LECO: Improving Early Exiting via Learned Exits and Comparison-based Exiting Mechanism

Jingfan Zhang[1], **Ming Tan**[2], **Pengyu Dai**[3,4], **Wei Zhu**[5*]
[1] University of Ottawa, Canada
[2] Southern University of Science and Technology, China
[3] Chongqing University of Post and Telecommunication, China
[4] Brunel University, London
[5] East China Normal University, China

Abstract

Recently, dynamic early exiting has attracted much attention since it can accelerate the inference speed of pre-trained models (PTMs). However, previous work on early exiting has neglected the intermediate exits' architectural designs. In this work, we propose a novel framework, Learned Exits and COmparison-based early exiting (LECO) to improve PTMs' early exiting performances. First, to fully uncover the potentials of multi-exit BERT, we design a novel search space for intermediate exits and employ the idea of differentiable neural architecture search (DNAS) to design proper exit architectures for different intermediate layers automatically. Second, we propose a simple-yet-effective comparison-based early exiting mechanism (COBEE), which can help PTMs achieve better performance and speedup trade-offs. Extensive experiments show that our LECO achieves the SOTA performances for multi-exit BERT training and dynamic early exiting.

1 Introduction

Despite achieving state-of-the-art (SOTA) performances on almost all the natural language processing (NLP) tasks (Lin et al., 2021), large pre-trained language models (PLMs) still have difficulty being applied to many industrial scenarios with low latency requirements. Many research works are devoted to speeding up the inference of BERT or other PLMs, such as network pruning (Zhu and Gupta, 2017; Xu et al., 2020a; Fan et al., 2019; Gordon et al., 2020), student network distillation (Sun et al., 2019; Sanh et al., 2019; Jiao et al., 2020), and early exiting (Teerapittayanon et al., 2016; Xin et al., 2020; Kaya et al., 2019; Xin et al., 2021). Due to its potential in applications, early exiting has attracted much attention in the research field (Xu et al., 2021a). Early exiting requires a multi-exit BERT, a BERT backbone with an intermediate classifier (or exit) installed on each layer. And then, a dynamic early exiting mechanism is applied during the forward pass to ensure efficient inference. Early exiting is in parallel with and can work together with static model compression methods (Tambe et al., 2020). However, the literature focuses less on the training of multi-exit BERT (Teerapittayanon et al., 2016; Xin et al., 2020; Liu et al., 2020; Xin et al., 2021) and there is no literature systematically discussing the architectural design of the intermediate exits.

In this work, we propose a novel framework, Learned Exits and COmparison-based Early exiting (LECO), designated to discover the full potentials of multi-exit BERT in early exiting. First, we design a suitable and comprehensive search space for architectural learning of the intermediate exits (see Figure 1). Our search space contains candidate activation functions, encoding operations, and pooling operations. We follow the differentiable neural architecture search (DNAS) framework like Liu et al. (2019a); Xie et al. (2019); Chen et al. (2021) to learn a set of intermediate exits with different architectures automatically. Second, reflecting on the limitations of the patience-based early exiting method PABEE (Zhou et al., 2020), we propose a comparison-based early exiting (COBEE) mechanism. COBEE makes early exiting decisions by comparing the predicted distributions of adjacent intermediate layers.

We conduct extensive experiments and ablation studies on the GLUE benchmark (Wang et al., 2018). We show that learned intermediate exits of LECO outperform the previous SOTA multi-exiting BERT training methods while adding fewer trainable parameters. Furthermore, our novel dynamic early exiting mechanism COBEE outperforms the previous SOTA early exiting mechanisms. Further analysis shows that: (a) our LECO framework can help to boost the performance of multi-exiting

*Corresponding author: michaelwzhu91@gmail.com

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics - Student Research Workshop, pages 298–309
July 10-12, 2023

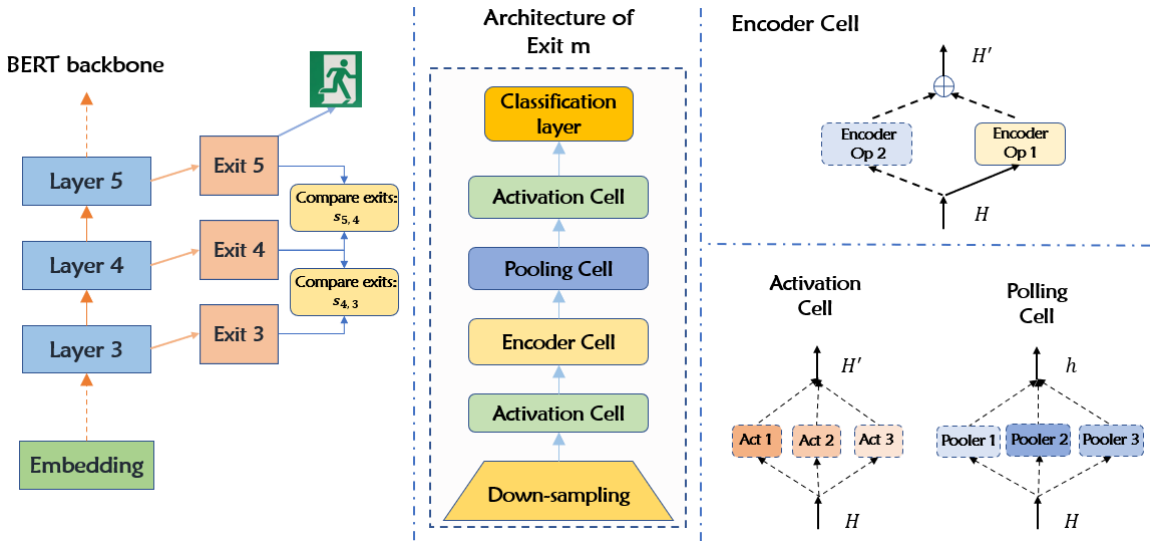


Figure 1: The overall framework of our LECO framework. **Left**: We compare the predicted distributions of adjacent PTMs' intermediate layers for mining exiting signals. **Middle**: the general architecture of intermediate exits. **Right**: Each edge in the the search cell is a weighted sum of multiple operations under the DNAS framework.

BERT under different training strategies. (b) our novel dynamic early exiting strategy outperforms the baseline early exiting methods.

Our contributions are as follows:

- We propose a novel framework, LECO, which constructs a search space for intermediate exits and employs a DNAS framework to learn the suitable exits for different layers.
- We propose a novel comparison-based early exiting criterion which can achieve better quality-speed tradeoffs for PTMs.
- We conduct experiments to show that our LECO achieves SOTA performances for multi-exit BERT training.

2 Related Work

2.1 Inference acceleration methods

Since the rise of BERT, there are quite large numbers of literature devoting themselves to speeding up the inference of BERT. Standard method include direct network pruning (Zhu and Gupta, 2017; Xu et al., 2020a; Fan et al., 2019; Gordon et al., 2020), distillation (Sun et al., 2019; Sanh et al., 2019; Jiao et al., 2020), Weight quantization (Zhang et al., 2020b; Bai et al., 2020; Kim et al., 2021) and Adaptive inference (Zhou et al., 2020; Xin et al., 2020; Liu et al., 2020). Among them, adaptive inference has drawn much attention. Adaptive inference aims to deal with simple examples with only shallow layers of PLMs, thus speeding up inference time on average.

Early exiting requires a multi-exit model, like a BERT backbone with an intermediate classifier (or exit) installed on each layer. Early exiting literature mainly focuses on the development of the early exiting strategies, that is, determining when an intermediate exit's prediction is suitable as the final model prediction. Score based strategies (Teerapittayanon et al., 2016; Xin et al., 2020; Kaya et al., 2019; Xin et al., 2021), prior based strategies (Sun et al., 2022) and patience based strategies (Zhou et al., 2020) have been proposed. Teerapittayanon et al. (2016) uses the entropy of an intermediate layer's predicted distribution to measure the in-confidence level and decide whether to exit early. PABEE asks the model to exit when the current layer's prediction is the same with the previous layers.

Our work complements the literature on early exiting by proposing the LECO framework to improve early exiting performance via the automatic architectural design of exit architectures and a novel early exiting mechanism.

2.2 Neural architecture search

With the rapid development and wide industrial applications, researchers have devoted great effect in manually designing neural netoweks (Krizhevsky et al., 2012; Simonyan and Zisserman, 2015; He et al., 2016; Huang et al., 2017; Wang et al., 2022). The trend is to stack more and more convolutional or transformer layers to construct a deep network. Recently, when trying

to avoid manual architecture design, researchers started considering developing algorithms to design neural networks automatically. Thus, a new research sub-field of automated machine learning (AutoML) (He et al., 2021) called neural architecture search is established (Zoph and Le, 2017).

In the early attempts, NAS requires massive computations, like thousands of GPU days (Zoph and Le, 2017; Zoph et al., 2018; Liu et al., 2018). Recently, a particular group of one-shot NAS, led by the seminal work DARTS (Liu et al., 2019a) has attracted much attention. DARTS formulates the search space into a super-network that can adjust itself in a continuous space so that the network and architectural parameters can be optimized alternately (bi-level optimization) using gradient descent. A series of literature try to improve the performance and efficiency of DARTS. SNAS (Xie et al., 2019) reformulate DARTS as a credit assignment task while maintaining the differentiability. P-DARTS (Chen et al., 2021) analyze the issues during the DARTS bi-level optimization, and propose a series of modifications. PC-DARTS (Xu et al., 2021b) reduces the memory cost during search by sampling partial channels in super-networks. Fair-DARTS (Chu et al., 2021) change the softmax operations in DARTS into sigmoid and introduce a penalty term to prune the architectural parameters according to the demand. Gao et al. (2020) make the hyper-network more close to the discretized sub-network by penalizing the entropy of the architecture parameters.

Our work contributes to the NAS literature by investigate the architectural search of intermediate exits to improve the early exiting performances.

3 Preliminaries

In this section, we introduce the necessary background for BERT early exiting. we consider the case of multi-class classification with K classes, $\mathcal{K} = \{1, 2, ..., K\}$. The dataset consists of N samples $\{(x_i, y_i), i \in \mathcal{I} = \{1, 2, ..., N\}\}$, where x_i is an input sentence consisting of L words, and $y_i \in \mathcal{K}$ is the label.

3.1 Early Exiting

Multi-exit PTM Early exiting is based on multi-exit PTM, which is a PTM backbone with classifiers (or exits) at each layer. With M layers, M classifiers $f_m(x; \theta_m)$ are designated at M layers of the PTM, each of which maps its input to the probability distribution on K classes. $f_m(x; \theta_m)$ can take the form of a simple linear layer (linear exit) following (Zhou et al., 2020). However, as is shown in Liu et al. (2020), adding an encoding operation like the multi-head self-attention layer (Vaswani et al., 2017) to the intermediate exits (MHA exits) can significantly boost the performance of intermediate layers, demonstrating the importance of architectural design.

Training We now introduce the three main multi-exit BERT training methods widely adopted in the literature.

JT. Perhaps the most straightforward fine-tuning strategy is to minimize the sum of all classifiers' loss functions and jointly update all parameters in the process. We refer to this strategy as JT. The loss function is:

$$\mathcal{L}_{JT} = \sum_{m=1}^{M} \mathcal{L}_m^{CE} \tag{1}$$

where $\mathcal{L}_m^{CE} = \mathcal{L}_m^{CE}(y, f_m(x; \theta_m))$ denotes the cross-entropy loss of the m-th exit. This method is adopted by Teerapittayanon et al. (2016); Kaya et al. (2019); Zhou et al. (2020); Zhu (2021).

2ST. The two-stage (2ST) (Xin et al., 2020; Liu et al., 2020) training strategy divides the training procedure into two stages. The first stage is identical to the vanilla BERT fine-tuning, updating the backbone model and only the final exit. In the second stage, we freeze all parameters updated in the first stage and fine-tune the remaining exits separately:

$$\text{Stage1} : \mathcal{L}_{stage1} = \mathcal{L}_M^{CE}(y_i, f_M(x_i; \theta_M)) \tag{2}$$

$$\text{Stage2} : \mathcal{L}_{stage2} = \mathcal{L}_m^{CE}, m = 1, ..., M-1. \tag{3}$$

where $\mathcal{L}_m^{CE} = \mathcal{L}_m^{CE}(y_i, f_m(x_i; \theta_m))$ denotes the cross-entropy loss of m-th exit.

ALT. It alternates between two objectives (taken from Equation 1 and 2) across different epochs, and it was proposed by BERxiT (Xin et al., 2021):

$$\text{Odd} : \mathcal{L}_{stage1} = \mathcal{L}_M^{CE}(y_i, f_M(x_i; \theta_M)) \tag{4}$$

$$\text{Even} : \mathcal{L}_{joint} = \sum_{m=1}^{M} \mathcal{L}_m^{CE} \tag{5}$$

For the search and training of our LECO method, we adopt the joint training (JT) method, following Teerapittayanon et al. (2016); Kaya et al. (2019); Zhou et al. (2020); Zhu (2021). LECO mainly

employs JT to fine-tune the PTM backbone and simultaneously learn the best exit architectures for all intermediate layers under a differentiable NAS framework.

Early exiting inference At inference, the multi-exit PLM can operate in two different modes: (a) static early exiting, that is, a suitable exit m^* is appointed to predict all queries. (b) Dynamic early exiting, the model starts to predict on the classifiers $f^{(1)}$, $f^{(2)}$, ..., in turn in a forward pass, until it receives a signal to stop early at an exit $m^* < M$, or arrives at the last exit M.

3.1.1 Inference speedup ratio

During inference, we will run the test samples with batch size one following Zhou et al. (2020); Teerapittayanon et al. (2016). We report the actual wall-clock run-time reduction as the efficiency metric. For each test sample x_i, denote the inference time cost under early exiting as t_i, and time cost under no early exiting as T_i. Then the average speedup ratio on the test set is calculated by $\text{Speedup} = 1 - \frac{\sum_1^{N_{test}} t_i}{\sum_1^{N_{test}} T_i}$, where N_{test} is the number of samples on the test set. We will run the test set ten times and report the average speedup ratio to avoid randomness of run-time.

3.2 Preliminaries on DARTS

Assume there is a pre-defined space of operations denoted by $\mathcal{O}$, where each element, $o(\cdot)$, denotes a neural network operation, such as convolutional operation, self-attention, and activation. DARTS (Liu et al., 2019a) operates on a search cell, a fully connected directed acyclic graph (DAG) with N nodes. Let (i, j) denote a pair of nodes. The core idea of DARTS is to initialize a super-network stacked with blocks with the same architecture as the DAG. During the search, each edge in the DAG is a weighted sum including all $|\mathcal{O}|$ operations in $\mathcal{O}$, $f_{i,j}(z_i) = \sum_{o\in\mathcal{O}} a^o_{i,j} \cdot o(z_i)$, where $a^o_{i,j} = \frac{\exp \alpha^o_{i,j}}{\sum_{o'\in\mathcal{O}} \exp \alpha^{o'}_{i,j}}$, z_i denotes the output of the i-th node, and $\alpha^o_{i,j}$ is the architectural parameters that represent the weight (or the importance score) of $o(\cdot)$ in edge (i, j). The output of a node is the sum of all input flow, i.e., $z_j = \sum_{i<j} f_{i,j}(z_i)$. The output of the entire cell is formed by summing the last two nodes.

This design makes the entire framework differentiable to layer weights and architectural parameters $\alpha^o_{i,j}$ so that it can perform architecture searches in an end-to-end fashion. The standard optimization method is the bi-level optimization proposed in DARTS. After the search process is completed, the discretization procedure extracts the final sub-network by dropping the operations receiving lower scores.

4 Search space of LECO

As depicted in Figure 1, we construct the search space of a LECO intermediate exit mimicking the MHA exit. Representations of the current BERT layer, $H^{(m)}_i$, will first be down-sampled to a smaller dimension $\mathcal{R}^{d_e}$ (e.g., 64) to keep the intermediate exit parameter-efficient.[1] Then, it will go through an activation cell, an encoder cell, a pooling cell, and finally, another activation cell. The whole DAG of the intermediate exit consists of 7 edges.

Activation cell Both activations cells are one-step DAGs (Figure 1), designated to choose the proper activation function from several candidates. Similar to So et al. (2019), the collection of activation functions we consider is: (a) **ReLU** (Agarap, 2018); (b) **GeLU** (Hendrycks and Gimpel, 2016); (c) **SWISH** (Ramachandran et al., 2017); (d) **Tanh** (Krizhevsky et al., 2012); (e) **NullAct**, which means making no changes to the input.

Encoder cell As is shown in Figure 1, different from Wang et al. (2020); Zhu et al. (2021a), we construct our encoder cell as a simple DAG, which consists of at most two encoder operations. Encoder operations 1 and 2 will encode the cell's input, and their outputs will be summed to be the output of the encoder cell. As an extension to the encoder search space of Wang et al. (2020); Zhu et al. (2021a); Chen et al. (2020), our collection of encoder operations consists of the following commonly used encoding operations: (a) 1-d convolutional layers, with stride 1, same padding, output filters equal to the input's dimension, and kernel size equal to 1, 3, or 5 (denoted as **conv**_k, $k = 1, 3, 5$); (b) multi-head self-attention layer (Vaswani et al., 2017), with $k = 2, 4, 8$ attention heads, head size equaling d_e/k (denoted as **mha**_k, $k = 2, 4, 8$); (c) skip-connection, denoted as **skip-connect**; (d) the null encoding operation that multiply zero tensors to the input (**null**).[2]

[1] Note that the parameters of the intermediate exits constitute at most 1.6% of the BERT's parameters.

[2] Selecting this operation means fewer operations will be included in the encoder DAG.

Pooling cell It is also a one-step DAG for selecting the proper pooling layer. The most commonly used pooling operation for PTM-based models is to extract the representations of the $[CLS]$ token (denoted as **cls_pool**). As is summarized in Gong et al. (2018), other commonly used pooling operations are: max pooling (**max_pool**); average pooling (**avg_pool**); self-attention based pooling (**sa_pool**).

Note that our search space contains the MHA exit (introduced in Section 3.1) as a special case. The above search space can result in 6.87e+34 combinations of different multi-exit BERT. We will mainly follow DARTS (Liu et al., 2019a) to search for the optimal architecture designs of exits. But different from (Liu et al., 2019a), we adopt a macro search space, that is, the exits from different layers have different architectural parameters, thus resulting different architectures for different layers.

5 Comparison-based Early Exiting

The patience-based mechanism (Zhou et al., 2020) validates the early exiting decisions among the previous layers, providing a promising direction for designing early exiting mechanisms. the early exiting condition in PABEE is coarse: it directly compares the predicted labels. However, tt is common for BERT to change its predictions after a few intermediate layers. Thus, PABEE's early exiting performances with low patience parameters may not be reliable. To summarize, we need a more fine-grained criterion to generate more reliable early exiting signals.

We now introduce our Comparison-based early exiting method, COBEE. The inference procedure is illustrated in Figure 1. Assume the forward pass has reached layer $m < M$. We now compare the predicted distributions of layer m and layer $m^{'}$ ($m > m^{'}$) as follows. Denote the label that receives the highest probability mass at layer m as k_m^*, and the probability distribution of exit m is denoted as $\mathbf{Pr}_m$, then the disagreement between layer m and layer $m^{'}$ is calculated as:

$$\text{Di}(\mathbf{Pr}_m, \mathbf{Pr}_{m^{'}}) = |\mathbf{Pr}_m(k_m^*) - \mathbf{Pr}_{m^{'}}(k_m^*)|. \quad (6)$$

For simplicity, we denote $\text{di}_{m,m^{'}} = \text{Di}(\mathbf{Pr}_m, \mathbf{Pr}_{m^{'}}) \in \mathbf{R}$. The smaller the value of $\text{di}_{m,m^{'}}$, the predicted distributions $\mathbf{Pr}_m$ and $\mathbf{Pr}_{m^{'}}$ are more consistent with each other. We use a counter cnt to store the number of times the disagreement scores between adjacent layers are less than the pre-defined exiting threshold τ. At layer m, cnt_m is calculated as:

$$cnt_m = \begin{cases} cnt_{m-1} + 1, & \text{if } \text{di}_{m,m-1} < \tau, \\ 0, & \text{otherwise.} \end{cases} \quad (7)$$

If $\text{di}_{m,m-1}$ is less than the pre-defined threshold, then the patience counter is increased by 1. Otherwise, the patience counter is reset to 0. If cnt_m reaches the pre-defined patience value t, the model stops inference and exits early. Otherwise, the model goes to the next layer. However, if the model does not exit early at intermediate layers, the model uses the final classifier f_M for prediction.

6 Experiments

6.1 Datasets

We evaluate our proposed approach to the classification tasks on GLUE benchmark (Wang et al., 2018). We only exclude the STS-B task since it is a regression task, and we exclude the WNLI task following previous work (Devlin et al., 2019; Jiao et al., 2020; Xu et al., 2020b). Since the original test sets are not publicly available, we follow Zhang et al. (2020a) and Mahabadi et al. (2021) to construct the train/dev/test splits as follows: (a) for datasets with fewer than 10k samples (RTE, MRPC, CoLA), we divide the original validation set in half, using one half for validation and the other for testing. (b) for larger datasets, we split 1k samples from the training set as the development set, and use the original development set as the test set. The detailed dataset statistics are presented in Table 1.

For MNLI, we report acc, which is the average of the accuracy scores on the matched and mismatched test set. For MRPC and QQP, we report acc-f1, which is the average of accuracy and F1 scores. For CoLA, we report mcc, which is the Matthews correlation. For all other tasks, we report accuracy (acc).

6.2 Baseline methods

We compare our LECO framework with the following baselines:

Multi-exiting model training For multi-exit model training, we compare: (a) Joint training (JT) (Zhou et al., 2020; Teerapittayanon et al., 2016), with both a linear exit and an MHA exit ($d_e = 64$); (b) two-stage training (2ST) (Liu et al., 2020; Xin et al., 2020), with an MHA exit ($d_e = 64$); (c) alternating training (ALT) in Xin et al. (2021); (d) the

Category	Datasets	|train|	|dev|	|test|	$\|\mathcal{Y}\|$	Type	Labels
Single-sentence	SST-2	66349	1000	872	2	sentiment	positive, negative
	CoLA	8551	521	522	2	linguistic acceptability	acceptable, not acceptable
Sentence-pair	MNLI	391702	1000	19647	3	NLI	entailment, neutral, contradiction
	MRPC	3668	204	204	2	paraphrase	equivalent, not equivalent
	QNLI	103743	1000	5463	2	NLI	entailment, not entailment
	QQP	362846	1000	40430	2	paraphrase	equivalent, not equivalent
	RTE	2490	138	139	2	NLI	entailment, not entailment

Table 1: The statistics of datasets evaluated in this work. For MNLI task, the number of samples in the test set is summed by matched and mismatched samples. $|\mathcal{Y}|$ is the number of classes for a dataset.

Gradient Equilibrium technique (GradEquil) (Li et al., 2019), which incorporates JT with gradient adjustments and is adopted by Liu et al. (2021); (e) Global Past Future (Liao et al., 2021) (Global-PF) which asks the lower layers to imitate the deeper layers; (f) GAML-BERT (Zhu et al., 2021b), which employs a mutual learning strategy to improve the performances of shallow exits.

Early exiting methods We compare the early exiting performances of our COBEE method on the multi-exit backbone trained under the LECO framework with the following methods: (a) Entropy-based method (Entropy) originated from (Teerapittayanon et al., 2016), which is equivalent to the maximum-probability based method Schwartz et al. (2020); (b) Patience-based method (Patience) (Zhou et al., 2020); (c) learning-to-exit based method (LTE) proposed by Xin et al. (2021), which train an extra meta-classifier to estimate the confidence on a sample and achieves the SOTA performances of early exiting. For comparison, we also run the patience-based method on the backbone obtained by the JT method with linear exits.

6.3 Experimental settings

Devices We implement LECO on the base of HuggingFace's Transformers. We conduct our experiments on Nvidia V100 16GB GPUs.

PTM models. We mainly adopt the ALBERT base (Lan et al., 2019) backbone. We will also include RoBERTa-base (Liu et al., 2019b), and DeBERTa-base (He et al., 2020) in the ablation studies.

Settings for Architecture search We add a LECO search cell (Figure 1) with dimension d_e equal to 32 on each intermediate layer of the PTM and adopt the DARTS (Liu et al., 2019a) method to learn the best exit architecture for each layer. AdamW optimizer (Loshchilov and Hutter, 2019) is used for both the model and architecture parameters. At the beginning of each epoch, the training set is randomly split into D_1 (for updating model parameters) and D_2 (for updating architecture parameters) with a ratio of $1:1$. The search will last for 30 epochs. The learning rate is 2e-5 for model parameters and 2e-4 for architectural parameters. The search procedure is run once on each GLUE task.

Settings for Architecture evaluation After the search procedure ends, the top-scored sub-network is discretized from the super-network at each layer and will be trained from scratch as the final learned exit. The learning rate is 2e-5, and AdamW optimizer (Loshchilov and Hutter, 2019) is used for optimization. We evaluate the dev set and save the checkpoint after each epoch. After training ends, we evaluate the best checkpoint on the test set. We train the final learned exits under 5 random seeds to obtain its average test performance.

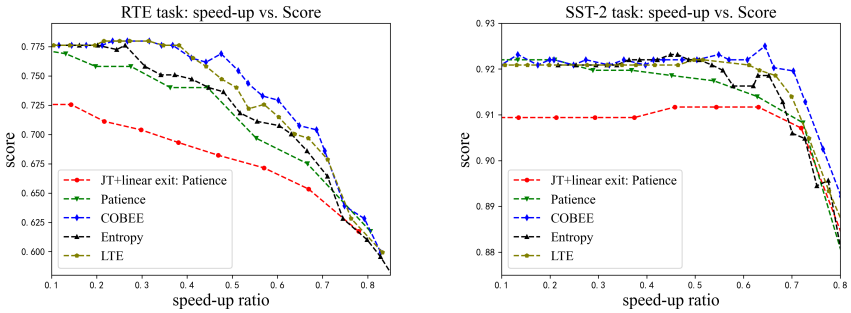
6.4 Main results

Comparison of multi-exit model training methods Table 2 reports the main results on the GLUE benchmark with ALBERT as the backbone model. All baseline models are run with the original authors' open-sourced codes. We report AVG, the cross-layer average score, and BEST, the best score among all the intermediate layers. From Table 2, Our LECO method outperforms the previous multi-exit model training methods in terms of the AVG scores (with statistical significance), demonstrating that our LECO framework effectively boosts the overall performances of intermediate exits and thus providing stronger backbones for early exiting.

Note that both 2ST + MHA exit (Liu et al., 2020) and JT + MHA exit introduce 66k parameters per exit, while the LECO method adds 25k-26k parameters per exit. The comparison among the three methods demonstrates that our LECO method does not rely only on adding more parameters to obtain performance improvements. The improvements of LECO result from better architectural designs for

	RTE		MRPC		CoLA		SST-2		QNLI		QQP		MNLI	
Baseline methods														
	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST
JT + linear exit	66.8	72.5	83.7	87.9	43.7	53.3	89.2	91.1	82.6	87.3	82.2	87.2	76.0	83.1
JT + MHA exit	68.1	76.9	84.1	88.2	43.6	57.5	88.2	91.5	82.8	87.6	82.4	87.1	76.8	83.2
GradEquil	67.3	77.4	84.2	89.3	43.6	56.1	89.2	91.8	82.4	88.0	82.7	87.0	76.5	83.6
ALT	68.5	77.8	84.6	88.3	44.1	57.3	88.9	91.6	82.3	87.8	82.5	86.8	76.6	83.2
GAML-BERT	68.8	77.6	84.9	88.8	45.0	57.9	89.1	92.3	82.6	87.9	82.6	87.5	75.9	83.4
Global-PF	68.5	78.1	84.9	88.6	45.1	57.7	88.9	92.6	82.5	88.1	82.6	87.4	76.5	83.3
2ST + MHA exit	68.9	77.5	85.1	89.2	45.0	57.9	89.3	92.4	82.5	88.0	82.7	87.3	76.2	82.7
Our proposed method														
LECO	**69.7***	77.9	**85.8***	89.4	**46.4***	58.0	**89.6***	92.5	**83.4***	88.1	**83.1***	87.4	**77.3***	83.4

Table 2: Average test performance of methods with ALBERT backbone on GLUE tasks across 5 random seeds. AVG represents cross-layer average score, and BEST represents best score among all layers. The $*$ symbol on the AVG scores means the results surpass the baseline method with statistical significance (by the Wilcoxon signed-rank test).



(a) RTE (b) SST-2

Figure 2: The speedup-score curves with different dynamic early exiting methods, on the RTE and SST-datasets.

exits of different depths.

Comparison of dynamic early exiting mechanisms We compare our COBEE method with the previous best-performing early exiting methods on the multi-exit ALBERT-base backbone trained under our LECO framework (as reported in Table 2). We also run the patience-based early exiting with the multi-exit ALBERT-base trained with the JT method. For the patience-based method (Zhou et al., 2020), early exiting is run on different patience parameters. For the other methods, we run early exiting under different confidence thresholds or patience parameters so that the speedup-performance curves consist of at least 20 points evenly distributed across the interval $(0, 1)$ of speedup ratios. The speedup-performance curves for the RTE and SST-2 tasks are plotted in Figure 2.

The following takeaways can also be made from Figure 2: (a) With the same backbone model, our COBEE method achieves better speedup-performance trade-offs than the previous SOTA early exiting methods, especially when the speedup ratio is large. (b) The comparison between Patience and JT+linear exit: Patience demonstrates that our LECO method can provide superior backbones for early exiting and consistently result in superior performances under different speedup ratios, even though introducing a more complex exit architecture. The learned exit architecture constitutes 0.25% of the parameters on each intermediate layer and increases 0.6% inference latency on average. However, the performance gains on the intermediate layers clearly out-weights the increased latency.

6.5 Discussions and ablation studies

Discussion on the learned architectures Table 6 of the Appendix A presents the best-learned exit architectures on each layer of ALBERT when the downstream task is MRPC or RTE. Three observations can be made: (a) although we allow at most two encoder operations in the encoder search cell, more than half of the learned exits include one valid encoding operation, making the exits more parameter efficient. (b) The learned archi-

Method	AVG score	
-	RTE	SST-2
LECO	69.7	89.6
2ST + MHA exit	68.9	89.3
2ST + LECO	69.6	89.5
ALT	68.5	88.9
ALT + LECO	69.3	89.4

Table 3: Comparisons of LECO with different multi-exit training methods. Cross-layer average performance (AVG) scores are reported.

tectures tend to use a pair of different activation functions, which is different from the combination of the Tanh-Tanh activation functions applied in the MHA exit (Liu et al., 2020). (c) Most exits do not select the **cls_pool** pooling operation, validating the necessity of our pooler search cell.

LECO works well with other multi-exit training strategies In the main experiments, we train LECO with the JT method. Table 3 demonstrates the results of LECO when trained with 2ST and ALT. The results show that LECO can effectively improve the performances of 2ST and ALT, and achieve comparable results with LECO combined with JT. However, the JT method is more convenient and takes less training time.

LECO works well with other pretrained backbones We now substitute the pretrained backbone to RoBERTa-base (Liu et al., 2019b) and DeBERTa-base (He et al., 2020), and the results are reported in Table 4. We can see that our LECO framework can also help to improve the average performance of multi-exit RoBERTa/DeBERTa model. An interesting take-away is that RoBERTa and DeBERTa can not outperform ALBERT in terms of AVG scores. We hypothesis that ALBERT shares parameters across transformer layers, thus the difference between shallow and deep layers are smaller than the other models.

Ablation on the search space We now conduct an ablation study to show the validity of our search space design. We consider reducing our search space $\mathcal{O}$ to a singleton step-by-step: (a) reduce the activation cells by only keeping the **Tanh** activation ($\mathcal{O}_1$); (b) further reduce the pooler cell to only include **cls_pool** ($\mathcal{O}_2$); (c) further reduce the encoder cell to only include **mha_dot**, and now the search space only contains the MHA exit. Table 5 reports the search results on different search spaces. From Table 5, we can see that dropping any components of the whole search space results in performance losses, demonstrating that our search space design is necessary and beneficial.

Method	AVG score	
-	RTE	SST-2
ALBERT backbone		
LECO	69.7	89.6
JT + MHA exit	68.1	88.2
RoBERTa backbone		
LECO	68.6	88.7
JT + MHA exit	66.5	87.4
DeBERTa backbone		
LECO	69.5	89.3
JT + MHA exit	66.9	88.1

Table 4: Comparisons of LECO with different pretrained backbones. Cross-layer average performance (AVG) scores are reported. We can see that RoBERTa and DeBERTa can not outperform ALBERT in AVG scores.

search space	AVG score	
-	RTE	SST-2
$\mathcal{O}$	69.7	89.6
$\mathcal{O}_1$	69.3	89.1
$\mathcal{O}_2$	68.9	88.7
MHA exit	68.1	88.2

Table 5: Experimental results for the ablation study of our LECO search space. Cross-layer average (AVG) performance scores are reported.

7 Conclusion

In this work, we propose a novel framework, LECO. Our contributions are three-fold. First, LECO designs a unified search space for architectural designs of intermediate exits. Second, we apply the differentiable NAS framework of DARTS to learn the optimal exit architectures automatically. Third, we propose a novel comparison based early exiting mechanism, COBEE. Experiments on the GLUE benchmark and ablation studies demonstrate that our LECO framework can achieve SOTA on multi-exit BERT training and outperforms the previously SOTA dynamic early exiting methods.

Limitation

Although our LECO framework is shown to be effective in improving the multi-exit BERT training, it still has certain limitations that need to be addressed in the future: (a) MHA exits and our learned exits indeed introduce new parameters and additional flops. We would like to explore more parameter-efficient methods to improve multi-exit

BERT training in future works. (b) In this work, we demonstrate our framework's performance on sentence classification or pair classification tasks. In future works, we would like to extend our work to broader tasks such as sequence labeling, relation extraction, and text generation. We would like to explore this aspect in the future.

Ethics Statement

Our LECO framework is designated to improve the training of multi-exit BERT and dynamic early exiting performances. Our work can facilitate the deployment and applications of pre-trained models on devices with less powerful computation capabilities, making the state-of-the-art models accessible for everyone. In addition, we hope this technology can help reduce the carbon footprints of NLP-based applications. Furthermore, the datasets we experiment with are widely used in previous work and, to our knowledge, does not introduce new ethical concerns.

References

Abien Fred Agarap. 2018. Deep learning using rectified linear units (relu). *ArXiv*, abs/1803.08375.

Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. 2020. Binarybert: Pushing the limit of bert quantization. *arXiv preprint arXiv:2012.15701*.

Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. 2020. Adabert: Task-adaptive bert compression with differentiable neural architecture search. In *IJCAI*.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. 2021. Progressive darts: Bridging the optimization gap for nas in the wild. *ArXiv*, abs/1912.10952.

Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. 2021. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12219–12228.

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.

Angela Fan, Edouard Grave, and Armand Joulin. 2019. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*.

Yuan Gao, Haoping Bai, Zequn Jie, Jiayi Ma, Kui Jia, and Wei Liu. 2020. Mtl-nas: Task-agnostic neural architecture search towards general-purpose multi-task learning. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11540–11549.

Jingjing Gong, Xipeng Qiu, Shaojing Wang, and Xuanjing Huang. 2018. Information aggregation via dynamic routing for sequence encoding. In *COLING*.

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*.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *ArXiv*, abs/2006.03654.

Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. Automl: A survey of the state-of-the-art. *Knowl. Based Syst.*, 212:106622.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv: Learning*.

Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269.

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. *ArXiv*, abs/1909.10351.

Y. Kaya, Sanghyun Hong, and T. Dumitras. 2019. Shallow-deep networks: Understanding and mitigating network overthinking. In *ICML*.

Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2021. I-bert: Integer-only bert quantization. In *International conference on machine learning*, pages 5506–5518. PMLR.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84 – 90.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. 2019. Improved techniques for training adaptive deep networks. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1891–1900.

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

Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. 2021. A survey of transformers. *ArXiv*, abs/2106.04554.

Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Loddon Yuille, Jonathan Huang, and Kevin P. Murphy. 2018. Progressive neural architecture search. In *ECCV*.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019a. Darts: Differentiable architecture search. *ArXiv*, abs/1806.09055.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Qi Ju. 2020. Fastbert: a self-distilling bert with adaptive inference time. *arXiv preprint arXiv:2004.02178*.

Xiangyang Liu, Tianxiang Sun, Junliang He, Lingling Wu, Xinyu Zhang, Hao Jiang, Zhao Cao, Xuanjing Huang, and Xipeng Qiu. 2021. Towards efficient nlp: A standard evaluation and a strong baseline. In *North American Chapter of the 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. 2019b. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *ICLR*.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. In *NeurIPS*.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2017. Swish: a self-gated activation function. *arXiv: Neural and Evolutionary Computing*.

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. *arXiv preprint arXiv:2004.07453*.

Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.

David R. So, Chen Liang, and Quoc V. Le. 2019. The evolved transformer. *ArXiv*, abs/1901.11117.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*.

Tianxiang Sun, Xiangyang Liu, Wei Zhu, Zhichao Geng, Lingling Wu, Yilong He, Yuan Ni, Guotong Xie, Xuanjing Huang, and Xipeng Qiu. 2022. A simple hash-based early exiting approach for language understanding and generation. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2409–2421, Dublin, Ireland. Association for Computational Linguistics.

Thierry Tambe, Coleman Hooper, Lillian Pentecost, En-Yu Yang, Marco Donato, Victor Sanh, Alexander M. Rush, David M. Brooks, and Gu-Yeon Wei. 2020. Edgebert: Optimizing on-chip inference for multi-task nlp. *ArXiv*, abs/2011.14203.

Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *ArXiv*, abs/1706.03762.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *BlackboxNLP@EMNLP*.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. 2022. Deepnet: Scaling transformers to 1, 000 layers. *ArXiv*, abs/2203.00555.

Yujing Wang, Yaming Yang, Yiren Chen, Jing Bai, Ce Zhang, Guinan Su, Xiaoyu Kou, Yunhai Tong, Mao Yang, and Lidong Zhou. 2020. Textnas: A neural architecture search space tailored for text representation. In *AAAI*.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2019. Snas: Stochastic neural architecture search. *ArXiv*, abs/1812.09926.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. Deebert: Dynamic early exiting for accelerating bert inference. *arXiv preprint arXiv:2004.12993*.

Ji Xin, Raphael Tang, Yaoliang Yu, and Jimmy Lin. 2021. Berxit: Early exiting for bert with better fine-tuning and extension to regression. In *Proceedings*

of the 16th conference of the European chapter of the association for computational linguistics: Main Volume, pages 91–104.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020a. Bert-of-theseus: Compressing bert by progressive module replacing. *arXiv preprint arXiv:2002.02925*.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020b. Bert-of-theseus: Compressing bert by progressive module replacing. In *EMNLP*.

Jingjing Xu, Wangchunshu Zhou, Zhiyi Fu, Hao Zhou, and Lei Li. 2021a. A survey on green deep learning. *ArXiv*, abs/2111.05193.

Yuhui Xu, Lingxi Xie, Wenrui Dai, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Hongkai Xiong, and Qi Tian. 2021b. Partially-connected neural architecture search for reduced computational redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43:2953–2970.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q. Weinberger, and Yoav Artzi. 2020a. Revisiting few-sample bert fine-tuning. *ArXiv*, abs/2006.05987.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020b. Ternarybert: Distillation-aware ultra-low bit bert. *arXiv preprint arXiv:2009.12812*.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33:18330–18341.

Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*.

Wei Zhu. 2021. Leebert: Learned early exit for bert with cross-level optimization. In *ACL*.

Wei Zhu, Yuan Ni, Xiaoling Wang, and Guo Tong Xie. 2021a. Discovering better model architectures for medical query understanding. In *NAACL*.

Wei Zhu, Xiaoling Wang, Yuan Ni, and Guo Tong Xie. 2021b. Gaml-bert: Improving bert early exiting by gradient aligned mutual learning. In *EMNLP*.

Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. *ArXiv*, abs/1611.01578.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning transferable architectures for scalable image recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8697–8710.

A Demonstrations of learned architectures

In the section, the learned exit architectures on the RTE and SST-2 tasks are presented in Table 6. Discussions on the observations from the learned architectures can be found in the main content.

task	layer index	activation 1	activation 2	pooler	encoder op 1	encoder op 2
SST-2	1	swish	leaky_relu	avg_pool	conv_3	null
	2	gelu	leaky_relu	max_pool	null	mha_4
	3	nullAct	swish	max_pool	mha_4	null
	4	swish	leaky_relu	cls_pool	conv_3	null
	5	swish	gelu	sa_pool	conv_5	skip-connect
	6	swish	swish	avg_pool	null	conv_5
	7	gelu	swish	max_pool	mha_4	conv_1
	8	nullAct	leaky_relu	max_pool	null	skip-connect
	9	tanh	gelu	cls_pool	conv_1	conv_1
	10	nullAct	gelu	cls_pool	skip-connect	mha_8
	11	nullAct	gelu	avg_pool	conv_3	null
	12	gelu	nullAct	cls_pool	conv_3	skip-connect
RTE	1	nullAct	tanh	sa_pool	null	conv_1
	2	swish	nullAct	avg_pool	conv_1	conv_5
	3	gelu	tanh	sa_pool	null	mha_2
	4	swish	nullAct	sa_pool	skip-connect	conv_3
	5	gelu	nullAct	sa_pool	conv_3	null
	6	gelu	tanh	sa_pool	mha_pdot	conv_3
	7	nullAct	tanh	sa_pool	conv_3	null
	8	leaky_relu	leaky_relu	max_pool	conv_1	null
	9	nullAct	swish	max_pool	null	conv_1
	10	swish	leaky_relu	max_pool	conv_1	null
	11	nullAct	gelu	cls_pool	skip-connect	mha_4
	12	nullAct	swish	cls_pool	mha_4	null

Table 6: The best architectures learned via our LECO framework. We can see that on the same task, BERT requires different intermediate exits to better exploit the representation capabilities on different layers.