SkipBERT: Efficient Inference with Shallow Layer Skipping

Jue Wang[1], **Ke Chen**[1], **Gang Chen**[1,2], **Lidan Shou**[1,2*] and **Julian McAuley**[3]
[1]College of Computer Science and Technology, Zhejiang University
[2]State Key Laboratory of CAD&CG, Zhejiang University
[3]University of California, San Diego
{zjuwangjue,chenk,cg,should}@zju.edu.cn,
jmcauley@ucsd.edu

Abstract

In this paper, we propose SkipBERT to accelerate BERT inference by skipping the computation of shallow layers. To achieve this, our approach encodes small text chunks into independent representations, which are then materialized to approximate the shallow representation of BERT. Since the use of such approximation is inexpensive compared with transformer calculations, we leverage it to replace the shallow layers of BERT to skip their runtime overhead. With off-the-shelf early exit mechanisms, we also skip redundant computation from the highest few layers to further improve inference efficiency. Results on GLUE show that our approach can reduce latency by 65% without sacrificing performance. By using only two-layer transformer calculations, we can still maintain 95% accuracy of BERT.[1]

1 Introduction

Pre-trained language models, such as ELMo (Peters et al., 2018), GPT (Radford et al., 2018), BERT (Devlin et al., 2019), XLNet (Yang et al., 2019), and RoBERTa (Liu et al., 2019), have yielded significant improvements to NLP tasks. Despite the gain in accuracy, these models have significant demands in computation and inference time, limiting their use in resource-constrained or latency-sensitive applications. Therefore, it is desirable to reduce the computational overhead of these models while retaining acceptable accuracy.

Knowledge distillation (KD, Hinton et al. 2015) facilitates the transfer of knowledge embedded in pre-trained language models into small student models (Sanh et al., 2019; Sun et al., 2019; Jiao et al., 2020), which usually reduces the redundant parameters of BERT in a uniform manner. Early exit mechanisms (Xin et al., 2020; Zhou et al., 2020; Liu et al., 2020) then use an adaptive number of transformer layers during inference, aiming to reduce redundant calculations from the highest few layers. However, since they build the sequence representation from scratch for each forward pass, they require a certain number of lower layers to capture basic syntactic and semantic information, making it difficult to further reduce inference costs. This naturally raises a question: Can we reduce the computation at the lower transformer layers?

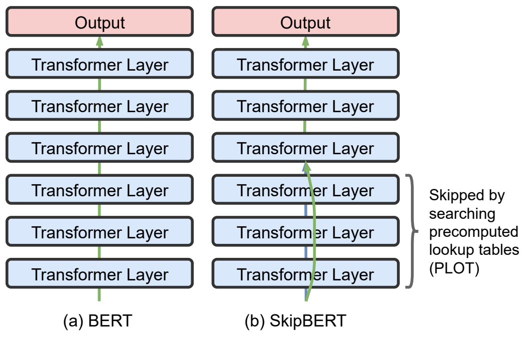


Figure 1: Comparison of inference between BERT and SkipBERT. The computation of shallow (lower) layers of SkipBERT are skipped by searching PLOT.

In this paper, we propose SkipBERT, a novel scheme that skips the computation at the shallow transformer layers of BERT. As revealed by Jawahar et al. (2019); Rogers et al. (2020), the lower layers of BERT mainly focus on short-distance context, while the higher layers are able to capture long-range dependencies. Therefore, it is reasonable to assume that, at lower layers, even if distant tokens are masked, the representation for each token will not vary dramatically. Here, by sweeping over the input text, we get short chunks (n-grams) and use their representations to approximate the hidden states of BERT's lower layers. We then precompute and store representations of text chunks in a *precomputed lookup table* (PLOT). Thus, during inference we only need to *access* PLOT to get the representations of short chunks, which is inexpen-

[*] Corresponding author
[1]Source code is available at https://github.com/LorrinWWW/SkipBERT.

sive compared with transformer computation.

Fig. 1 compares the inference procedure between vanilla BERT and our proposed SkipBERT. In BERT, the input text needs to be processed by a large number of transformer layers in turn, leading to high latency in inference. In comparison, SkipBERT precomputes the hidden states of lower transformer layers, which are accessed via table lookups, rather than computed in inference-time.

Moreover, SkipBERT exhibits effective compatibility with early exit mechanisms: Since the initial sequence representation in our work is partially contextualized (thanks to PLOT) rather than individual word embeddings, SkipBERT allows exiting from a relatively earlier layer than typical BERT variants, while maintaining good accuracy. We empirically verify this in Section 4.5. Therefore, our approach can skip the calculations of lower and higher layers for the same input, thereby further improving the inference speed.

Our contributions are listed as follows:

- We present SkipBERT to avoid computation at BERT's lower layers during inference. Instead, we construct PLOT and use it to approximate their hidden states.
- We incorporate early exit mechanisms as an enhancement to skip redundant computation, leading to further network acceleration.
- We conduct extensive experiments on GLUE. Compared with BERT, SkipBERT is capable of accelerating inference by up to 65% without compromising GLUE score, or accelerating by 82% while retaining 95% accuracy.

2 Related Work

Knowledge Distillation (Hinton et al., 2015) provides an effective way to transfer the knowledge embedded in a teacher network to a student network. The student network is usually more lightweight than the teacher network and thus more computationally efficient. The student network can be structurally identical to the teacher but contains fewer layers or hidden units, e.g. BERT-PKD (Sun et al., 2019), DistilBERT (Sanh et al., 2019), TinyBERT (Jiao et al., 2020), MiniLM (Wang et al., 2020), and BERT-EMD (Li et al., 2020). Meanwhile, some work adopts specifically designed networks, e.g. SqueezeBERT (Iandola et al., 2020) and MobileBERT (Sun et al., 2020), to reduce the computation per layer.

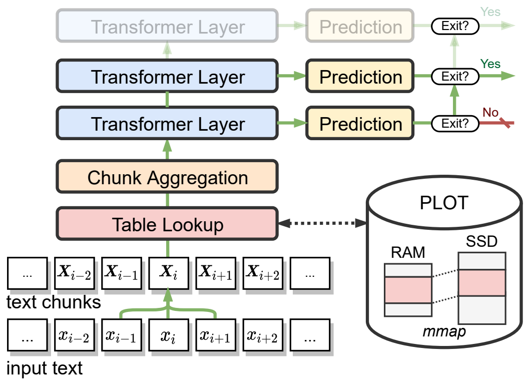


Figure 2: The overview of our system during inference. PLOT stores hidden states of the local transformer layers. The global transformer layers are enhanced with early exit mechanisms.

Input-adaptive inference allows models to choose different computational paths according to the input during inference. In this way, simpler input samples usually require less calculation to make predictions. Recently, DeeBERT (Xin et al., 2020) adapts confidence-based BranchyNet (Teerapittayanon et al., 2016), which uses entropy as an early-exit criterion. FastBERT (Liu et al., 2020) uses self-distillation to train the branch classifiers. RightTool (Schwartz et al., 2020) leverages the same early-exit criterion as in the Shallow-Deep Network (Kaya et al., 2019), i.e., softmax scores of predictions. PABEE (Zhou et al., 2020) stops inference when the intermediate predictions of the internal classifiers remain unchanged consecutively.

Precomputation has also been studied in information retrieval, where documents are assumed to be stored at local database so their representation can be precomputed (Gao et al., 2020). However, this method may not be suitable for other tasks where the input text is unknown before inference.

3 Model

During training, SkipBERT consists of two groups of transformer layers, *local* transformer layers for encoding short-distance context, and *global* transformer layers for leveraging the full context. Once pre-training finishes, our approach will replace local transformer layers with PLOT, which stores the hidden states of local transformer layers; we also enhance global transformer layers with early exit mechanisms to further accelerate inference speed. Fig. 2 presents the overview of our system.

3.1 Preparing Inputs

We define the input text as a sequence of tokens $\boldsymbol{x} = [x_i]_{0 \leq i < n}$ in BERT's input style, where n is the number of input tokens.

As shown in Fig. 3, we sweep over the input text to get three-token chunks (tri-grams, $\boldsymbol{X}_i = [x_{i-1}, x_i, x_{i+1}]$), which will also be taken as the index entries of PLOT later. We let cross-border tokens be padding tokens, i.e. $x_{-1} = x_n = x_{[\mathrm{PAD}]}$.

We show in Section 4.8.2 that using longer text chunks (e.g. 5-grams) will improve accuracy since they can bring more context information than tri-grams. However, the number of 5-grams is too large to be enumerated and stored, and thus it is hard to use them in actual applications.

3.2 Leveraging Local Context

Fig. 3 illustrates our procedure to leverage local context. By mapping each word to a d-dimensional embedding, we denote the chunk embeddings by $\tilde{\boldsymbol{X}}_i \in \mathbb{R}^{3\times d}$. For local transformer layers, we inject position embeddings $\boldsymbol{P} \in \mathbb{R}^{3\times d}$, and define the initial chunk representations as follows:

$$\boldsymbol{H}_i^{(0)} = \mathrm{LN}(\tilde{\boldsymbol{X}}_i + \boldsymbol{P}) \qquad (1)$$

where $\mathrm{LN}(\cdot)$ is layer normalization.

We use L_{loc} transformer layers to leverage the local context of each text chunk. For layer $0 \leq m < L_{\mathrm{loc}}$, we have:

$$\boldsymbol{H}_i^{(m+1)} = \mathrm{Transformer}^{(m)}(\boldsymbol{H}_i^{(m)}). \qquad (2)$$

Note that since each chunk is short enough, it would be possible to precompute these representations before inference. More importantly, these representations are good approximations of those produced by the respective shallow layers of BERT. Thus, given a tri-gram, the embedding produced from $(L_{\mathrm{loc}} - 1)$-th layer is taken as its respective data entry stored in PLOT. We also precompute bi-grams and uni-grams following the same procedure of tri-grams. When a lookup of tri-gram fails (out-of-vocabulary, OOV), the system will resort to bi-grams or uni-grams as an alternative.

Specifically, we randomly replace $\delta\%$ of tri-grams by bi-grams during training. On the one hand, such random replacement allows the model to encounter bi-grams during training, so as to better handle OOVs in inference; on the other hand, it can also be considered a variant of Dropout (Srivastava et al., 2014), which drops tokens rather than

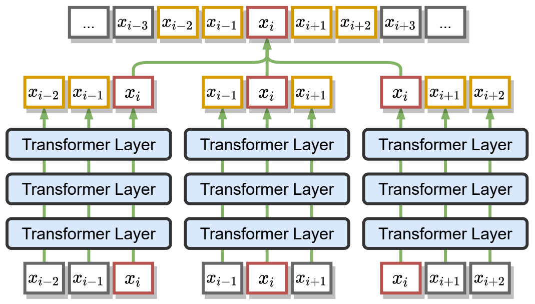


Figure 3: Illustration of how we leverage the local context for token x_i. Yellow-bordered boxes indicate embeddings that interact with token x_i; grey-bordered boxes are independent of x_i.

hidden units, thereby improving the robustness of our approach. Section 4.7 shows $\delta = 10\%$ works well with different OOV rates. We also show in Section 4.8.2 that even bi-grams have a clear advantage over the baseline, which can be seen as an extreme case when all tri-gram lookups fail.

3.3 Aggregating Text Chunks

Now we get a list of contextualized chunk embeddings. Here we aggregate them to form a feature sequence corresponding to the original input text.

Each token occurs at three consecutive tri-grams, as shown in Fig. 3. By calculating a weighted sum of embeddings that correspond to the same token, we can leverage its context of five tokens:

$$\tilde{h}_i = \sum_{j=-1,0,1} H_{i+j,1-j}^{(L_{\mathrm{loc}})} \cdot \mathrm{Gate}(H_{i+j,1-j}^{(L_{\mathrm{loc}})}), \qquad (3)$$

where $\mathrm{Gate}(\cdot)$ is a sigmoid-based gating mechanism such that $\mathrm{Gate}(x) = \sigma(v_{\mathrm{G}} \cdot x + b_{\mathrm{G}})$, where v_{G} is a learnable vector and b_{G} is a learnable scalar.

Note that these embeddings do not have a sense of the order of the sequence. So we need to inject position and segment embeddings before sending them to the subsequent transformer layers:

$$h_i^{(0)} = \begin{cases} \mathrm{LN}(\tilde{h}_i + \tilde{p}_i + \tilde{s}_A), \text{ if } x_i \in A, \\ \mathrm{LN}(\tilde{h}_i + \tilde{p}_i + \tilde{s}_B), \text{ if } x_i \in B, \end{cases} \qquad (4)$$

where $\tilde{p}_i$ and $\tilde{s}_{A/B}$ are position and segment embeddings respectively as in Devlin et al. (2019).

3.4 Leveraging Global Context

We denote by $\boldsymbol{h}^{(0)} = [h_i^{(0)}]_{0 \leq i < n}$ the aggregated sequence representation. We use L_{glo} transformer layers to further contextualize it. For layer $0 \leq m < L_{\mathrm{glo}}$, we have:

$$\boldsymbol{h}^{(m+1)} = \mathrm{Transformer}^{(m+L_{\mathrm{loc}})}(\boldsymbol{h}^{(m)}). \qquad (5)$$

Since we focus on text classification and regression tasks, we use the representation corresponding to token $x_{\text{[CLS]}}$ to compute logit scores:

$$z = \text{Classifier}(h_{\text{[CLS]}}^{(L_{\text{glo}})}) \tag{6}$$

where Classifier$(\cdot)$ is a two-layer feedforward neural network.

When an early exit mechanism is activated, we compute logit scores for each global transformer layer as follows:

$$z^{(m)} = \text{Classifier}^{(m)}(h_{\text{[CLS]}}^{(m)}) \tag{7}$$

We adopt a simple confidence-based early exit mechanism, i.e., once the prediction's maximum logit score is higher than a pre-defined threshold, the result will be returned without passing through the next transformer layers.

3.5 Training

We mainly adopt the two-stage learning procedure proposed in TinyBERT (Jiao et al., 2020). It includes *general distillation* (GD) conducted on large-scale unlabeled corpora, and *task-specific distillation* (TD) to learn from fine-tuned BERT.

General Distillation We perform distillation on the hidden states and attention scores. We compute loss on the chunk aggregation layer and global transformer layer. The local transformer layers are trained with supervision signals from upper layers. The loss is defined as follows:

$$\mathcal{L}_{\text{GD}} = \mathcal{L}_{\text{att}} + \mathcal{L}_{\text{hid}} \tag{8}$$

and we define $\mathcal{L}_{\text{att}}$ and $\mathcal{L}_{\text{hid}}$ as the mean-squared error (MSE) of attention scores and hidden states between the teacher (T) and student (S):

$$\mathcal{L}_{\text{att}} = \sum_{m=1}^{L_{\text{glo}}} \text{MSE}(\boldsymbol{a}_{\text{S}}^{(m)}, \boldsymbol{a}_{\text{T}}^{(g_{\text{att}}(m))}) \tag{9}$$

$$\mathcal{L}_{\text{hid}} = \sum_{m=0}^{L_{\text{glo}}} \text{MSE}(\boldsymbol{h}_{\text{S}}^{(m)} \boldsymbol{W}_{\text{hid}}, \boldsymbol{h}_{\text{T}}^{(g_{\text{hid}}(m))}) \tag{10}$$

where $\boldsymbol{a}^{(m+1)}$ and $\boldsymbol{h}^{(m+1)}$ represent the attention score matrix and hidden states of the m-th transformer layer; $\boldsymbol{h}^{(0)}$ is the outputs of chunk aggregation layer; $\boldsymbol{W}_{\text{hid}}$ is a learnable matrix to transform the hidden states of the student into the same space as the teacher; $g_{\text{att}}(\cdot)$ and $g_{\text{hid}}(\cdot)$ define the layer mapping function between the student and teacher.

For attention-based distillation, we use the uniform mapping strategy to leverage the heterogeneous attention patterns across different layers. For hidden states-based distillation, we use top mapping strategy since the initial sequence representation (outputs of chunk aggregation) are already partially contextualized. The detailed illustration of layer mapping is presented at Appendix E.

Task-Specific Distillation We start from the generally distilled SkipBERT, and use fine-tuned BERT as the teacher for task-specific distillation. The loss is defined as follows:

$$\mathcal{L}_{\text{TD}} = \beta(\mathcal{L}_{\text{att}} + \mathcal{L}_{\text{hid}}) + \mathcal{L}_{\text{pred}} \tag{11}$$

where β is a factor to control the loss weight; $\mathcal{L}_{\text{pred}}$ is the prediction loss that will be defined below.

For classification, the loss function $\mathcal{L}_{\text{pred}}$ is calculated via cross entropy:

$$\mathcal{L}_{\text{pred}} = \text{CE}(z_{\text{S}}/\tau, z_{\text{T}}/\tau) \tag{12}$$

where z_{S} are the logits predicted by the student; z_{T} are the logits predicted by the teacher; τ is the temperature to smooth the probability distribution to facilitate distillation training. For regression, the loss is instead calculated by MSE, i.e., $\mathcal{L}_{\text{pred}} = \text{MSE}(z_{\text{S}}, z_{\text{T}})$.

Early Exit Specifically, when SkipBERT enables early exit mechanisms, we need to train internal classifiers to predict based on the hidden states of their respective layers. Overall, we train the model to minimize a weighted average loss as follows:

$$\mathcal{L}_{\text{pred}} = \frac{\sum_{m=1}^{L_{\text{glo}}} m \mathcal{L}_{\text{pred}}^{(m)}}{\sum_{m=1}^{L_{\text{glo}}} m} \tag{13}$$

where $\mathcal{L}_{\text{pred}}^{(m)}$ is the loss between the predictions of the teacher and the m-th intermediate classifier of the student.

3.6 Constructing PLOT

Considering that the local transformer layers mostly capture generalized knowledge, which do not vary significantly across different tasks, we *do not* update the local transformer layers during fine-tuning. Therefore, once general distillation is finished, we can compute their hidden states to construct PLOT.

To ensure fast response, PLOT should ideally be loaded in the server's RAM during inference. However, such a table could be too large to fully fit into

Model	MACs	Latency	GLUE Score	CoLA (8.5k)	SST-2 (67k)	MRPC (3.5k)	STS-B (5.7k)	QQP (364k)	MNLI (393k)	QNLI (108k)	RTE (2.5k)	WNLI (0.6k)
BERT$_{12}$	10.9G	100%	78.3	52.1	93.5	88.9/84.8	87.1/85.8	71.2/89.2	84.6/83.4	90.5	66.4	65.1
BERT$_6$-PKD	5.4G	-49%	-	43.5	92.0	85.0	-/81.6	70.7/-	81.5/81.0	89.0	65.5	-
DistilBERT$_6$	5.4G	-49%	-	49.0	92.5	86.9	-/81.3	70.1/-	82.6/81.3	88.9	58.4	-
BERT-of-Theseus$_6$	5.4G	-49%	77.1	47.8	92.2	87.6/83.2	85.6/84.1	71.6/89.3	82.4/82.1	89.6	66.2	65.1
TinyBERT$_6$v2	5.4G	-49%	-	46.1	92.6	88.0/-	-/83.9	71.3/-	84.4/83.1	89.8	69.7	-
BERT-EMD$_6$	5.4G	-49%	78.7	47.5	**93.3**	**89.8/86.4**	**87.6/86.8**	**72.0/89.3**	**84.7**/83.5	**90.7**	**71.7**	65.1
SkipBERT$_{6+6}$	5.4G	-49%	**78.9**	**52.7**	**93.3**	88.9/85.0	87.0/85.8	71.9/89.2	84.3/**84.2**	90.6	70.6	65.1
w/ exit	**3.7G**	**-65%**	78.3	50.8	91.9	88.6/84.8	86.7/85.4	71.8/89.0	83.8/83.8	90.2	69.8	65.1
BERT$_{\text{mini4}}$	**0.4G**	-66%	65.8	0.0	85.9	81.1/71.8	75.4/73.3	66.4/86.2	74.8/74.3	84.1	57.9	62.3
BERT$_{\text{small4}}$	1.6G	-66%	71.2	27.8	89.7	83.4/76.2	78.8/77.0	68.1/87.0	77.6/77.0	86.4	61.8	62.3
DistilBERT$_4$	3.6G	-66%	-	32.8	**91.4**	82.4/-	-/76.1	68.5/-	78.9/78.0	85.2	54.1	-
BERT$_4$-PKD	3.6G	-66%	-	24.8	89.4	82.6/-	-/79.8	70.2/-	79.9/79.3	85.1	62.3	-
TinyBERT$_4$v2	0.6G	-66%	-	25.3	90.0	85.4/-	-/80.4	68.9/-	81.2 80.3	86.2	63.9	-
BERT-EMD$_4$	0.6G	-66%	73.6	25.6	91.0	87.6/82.4	83.6/82.3	69.3/87.9	**82.1**/80.6	87.2	**66.2**	65.1
SkipBERT$_{6+4}$	0.6G	-66%	**75.6**	**39.8**	91.3	**87.7**/82.7	**84.1/82.8**	**70.4/88.3**	82.0/**81.6**	**88.5**	66.1	65.1
w/ exit	0.5G	-74%	75.1	36.0	91.2	87.6/**83.5**	84.1/82.8	70.4/88.3	81.9/81.3	88.5	64.8	65.1
SkipBERT$_{6+2}$	1.8G	**-82%**	74.0	36.0	90.9	85.9/80.5	82.0/80.6	70.2/88.6	80.2/79.9	86.6	63.6	65.1

Table 1: Results on the GLUE benchmark. MACs are multiply–accumulate operations of text with 128 tokens. The results of DistilBERT$_4$, BERT-PKD, TinyBERTv2 are taken from Li et al. (2020). The results of BERT$_{\text{mini4/small4}}$ are taken from google-research/bert. The result of DistilBERT$_6$ is taken from (Jiao et al., 2020). Others are taken from the official GLUE website. "w/ exit": enabling the early exit mechanism.

RAM. Hence we propose to adopt memory-mapped files (*mmap*), which allows for file access via the virtual memory mechanism. By using *mmap*, the frequently used chunk embeddings reside in RAM for fast lookup, while the rare chunks can be stored on SSD, and will be loaded to RAM only when the system demand-pages them. Appendix D presents a simple implementation of PLOT.

4 Experiments

4.1 Data

We use the corpora of Wikipedia[2] and BooksCorpus[3] (Zhu et al., 2015) to perform general distillation. For task-specific distillation, we mainly evaluate SkipBERT and compare it with other baselines on the GLUE benchmark (Wang et al., 2018). Appendix F provides some details.

4.2 Setup

We denote by SkipBERT$_{6+6}$ the scheme with 6 local transformer layers (converted to PLOT) and 6 global transformer layers, each having a hidden size of 768 and intermediate size of 3072. For direct comparisons with 4-layer baselines, we instantiate SkipBERT$_{6+4}$ with 4 thin global transformer layers (hidden size of 312 and intermediate size of 1200). We also instantiate SkipBERT$_{6+2}$ with only 2 global transformer layers to further reduce the latency. Appendix C presents detailed settings.

Training For general distillation, we randomly initialize SkipBERT, and pre-train it with Lamb optimizer (You et al., 2019). We use linear learning rate decay with the peak learning rate of 1e-3 and a batch size of 2048 for around 80k steps, including 4000 warm-up steps.

For task-specific distillation, under the supervision of a fine-tuned BERT, we use AdamW (Kingma and Ba, 2015) to train 20 epochs with a learning rate of 2e-5. We slightly tune the hyper-parameters across different tasks, and the details can be found in Appendix B. We do not use any data augmentation strategies.

Inference Following prior work, we evaluate *latency* by performing inference on a per-instance basis, i.e. the batch size for inference is set to 1. This is a common latency-sensitive scenario when processing individual requests from different users. We note that latency on modern GPUs is not sensitive to the hidden size, but mainly depends on the number of sequential operations, i.e. the number of network layers. We report the median performance over 5 runs with different random seeds.

4.3 Results on GLUE

We submitted our model predictions to the official GLUE evaluation server[4] to obtain results on the

[2]`https://dumps.wikimedia.org/`

[3]`https://yknzhu.wixsite.com/mbweb`

[4]`https://gluebenchmark.com/`

test set, as summarized in Table 1. We present the results of TinyBERT v2 reported by Li et al. (2020) as the v2 model employs more training corpora than v1, and they eliminate the data augmentation strategy for a fair comparison.

By comparing with baselines (we compare with 6-layer models and 4-layer models separately), we can see that SkipBERT outperforms all compared approaches in terms of GLUE score. Compared with TinyBERT, as we mainly follow their distillation process, our approach shows clear advantages on all tasks. BERT-EMD employs a more sophisticated task-specific distillation process based on general-distilled TinyBERT, and further improves the overall performance. Nevertheless, SkipBERT still maintains an advantage in the overall score.

Specifically, SkipBERT$_{6+4}$ has a similar inference speed to the 4-layer baselines, but achieves higher accuracy on most tasks. We consider that a 4-layer model is somewhat too shallow to capture complex dependencies from scratch. In contrast, SkipBERT effectively compensates by adding "more layers" in effect, even though their computation is skipped by PLOT search during inference. These layers are useful to capture the basic linguistic information, thereby reducing the burden on subsequent layers. Moreover, our method can further reduce the latency with only a slight loss in accuracy. SkipBERT$_{6+2}$ which performs only two-layer transformer calculations maintains accuracy comparable to 4-layer models.

For the 6-layer track, TinyBERT and BERT-EMD both achieve performance comparable to the teacher model. However, SkipBERT$_{6+6}$ also shows competitive results, especially for the challenging CoLA task (predicting linguistic acceptability judgments), on which previous methods do not work well. The local transformer layers of SkipBERT can effectively capture the short-distance grammatical knowledge, e.g. subject-verb-object word order and verbal argument structure, etc., which is considered crucial to CoLA (Warstadt et al., 2019).

The early exit mechanism, tagged by "w/ exit" in Table 1, provides a flexible way to tune the speed-accuracy tradeoff. With early exit enabled, both SkipBERT$_{6+6}$ and SkipBERT$_{6+4}$ achieve further improvements on inference speed with only a minor decrease in accuracy. More exploration will be done in Section 4.5.

Model	EM	F1
BERT$_{12}$	80.9	88.3
BERT$_4$-PKD	70.1	79.5
DistilBERT$_4$	71.8	81.2
TinyBERT$_{4\text{v1}}$	72.7	82.1
TinyBERT$_{4\text{v2}}$†	73.9	82.6
SkipBERT$_{6+4}$	**76.2**	**84.5**
BERT$_6$-PKD	77.1	85.3
DistilBERT$_6$	78.1	86.2
TinyBERT$_{6\text{v1}}$	79.7	87.5
TinyBERT$_{6\text{v2}}$†	81.6	88.8
SkipBERT$_{6+6}$	**82.1**	**89.1**

Table 2: Development results on SQuaD v1.1. Results of BERT-PKD, DistilBERT and TinyBERT$_{\text{v1}}$ are taken from Jiao et al. (2020). † denotes our reproduction.

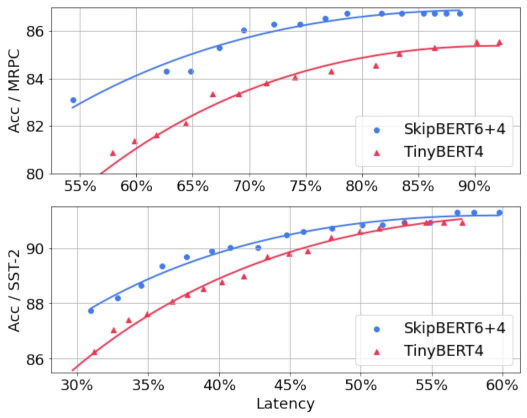


Figure 4: Accuracy-latency curve on the development set of MRPC and SST-2. Both models are equipped with the same early exit mechanism.

4.4 Results on SQuAD

We also investigate the effectiveness of SkipBERT on the reading comprehension task, SQuAD v1.1 (Rajpurkar et al., 2016a). Following previous work, we treat this task as sequence labeling and predict the possibility of each token as the start or end of answer span. Table 2 shows that SkipBERT outperforms all the baselines with large margins. This experiment shows that our approach also works well for relatively complicated task forms.

4.5 Accuracy-Latency Curve

Here we investigate the compatibility between early exit mechanisms and SkipBERT. We draw the accuracy-latency curve by tuning the early exit threshold. The goal is to enlarge the area under the curve – so the model can maintain good accuracy when the average exit layer is small. Fig. 4 compares the results of TinyBERT$_4$ v2 and SkipBERT$_{6+4}$, both using the same early exit mech-

Operation	MACs $d = 768$ / 312	Latency $d = 768$ / 312
Retrieve Text Chunks	-	155 µs / 114 µs
- Chunks to IDs	-	31.4 µs
- Retrieve from *mmap**	-	22.8 µs / 18.4 µs
- Send to GPU	-	101 µs / 64.1 µs
Aggregate Text Chunks	-	48.4 µs / 42.7 µs
Each Transformer Layer	**906M / 146M**	**1.22 ms / 1.18 ms**
Each Classifier	591K / 98K	116 µs / 112 µs

Table 3: Breakdown of computation and average latency of text with 128 tokens (including padding tokens). * varies depending on the input and cache memory, and we report the average value here.

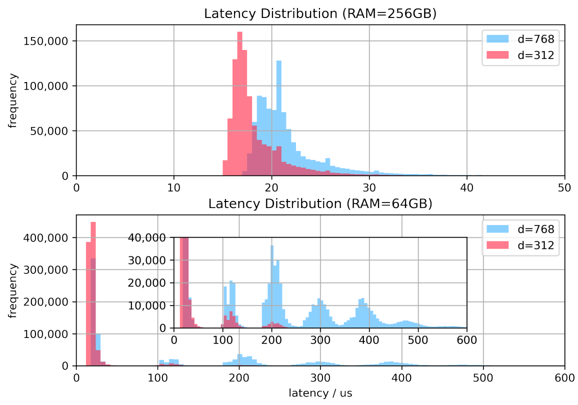


Figure 5: Latency distribution of reading from *mmap*. We evaluate on GLUE training sets, which include text from various sources, to simulate realistic workloads. We zoomed in on the long tail with RAM size of 64GB.

anism. We observe that SkipBERT consistently outperforms TinyBERT on both MRPC and SST-2. Specifically, the curve of SkipBERT is "flatter" than that of TinyBERT, which indicates that even if SkipBERT is forced to exit at a relatively shallow layer, it can still maintain a desirable accuracy. Compared with baselines, our approach starts inference based on PLOT search results rather than from scratch, so even at a lower layer, the representation is well-learned for making predictions.

4.6 Breakdown of Computation and Latency

Table 3 presents the breakdown of computation and average latency of SkipBERT. Detailed hardware information can be found at Appendix A. We can observe that the transformer layers account for the majority of inference time.

We note that there may be some variation in the latency of retrieving data from *mmap*, depending on the cache memory managed by the operating system. Fig. 5 presents the latency distribution of retrieving chunks contained in a text sequence with

OOV: PLOT size:	0% -	11.2% 168G	14.5% 59.2G	24.7% 12.1G
$\delta = 0\%$	88.0/91.4	87.0/90.6	86.3/90.1	85.8/89.6
$\delta = 5\%$	88.2/91.4	**88.2/91.7**	88.0/91.3	87.3/91.0
$\delta = 10\%$	**88.2/91.6**	88.0/91.4	**88.0/91.4**	**88.0/91.2**
$\delta = 20\%$	88.0/91.4	87.7/91.2	88.0/91.3	88.0/91.2

Table 4: Influence of space costs on the model accuracy. An OOV here means a tri-gram which can not be found at PLOT. We randomly replace $\delta\%$ of tri-grams with bi-grams during training. We evaluate on MRPC.

128 tokens under different RAM sizes. We perform experiments in Docker containers to limit the RAM size; more results can be found in Appendix G. The upper half of Fig. 5 shows that with enough RAM, the system can directly collect chunk embeddings from RAM, yielding latencies clustered around 20 µs. Meanwhile, with a smaller RAM as shown in the lower half of Fig. 5, most of the latency is still around 20 µs but a small portion of items take several hundred µs due to cache misses. We also observe that the long tail of latency is distributed in several clusters, mainly due to I/O queuing. However, even under heavy I/O load, retrieving data from *mmap* takes less time than the computation of a single transformer layer.

4.7 Space Costs and OOV

The previous sections prioritize accuracy and efficiency by sacrificing space. Reducing the space costs (by dropping less frequent chunks) allows users to use more economical hardware, but it will lead to OOV issues which may compromise accuracy. Here we only count OOV for tri-grams, since OOVs for bi-grams rarely occur (<0.5%) and have little impact on the final performance.

We collect tri-grams on news corpora[5] and training sets of GLUE to construct PLOT.

Table 4 shows results by reducing the space costs. $\delta = 0\%$ means that the model does not see any bi-grams during training. In this case, if the model encounters a tri-gram lookup failure and reverts to bi-grams, the performance will suffer to some extent. When we randomly replace $\delta\%$ of tri-grams with bi-grams during training, the model becomes more robust to OOVs, and can even slightly improve accuracy. We find $\delta = 10\%$ works well for all cases, and thus we use it as the default value.

Generally, our method can maintain the advan-

[5]The monolingual data from WMT 2011, http://www.statmt.org/wmt11/translation-task.html

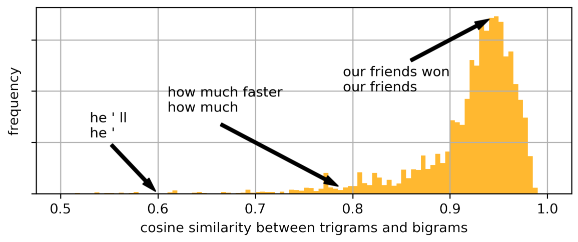


Figure 6: Distribution of cosine similarity between tri-grams and bi-grams.

Model	CoLA	MRPC	MNLI
BERT_4	23.7	85.0/89.6	79.8/79.6
FFN+BERT_4	25.4	85.3/89.9	79.7/79.8
Conv+BERT_4	24.1	84.6/89.6	79.6/79.7
SkipBERT_{2+4}	28.6	86.0/90.0	80.5/81.0
SkipBERT_{4+4}	31.4	86.0/90.2	80.9/81.1
SkipBERT_{6+4}	**32.9**	**86.3/90.4**	80.9/81.1
SkipBERT_{8+4}	32.4	85.0/89.5	**81.0/81.2**

Table 5: Effect of the number/type of skipped layers.

Model	r.f.	CoLA	MRPC	MNLI
BERT_4	-	23.7	85.0/89.6	79.8/79.6
SkipBERT_{6+4}				
w/ 1-gram	1	23.5	85.3/89.5	79.8/79.5
w/ 2-gram	3	29.1	86.3/90.1	80.8/81.0
w/ 3-gram ctr. only	3	32.3	86.0/90.0	80.7/81.1
w/ 3-gram	5	32.9	86.3/90.4	80.9/81.1
w/ 5-gram ctr. only	5	31.9	**87.5/91.2**	**81.2/81.4**
w/ 5-gram	9	**34.5**	86.8/90.7	81.1/81.3

Table 6: Effect of short context. r.f.: receptive field of local layers; ctr. only: only use the embedding of the center token of each chunk; 5-gram results are computed on the fly without using PLOT.

tage even if the OOV rate is at a moderate level. As we will see later at Section 4.8.2, if we only use bi-gram embeddings, i.e. the OOV rate is 100%, our approach is still better than the baseline that does not apply PLOT.

To understand why the backoff strategy, namely to replace tri-grams with bi-grams for OOVs, does not hurt accuracy, we investigate the similarity between them. As shown in Fig. 6, most of them are similar, confirming the feasibility of our backoff strategy; but there is also a long tail where bi-grams cannot well compensate for the missing tri-grams. Fig. 6 also shows some examples with different similarities. Generally, auxiliary tokens that do not contain much meaning by themselves tend to rely more on context. Meanwhile, tokens rich in semantics, e.g. noun phrases, do not vary much in embedding under different ranges of context.

4.8 Ablation Study

We conduct an ablation study in this subsection. We only pre-train SkipBERT on the Wikipedia corpus for 1 epoch for fast validation. We also prepare a generally distilled small BERT_4 (the model architecture is identical to TinyBERT_4) with the same setup and corpus as a baseline. We report the results on the development set.

4.8.1 Tuning the Number of Skipped Layers

Table 5 compares the results with different numbers of local transformer layers. BERT_4 is a baseline that does not employ any skipping mechanism. We can see that all settings that use additional local transformer layers have better performance than BERT_4, indicating the effectiveness of our approach. In general, the performance increases when we gradually enlarge the number of local transformer layers. CoLA benefits most from the local transformer layers due to better modeling of short-distance context. However, when it reaches a certain number of local transformer layers, the improvement becomes minimal. We believe that since each token only has the context of five tokens, too many layers may increase the risk of overfitting, which harms the performance. Thus we adopt 6 layers as our default setting.

We also construct variants that replace local transformer layers with a single-layer FFN or CNN, which are computationally lightweight and thus may not need precomputation. However, their accuracy improvement against BERT_4 is very limited, which shows that even for short-distance context, using a relatively complex and deep network is beneficial to the final performance.

4.8.2 Effect of Short Context

We investigate the effect of short-distance context leveraged in the local transformer layers of SkipBERT. Table 6 presents the comparisons of using different ranges of short-context in local transformer layers. 1-grams are equivalent to conventional word embeddings, and the performance is similar to the baseline. When using 2-grams, SkipBERT obtains notable improvements since each token can now access its direct neighbors in local transformer layers. 3-grams and 5-grams bring consistent improvements to all tasks. Generally, the results are improved when we broaden the receptive field of local transformer layers, showing that more contexts are always beneficial. However, due to its large number, it would be hard to enumerate n-grams with $n > 3$. It might require certain pruning or compression strategies, which we leave as future work.

In addition, we also study the effect of the weighted sum used in chunk aggregation, Eq. (3). We add comparison against a variant that only uses the embedding of the central token of each chunk, denoted "ctr. only". Table 6 shows that the weighted sum brings improvements over all tasks for the 3-gram setting. However, it is not as effective for the 5-gram setting on MRPC and MNLI. We believe using a weighted sum for five chunks may confuse important semantics and thus affect the accuracy; while for 3-gram setting, this problem is not as serious, and using a weighted sum for neighbor chunks can bring more context information to improve the accuracy.

4.8.3 Effect of Distillation Objective

Model	CoLA	MRPC	MNLI
SkipBERT$_{6+4}$	**32.9**	**86.3/90.4**	80.9/81.1
w/o GD-att	29.9	81.9/87.2	**81.3/81.6**
w/o GD-hid	28.7	85.5/90.0	80.9/81.1
w/o TD-att	31.3	85.8/89.6	80.1/80.1
w/o TD-hid	30.5	85.8/89.9	80.4/80.1

Table 7: Effect of different distillation objectives.

We here show the effects of different distillation objectives. We try to eliminate attention-based or hidden state-based distillation. Results in Table 7 indicate that all distillation objectives are helpful both in the general distillation and task-specific distillation process. In general distillation, both attention and hidden states-based distillation are critical to the final performance of relatively small datasets, e.g. CoLA and MRPC. But for large-scale datasets, e.g. MNLI, removing attention based distillation even improves the performance, which may imply that the student model can benefit more from a fine-tuned teacher model as long as the downstream task has enough data.

In the task-specific distillation, the two distillation objectives are marginally helpful for CoLA and MRPC, while acting more importantly for MNLI. The original TinyBERT uses a data augmentation strategy for all tasks during fine-tuning, which significantly enlarges the training set and makes the effect of task-specific distillation more significant.

5 Conclusion

In this paper, we proposed SkipBERT, a straightforward yet effective approach to skip the computation of BERT's shallow layers. We used representations of short text chunks to approximate BERT's shallow representation, and stored them in PLOT for fast retrieval during inference. Empirical results showed that SkipBERT could achieve performance comparable to BERT while significantly reducing inference time. In the future, we would like to leverage discontinuous text chunks to further improve the accuracy and inference speed. We will also try to reduce storage requirements with appropriate pruning and compression strategies.

Acknowledgements

This work was supported by the Key Research and Development Program of Zhejiang Province of China (No. 2021C01009), NSF of China Grant No. 62050099, and the Fundamental Research Funds for the Central Universities.

References

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. In *Proc. of TAC*.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proc. of SemEval*.

Zihan Chen, Hongbo Zhang, Xiaoji Zhang, and Leqi Zhao. 2018. Quora question pairs.

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

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proc. of IWP@IJCNLP*.

Luyu Gao, Zhuyun Dai, and Jamie Callan. 2020. Modularized transfomer-based ranking framework. In *Proc. of EMNLP*.

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

Forrest N Iandola, Albert E Shaw, Ravi Krishna, and Kurt W Keutzer. 2020. Squeezebert: What can computer vision teach nlp about efficient neural networks? *arXiv preprint arXiv:2006.11316*.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does bert learn about the structure of language? In *Proc. of ACL*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. Tinybert: Distilling bert for natural language understanding. In *Proc. of EMNLP: Findings*.

Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. 2019. Shallow-deep networks: Understanding and mitigating network overthinking. In *Proc. of ICML*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proc. of ICLR*.

Hector Levesque, Ernest Davis, and Leora Morgenstern. 2011. The winograd schema challenge. In *Proc. of AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*.

Jianquan Li, Xiaokang Liu, Honghong Zhao, Ruifeng Xu, Min Yang, and Yaohong Jin. 2020. Bert-emd: Many-to-many layer mapping for bert compression with earth mover's distance. In *Proc. of EMNLP*.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and QI JU. 2020. Fastbert: a self-distilling bert with adaptive inference time. In *Proc. of ACL*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint*.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. Technical report, OpenAI.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016a. Squad: 100,000+ questions for machine comprehension of text. In *Proc. of EMNLP*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016b. Squad: 100,000+ questions for machine comprehension of text. In *Proc. of EMNLP*.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in bertology: What we know about how bert works. *TACL*.

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

Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A Smith. 2020. The right tool for the job: Matching model and instance complexities. In *Proc. of ACL*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc. of EMNLP*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model compression. In *Proc. of EMNLP-IJCNLP*.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. Mobilebert: a compact task-agnostic bert for resource-limited devices. In *Proc. of ACL*.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *Proc. of ICPR*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: Amulti-task benchmark and analysis platform for natural language understand. In *Proc. of ICLR*.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Proc. of NeurIPS*.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2019. Neural network acceptability judgments. In *Proc. of ACL*.

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

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. Deebert: Dynamic early exiting for accelerating bert inference. In *Proc. of ACL*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Proc. of NeurIPS*.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. Large batch optimization for deep learning: Training bert in 76 minutes. In *Proc. of ICLR*.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. Bert loses patience: Fast and robust inference with early exit. In *Proc. of NeurIPS*.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proc. of ICCV*.

A Hardware Information

Computation Related We test on *Intel(R) Xeon(R) Gold 6240C* CPU with 24.75M Cache and 2.60 GHz (3.90 GHz maximum). The GPU model is *Tesla V100* with 32GB Graphics RAM.

Data Retrieval Related The server has 384GB RAM. We store PLOT as *mmap* files and read them from SSD. The SSD used is *Toshiba PX04PMC160* 1.6TB with NVMe driver. It contains four caches of Micron's 5ME77 D9QBJ, DDR3L 1600 MHz. Each one has 512MB capacity, and thus the four make up a 2GB cache capacity. We also note that the CPU used in the experiments contains 48 lanes of PCIe 3.0 bandwidth and throughput for demanding I/O-intensive workloads.

We attach below the results of random 4k read benchmark on our SSD with *fio*,[6] and it takes 107.24 µs on average to retrieve 4k data randomly, which is in line with our latency tests.

```
4kQD32read: (groupid=4, jobs=1): err= 0: pid=330145:
     Thu Sep 16 03:12:05 2021
  read: IOPS=297k, BW=1162MiB/s (1218MB/s)(5000MiB
     /4303msec)
    slat (nsec): min=1335, max=89614, avg=1892.79,
     stdev=790.98
    clat (usec): min=24, max=497, avg=105.31, stdev
     =28.29
     lat (usec): min=26, max=499, avg=107.24, stdev
     =28.33
```

B Fine-tuning Details

In this section, we introduce the detailed settings during fine-tuning. We set the maximum sequence length to 128 for the GLUE benchmark. We train 20 epochs with a learning rate of 2e-5. We choose batch sizes from {16, 32}, and β from {0.1, 0.2}.

According to TinyBERT (Jiao et al., 2020), it is useful to first perform the intermediate layer distillation (i.e. with no prediction loss) on the augmented dataset using a fine-tuned teacher model for several epochs. In our experiments, without data augmentation, this strategy is still useful for CoLA, SST-2, MRPC, STS-B, and RTE but not as useful for other tasks. Specifically, we train additional 10 epochs with no prediction loss for CoLA and STS-B, and 1 epoch for SST-2, MRPC, and RTE.

C Model Architecture

Table 8 presents the neural network architecture of SkipBERT$_{6+6}$, SkipBERT$_{6+4}$ and SkipBERT$_{6+2}$, and we compare with BERT$_{12}$ as a reference. The local transformer layers of SkipBERT only appear in the training phase, and we replace them with PLOT in inference-time. We use the same settings of local transformer layers for SkipBERT$_{6+6}$, SkipBERT$_{6+4}$ and SkipBERT$_{6+2}$, since they do not affect inference speed. We reduce the hidden size of SkipBERT$_{6+4}$ so as to be able to compare with recent 4-layer counterparts, and we add a linear layer between local and global transformer layers to match their hidden size. We also instantiate SkipBERT$_{6+2}$ with only 2 global transformer layers to further reduce the latency, where the hidden size setting is the same to SkipBERT$_{6+6}$.

D PLOT Implementation

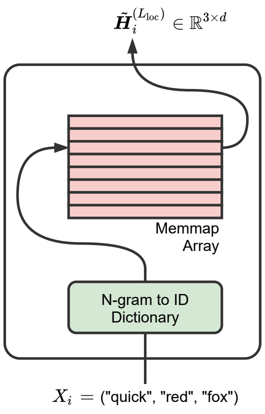

$\tilde{\boldsymbol{H}}_i^{(L_{\text{loc}})} \in \mathbb{R}^{3\times d}$
Memmap Array
N-gram to ID Dictionary
X_i = ("quick", "red", "fox")

Figure 7: Simple Implementation of PLOT

We here provide a simple implementation of PLOT, as shown in Fig. 7. We store chunk representations in a *mmap* file as a huge numpy array[7]. We use python built-in dictionary to map text chunks (n-grams) to their corresponding IDs, which are the offsets to their corresponding representations in the *mmap* file. By such a two-step procedure, we can get the chunk representation without transformer layer computation.

[6] https://fio.readthedocs.io/en/latest/fio_doc.html

[7] https://numpy.org/doc/stable/reference/generated/numpy.memmap.html

			BERT$_{12}$	SkipBERT$_{6+6}$	SkipBERT$_{6+4}$	SkipBERT$_{6+2}$
Params			110M	110M	71M	81M
Global Layers	MHA	d_{out}	768	768	312	768
		n_{head}	12	12	12	12
		d_{in}	768	768	312	768
	FFN	d_{out}	768	768	312	768
		d_{ffn}	3072	3072	1200	3072
		d_{in}	768	768	312	768
			$\times 12$	$\times 6$	$\times 4$	$\times 2$
Chunk Aggregation		$d_{\text{in/out}}$	768	768	312	768
Local Layers	Linear	d_{out}		-	312	-
		d_{in}			768	
	MHA	d_{out}	-	768	768	768
		n_{head}		12	12	12
		d_{in}		768	768	768
	FFN	d_{out}		768	768	768
		d_{ffn}		3072	3072	3072
		d_{in}		768	768	768
				$\times 6$	$\times 6$	$\times 6$
Embedding		d_{emb}	768	768	768	768

Table 8: The detailed model architecture of BERT$_{12}$, SkipBERT$_{6+6}$, SkipBERT$_{6+4}$ and SkipBERT$_{6+2}$. MHA: Multi-Head Attention. FFN: Feed-Forward Networks. d_{emb}, d_{in}, d_{out}, d_{ffn}, and n_{head} denote the embedding size, input hidden size, output hidden size, FFN intermediate size, number of attention heads.

We note that the implementation of PLOT can be extended to a distributed key-value store to support parallel accesses and larger keys (longer n-grams) with reasonable engineering efforts. We welcome community participation to further optimize the structure in the future.

E Distillation Layer Mapping

Fig. 8 presents the layer mapping of different SkipBERT variants.

We use the uniform mapping strategy for attention-based distillation to leverage the heterogeneous attention patterns across different layers. For SkipBERT$_{6+6}$, we match the student's attention scores with the attention scores of every two transformer layers of the teacher (BERT-base). For SkipBERT$_{6+4}$, we match with every three transformer layers of the teacher. And for SkipBERT$_{6+2}$, we match the 3rd and 7th transformer layer of the teacher.

We use the top mapping strategy for hidden states-based distillation. That it, we match the student's hidden states with the top few transformer layers of the teacher. Since the initial sequence representation (outputs of chunk aggregation) are already partially contextualized, SkipBERT can learn from higher transformer layers of the teacher while skipping shallow layers.

F Data and Tasks

We evaluate our approach on the GLUE benchmark. This benchmark consists of a diverse set of 9 NLU tasks:

CoLA Corpus of Linguistic Acceptability, a single-sentence classification task to predict whether a sentence can be accepted a grammatically correct one. (Warstadt et al., 2019)

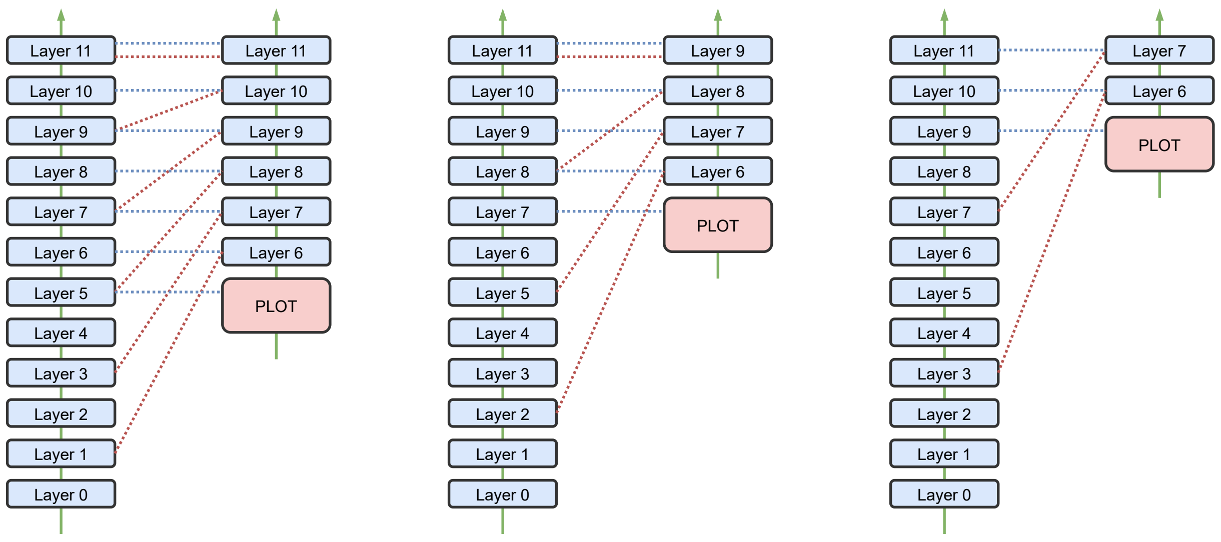
SST-2 Stanford Sentiment Treebank, a single-sentence classification task to predict the sentiment of movie reviews. (Socher et al., 2013)

MRPC Microsoft Research Paraphrase Corpus, a paraphrase identification task to predict whether two sentences are paraphrases of each other. (Dolan and Brockett, 2005)

STS-B Semantic Textual Similarity Benchmark, a regression task to evaluate the similarity of two pieces of texts by a score from 1 to 5. (Cer et al., 2017)

QQP Quora Question Pairs, a bi-sentence classification task to determine whether two questions are semantically equivalent. (Chen et al., 2018)

MNLI Multi-Genre Natural Language Inference, a bi-sentence classification task. Given a pair of premise and hypothesis, the task aims to predict whether the hypothesis is an entailment, contradiction, or neutral with respect to the premise.



(a) SkipBERT$_{6+6}$ (b) SkipBERT$_{6+4}$ (c) SkipBERT$_{6+2}$

Figure 8: Mapping between the teacher (BERT-base) and student. Red dotted lines indicate the layer mapping for attention-based distillation; blue dotted lines indicate the layer mapping for hidden states-based distillation.

(Williams et al., 2018)

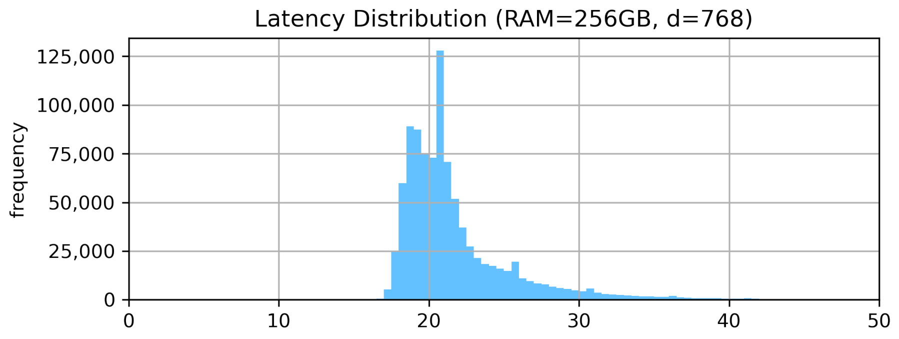
QNLI Question Natural Language Inference, a bi-sentence classification task. Given a pair of question and context, the task aims to predict whether the context contains the answer to the question. (Rajpurkar et al., 2016b)

RTE Recognizing Textual Entailment, a bi-sentence classification task, determining whether the meaning of one sentence is entailed from the other sentence. (Bentivogli et al., 2009)

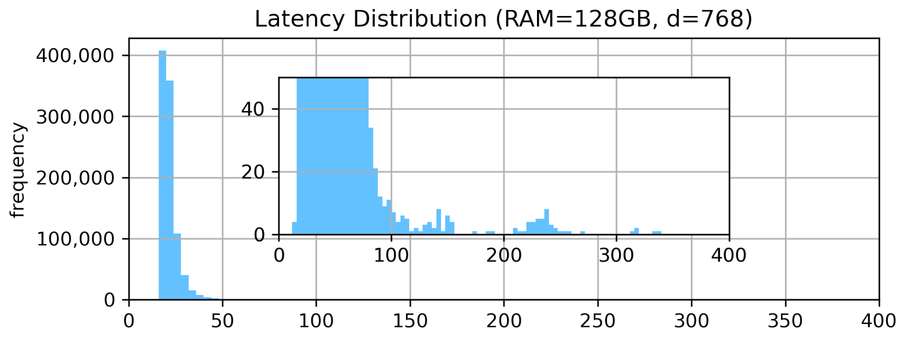
WNLI Winograd Schema Challenge, aiming to predict if the original sentence entails the sentence with the pronoun substituted. (Levesque et al., 2011)

G Latency Distribution of Retrieving Data from Mmap

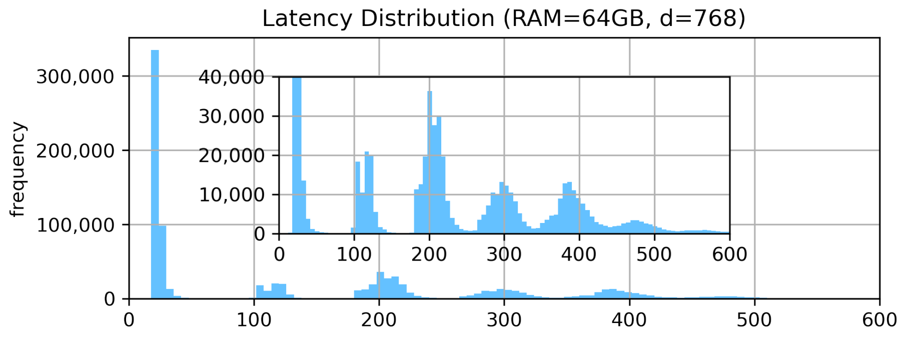
We perform experiments in Docker containers with different RAM limitations, including 256GB, 128GB, 64GB, and 32GB. We evaluate cases with different hidden sizes, and the results are shown in Fig. 9 and Fig. 10.



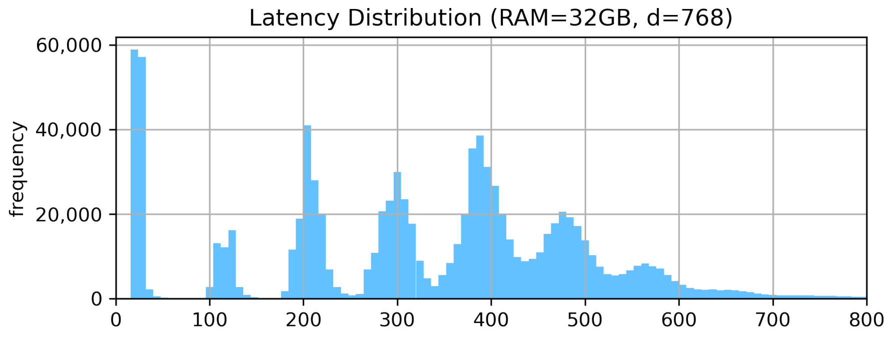
(a) RAM=256GB, *d*=768



(b) RAM=128GB, *d*=768

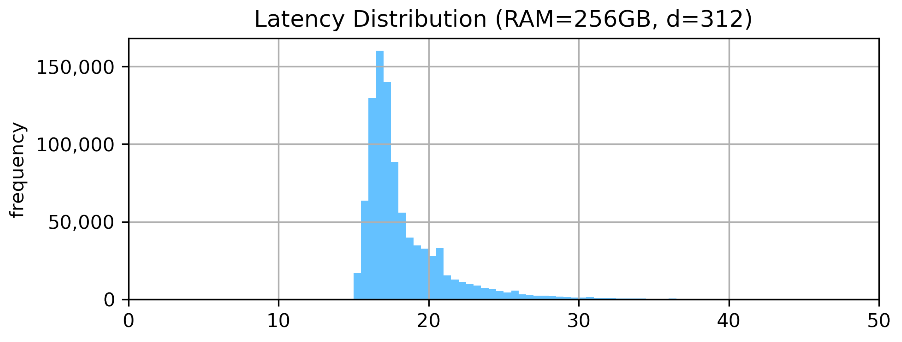


(c) RAM=64GB, *d*=768

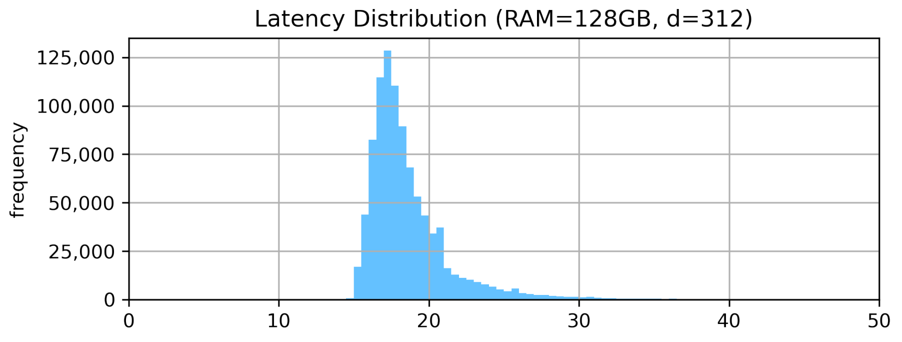


(d) RAM=32GB, *d*=768

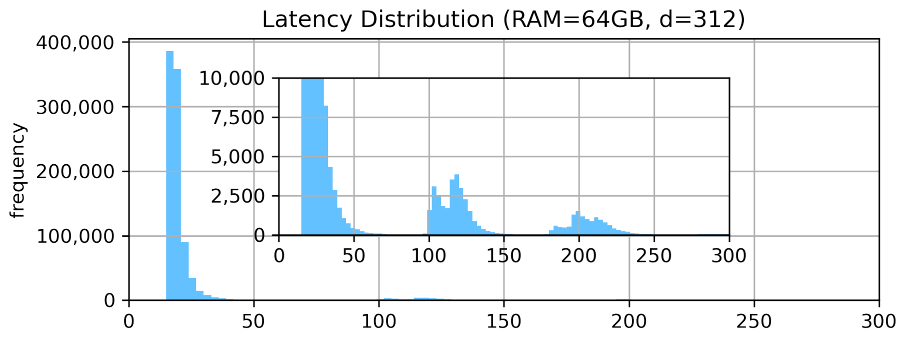
Figure 9: Latency distribution (*d*=768)



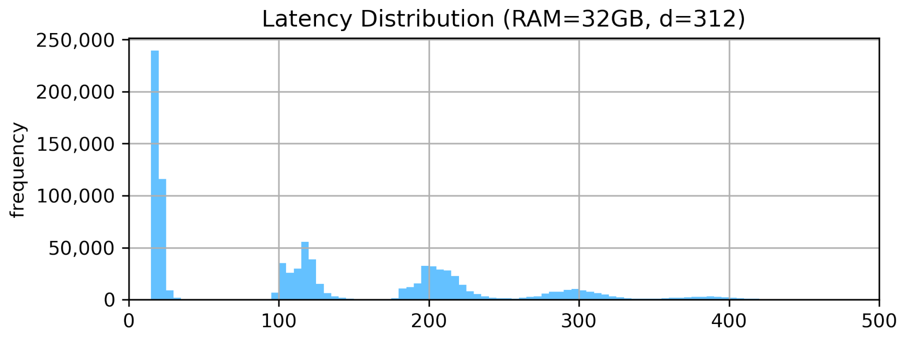
(a) RAM=256GB, *d*=312



(b) RAM=128GB, *d*=312



(c) RAM=64GB, *d*=312



(d) RAM=32GB, *d*=312

Figure 10: Latency distribution (*d*=312)