Front. Comput. Sci., 2024, 18(3): 183308
https://doi.org/10.1007/s11704-022-2341-9

RESEARCH ARTICLE

Accelerating BERT inference with GPU-efficient exit prediction

Lei LI[1], Chengyu WANG[2], Minghui QIU[2], Cen CHEN (✉)[1], Ming GAO[1,3], Aoying ZHOU[1]

1 Shanghai Engineering Research Center of Big Data Management, School of Data Science and Engineering, East China Normal University, Shanghai 200062, China
2 Alibaba Group, Hangzhou 311121, China
3 KLATASDS-MOE, School of Statistics, East China Normal University, Shanghai 200062, China

Abstract BERT is a representative pre-trained language model that has drawn extensive attention for significant improvements in downstream Natural Language Processing (NLP) tasks. The complex architecture and massive parameters bring BERT competitive performance but also result in slow speed at model inference time. To speed up BERT inference, FastBERT realizes adaptive inference with an acceptable drop in accuracy based on knowledge distillation and the early-exit technique. However, many factors may limit the performance of FastBERT, such as the teacher classifier that is not knowledgeable enough, the batch size shrinkage and the redundant computation of student classifiers. To overcome these limitations, we propose a new BERT inference method with GPU-Efficient Exit Prediction (GEEP). GEEP leverages the *shared exit loss* to simplify the training process of FastBERT from two steps into only one step and makes the teacher classifier more knowledgeable by feeding diverse Transformer outputs to the teacher classifier. In addition, the *exit layer prediction* technique is proposed to utilize a GPU hash table to handle the token-level exit layer distribution and to sort test samples by predicted exit layers. In this way, GEEP can avoid batch size shrinkage and redundant computation of student classifiers. Experimental results on twelve public English and Chinese NLP datasets prove the effectiveness of the proposed approach. The source codes of GEEP will be released to the public upon paper acceptance.

Keywords BERT, FastBERT, inference acceleration, model distillation, early exit, text classification

1 Introduction

In recent years, pre-trained language models [1−3] have drawn extensive attention for their significant improvements in downstream Natural Language Processing (NLP) tasks. Among these models, BERT [1] is the most representative and inspires a lot of follow-up works in the research community. The complex architecture design and massive parameters bring superior performance to pre-trained language models but often result in slow inference speed, which limits their usage in real-world applications. To improve the usability of these models, many approaches are proposed to accelerate BERT-ish models. For example, knowledge distillation [4] seeks to distill knowledge from a large pre-trained teacher model to a smaller yet effective student model. Early-exit networks [5] comprise a backbone architecture and additional exit heads (or classifiers) along with the backbone model layers. In the inference stage, the network adaptively chooses an early-exit head to yield outputs and circumvents the rest of the model to achieve a shorter inference time.

Recently, FastBERT [6] is proposed to integrate advantages of both knowledge distillation and early-exit networks. As shown in Fig. 1(a), FastBERT consists of a backbone and several branches.[1)] The backbone is built upon 12 layers of Transformer encoders with an additional teacher classifier, while the branches include 11 student-classifiers to enable early outputs. Specifically, the model is trained in preparation for downstream inference with three steps: (1) pre-training the backbone or using a pre-trained backbone; (2) backbone fine-tuning; and (3) freezing the backbone and performing a self-distillation stage to align each student classifier with the teacher classifier.

During inference, a batch of data flows through the Transformer layers from the bottom to the top and is fed into the corresponding student classifiers. The uncertainty of a student classifier's output p_s is computed by means of the normalized entropy, shown as follows:

$$Uncertainty(p_s) = \frac{\sum_{i=1}^{N} p_s(i) \log p_s(i)}{\log \frac{1}{N}}, \quad (1)$$

where p_s is the distribution of output probability, and N is the number of labeled classes. A sample with a lower uncertainty score than the pre-determined threshold will yield an output

Received June 7, 2022; accepted November 30, 2022

E-mail: cenchen@dase.ecnu.edu.cn

1) The statistical computation in this work is primarily based on the base version of the BERT model unless otherwise specified.

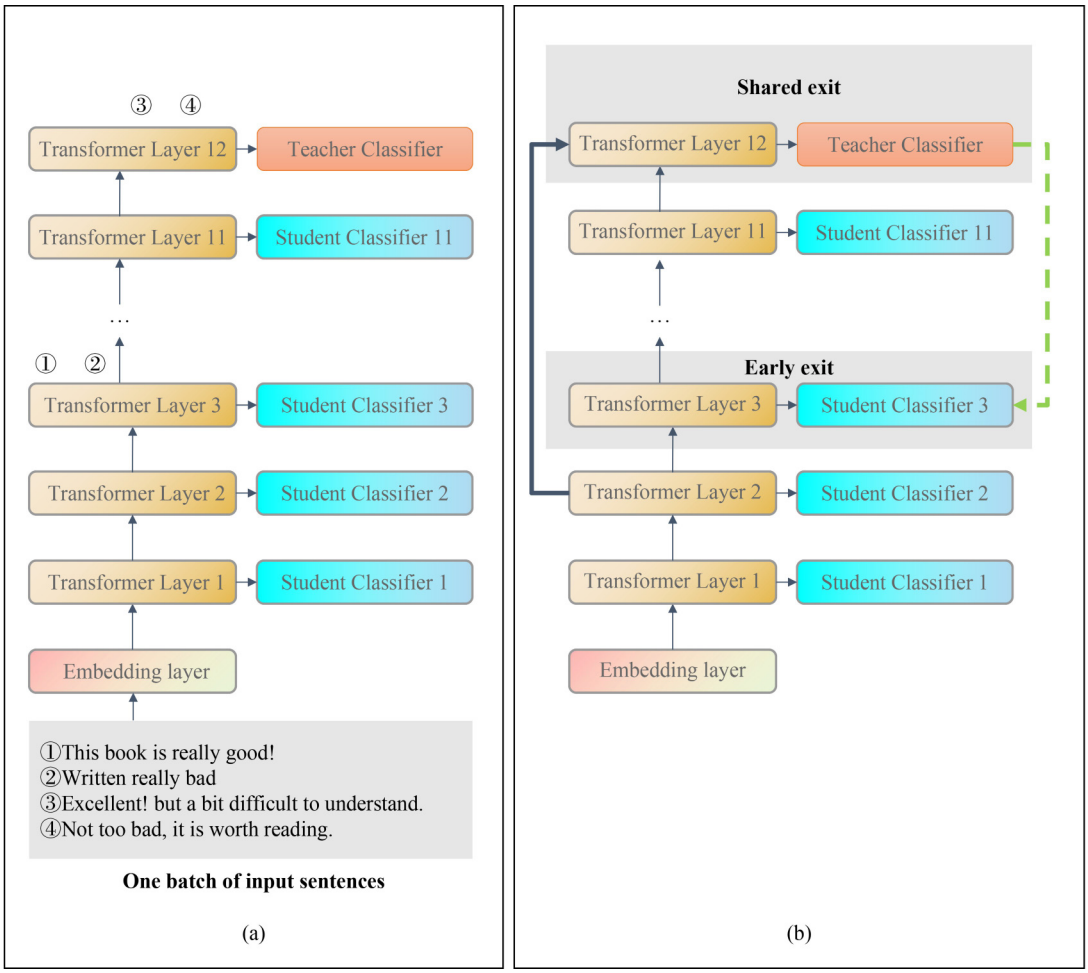

Fig. 1 (a) FastBERT with 12 transformer layers and 11 early exits. A batch of four samples is fed into the network. Samples 1 and 2 (easy samples) exit at Layer 3, While Samples 3 and 4 (hard samples) exit at Layer 12. (b) The shared exit layer in our GEEP method

without engaging subsequent computation. As shown in Fig. 1(a), we suggest that FastBERT may have three deficiencies summarized below:

1. FastBERT seeks to align student classifiers with the teacher classifier to enable early exits. The student classifier only has access to bottom-layer features, while the teacher classifier deals only with the last-layer features. The difference in feature representations from different layers makes it challenging to align student classifiers with the teacher classifier.
2. We observe that there exists a *batch size shrinkage* problem in early-exit networks that may hurt the model efficiency. For example, in Fig. 1(a), Samples 1 and 2 are easy samples that can be predicted correctly by Student Classifier 3. Samples 3 and 4 are hard samples where only the teacher classifier can generate outputs with high confidence. Roughly speaking, the batch size maintains to be 4 in about $3/12 = 25\%$ of the time. For the rest of the time, the actual batch size is 2. This makes it less efficient for batched computation.
3. FastBERT suffers from additional computation burden for hard examples such as Samples 3 and 4. In general, the computation of a Transformer block requires 1809.9M FLOPS (Floating-point operations), while a student classifier needs $46.1M$ FLOPS. For Samples 3 and 4, FastBERT incurs a redundant computation of $46.1M * 11 = 507.1M$ FLOPS (around 28% additional computation) from the 11 student classifiers.

To overcome these problems, we propose a new BERT inference method with GPU-Efficient Exit Prediction (GEEP). First, we propose a *shared exit loss* in GEEP to enable the teacher classifier to have access to diverse Transformer layers' outputs instead of the last Transformer layer. This can help better align student classifiers with the teacher classifier. Furthermore, by using a GPU hash table, GEEP efficiently predicts the exit layer of each test sample based on the exit layer information of train samples. During model inference, GEEP groups test instances with the same exit layer together to avoid *batch size shrinkage*. In addition, we directly skip redundant student classifications for hard samples to remove redundant computation of student classifiers. The source code of GEEP is available in the EasyNLP framework [7,8].

In summary, the main contributions are as follows:

1. We formally propose a novel framework named GEEP to accelerate BERT inference.
2. GEEP leverages the shared exit loss to make the teacher classifier more suitable for student classifiers to learn

from. In addition, GEEP utilizes a GPU hash table to handle early-exit layer prediction to avoid *batch size shrinkage* and redundant computation of student classifiers.

3. Extensive experiments on twelve datasets show the proposed GEEP approach outperforms competing baselines at a large margin. It helps to achieve ~2x speedup at inference without sacrificing much performance.

The rest of this paper is summarized as follows. Section 2 briefly overviews the related work. Our GEEP framework is elaborated in Section 3. The experiments are presented in Section 4. Finally, we draw the conclusion and discuss the future work in Section 5.

2 Related work

In this section, we summarize the related work of GEEP on various aspects, including knowledge distillation, early-exit networks, and hashing techniques for GPU.

2.1 Knowledge distillation

Deep neural networks have been successful in both industry and academia due to their scalability to encode massive data and maneuver billions of model parameters. However, it is challenging to deploy these cumbersome deep models on devices with limited resources.

Recently, knowledge distillation has received increasing attention from researchers [4] since it realizes model compression and acceleration by learning a small student model from a large teacher model. Transferring the information from a large model into a small model without a significant drop in accuracy is first proposed as a model compression approach by [9]. This approach is later formally popularized as knowledge distillation by [10].

In vanilla knowledge distillation, the knowledge is transferred from a teacher model into other student models. For example, [11] transfers knowledge from a 12-layer BERT into smaller and faster student models with fewer layers. In comparison, self-distillation [6,12] uses a single model in which knowledge from the deeper sections (top layers) of the network is distilled into its shallow sections (bottom layers).

Self-distillation enables us to set multiple exits on different sections or layers of a cumbersome model (i.e., BERT). For easy samples, we can early exit from models to avoid huge computational complexity and massive storage requirements.

2.2 Early-exit networks

Neural networks are becoming more over-parameterized due to recent advances in model design [5]. Based on the fact that not all inputs need complete computation to yield confident outputs, the adaptive inference is receiving attention as a prominent approach for accelerating deep neural networks.

Particularly, early-exit networks do not introduce extra models and provide dynamic inference time in a wide range by adjusting the specific threshold, carrying complementary performance gains to other efficiency optimizations [13].

In general, early-exit networks comprise a backbone architecture and additional exit heads (or classifiers) along its depth. In inference mode, a sample flows through the backbone and each exit sequentially. If an exit criterion is satisfied, the model yields output and circumvents the rest of the model. Typical methods employ the softmax of an exit to quantify the confidence of the network for a given prediction [6,13]. In the literature, [14] proposes an approach keeping per class statistics at each layer. [15] calculate classifiers' trust scores based on sample distances to a calibration set. [16] employ a policy that aggregates multiple exits on the same output. It is worth noting that too many redundant early exits can yield an extreme computation load and achieve a worse efficiency than the backbone model. In addition, using too many early classifiers can counteract convergence during end-to-end training. HASHEE [17] replaces the learn-to-exit modules with hash functions to assign each token to a fixed exiting layer. However, HASHEE does not introduce extra trainable parameters so that it can not enhance the performance of the teacher classifier. HASHEE also does not verify that token-level exit information is transferable from the training dataset to the test dataset.

2.3 Hashing techniques for GPU

Hash tables are effective data structures for manipulating sparse data, with widespread usage in various domains. Emerging many-core architectures, particularly Graphical Processing Units (GPUs) are specifically designed for data-parallel computation, in which the same operation is performed by multiple threads in parallel. To accommodate the oncoming shift towards large-scale sparse data processing, GPU-based hashing acknowledges these needs by utilizing lock-free shared-memory [18].

Open-addressing is an effective hash method in which a key is inserted into the hash table by searching through alternate table locations until a location is found to place the element [19]. [20] develops an open-addressing approach based on multi-level bounded linear probing, where the hash table has multiple levels. When an attempt of probing fails, it moves to the next level and uses a different hash function. In cuckoo-based hashing [21−23], each key is assigned two locations in the hash table, as specified by primary and secondary hash functions. When inserting a new key, its first location is probed with the primary function. If the slot is empty, then the key is inserted, and the probe sequence ends. Otherwise, a collided key already occupies the slot, and the cuckoo eviction procedure begins.

3 Methodology

In this section, we elaborate the GEEP technique for BERT acceleration in detail. Briefly speaking, our GEEP method mainly consists of two components, namely *shared exit loss* and *exit layer prediction*. In our work, we argue that the performance of student classifiers can be further optimized so that we can achieve better accuracy with the same consumption time. As there is a trade-off between speed and accuracy, we seek to predict faster while maintaining the same or similar accuracy.

3.1 Preliminaries

As aforementioned, FastBERT [6] is trained in preparation for

downstream inference with three steps: backbone pre-training, backbone fine-tuning, and self-distillation. The backbone model includes three parts: the embedding layer, the encoder stacking Transformer layers [24], and the teacher classifier.

For an input sentence $s = [w_0, w_1, ..., w_n]$ with length n, it is first transformed by the embedding layer of the backbone to a sequence of vector representations:

$$e = Embedding(s). \tag{2}$$

Next, the Transformer layers in the encoder perform a layer-by-layer feature aggregation:

$$h_i = Transformer\_i(h_{i-1}), \tag{3}$$

where h_i $(i = 1, ..., L)$ represents the output features at the ith Transformer layer, and $h_0 = e$. L is the number of Transformer layers (we have $L = 12$ for BERT-base).

The output of the final Transformer layer is fed into the teacher classifier, which includes three layers:

1. a fully-connected layer narrowing the dimension from 768 to 128.
2. a self-attention operation joining a fully-connected layer without changes in vector size.
3. a fully-connected layer with a $softmax$ function projecting vectors to an N-class indicator p_t as in Eq. (4), where N is the task-specific number of classes:

$$p_t = Teacher\_Classifier(h_L), p_t \in \mathbb{R}^{N\times 1}. \tag{4}$$

Finally, we use the cross-entropy loss between p_t and the ground truth p_g to optimize the backbone. In the *self distillation* step, the teacher classifier produces a high-quality soft-label p_t for each sample. The ith student classifier produces a prediction p_{s_i}. The objective is to minimize the KL-Divergence $D_{KL}(p_{s_i}\|p_t)$. As there are $L-1$ student classifiers in the FastBERT, the total loss is thus defined as:

$$Loss(p_{s_1}, ..., p_{s_{L-1}}, p_t) = \sum_{i=1}^{L-1} D_{KL}(p_{s_i}\|p_t). \tag{5}$$

3.2 The design of shared exit layers

The fundamental problem is that there exists a gap between the teacher classifier and the student classifiers. For example, in Fig. 1(b), for the early-exit network in Transformer Layer 3, Student Classifier 3 seeks to learn knowledge from the teacher classifier (as shown in the dashed line on the right side of the figure). Clearly, Transformer Layer 12 and the teacher classifier only deal with the last-layer features from Transformer Layer 11, while Transformer 3 and Student Classifier 3 only have access to the information from Transformer Layer 2. Since there exists a gap between features from Transformer Layers 2 and 11 [25], it is difficult for the student classifier to produce probability distributions that are similar to the teacher classifier.

To avoid this problem, we employ Transformer Layer 12 and the teacher classifier as the *shared exit*. The output of Transformer Layer 2 is sent to Transformer Layer 12 (the bold line on the left side of the figure) to force the teacher classifier to make predictions based on the output of Transformer Layer 2. Then the teacher classifier is able to teach Student Classifier 3 to make predictions based on the output of Transformer Layer 2. Such mechanism can be naturally extended to other Transformer layers so that we obtain a new loss called the *shared exit loss*:

$$\sum_{i=1}^{L-1} CEL(p_{t_i}, p_g), \tag{6}$$

where CEL refers to the cross-entropy loss and p_g is the ground truth distribution. We construct multiple sub-networks by concatenating Transformer Layers 1 to i, Transformer Layer L, and the teacher classifier sequentially. Then we denote p_{t_i} as the output of the teacher classifier in sub-networks. Equation (6) indicates that the teacher classifier should consider diverse Transformer outputs to make it easier for the students to learn from.

To simplify the training process, we combine the *backbone fine-tuning* step and the *self-distillation* step together into one step. We combine Eq. (5) and Eq. (6) to produce a combined loss as follows:

$$\sum_{i=1}^{L-1} CEL(p_{t_i}, p_g) + \sum_{i=1}^{L-1} D_{KL}(p_{s_i}\|p_t). \tag{7}$$

Note that, during the optimization of the KL-Divergence, the parameters of the teacher classifier should be kept fixed for the students to learn from. To achieve this, for the KL-divergence loss, we place a stop-gradient (`stop_grad`) operation on the teacher part, as shown in Eq. (8):

$$\sum_{i=1}^{L-1} CEL(p_{t_i}, p_g) + \sum_{i=1}^{L-1} D_{KL}(p_{s_i}\|\texttt{stop\_grad}(p_t)). \tag{8}$$

We suggest despite the fact that Eq. (8) increases the complexity of the training process, which is infrequent and can be done offline.

It should be noted that in previous research, residual operators in the BERT architecture enable the final Transformer layer to virtually access the features from lower layers [26]. However, in this approach, the final Transformer layer receives modified low-level features after self-attention operations. In contrast, Eq. (8) lets the final Transformer layer receive original features from low-level layers, which are exactly encountered by early exits (including a low-level Transformer layer and a student classifier). In addition, cross-layer feeding features produce gradients through the first term in Eq. (8), which is not affected by the stop-gradient operation in the second term.

3.3 Exit layer prediction

In FastBERT, a token (e.g., **good**) occurs in multiple input sequences, which may exit from one of 12 classifiers (one teacher and 11 students). Modern machine learning approaches rely on a hypothesis that the training data and the testing data are independently and identically distributed [27]. We extend this hypothesis as follows.

Hypothesis 1 The token-level exit layer distributions in the training data and the testing data are similar.

Based on Hypothesis 1, we can roughly predict the exit

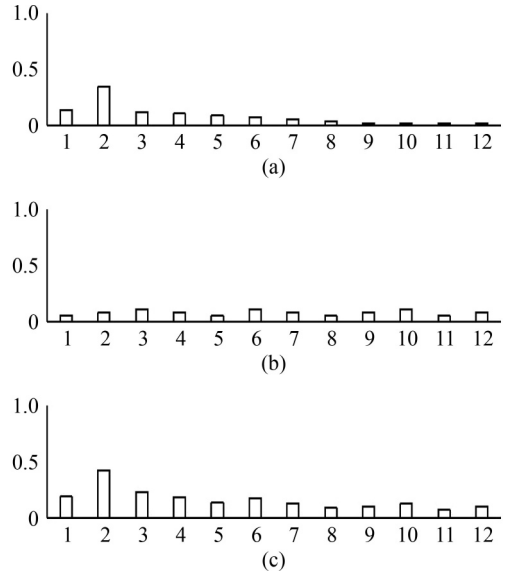

Fig. 2 Exit Layer Distributions for Tokens. (a) Exit layer distribution of token **good**; (b) exit layer distribution of token **very**; (c) the *exit score* of sequence **very good**

layer of samples in the testing set. Given an input sequence "**very good**", we calculate the *exit scores* by summing up exit layer distributions (obtained from the training dataset) of tokens layer-by-layer. As in Fig. 2, the input sequence **very good** is probable to exit from Layer 2 which has the highest *exit score*.

After predicting the exit layers of all samples in the testing set, we sort samples by the exit layer. We feed the sorted testing set to the model batch by batch. Obviously, samples in one batch have similar exit layers. Assuming that the first sample in one batch has an exit layer $j \in [1, L]$, we force the model to skip student classifiers below classifier j, and let all samples in the batch exit from student classifier j. This can help to skip some redundant exit computations. The pseudo-code for the inference process is summarized in Algorithm 1, where $Classifier\_i$ is the student classifier with $i \in [1, L-1]$, and $Classifier\_i$ is the teacher classifier when $i = L$. We denote the time consumption of Steps 2,3 and 4-13 as $T_{predict}$, T_{sort} and $T_{inference\_sorted}$ respectively. We do not consider the time consumption of Step 1 since it can be done offline. We use $T_{inference}$ to express the time consumption of regular inference on the original (unsorted) testing set.

Algorithm 1 Pseudo-code for GEEP inference over the testing dataset

```
1:  Use db to record exit distribution of tokens in the training
    dataset D_train.
2:  Based on db, predict the exit layer of each sample in the
    testing dataset D_test.
3:  Sort samples in the testing dataset D_test by the exit layer,
    then obtain D_test_sorted.
4:  for batch_data in D_test_sorted:
5:      hidden ← Embedding(batch_data)
6:      exit_layer ← batch_data[0][exit_layer]
7:      for i ← 1 to L:
8:          hidden ← Transformer_i(hidden, mask)
9:          if i < exit_layer then
10:             continue
11:         logits ← Classifier_i(hidden, mask)
12:         if i = exit_layer then
13:             break
```

From the algorithm, we can see that Steps 4 to 13 are expected to avoid *batch size shrinkage* and computation of redundant student classifiers. Therefore, the time $T_{inference\_sorted}$ is reduced. However, we also introduce extra time consumption $T_{predict}$ and T_{sort}. Overall, if GEEP successfully accelerates model inference, we have:

$$T_{predict} + T_{sort} + T_{inference\_sorted} < T_{inference}. \quad (9)$$

To make sure the speedup criterion is satisfied, we reduce the time consumption of $T_{predict}$ by building a high-performance databank *db*. In GEEP, we implement *db* in the form of a GPU hash table using CUDA. We do not employ neural network approaches since they are time-consuming and make the total inference time longer than FastBERT. As in Fig. 3, using a fine-tuned early-exit model to predict a sample in the training data, the sample may exit from Student Classifier 4. For each token in the sample, we construct a 32-bit key by putting the layer at 31 to 27 bits (the capacity is $2^5 = 32$) and putting the input id at 26 to 0 bits (the input id is provided by the tokenizer). A location is obtained by hashing the 32-bit key modulo the size of the hash table.

The hash table contains a key array and a value array. We check the location in the key array. If it is empty, then the 32-bit key is stored in the key array and the same location in the value array increase by 1, i.e., working as a counter. If the location is not empty, we search the next consecutive locations in limited steps until an empty location is found. For a sample in the testing set, each token is combined with different layers to serve as keys to retrieve exit layer frequencies that can be normalized to a token-level exit layer distribution. We calculate the *exit scores* of a sample by summing up token-level exit layer distributions.

A sample is expected to exit from the layer such that the layer has the highest *exit score*. We treat the predicted exit layer as an adjustable parameter. We will not let the GEEP model exit from the layer predicted by "exit scores". In contrast, we increase or decrease the predicted layer to adjust the accuracy and inference time of the GEEP model.

In addition, we provide an approximate analysis of the memory consumption of the GPU hash table. In our case, the hash table occupies:

$$32bit \times 2 \times 12 \times 50000 / 0.75 \approx 6.1\text{MB}, \quad (10)$$

where 2 represents the fact that the key and value both use 32 bits to store, 12 is the layer number of the BERT-base model, 50000 is the vocabulary size of the BERT-base model, and 0.75 is the load factor for the hash table. Based on our well-designed bit operations, the GPU table is small enough to co-exist with the pre-trained model without modifying the architecture or reducing the batch size.

4 Experimental results

In this section, we evaluate the performance of GEEP on various datasets. We also compare it against strong baselines to prove its superiority.

4.1 Datasets and experimental settings

To study the effectiveness of GEEP and make a fair

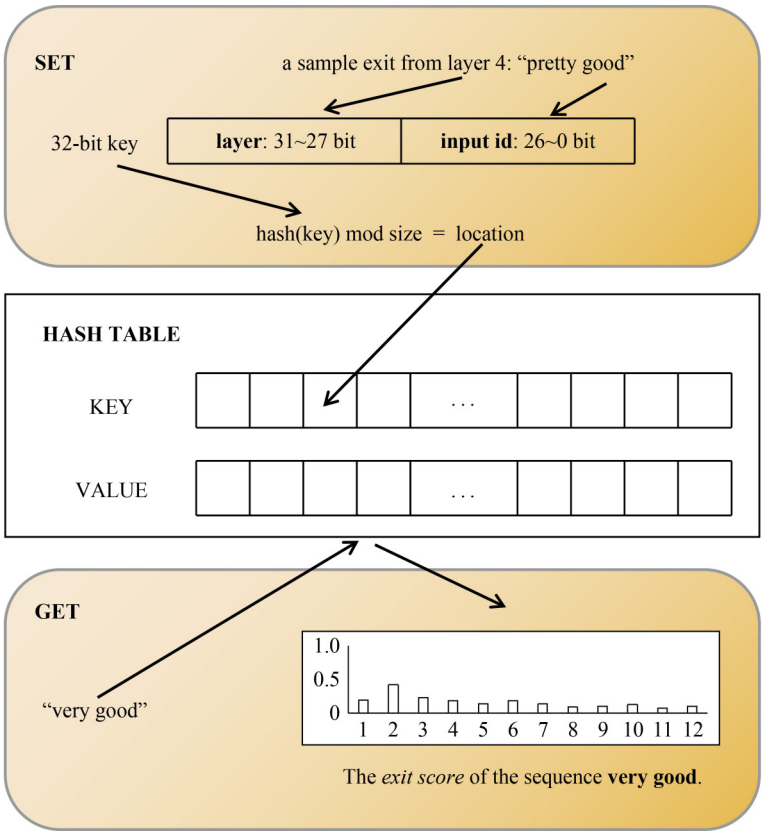

Fig. 3 The GPU Hash Table, with an Example of Its Processing Steps in GEEP.

comparison against baselines, we report results on twelve datasets used by FastBERT [6]. The six Chinese datasets include five sentence classification tasks (ChnSentiCorp, Book review, Shopping review, Weibo, and THUCNews) and a sentence matching task LCQMC [28]. The six English datasets (Ag.News, Amz.F, DBpedia, Yahoo, Yelp.F, and Yelp.P) are sentence classification tasks and were released in [29]. The statistics of the data splits are shown in Table 1. Considering the absence of development sets in English datasets, all development sets in both English and Chinese are not used, and the experimental results are obtained on the testing datasets. In the experiments, all results are averaged over five trials.

We compare our GEEP method against these baselines:

- **BERT:** We use the BERT-base model [1] that includes 12 layers, 768 hidden dimensions, 12 attention heads and 110M parameters. For English datasets, we load parameters from **BERT-Base, Uncased**. For Chinese datasets, we load parameters from **BERT-Base, Chinese**.
- **FastBERT:** FastBERT [6] integrates advantages of knowledge distillation and early-exit networks. We employ the implementation of FastBERT provided by the authors. As FastBERT shows better performance than DistilBERT [11], we do not regard DistilBERT as a strong baseline here.

Table 1 Data splits of all the datasets

Name	#Train	#Dev.	#Test
ChnSentiCorp	9,600	1,200	1,200
Book review	20,000	10,000	10,000
Shopping review	20,000	10,000	10,000
Weibo	99,988	10,000	10,000
THUCNews	50,000	5,000	10,000
LCQMC	238,766	8,802	12,500
Ag.News	120,000	0	7,600
Amz.F	3,000,000	0	650,000
DBpedia	560,000	0	70,000
Yahoo	1,400,000	0	600,00
Yelp.F	650,000	0	50,000
Yelp.P	560,000	0	38,000

In computing-related research, floating-point operations per second (FLOPS) is a measure of computer performance, useful in fields of scientific computations that require floating-point calculations. Many studies [30,31] utilize FLOPS to measure the computational complexity of blocks, layers and models of neural networks. Although we can roughly observe that larger FLOPS result in longer inference time. We argue that the FLOPS value is not the perfect solution for measuring model efficiency in some circumstances. Some studies transfer partial computing load from neural networks to help functions such as calculating early-exit criteria and filtering out easy samples [6]. The time consumption of these help functions will be neglected in calculating FLOPS. Thus, we report the actual time consumption to study the efficiency of models in our experimental results. We train all models using the Adam optimizer. All experiments are conducted on a server with 8 core, 32G memory and an NVIDIA V100 GPU (16G). GEEP does not depend on a specific type of GPU. However, some

hardware properties may slightly improve the GEEP performance: (1) Larger GPU memory will improve the batch size and speed up the training process of the Shared Exit Loss; (2) More CUDA cores manipulating parallelly GPU hashtables will result in a shorter time for computing exit scores. (3) We are glad to offer a list of the best GEEP settings for various hardware (e.g., CPU, GPU, TPU) in future work.

4.2 Detailed performance comparison

To examine the effectiveness of GEEP when a wide range of speedup values are reached, we present the comparison between GEEP and baselines in terms of accuracy, inference time, and speedup, as shown in Table 2. Results in bold fonts mean that our GEEP can have both faster inference speed and higher accuracy at various speedup times for the dataset. Among the twelve datasets, our GEEP method achieves better results than FastBERT in nine datasets (75% of all datasets), and comparable performance for the rest of the datasets. With a minor performance drop, i.e., within 1% drop of accuracy, the proposed GEEP manages to achieve 1.49x to 2.36x speedup on average compared to the original BERT model, while FastBERT has around 1.39x speedup on average. If we are allowed to have a performance drop within 3% in accuracy, the GEEP method can achieve 2.35x to 6.52x speedup. In all, the results show that the proposed GEEP method is able to achieve a faster inference speed than FastBERT and BERT while maintaining similar model performance in terms of accuracy.

We also present the detailed accuracy-time curves in Fig. 4 where the horizontal axis represents inference time (seconds), and the vertical represents accuracy (%). Obviously, a model with better performance tends to have a curve closer to the upper part of the figure. The results in the first and second rows in Fig. 4 also support the claim that our GEEP method is effective in a wide range of speedups, maintaining better efficiency and effectiveness compared to FastBERT. Specifically, with a minor performance drop (less than 1% of accuracy), GEEP can achieve around 2x speedup. For the results in the third row, the performance of FastBERT and GEEP is pretty close. In general, the proposed GEEP achieves relatively better performance than FastBERT as the curves tend to be closer to the upper part of the figure. Particularly, the GEEP method has a much better performance when the speedup rate is high (as shown in the left part of the figures). This shows it is helpful for scenarios that require high inference speedup.

4.3 Verification of Hypothesis 1

Note that our method is based on Hypothesis 1, where we assume that the token-level exit layer distributions are similar in training and testing sets. To verify this hypothesis, we first compute the token-level exit layer distributions d_1,d_2 for each token from training data and testing data, respectively. Next, we calculate the Jensen-Shannon Divergence (JSD) for each token as follows:

$$JSD(d_1||d_2) = \frac{1}{2}KLD(d_1||M) + \frac{1}{2}KLD(d_2||M), \qquad (11)$$

Table 2 Comparison of accuracy (A), time (T), and speedup (S) between GEEP and the baselines over all the 12 datasets

	ChnSentiCorp		Book review		Shopping review		Weibo		THUCNews		LCQMC	
	A/%	T/s Speedup	A/%	T/s Speedup	A/%	T/s Speedup	A/%	T/s Speedup	A/%	T/s Speedup	A/%	T/s Speedup
BERT	94.50	3.66 1x	87.21	26.62 1x	96.79	26.52 1x	97.75	26.58 1x	96.69	26.56 1x	86.60	33.10 1x
FastBERT	92.00	1.50 2.44x	86.50	20.18 1.32x	96.25	11.90 2.23x	97.79	15.51 1.71x	96.59	11.97 2.22x	83.90	28.23 1.17x
	90.58	0.99 3.68x	85.81	14.40 1.85x	96.08	9.64 2.75x	97.80	9.28 2.87x	96.11	6.61 4.02x	79.70	20.38 1.62x
	88.92	0.70 5.23x	83.98	7.99 3.33x	95.96	8.10 3.27x	97.74	3.35 7.94x	95.21	3.90 6.81x	73.63	10.12 3.27x
GEEP	92.08	1.07 **3.42x**	86.52	19.64 **1.36x**	96.45	11.12 **2.39x**	97.73	15.36 1.73x	96.55	13.25 2.00x	86.40	19.29 **1.72x**
	91.08	0.81 **4.50x**	86.28	10.86 **2.45x**	96.47	8.99 **2.95x**	97.80	8.94 2.97x	96.26	8.96 2.96x	85.26	11.18 **2.96x**
	89.17	0.56 **6.52x**	84.09	7.02 **3.79x**	96.21	6.83 **3.88x**	97.75	4.66 5.71x	95.19	4.73 5.61x	80.52	5.85 **5.66x**

	Ag.news		Amz.F		DBpedia		Yahoo		Yelp.F		Yelp.P	
	A/%	T/s Speedup	A/%	T/s Speedup	A/%	T/s Speedup	A/%	T/s Speedup	A/%	T/s Speedup	A/%	T/s Speedup
BERT	94.54	20.28 1x	65.53	1717.61 1x	99.31	184.52 1x	77.34	158.28 1x	65.89	131.83 1x	95.97	100.14 1x
FastBERT	94.38	16.00 1.27x	63.34	1470.28 1.17x	99.29	43.85 4.21x	76.52	131.98 1.20x	63.29	114.89 1.15x	95.69	71.91 1.39x
	93.88	10.19 1.99x	62.44	1032.32 1.66x	99.24	31.20 5.91x	75.97	105.88 1.49x	62.06	91.73 1.44x	94.99	50.94 1.97x
	93.28	5.97 3.40x	61.80	598.32 2.87x	99.14	24.23 7.62x	75.51	63.95 2.48x	60.88	61.81 2.13x	94.13	35.16 2.85x
GEEP	94.52	11.69 **1.73x**	63.78	1428.94 **1.20x**	99.27	63.13 2.92x	76.99	119.60 **1.32x**	65.34	109.92 **1.20x**	95.82	67.15 **1.49x**
	94.41	6.77 **2.99x**	63.64	867.25 **1.98x**	99.24	47.92 3.85x	77.06	93.40 **1.69x**	65.44	88.39 **1.49x**	95.44	42.46 **2.36x**
	93.72	3.56 **5.70x**	62.82	447.34 **3.84x**	99.18	32.98 5.59x	76.69	54.25 **2.92x**	64.71	56.07 **2.35x**	94.97	26.17 **3.83x**

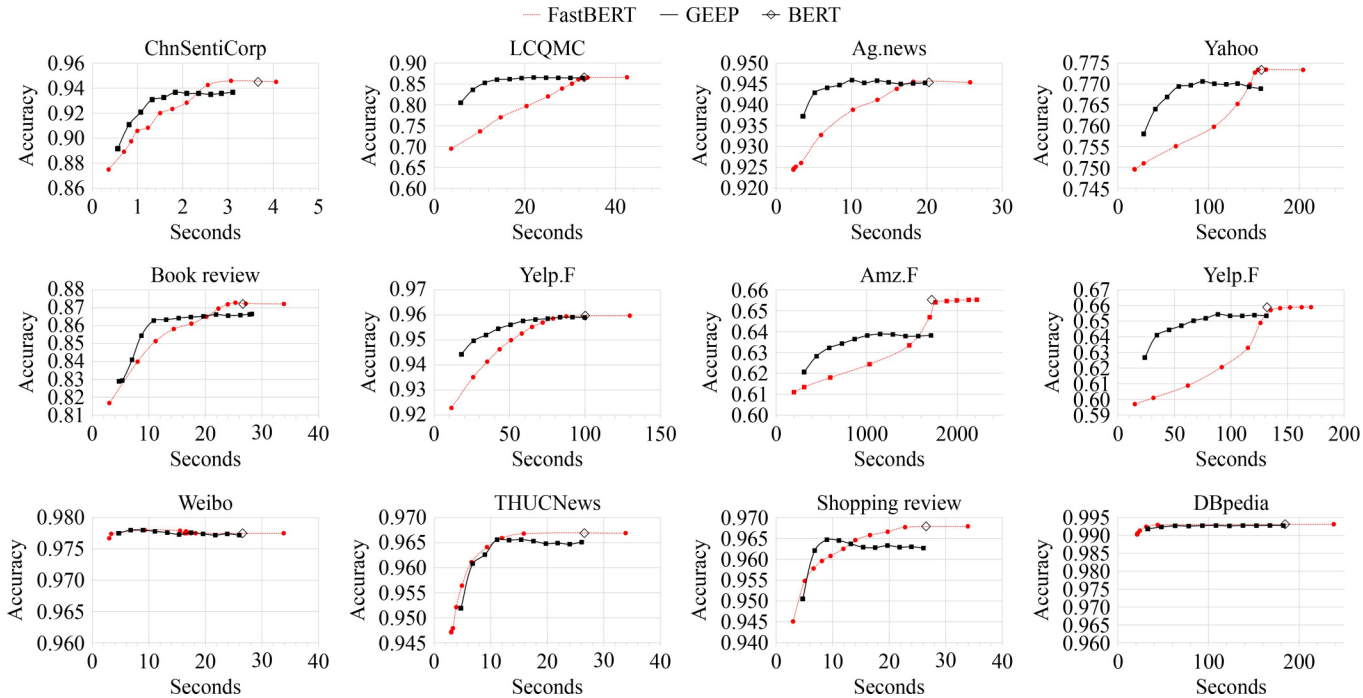

Fig. 4 The Accuracy-time Curve of the GEEP and Baselines in All the Datasets.Curves are made by connecting points obtained from experiments

where KLD is the KL-Divergence and $M = \frac{1}{2}(d_1 + d_2)$. The JSD score lies within the range $[0, 1]$, and a smaller JSD indicates that the two distributions are more similar. Furthermore, we provide a "RANDOM" baseline that is calculated as follows:

$$\frac{\sum_{i=1}^{50000} JSD(d_3 \| d_4)}{50000}, \tag{12}$$

where 50000 is a common vocabulary size. d_3 and d_4 are random probability vectors obtained by:

$$d_3 = rand1(12)/S1. \tag{13}$$

$$d_4 = rand2(12)/S2, \tag{14}$$

Note that $rand1(12)$ and $rand2(12)$ are the functions that produce 12 random values from the uniform distribution between [0,1]. S1 is the sum of $rand1(12)$ and S2 is the sum of $rand2(12)$. We then employ Eq. (12) to generate a token-level exit layer distribution. As shown in Table 3, we present the average of JSD for all tokens in various datasets and compare it with the RANDOM baseline. We can observe that all averaged JSD scores obtained from experimental datasets are much smaller than the baseline (0.0083). This evidence can support Hypothesis 1, where the token-level exit layer distributions in the training data and the testing data are similar. Although Hypothesis 1 is highly general towards downstream tasks. We also suggest that users should validate Hypothesis 1 on their own data before deploying GEEP.

Table 3 The average of JSD scores for all tokens in all the datasets

Name	JSD
RANDOM	**0.0083**
ChnSentiCorp	0.0064
Book review	0.0066
Shopping review	0.0040
Weibo	0.0023
THUCNews	0.0031
LCQMC	0.0073
Ag.News	0.0042
Amz.F	0.0040
DBpedia	0.0008
Yahoo	0.0051
Yelp.F	0.0035
Yelp.P	0.0054

4.4 Inference time analysis

Recall that the GEEP inference algorithm on testing sets is presented in Algorithm 1. We now present a deep analysis of the time consumption of different steps during inference. The time distributions w.r.t. Algorithm 1 are shown in Table 4, where P, I, (s), (%) represent Predict, Inference, seconds, percentage respectively. The time for sorting is less than 1% of the total time, so that is not shown in Table 4. The previous

Table 4 The time distribution for the inference algorithm of GEEP

Name	P/s	P/%	I/s	I/%
ChnSentiCorp	0.22	39.22	0.34	60.44
Book review	1.83	21.01	6.85	78.85
Shopping review	1.86	39.48	2.83	60.17
Weibo	1.82	39.13	2.82	60.57
THUCNews	1.85	39.04	2.87	60.62
LCQMC	2.29	39.18	3.54	60.61
Ag.News	1.38	38.83	2.17	60.89
Amz.F	121.70	39.23	186.34	60.07
DBpedia	12.90	39.13	19.87	60.24
Yahoo	11.12	39.02	17.16	60.23
Yelp.F	9.32	39.23	14.26	60.01
Yelp.P	7.05	39.31	10.76	59.94

experimental results in Fig. 4 have already proven that GEEP consumes less time in inference, compared to FastBERT and the original BERT model. From Table 4, we can observe that time for prediction (i.e., producing exit scores by querying GPU hash tables) costs about 40% of the total time. This shows that the inference time of the proposed GEEP method can be further reduced with a more efficient GPU hash table implementation, e.g., [23]. We leave it as future work.

4.5 Ablation study

GEEP has two major components, i.e., the *exit layer prediction* (ELP) and the *shared exit loss* (SEL). In this part, we proceed to present an ablation study of GEEP to examine the relative importance of ELP and SEL. As shown in Fig. 5, we find that GEEP w/o. ELP (i.e., only Shared Exit Loss) performs better in higher speedup rates (which takes less time), while GEEP w/o. SEL (i.e., only GPU Hash Table) performs better in lower speedup rates (which takes more time). Overall speaking, GEEP achieves better performance on average, compared to existing approaches. In addition, the green line represents GEEP without Exit Layer Prediction, which is equivalent to FastBERT trained by Shared Exit Loss, which still introduces the redundant sub-classifier computation described in Fig. 1. For similar accuracy, it consumes more time than GEEP, and its curve is longer on the x-axis, which represents inference time.

Figure 6 shows more detailed performances of mentioned models in this paper. Fig. 4 and Fig. 5 are derived from Fig. 6.

4.6 Discussion for industrial applications

Furthermore, GEEP consists of two components, i.e., SEL and ELP, that can work independently. For industrial applications, a user can draw a performance curve similar to Fig. 5 based on the industrial dataset. Based on the online serving requirements, the user can choose the best setting accordingly (i.e., choosing a setting with the highest accuracy that satisfies the time consumption requirements).

For online serving, a high QPS (Query Per Second) application could obtain higher speedup using GEEP, compared to FastBERT. Because we can treat it as an offline processing task in a short time window (e.g., 1 second). For low QPS applications (i.e., data comes in a streaming fashion), GEEP performs comparable to or slightly better than FastBERT. As GEEP pre-computes the exit layer of the tokens in the input, this helps to decide which layer to obtain the results. While FastBERT needs to calculate the sub-classifier forwarding and the normalized cross-entropy at each layer sequentially, which consumes a bit more time.

4.7 Cascade mode

We go through newer early exit methods such as CascadeBERT [32], PABEE [16], Early Exiting with Ensemble [33], LeeBERT [34], and DeeBERT [35]. Among these work, Early Exiting with Ensemble [33] and LeeBERT [34] do not provide the source code yet. CascadeBERT [32] reports that it outperforms PABEE [16] and DeeBERT [35] in almost all experiments. Thus, we compare our GEEP with CascadeBERT on three GLUE [36] tasks including SST-2, QNLI, and RTE. CascadeBERT integrates a small BERT (2 layers) and a big BERT (12 layers) so we build a GEEP (cascade) model which only consists of the transformer backbone (with the teacher classifier) and the student classifier 2 in Fig. 1. As shown in Table 5, we report accuracy of BERT, GEEP (cascade) and CascadeBERT. These models are trained on the training split for 5 epochs, and tested on the validation

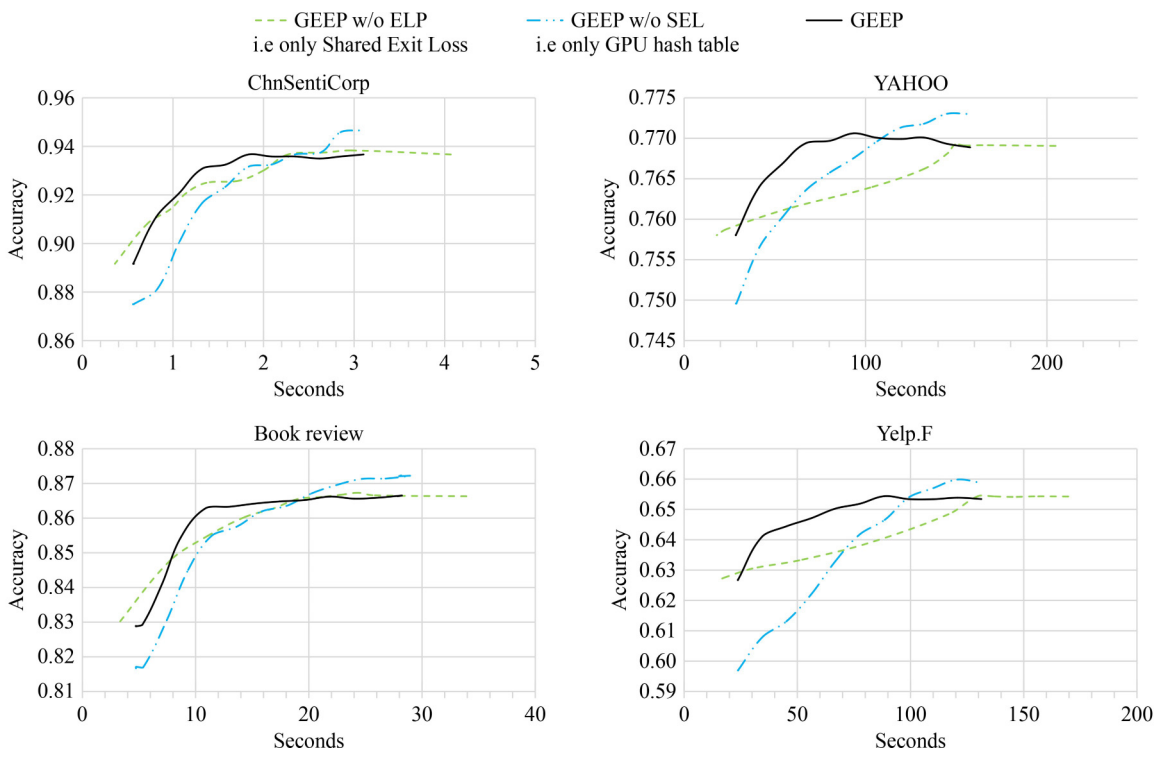

Fig. 5 Ablation study of GEEP. Curves are made by connecting points obtained from experiments

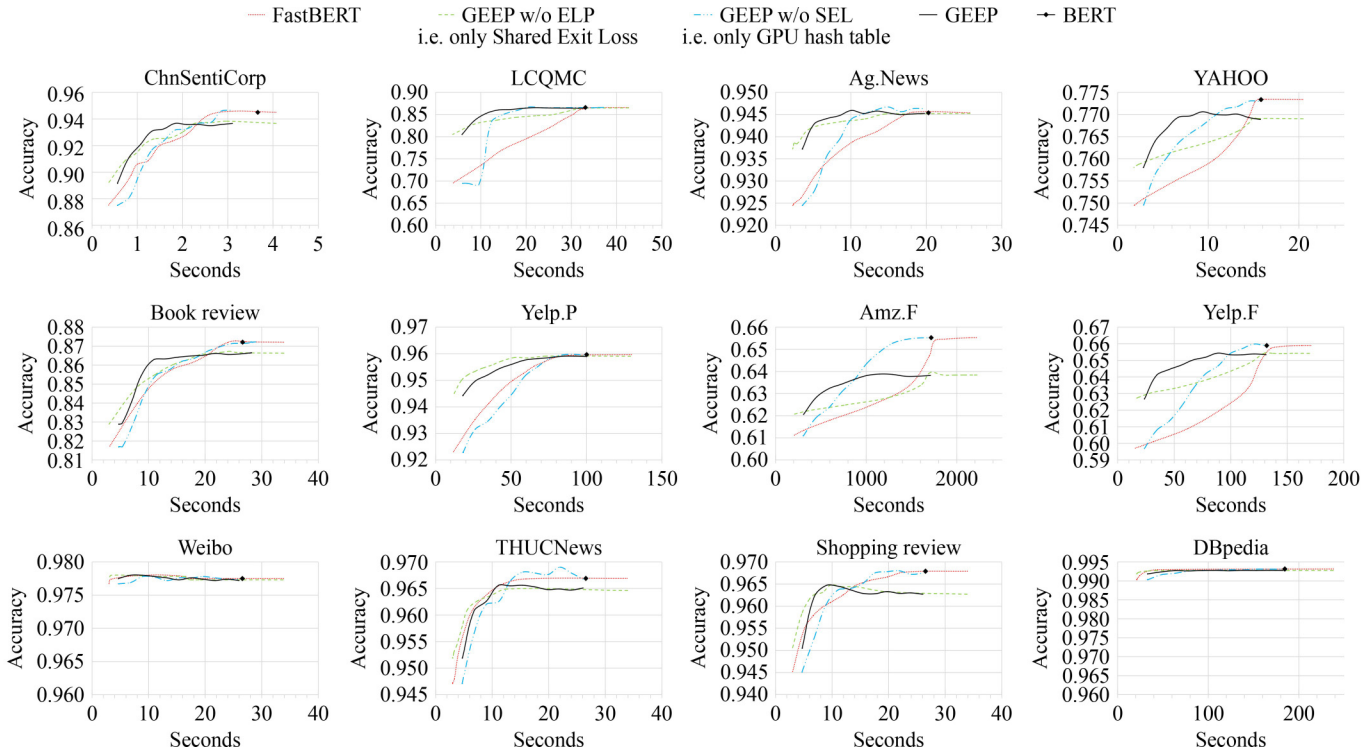

Fig. 6 Detailed performances of mentioned models in this paper

Table 5 Accuracy of GEEP (cascade) and CascadeBERT on GLUE tasks

Model	SST-2/%	QNLI/%	RTE/%
BERT	90.37	89.60	67.10
CascadeBERT	**87.84**	84.49	**62.09**
GEEP (cascade)	86.60	**85.74**	61.73

split of corresponding datasets. Both CascadeBERT and GEEP (cascade) are adjusted to speed up about 6 times. Better accuracy values from CascadeBERT or GEEP (cascade) are in bold. We can observe that GEEP (cascade) provides competitive performance, compared to CascadeBERT. In addition, GEEP (cascade) relies on fewer parameters since it includes 12 layers transformer but CascadeBERT owns $2+12=14$ layers. In future work, we plan to figure out a best combination of cascaded classifiers (i.e., [2,6,8,12]) for each dataset.

4.8 Accuracy drop and training data size

Shared Exit Loss proposed in this paper utilizes a paradigm of shared parameters, which is a cost-effective way for model parameterization. However, shared parameters do not always work well, especially for tasks dealing with large training data. In this part, we try to figure out the relation between the accuracy drop and the training data size. As shown in Table 6, we give accuracy (%) of BERT and SEL (i.e., BERT with Shared Exit Loss, without Exit Layer Prediction) on twelve datasets. Then we can calculate the Accuracy Drop = BERT − SEL. The **LOG10** column contains the LOG10 value of the training data size for each dataset (see Table 1). For example, the Ag.News dataset includes 120,000 samples for training, and its LOG10 value is $\log_{10}(120,000) \approx 5.07$. We draw a point for each dataset in Fig. 7 whose x-axis is Log_{10}(Training Data Size) (i.e., LOG10 values in Table 6),

Table 6 Accuracy (%) of BERT and SEL (BERT with shared exit loss)

Dataset	BERT/%	SEL/%	LOG10
ChnSentiCorp	94.50	93.67	3.98
Book review	87.21	86.63	4.30
Shopping review	96.79	96.27	4.30
Weibo	97.75	97.73	4.99
THUCNews	96.69	96.46	4.69
LCQMC	86.60	86.51	5.37
Ag.News	94.54	94.53	5.07
Amz.F	65.53	63.84	6.47
DBpedia	99.31	99.28	5.74
Yahoo	77.34	76.91	6.14
Yelp.F	65.89	65.43	5.81
Yelp.P	95.97	95.91	5.74

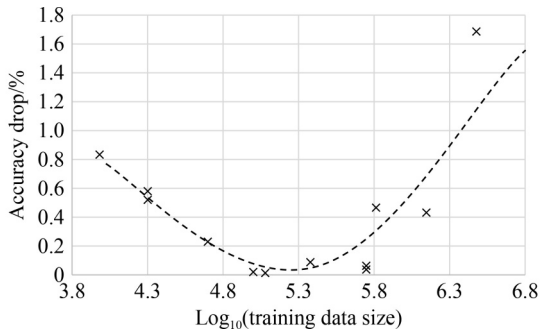

Fig. 7 Accuracy drop and Log_{10}(training data size)

and y-axis is Accuracy Drop. We draw a curve to fit these points, and we observe that this curve firstly decreases when Log_{10}(Training Data Size) is smaller than about 5 (it means the training data contains about 100,000 samples). When Log_{10}(Training Data Size) is larger than 5, the curve increases.

We offer a conjectural explanation for this curve that when training data contains samples less than 100,000 and the Accuracy Drop is mainly caused by insufficient training data and reaching local optimum with poor generalization. When training data contains samples more than 100,000 and the Accuracy Drop is mainly caused by shared parameters which limit the expression power of the model. In conclusion, we may not need to increase the parameters of the Shared Exit when the training data contains samples less than 100,000.

5 Conclusion

In this paper, we propose GEEP which integrates two approaches to enhance fast adaptive inference of pre-trained language models such as BERT. The shared exit loss makes the teacher classifier in GEEP more knowledgeable, and the exit layer prediction avoids batch size shrinkage and redundant computation. The former improves model effectiveness, and the latter improves efficiency. Extensive experiments show the advantages of the GEEP method against competitive baselines.

In "Fig. 4, we observe that GEEP (the black cure) introduces a drop in accuracy, compared to the original BERT (the rhombus), for datasets, e.g., Yahoo and Amz.F. Thus, we can confirm that multi-task learning protocol of the shared exit loss, do harm to the model training. In future work, we will try to enlarge the capacity of the Shared Exit and find an approach to weight each component in the shared exit loss, for acquiring the pareto optimality of GEEP which exits from each student classifier.

Acknowledgements This work has been supported by the National Natural Science Foundation of China (Grant Nos. U1911203, 61877018, 61977025, 62202170), and Alibaba Group through the Alibaba Innovation Research Program.

References

1. Devlin J, Chang M W, Lee K, Toutanova K. BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). 2019, 4171−4186
2. Radford A, Narasimhan K. Improving language understanding by generative pre-training. See cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf website. 2018
3. Yang Z, Dai Z, Yang Y, Carbonell J G, Salakhutdinov R, Le Q. XLNet: generalized autoregressive pretraining for language understanding. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems. 2019, 517
4. Gou J, Yu B, Maybank S J, Tao D. Knowledge distillation: a survey. International Journal of Computer Vision, 2021, 129(6): 1789–1819
5. Laskaridis S, Kouris A, Lane N D. Adaptive inference through early-exit networks: design, challenges and directions. In: Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning. 2021, 1−6
6. Liu W, Zhou P, Wang Z, Zhao Z, Deng H, Ju Q. FastBERT: a self-distilling BERT with adaptive inference time. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. 2020, 6035−6044
7. Wang C, Qiu M, Zhang T, Liu T, Li L, Wang J, Wang M, Huang J, Lin W. EasyNLP: A comprehensive and easy-to-use toolkit for natural language processing. 2022, arXiv preprint arXiv: 2205.00258
8. Wang C, Qiu M, Huang J. Building natural language processing applications with EasyNLP. In: Proceedings of the 31st ACM International Conference on Information & Knowledge Management. 2022, 5100−5101
9. Buciluǎ C, Caruana R, Niculescu-Mizil A. Model compression. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2006, 535−541
10. Hinton G, Vinyals O, Dean J. Distilling the knowledge in a neural network. 2015, arXiv preprint arXiv: 1503.02531
11. Sanh V, Debut L, Chaumond J, Wolf T. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. 2019, arXiv preprint arXiv: 1910.01108
12. Zhang L, Song J, Gao A, Chen J, Bao C, Ma K. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In: Proceedings of 2019 IEEE/CVF International Conference on Computer Vision (ICCV). 2019, 3712−3721
13. Berestizshevsky K, Even G. Dynamically sacrificing accuracy for reduced computation: Cascaded inference based on softmax confidence. In: Proceedings of the Artificial Neural Networks and Machine Learning-ICANN 2019: Deep Learning: the 28th International Conference on Artificial Neural Networks. 2019, 306−320
14. Gormez A, Koyuncu E. Class means as an early exit decision mechanism. 2021, arXiv preprint arXiv: 2103.01148v1
15. Jiang H, Kim B, Guan M Y, Gupta M. To trust or not to trust a classifier. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. 2018, 5546−5557
16. Zhou W, Xu C, Ge T, McAuley J J, Xu K, Wei F. BERT loses patience: fast and robust inference with early exit. In: Proceedings of the Conference on Neural Information Processing Systems. 2020, 18330−18341
17. Sun T, Liu X, Zhu W, Geng Z, Wu L, He Y, Ni Y, Xie G, Huang X, Qiu X. A simple hash-based early exiting approach for language understanding and generation. In: Proceedings of Findings of the Association for Computational Linguistics: ACL 2022. 2022, 2409−2421
18. Lessley B, Childs H. Data-parallel hashing techniques for GPU architectures. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(1): 237–250
19. Cormen T H, Leiserson C E, Rivest R L, Stein C. Introduction to Algorithms. 3rd ed. Massachusetts: The MIT Press, 2009
20. Bordawekar R. Evaluation of parallel hashing techniques. In: Proceedings (Findings) of the GPU Technology Conference. See on-demand.gputechconf.com/gtc/2014/presentations/S4507-evaluation-of-parallel-hashing-techniques.pdf website. 2014, 1−27
21. Pagh R, Rodler F F. Cuckoo hashing. Journal of Algorithms, 2004, 51(2): 122–144
22. Breslow A D, Jayasena N S. Morton filters: faster, space-efficient cuckoo filters via biasing, compression, and decoupled logical sparsity. Proceedings of the VLDB Endowment, 2018, 11(9): 1041–1055
23. Alipourfard O, Moshref M, Zhou Y, Yang T, Yu M. A comparison of performance and accuracy of measurement algorithms in software. In: Proceedings of the Symposium on SDN Research. 2018, 18
24. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A N, Kaiser Ł, Polosukhin I. Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017, 6000−6010
25. Voita E, Sennrich R, Titov I. The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives. In: Proceedings of 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). 2019, 4396−4406
26. Xiong R, Yang Y, He D, Zheng K, Zheng S, Xing C, Zhang H, Lan Y, Wang L, Liu T. On layer normalization in the transformer architecture. In: Proceedings of the 37th International Conference on Machine Learning. 2020, 10524−10533
27. Cover T M, Thomas J A. Elements of Information Theory. 2nd ed. Hoboken: John Wiley & Sons, Inc., 2006, 57−58
28. Liu X, Chen Q, Deng C, Zeng H, Chen J, Li D, Tang B. LCQMC: A large-scale Chinese question matching corpus. In: Proceedings of the 27th International Conference on Computational Linguistics. 2018, 1952−1962
29. Zhang X, Zhao J, LeCun Y. Character-level convolutional networks for text classification. In: Proceedings of the 28th International Conference on Neural Information Processing Systems. 2015, 649−657

30. Jiao X, Yin Y, Shang L, Jiang X, Chen X, Li L, Wang F, Liu Q. TinyBERT: distilling BERT for natural language understanding. In: Proceedings of Findings of the Association for Computational Linguistics: EMNLP 2020. 2020, 4163−4174
31. Chen X, He B, Hui K, Sun L, Sun Y. Simplified tinyBERT: Knowledge distillation for document retrieval. In: Proceedings of the 43rd European Conference on Information Retrieval. 2021, 241−248
32. Li L, Lin Y, Chen D, Ren S, Li P, Zhou J, Sun X. CascadeBERT: Accelerating inference of pre-trained language models via calibrated complete models cascade. In: Proceedings of Findings of the Association for Computational Linguistics: EMNLP 2021. 2021, 475−486
33. Sun T, Zhou Y, Liu X, Zhang X, Jiang H, Cao Z, Huang X, Qiu X. Early exiting with ensemble internal classifiers. 2021, arXiv preprint arXiv: 2105.13792
34. Zhu W. LeeBERT: Learned Early Exit for BERT with cross-level optimization. In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). 2021, 2968−2980
35. Ji X, Tang R, Lee J, Yu Y, Lin J. DeeBERT: dynamic early exiting for accelerating BERT inference. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. 2020, 2246−2251
36. Wang A, Singh A, Michael J, Hill F, Levy O, Bowman S. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In: Proceedings of 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP. 2018, 353−355

Lei Li received his master degree in computer technology from Yunnan University, China in 2019. He is a PhD candidate in software engineering at East China Normal University, China, under the supervision of Professor Ming Gao. He is interested in Natural Language Processing and efficient model inference.

Chengyu Wang is an algorithm expert at Alibaba Group. He has obtained his PhD degree from East China Normal University (ECNU), China. Currently, he works on deep learning algorithms on various topics for Alibaba Cloud Machine Learning Platform of AI (PAI), and builds NLP toolkits named EasyTransfer and EasyNLP for Alibaba Cloud. He has published 70+ research papers in international conferences and journals, such as ACL, KDD, WWW, SIGIR, AAAI, TKDE, and WSDM.

Minghui Qiu held a PhD degree from School of Information Systems, Singapore Management University, Singapore, under the supervision of Associate Prof. Jing Jiang and Prof. Ee-peng Lim. From 2013 to 2014, he visited Language Technologies Institute, Carnegie Mellon University, USA, working with Noah Smith and Alex Smola. In the summer of 2014, he worked as an intern at Google Inc., Mountain View, CA, with Amr Ahmed and Yuan Wang. Recently, he is a senior algorithm expert in Alibaba cloud, working on deep learning and transfer learning for many NLP tasks, including paraphrastic sentence/doc embedding, neural conversation models, and sequence labeling. He is responsible for building the NLP and transfer learning toolkit named EasyNLP for Alibaba Cloud, supporting 10+ business units and 20+ applications in Alibaba Group.

Cen Chen is currently a tenure-track Associate Professor at East China Normal University, China. Before that, she worked as an algorithm expert at Ant Group from Aug 2017 to Aug 2021 (selected as Alistar 2017). She obtained a PhD degree from Singapore Management University under the supervision of Professor Lau Hoong Chuin and Associate Professor Cheng Shihfen from Jan 2013 to Jun 2017. From Aug 2015 to June 2016, she visited the Robotics Institute, Carnegie Mellon University, USA, working with Professor Stephen F. Smith and Dr. Zack Rubinstein. Her research focuses on analyzing, modeling, and designing of intelligent systems for supporting business and/or financial decisionmaking. Recent works include federated learning, transfer learning, and retrieval-based QA.

Ming Gao is working as a professor at School of Data Science and Engineering (DASE), East China Normal University, China. Prior to joining ECNU, he worked with Prof. Ee-Peng Lim as a Postdoctoral Fellow at Social Network Mining Research Group in School of Information System, Singapore Management University, Singapore. Before that, he started his PhD program in 2008 at Fudan University, China. From Aug. 2010 to Feb. His main research interests are knowledge graph, knowledge engineering, user profiling, social mining, and uncertain data management.

Aoying Zhou is a professor on computer science at East China Normal University (ECNU), China, where he is heading the School of Data Science and Engineering. Before joining ECNU in 2008, he worked for Fudan University at the Computer Science Department for 15 years. He is the winner of the National Science Fund for Distinguished Young Scholars supported by NSFC. He is now acting as a vice director of ACM SIGMOD China and Database Technology Committee of China Computer Federation. He is serving as a member of the editorial boards of the VLDB Journal, the WWW Journal, and so on. His research interests include data management, in-memory cluster computing, big data benchmarking, and performance optimization.