the decision boundary have a lower certainty level, and this is explicitly expressed as a lower confidence of the output probability distribution. But even when certainty cannot be explicitly measured, we can still train an auxiliary LTE module to estimate such a metric.

Concretely, the LTE module is a simple one-layer fully-connected network. It takes as input the hidden state $\mathbf{h}_i$ and outputs the certainty level $u_i$ of the sample at the $i^{th}$ layer:

$$u_i = \sigma(\mathbf{c}^\top \mathbf{h}_i + b), \tag{8}$$

where $\sigma$ is the sigmoid function, $\mathbf{c}$ is the weight vector, and $b$ is the bias term.

The loss function for the LTE module is a simple MSE between $u_i$ and the "ground truth" certainty level at the $i^{th}$ layer $\tilde{u}_i$:

$$J_i = ||u_i - \tilde{u}_i||_2^2. \tag{9}$$

For classification, the ground truth certainty level is whether the classifier makes the correct prediction:

$$\tilde{u}_i = \mathbb{1}[\arg\max_j g_i^{(j)}(\mathbf{h}_i; w_i) = y], \tag{10}$$

where $g_i$ is the output probability distribution at the $i^{th}$ layer and $g_i^{(j)}$ is its $j^{th}$ entry. For regression, the ground truth certainty level is negatively related to the prediction's absolute error:

$$\tilde{u}_i = 1 - \tanh(|g_i(\mathbf{h}_i; w_i) - y|). \tag{11}$$

To apply LTE, we initialize the LTE module together with classifiers and it is shared among all layers. We train the LTE module jointly with the rest of the model by *substituting* $L_i$ in Equation 3–7 with $L_i + J_i$. At inference time, if the predicted certainty level is higher than the chosen threshold, the inference sample performs early exiting.

| Dataset | Labels | 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 |
| STS-B | 1 | 8.6k / 1.5k / 1.4k |
| SICK | 1 | 4.4k / 4.9k / – |

Table 1: Statistics of datasets.

## 5 Experiments

### 5.1 Setup

We conduct experiments on six classification datasets of the GLUE benchmark (Wang et al., 2018a); since there is only one regression dataset, STS-B (Cer et al., 2017), in GLUE, we additionally use another regression dataset, SICK (Marelli et al., 2014). Statistics of these datasets are listed in Table 1. Our implementation is adapted from the Huggingface Transformer Library (Wolf et al., 2020). We conduct searches on experiment settings such as the optimizer, learning rates, hidden state sizes, and dropout probabilities, and discover that it is best to keep original settings from the library. Random seeds are also unchanged from the library for fair comparisons.[6]

Most results in this paper use the dev split, since the large number of evaluations we need are forbidden by the GLUE evaluation server. The only exception is Table 2, where we report model quality–efficiency trade-offs on the test split.

### 5.2 Layer-wise Scores Comparison

We discuss three fine-tuning strategies in Section 3: *Joint* (also used in RTJ), *Two-stage* (also used in DeeBERT), and *Alternating* (proposed in this paper). In tables and figures, they are labeled respectively as JOINT, 2STG, and ALT. Figures 2 and 3 compare these three fine-tuning strategies by showing their *layer-wise score* curves: each point in the curve shows the output score at a certain exit layer, i.e., all samples are required to exit at this layer for evaluation. More specifically, we report *relative* scores, and the 100% baseline is the original score of the vanilla Muppet without early exiting, and

[6]Detailed experiment settings are in Appendix B.

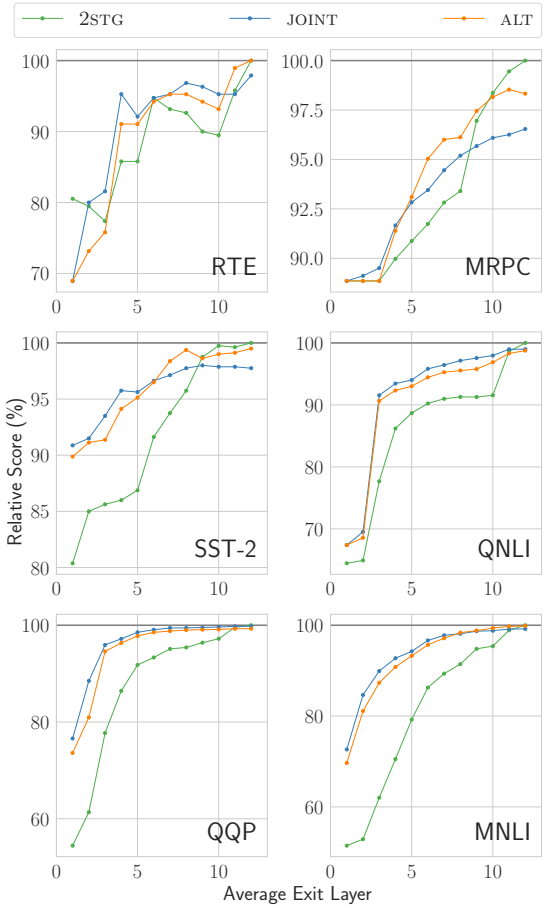


Figure 2: Layer-wise scores of different fine-tuning strategies for $\text{BERT}_{\text{BASE}}$.

this is also the score of the final layer of *Two-stage* because of parameter freezing in its second stage.

For $\text{BERT}_{\text{BASE}}$, we show plots for all six classification datasets, ordered by their training set sizes from smallest to largest. As we will see in later analyses, low-resource datasets show the most difference. Therefore, for $\text{RoBERTa}_{\text{BASE}}$ and $\text{ALBERT}_{\text{BASE}}$, we only show plots for RTE and MRPC (with training set size smaller than 6% of others) due to space limitations.[7]

We observe the following from the figures:

- *Two-stage* is unsatisfying. While it achieves the best score at the final layer, it comes at a large cost of other layers, especially for non-low-resource datasets.

- *Alternating* is better than *Joint* in later layers, and weaker in earlier layers. However, as we will see in the next section, when we evaluate quality–efficiency trade-offs of confidence-based early exiting, the weakness of *Alternating* in earlier layers is no longer substantial, while its advantage is preserved.

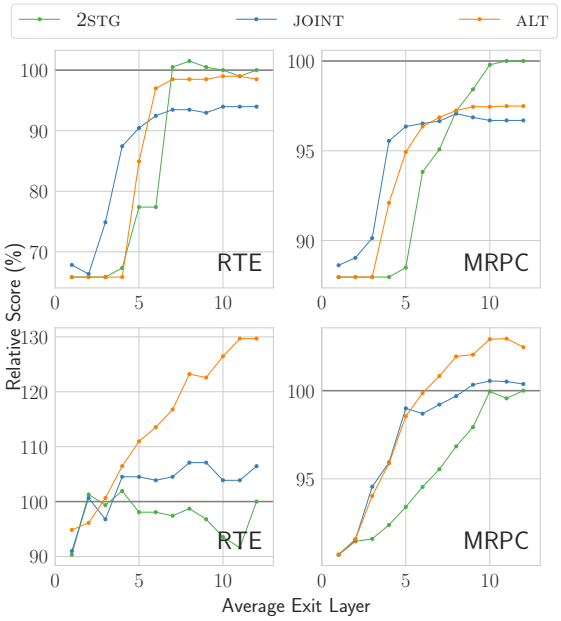


Figure 3: Layer-wise scores of different fine-tuning strategies for $\text{RoBERTa}_{\text{BASE}}$ (top two) and $\text{ALBERT}_{\text{BASE}}$ (bottom two).

- The difference between *Joint* and *Alternating* is larger for low-resource datasets, where the training set is insufficient to fine-tune all layers well simultaneously.

- Interestingly, for $\text{ALBERT}_{\text{BASE}}$, *Alternating*'s relative scores are higher than 100% in the final layers. We speculate that this is because of the parameter sharing nature of ALBERT and the small sizes of the datasets: better supervision for intermediate layers also helps the final layer.

### 5.3 Early Exiting Trade-offs Comparison

From the previous section, we see that *Two-stage* is visibly less preferable than the other two. Therefore in this section, we compare quality–efficiency trade-offs of *Joint* and *Alternating* when confidence threshold is used for making exiting decisions.

Specifically, we use *average exit layer* of all inference samples as the metric of efficiency for the following reasons: (1) it is linear w.r.t. the actual amount of computation; (2) according to our experiments, it is proportional to actual wall-clock runtime, and is also stable across different runs.[8]

[7] Results for other datasets are in Appendix C.

[8] Direct runtime measurement has the randomness caused

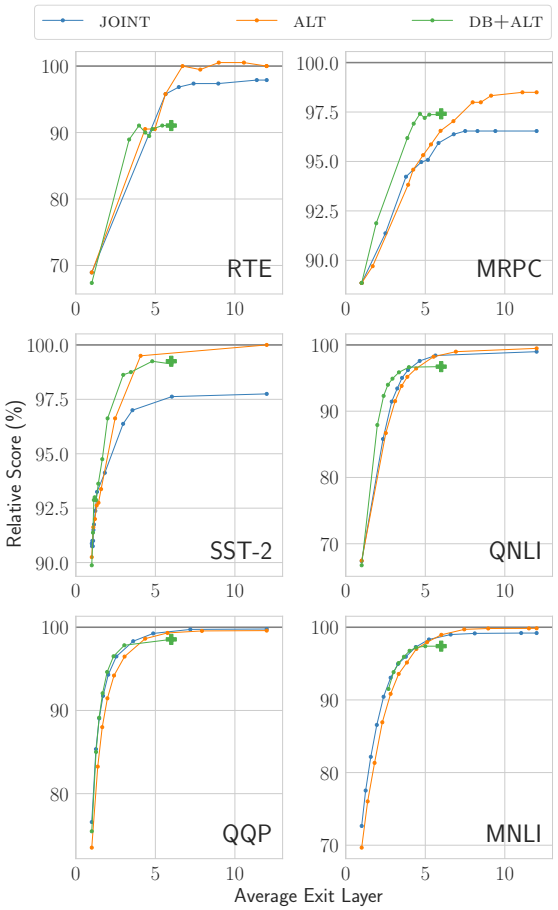


Figure 4: Quality–efficiency trade-offs using confidence for exiting decisions for $\text{BERT}_{\text{BASE}}$.

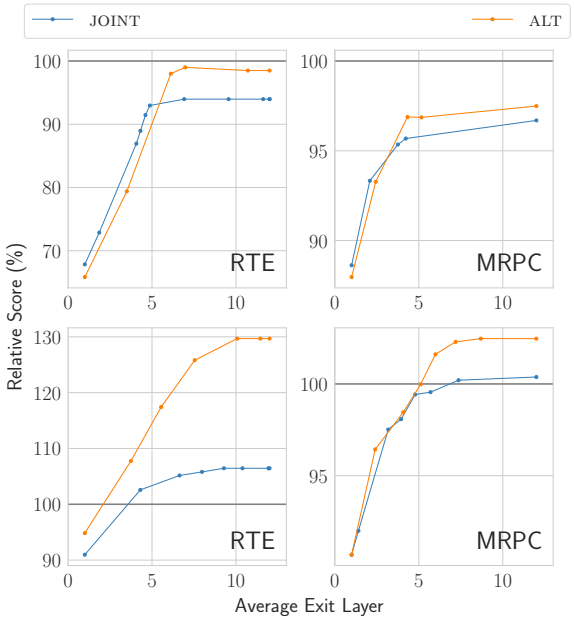


Figure 5: Quality–efficiency trade-offs using confidence for exiting decisions for $\text{RoBERTa}_{\text{BASE}}$ (top two) and $\text{ALBERT}_{\text{BASE}}$ (bottom two).

We visualize the trade-offs in Figures 4 and 5, and also show detailed numbers in Table 2 using results from the test set. Dots in the figures and ALT rows in the table are generated by varying the confidence threshold, and the thresholds are chosen to show trade-offs at different average exit layers. In addition to the comparison between *Joint* and *Alternating*, we add another strong baseline, DistilBERT (Sanh et al., 2019). We apply *Alternating* fine-tuning and early exiting on top of DistilBERT (labeled as DB+ALT), and the rightmost point of the curve is DistilBERT itself without early exiting (the green ✚). Observations from the table and figures are as follows:

- On the test set, early exiting with *Alternating* fine-tuning saves a large amount of inference computation, with only minimal quality degradation, compared with vanilla Muppets.
- Compared with *Joint*, *Alternating* inherits its benefits from the previous section: better trade-offs at higher scores (larger average exit layer). Additionally, its improvements are larger in smaller datasets.
- *Alternating*'s weakness at more aggressive exiting (smaller average exit layer) is minimized. Take Figure 4 as an example, we report *the area of one curve above the other* as a numerical metric: JOINT over ALT and ALT over JOINT is respectively (0.4, 13.5) for RTE, (0.9, 8.8) for MRPC, and (0.2, 18.2) for SST-2. The advantage of *Alternating* indicates that later layers intrinsically contribute more to early exiting performance, partly because the final layer's score is the upper bound for all previous layers (ignoring randomness in training). This shows that the *Joint* fine-tuning strategy, which treats all layers equally, is not ideal.
- In most cases, *Alternating* outperforms DistilBERT, which requires distillation in pre-training and is therefore much more resource-demanding. It also further improves model efficiency on top of DistilBERT, indicating that early exiting is *cumulative* with other acceleration methods.

### 5.4 Learning to Exit Performance

To examine the effectiveness of LTE, we apply it on top of models fine-tuned with *Alternating*. We show the results in Figure 6 on four datasets.

by other processes on the same machine. Detailed discussions can be found in Appendix D.

| | RTE | | MRPC | | SST-2 | | QNLI | | QQP | | MNLI-(m/mm) | | STS-B | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Score | Layer | Score | Layer | Score | Layer | Score | Layer | Score | Layer | Score | Layer | Score | Layer |
| **BERT$_{\text{BASE}}$** | | | | | | | | | | | | | | |
| RAW | 66.4 | 12 | 88.9 | 12 | 93.5 | 12 | 90.5 | 12 | 71.2 | 12 | 84.6/83.4 | 12 | 85.8 | 12 |
| ALT | 101% | −44% | 99% | −30% | 98% | −65% | 99% | −42% | 99% | −56% | 99%/99% | −37% | 95% | −50% |
| | 99% | −54% | 97% | −56% | 96% | −79% | 98% | −63% | 97% | −75% | 97%/97% | −57% | 91% | −67% |
| | 96% | −64% | 94% | −74% | 94% | −87% | 95% | −71% | 93% | −84% | 93%/92% | −72% | 85% | −75% |
| DB | 86% | −50% | 97% | −50% | 98% | −50% | 97% | −50% | 98% | −50% | 97%/97% | −50% | 94% | −50% |
| DB+ALT | 86% | −55% | 97% | −60% | 98% | −75% | 97% | −72% | 98% | −76% | 96%/97% | −66% | 93% | −66% |
| **BERT$_{\text{LARGE}}$** | | | | | | | | | | | | | | |
| RAW | 70.1 | 24 | 89.3 | 24 | 94.9 | 24 | 92.7 | 24 | 72.1 | 24 | 86.7/85.9 | 24 | 86.5 | 24 |
| ALT | 95% | −33% | 99% | −32% | 100% | −32% | 97% | −62% | 98% | −74% | 99%/99% | −36% | 97% | −39% |
| | 94% | −46% | 98% | −46% | 99% | −61% | 95% | −73% | 96% | −82% | 96%/97% | −57% | 90% | −62% |
| | 88% | −62% | 94% | −71% | 96% | −78% | 91% | −83% | 91% | −89% | 90%/90% | −75% | 76% | −80% |

Table 2: Test set results comparing baselines (raw BERT/RoBERTa, from the original paper), DistilBERT, and early exiting with *Alternating* fine-tuning. Metric for model quality: score for RAW baselines and relative scores for others. Metric for model efficiency: used layers for RAW; relative saved layers for others (w.r.t. raw models).

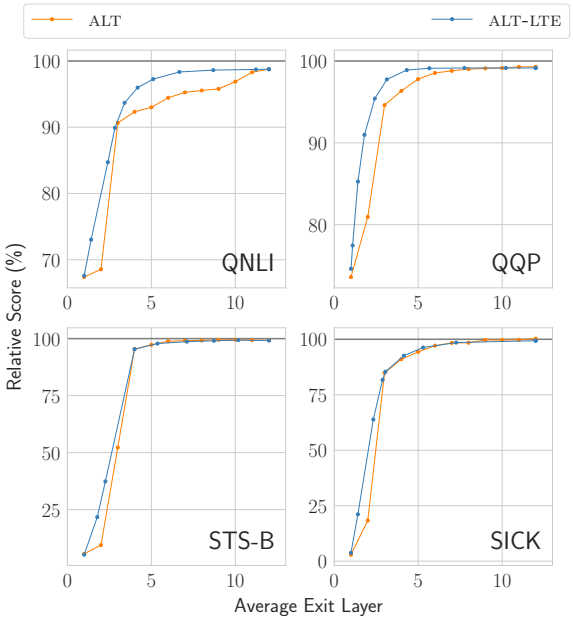


Figure 6: Comparing layer-wise score of *Alternating* with LTE-based early exiting on top of *Alternating*. BERT$_{\text{BASE}}$ is the backbone.

| Method | Rel. Score | Speedup | Avg. Exit Layer |
|---|---|---|---|
| PABEE | 99% | 2.1 | 5.7 |
| | 98% | 2.4 | 5.0 |
| ALT-LTE | 99% | 2.2 | 5.4 |
| | 98% | 2.6 | 4.6 |

Table 3: Comparing LTE with PABEE on STS-B.

We use the *layer-wise* score of *Alternating* as the baseline: if we want to save $x\%$ inference runtime, a straightforward way is to use the first $(100-x)\%$ layers for every sample, regardless of its difficulty. LTE is expected to dynamically allocate resources based on a sample's difficulty and therefore outperform this baseline. From the figures for QNLI and QQP, we observe that the blue curves are substantially above the orange curves, i.e., LTE provides better accuracy–efficiency trade-offs than the layer-wise baseline, achieving the same model quality with less computation. For regression tasks STS-B and SICK, the layer-wise baseline reaches its maximum score at relatively early layers, leaving little room for LTE to perform. Nevertheless, LTE still outperforms the baseline, especially in earlier layers (note that the $y$-axis is from 0 to $100\%$).

We also compare LTE with the concurrent patience-based baseline PABEE (Zhou et al., 2020) in Table 3, showing their speedups and average exit layers at the same relative scores. PABEE does not provide exact speedup numbers; therefore we estimate the values from their figures. We can see that *Alternating* fine-tuning plus LTE is marginally better than PABEE on regression tasks.

We further compare LTE-predicted certainty for each layer with layer-wise scores in Figure 7, where we observe large differences of predicted certainty both within and across layers. Also, predicted certainty is generally positively correlated with scores. This further demonstrates that the LTE module successfully captures certainty information based on the model's hidden state.

LTE extends confidence-based early exiting to

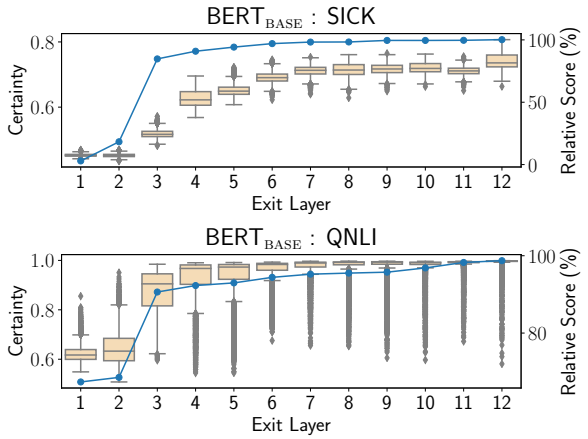


Figure 7: Comparison of each layer's score and learned certainty. Yellow box-plots: distribution of certainty at each layer; blue curve: relative score.

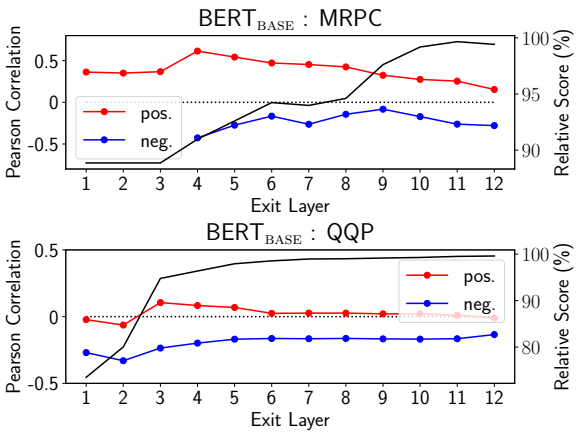


Figure 8: Comparison between BLEU–confidence correlation (red and blue) and layer-wise scores (black).

tasks other than classification. Furthermore, our LTE module is more straightforward and intuitive than DAT (Elbayad et al., 2020), yet achieves comparable results with classification tasks.

### 5.5 Prediction Confidence as a Probe

So far, we have only regarded prediction confidence as something produced by the black-box model and use it for making early exiting decisions. In this section, we show an example of how confidence is related to a human-interpretable feature, demonstrating its potential to reveal the inner mechanisms of Muppet models.

We choose two datasets, MRPC and QQP, where the task is to predict whether two input sequences are semantically equivalent. Intuitively, the BLEU score (Papineni et al., 2002) between the two sequences, which measures n-gram matching, may be related to the prediction. At each output layer, we first divide all dev set samples into two subsets by whether they are predicted as positive or negative; then, we calculate the BLEU-4 score for each sample, and calculate the Pearson correlation between BLEU scores and confidence in each subset; finally, we compare the correlation for both subsets in each layer, along with the layer-wise relative scores, in Figure 8.

We notice that the BLEU scores and predicted confidence show the strongest correlation in layers where the model quality starts to improve (layer 4–5 in MRPC[9] and 2–3 in QQP). After these layers, the correlation gradually weakens. It suggests that in the early layers, the model relies more on simple features such as n-gram matching for making semantic judgments: the higher the BLEU score is, the more certain it is for making positive predictions and the less certain it is for negative ones; however, with more layers, the model acquires the ability to look beyond the BLEU score, reducing its reliance on n-gram matching and achieving better performance. Therefore, with MRPC as an example, analyzing differences between layers 3 and 4 may reveal how the model detects n-gram matching, and analyzing differences between layers 4 and 6 may reveal advanced semantic features learned by the model.

[9]In MRPC, the first three layers only make positive predictions due to the highly imbalanced training label distribution.

## 6 Conclusion and Future Work

To improve early exiting for Muppets, we present BERxiT, including the *Alternating* fine-tuning strategy which outperforms methods from previous papers and the LTE idea which extends early exiting to a broader range of tasks. Experiments show the effectiveness of *Alternating* in providing better quality–efficiency trade-offs and the successful application of LTE to regression tasks. They also show that early exiting is cumulative with other acceleration methods such as DistilBERT and has the potential for model interpretation.

**Future Work** The fundamental question of early exiting for Muppets is how many transformer layers are sufficient for making good predictions. We draw inspiration from the *Limit* performance of Muppets: the score of *Limit* at the $i^{th}$ layer is obtained by taking the first $i$ transformer layers from the pre-trained Muppet model, attaching a classifier to the $i^{th}$ layer, and fine-tuning this single-output

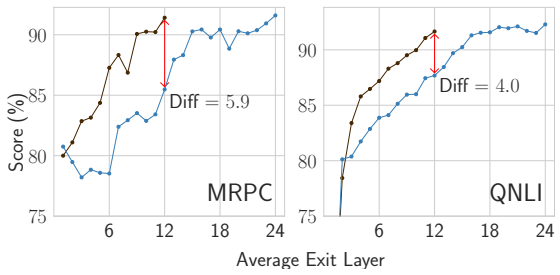


Figure 9: Comparison between LIMIT of $\text{BERT}_{\text{BASE}}$ (brown) and $\text{BERT}_{\text{LARGE}}$ (blue). Red arrow: difference between the two models when they both use 12 layers.

model. *Limit* estimates the upper bound for any fine-tuning methods by removing inter-classifier interference. We compare the *Limit* performance of $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$ in Figure 9, and notice that with the same number of layers (and identical fine-tuning strategy), $\text{BERT}_{\text{BASE}}$ almost always outperforms $\text{BERT}_{\text{LARGE}}$ by a large margin. This suggests that most transformer layers' potential to provide information for early exiting is limited by the single-output nature of pre-training. If we want to further improve early exiting Muppets for better trade-offs, adding more exiting paths in pre-training would be a promising direction.

## Acknowledgements

We thank anonymous reviewers for their insightful suggestions. We also gratefully acknowledge funding support from the Natural Sciences and Engineering Research Council (NSERC) of Canada. Computational resources used in this work were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute.

## References

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. Association for Computational Linguistics.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *International Conference on Learning Representations*.

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

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020. Depth-adaptive transformer. In *International Conference on Learning Representations*.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*.

Mitchell A Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing BERT: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307*.

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

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

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

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. 2018. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*.

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 the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2849–2858, New York, New York, USA. PMLR.

Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. 2017. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

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

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*.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 216–223, Reykjavik, Iceland. European Language Resources Association (ELRA).

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Mary Phuong and Christoph H. Lampert. 2019. Distillation-based training for multi-exit architectures. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

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

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 *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018a. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. 2018b. SkipNet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi 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 Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogerio Feris. 2018. BlockDrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

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

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

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. 2019. Slimmable neural networks. In *International Conference on Learning Representations*.

Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. 2019. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

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 H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18330–18341. Curran Associates, Inc.

## A Negative Results for Self-Distillation

FastBERT (Liu et al., 2020) does not provide result for $\text{BERT}_{\text{LARGE}}$ or $\text{RoBERTa}_{\text{LARGE}}$. We show $\text{BERT}_{\text{LARGE}}$ and $\text{RoBERTa}_{\text{LARGE}}$ layer-wise scores for different fine-tuning strategies in Figure 10. SD in the legend stands for self-distillation. We can see that for *Two-stage* and *Alternating*, the patterns are similar to those of $\text{BERT}_{\text{BASE}}$: *Alternating* better in earlier layers while *Two-stage* better in later layers.

However, self-distillation's behavior is inconsistent between models and datasets. While it performs as expected for $\text{BERT}_{\text{LARGE}}$ in SST-2 and MNLI, and for $\text{RoBERTa}_{\text{LARGE}}$ in MRPC, self-distillation fails to improve after the first few layers for $\text{BERT}_{\text{LARGE}}$ in MRPC and for $\text{RoBERTa}_{\text{LARGE}}$ in SST-2 and MNLI, and most layers' quality is considerably worse than *Alternating*. We therefore consider self-distillation an unstable and premature fine-tuning strategy.

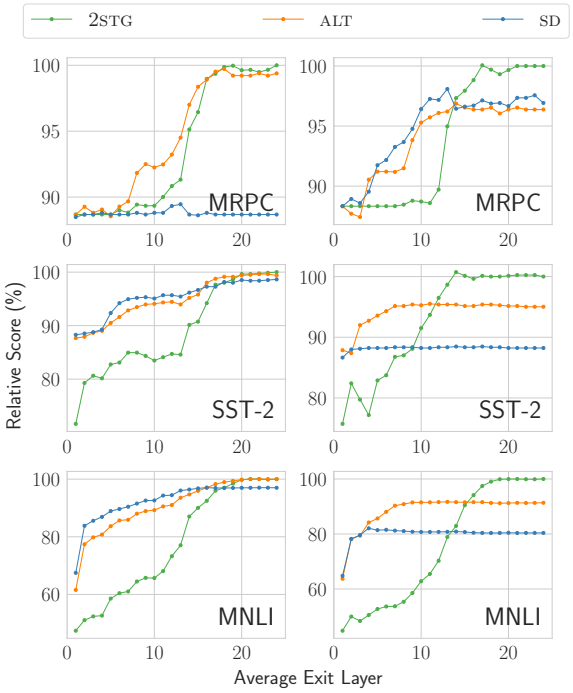


Figure 10: Layer-wise score for different fine-tuning strategies on $\text{BERT}_{\text{LARGE}}$ (left column) and $\text{RoBERTa}_{\text{LARGE}}$ (right column).

## B Additional Experiment Setting

For pre-trained models, we use the following ones provided by the Huggingface Transformer Library (Wolf et al., 2020) as backbone models:

- BERT-BASE-UNCASED
- BERT-LARGE-UNCASED
- ROBERTA-BASE
- ROBERTA-LARGE
- ALBERT-BASE-V2
- DISTILBERT-BASE-UNCASED

For BERT, ALBERT, and DistilBERT, we fine-tune for 3 epochs; for RoBERTa, we fine-tune for 10 epochs; no early-stopping or checkpoint selection is performed.

Experiments are done on a single NVIDIA P100 GPU with CUDA 10.1. For inference, we use a batch size of 1 (since we need to perform early exiting based on each individual sample's difficulty). Inference runtime for the entire dev set for all models and datasets is shown in Table 4. RoBERTa has the same model structure as BERT, and therefore its runtime is also very close to that of BERT. Note that this is affected by competing processes, and may vary between different runs.

Numbers of parameters for BERT and ALBERT backbone models can be found in the paper by Lan et al. (2020). RoBERTa shares the same model structure with BERT and has the same number of parameters. Numbers of parameters for early-exiting-specific modules, such as additional classifiers and the LTE module, are on the order of thousands, and are therefore negligible compared with those of backbone models (millions).

## C Additional Experiment Results

In the main paper, we report results of $\text{RoBERTa}_{\text{BASE}}$ and $\text{ALBERT}_{\text{BASE}}$ only on the two smallest datasets. Results of the other datasets are provided in Figure 11. We can see that while *Two-stage* is visibly less preferable, *Joint* and *Alternating* are close to each other with larger dataset sizes, and this is the reason why we keep only the low-resource datasets in the main paper.

## D Analyses of Efficiency Metric

In our experiments, we use *average exit layer* as the metric of efficiency for the following three reasons.

**It is linear w.r.t. the amount of computation.** Inference time computation in our model occurs in the following parts: the embedding layer, transformer layers, classifiers, and the LTE module (if used). If a layer is chosen, i.e., the exit layer is after

| | RTE | MRPC | SST-2 | QNLI | QQP | MNLI | STS-B | SICK |
|---|---|---|---|---|---|---|---|---|
| BERT-base | 5.8 | 8.4 | 18.0 | 110.4 | 856.8 | 209.4 | 33.2 | 107.8 |
| BERT-large | 11.6 | 17.4 | 35.9 | 223.2 | 1952.8 | 400.3 | 61.3 | 209.8 |
| ALBERT-base | 6.5 | 9.4 | 18.5 | 114.6 | 864.8 | 204.8 | 31.6 | 104.2 |
| DistilBERT | 3.0 | 4.3 | 9.1 | 55.4 | 407.1 | 106.2 | 15.9 | 50.9 |

Table 4: Inference runtime in seconds for each model and dataset.

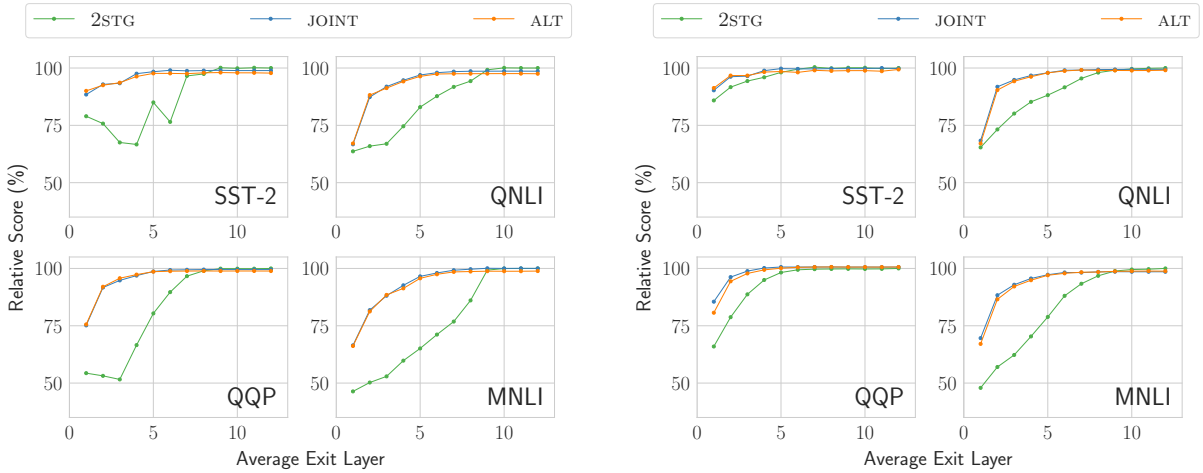


Figure 11: Additional layer-wise score for different fine-tuning strategies on $\text{RoBERTa}_{\text{BASE}}$ (left four) and $\text{ALBERT}_{\text{BASE}}$ (right four).

it, all components of the layer (transformer, classifier, LTE module) are used and incur computation cost. Additionally, embedding look-up (selecting a column in the matrix) is much faster than the above components (involving matrix-vector multiplication), and can therefore be neglected.

**It is stable across different runs.** With a fine-tuned model, an inference sample's exit layer only depends on the confidence (or LTE-predicted certainty) at each layer and the threshold. On the other hand, direct measurement of wall-clock runtime is frequently affected by competing processes and fluctuates between different runs.

**The computation overhead of early exiting is negligible.** With the above reasons, there is only one concern left for using average exit layer as the efficiency metric: how do additional layers in our model (including the additional classifier and possibly the LTE module) compare with transformer layers in the original BERT paper? We estimate FLOPS used in one sample's inference as follows. Since we will eventually end up with orders of magnitude differences, we use the *big-theta* asymptotic notation for estimation.

Most computation is incurred for matrix and vector multiplication. Using the naïve implementation, the cost for multiplying two vectors in $\mathbb{R}^d$ is $\Theta(d)$, and the cost for multiplying a matrix in $\mathbb{R}^{d_1 \times d_2}$ and a vector in $\mathbb{R}^{d_2}$ is $\Theta(d_1 d_2)$.

We denote $d$ as the hidden state size of our model (768 for base models and 1024 for large models), $c$ as the number of classes (less than 4 in our experiments), $n$ as the sequence length (typically in the hundreds), and $h$ as the number of heads in multi-head attention (12 for base and 16 for large).

The classifier is a one-layer fully-connected layer, mapping a vector in $\mathbb{R}^d$ to an output in $\mathbb{R}^c$, therefore the cost is $\Theta(cd)$. Similarly, the cost of the LTE module is $\Theta(2d)$, since its output is always a vector in $\mathbb{R}^2$.

The transformer layer mainly consists of multi-head self-attention, a fully-connected layer, and two layer normalization modules. Layer normalization is much faster than the other two and we therefore neglect it. The fully connected layer maps $n$ vectors from $\mathbb{R}^d$ to $\mathbb{R}^d$, therefore the cost is $\Theta(nd^2)$. The multi-head attention[10] computes $h$ individual uni-head attention. For each uni-head attention, the mapping from original query, key, and value

[10] Details can be found in the paper by Vaswani et al. (2017).

vectors ($\mathbb{R}^d$) to head-specific ones ($\mathbb{R}^{d/h}$) incurs a cost of $\Theta(nd^2/h)$; calculating the attention results incurs a cost of $\Theta(nd/h)$. Therefore the total cost here is $\Theta(nd^2 + nd) = \Theta(nd^2)$. Finally, results of each head are combined and one more matrix–vector multiplication is needed, incurring a cost of $\Theta(nd)$. The total cost of one transformer layer is therefore $\Theta(nd^2)$.

Comparing the above, the ratio of a transformer layer to a classifier/LTE module is

$$\frac{\Theta(nd^2)}{\Theta(cd + 2d)} = \Theta(nd). \tag{12}$$

Considering the value of $n$ and $d$, the classifier and the LTE module are several orders of magnitude lighter than the transformer layer. Even with advanced algorithms and parallel hardware that may accelerate transformer layers, we can still safely come to the conclusion that the computation overhead of early exit is negligible.