# BLOCK SKIM TRASFORMER FOR EFFICIENT QUESTION ANSWERING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Transformer based encoder models have achieved promising results on natural language processing (NLP) tasks including question answering (QA). Different from sequence classification or language modeling tasks, hidden states at all position are used for the final classification in QA. However, we do not always need all the context to answer the raised question. Following this idea, we proposed Block Skim Transformer (BST ) to improve and accelerate the processing of transformer QA models. The key idea of BST is to identify the context that must be further processed and the blocks that could be safely discarded early on during inference. Critically, we learn such information from self attention weights. As a result, the model hidden states are pruned at sequence dimension, achieving significant inference speedup. We also show that such extra training optimization objection also improves model accuracy. As an plugin to the transformer based QA models, BST is compatible to other model compression methods without changing existing network architectures. BST improves QA models's accuracies on different datasets and achieves $1.6\times$ speedup on $\text{BERT}_{\text{large}}$ model.

## 1 INTRODUCTION

With the rapid development of neural network in NLP tasks, the Transformer (Vaswani et al., 2017) that uses multi-head attention (MHA) mechanism is a recent huge leap (Goldberg, 2016). It has become a standard building block of recent NLP models. The Transformer-based BERT (Devlin et al., 2018) model further advances the model accuracy by introducing self-supervised pre-training and has reached the state-of-the-art accuracy on many NLP tasks.

One of the most challenging tasks in NLP is question answering (QA) (Huang et al., 2020). Our key insight is that when human beings are answering a question with a passage as a context, they do *not* spend the same level of comprehension for each of the sentences equally across the paragraph. Most of the contents are quickly skimmed over with little attention on it. However, in the Transformer architecture all tokens go through a same amount of computation, which suggests that we can take advantage of that by discarding many of the tokens in early layers of the Transformer. This redundant nature of transformer induces high execution overhead on the input sequence dimension.

To mitigate the inefficiencies in QA tasks, we propose to assign more attention to some blocks that are more likely to contain actual answer while terminating other blocks early during inference. By doing so, we reduce the overhead of processing irrelevant texts and accelerate the model inference. Meanwhile, by feeding the attention mechanism with the knowledge of answer position directly during training, the attention mechanism and QA model's accuracy are improved.

In this paper, we provide the first empirical study on attention feature map to show that an attention map could carry enough information to locate the answer scope. We then propose Block Skim Transformer (BST), a plug-and-play module to the transformer-based models, to accelerate transformer-based models on QA task. By handling the attention weight matrices as feature maps, the CNN-based Block Skim module extracts information from attention mechanism to make skim decision. With the predicted block mask, BST skips irrelevant context blocks, which do not enter subsequent layers' computation. In addition, we devise a new training paradigm that jointly trains Block Skim objective with the native QA objective, where extra optimization signals regarding the question position are given to attention mechanism directly.

In our evaluation, we show BST improves the QA accuracy and F1 score on all the datasets and models we evaluated. Specifically, $\text{BERT}_{\text{large}}$ is accelerated for 1.6× without any accuracy loss and nearly 1.8× with less than 0.5% F1 score degradation.

This paper contributes in the following 3 aspects.

- We for the first time show that an attention map is effective for locating the answer position in the input sequence.
- We propose Block Skim Transformer (BST), which leverages the attention mechanism to improve and accelerate transformer models on QA tasks. The key is to extract information from attention mechanism during processing and intelligently predict what blocks to skim.
- We evaluate BST on several Transformer-based model architectures and QA datasets and demonstrate BST 's efficiency and generality.

## 2 Related Work

**Recurrent Models with Skimming.** The idea to skip or skim irrelevant section or tokens of input sequence has been studied in NLP models, especially recurrent neural networks (RNN) (Rumelhart et al., 1986) and long short-term memory network (LSTM) (Hochreiter & Schmidhuber, 1997). LSTM-Jump (Yu et al., 2017) uses the policy-gradient reinforcement learning method to train a LSTM model that decides how many time steps to jump at each state. They also use hyper-parameters to control the tokens before jump, maximum tokens to jump, and maximum number of jumping. Skim-RNN (Seo et al., 2018) dynamically decides the dimensionality and RNN model size to be used at next time step. In specific, they adopt two "big" and "small" RNN models and select the "small" one for skimming. Structural-Jump-LSTM (Hansen et al., 2018) use two agents to decide whether jump a small step to next token or structurally to next punctuation. Skip-RNN (Campos et al., 2017) learns to skip state updates thus results in reduced computation graph size. The difference of BST to these works are two-fold. Firstly, the previous works make skimming decisions based on the hidden states or embeddings during processing. However, we are the first to analyze and utilize the attention relationship for skimming. Secondly, our work is based on Transformer model (Vaswani et al., 2017), which has outperformed the recurrent type models on most NLP tasks.

**Transformer with Input Reduction.** On contrast to aforementioned recurrent models, in the processing of Transformer-based model, all input sequence tokens are calculated in parallel. As such, skimming can be regarded as reduction on sequence dimension. PoWER-BERT (Goyal et al., 2020) extracts input sequence token-wise during processing based on attention scores to each token. During the fine-tuning process for downstream tasks, Goyal et al. proposes soft-extract layer to train the model jointly. Funnel-Transformer (Dai et al., 2020) proposes a novel pyramid architecture with input sequence length dimension reduced gradually regardless of semantic clues. For tasks requiring full sequence length output, like masked language modeling and extractive question answering, Funnel-Transformer up-sample at the input dimension to recover. Universal Transformer (Dehghani et al., 2018) proposes a dynamic halting mechanism that determines the refinement steps for each token. Different from these works, BST utilizes attention information between question and token pairs and skims the input sequence at the block granularity accordingly.

**Efficient Transformer.** There are also many attempts for designing efficient Transformers (Zhou et al., 2020; Wu et al., 2019; Tay et al., 2020). Well studied model compression methods for Transformer models include pruning (Guo et al., 2020), quantization (Wang & Zhang, 2020), distillation (Sanh et al., 2019), weight sharing. Plenty of works and efforts focus on dedicated efficient attention mechanism considering its quadratic complexity of sequence length (Kitaev et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020). BST is orthogonal to these techniques on the input dimension and therefore is compatible with them. We demonstrate this feasibility with the weight sharing model Albert (Lan et al., 2019) in Sec. 5.

## 3 Problem Formulation: Is Attention Effective for Skim

**Transformer.** Transformer model with multi-head self attention mechanism calculate hidden states for each position as weighted sum of input hidden states. The weight vector is calculated by

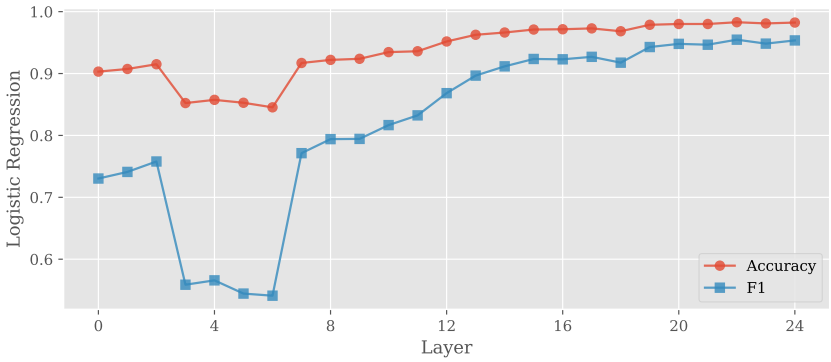


Figure 1: Accuracy and F1 score performance of logistic regression model predicting whether a block contains answer. The logistic regression model is built with attention values between question and target block as input feature.

parameterized linear projection query Q and key K as eq. 1. Given a sequence of input embeddings, the output contextual embedding is composed by the input sequence with different attention at each position.

$$Attention_{(Q,K)} = Softmax(\frac{QK^T}{\sqrt{d_k}}), \tag{1}$$

where $Q, K$ are query and key matrix of input embeddings, $d_k$ is the length of a query or key vector. Multiple parallel groups of such attention weights, also referred as attention heads, make it possible to attend to information at different positions.

QA is one of the ultimate downstream tasks in the NLP. Given a text document and a question about the context, the answer is a contiguous span of the text. To predict the start and end position of the input context given a question, the embedding of each certain token is processed for all transformer layers in the encoder model. In many end-to-end open domain QA systems, information retrieval is the advance procedure at coarse-grained passage or paragraph level. Under the characteristic of extractive QA problem that answer spans are contiguous, our question is that whether we can utilize such idea at fine-grained block granularity during the processing of transformer. Is the attention weights effective for distinguish the answer blocks?

To answer the above question, we build a simple logistic regression model with attention matrix from each layer to predict whether an input sentence block contains the answer. The attention matrices are profiled from a $\text{BERT}_{\text{large}}$ SQuAD QA model and reduced to block level following Equation 2 (Clark et al., 2019). We use the the attention between question and each block as input data for a logistic regression model to predict whether it contains the answer. The result is shown in Fig. 1 with using attention matrices from different layers. Simple logistic regression with hand crafted feature from attention weight achieves quit promising classification accuracy. This suggests that the attending relationship between question and targets is indeed capable for figuring out answer position.

$$BlockAttention(a,b) = \frac{1}{b-a}\sum_{i=a}^{b}\sum_{j=a}^{b} Attention(i,j) \tag{2}$$

# 4 Block Skimming Transformer (BST)

## 4.1 Architecture Overview of BST

We propose the Block Skimming Transformer (BST) model to accelerate the question answering task without degrading the answer accuracy. Unlike the conventional Transform-based model that uses all input tokens throughout the entire layers, our BST model accurately identifies the irrelevant contexts for the question in the early layers, and remove those irrelevant contexts in the following layers. As such, our model reduces the computation requirement and enables the fast question answering.

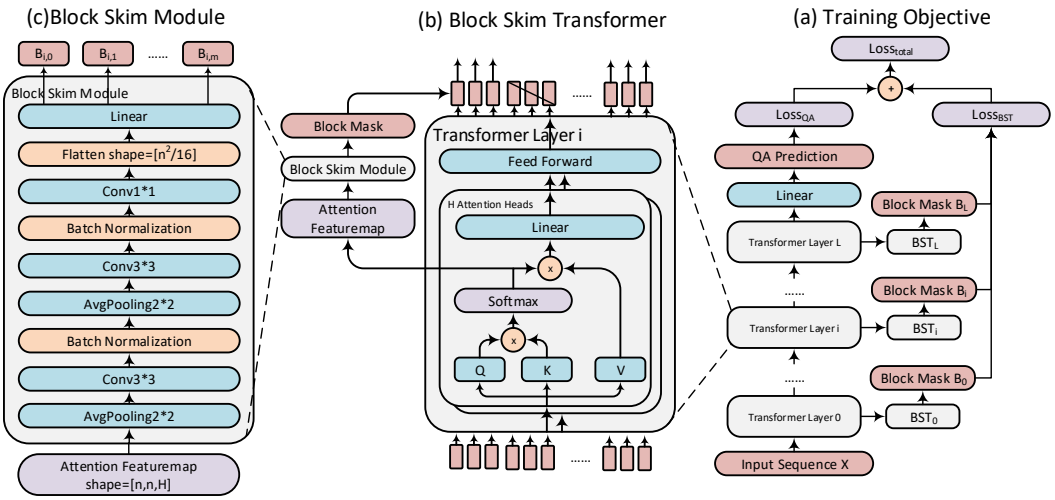


Figure 2: (a) The overview of Block Skimming Transformer. (b) A BST layer is composed of a Transformer layer and a Block Skimming Module. (C) The CNN-based BSM module details.

In Sec. 3, we have shown that it is feasible to identify those tokens that are irrelevant to the question through a hand-crafted feature using the attentions relationship among tokens. However, using this approach could significantly hurt the question answering task accuracy as we show later. As such, we propose an end-to-end learnable feature extractor that captures the attention behavior better.

Fig. 2 shows the overall architecture of our BST model, where a layer is composed of a Transformer layer and a learnable Block Skim Module (BSM). The BSM adopts the convolutional neural network for feature extraction. The input is attention matrices of attention heads, which are treated as feature maps of multiple input channels. The output is a block-level mask that corresponds to the relevance of a block of input tokens to the question.

In each BSM module, we use convolution to collect local attending information and use pooling to reduce the size of feature maps. Two $3 \times 3$ convolution and one $1 \times 1$ convolution are connected with pooling operations intersected. For all the convolution operations, ReLU funcition (Hahnloser & Seung, 2001) is used as activation function. To locate the answer context blocks, we use a linear classification layer to calculate the score for each block. Also, two Batch Normalization layers (Ioffe & Szegedy, 2015) are inserted to improve the model accuracy.

Formally, we denote the input sequence of a transformer layer as $X = (x_0, x_1, \ldots, x_n)$. Then the attention matrices of this layer is denoted as $Attention(X)$. Given the attention output of a transformer layer, the $k_{th}$ block prediction result $B$ is represented as $B = BST(Attention(X))$, where BST is the proposed architecture. The main functions of BST is expressed as Eq. 3.

$$BST(Attention) = Linear(Conv_{1\times1}(Conv_{3\times3}(Pool(Conv_{3\times3}(Pool(Attention)))))) \tag{3}$$

### 4.2 Joint Training of QA and Block-Skim Classifiers

There are two types of classifiers in our BST model, where the first is the original QA classifier at the last layer and the second is the block-level relevance classifier at each layer. We jointly train these classifiers so that the training objective is to minimize the sum of all classifiers' losses.

The loss function of each block-level classifier is calculated as the cross entropy loss against the ground truth label whether a block contains answer tokens or not. Equation 4 gives the formal definition. The total loss of the block-level classifier $\mathscr{L}_{BST}$ is the sum of all blocks that only contain passage tokens. The reason is that we only want to throw way blocks with irrelevant passage tokens instead of questions. Blocks that have question tokens or padding tokens are not used in the training

process. To be more detailed, such blocks are pre-processed and dropped during the training process.

$$\mathscr{L}_{BST} = \sum_{b_i \in \{\texttt{passage blocks}\}} CELoss(b_i, y_i)$$
$$y_i = \begin{cases} 1 & \text{, block i has answer tokens} \\ 0 & \text{, block i has no answer tokens} \end{cases} \tag{4}$$

To calculate the final total loss $\mathscr{L}_{total}$, we introduce two hyper-parameters in Equation 5. We first use the hyper-parameter $\alpha$ so that different models and settings could adjust the ratio between the QA loss and block-level relevance classifier loss. We then use the other hyper-parameter $\beta$ to balance the loss from positive and negative relevance blocks because there are typically many more blocks that contain no answer tokens (negative bocks) than the blocks that do contain answer tokens (positive bocks). We explain how to tune those hyper-parameters for different models and settings later.

$$\mathscr{L}_{total} = \mathscr{L}_{QA} + \alpha \sum_{i_{th} \text{ layer}} (\beta \mathscr{L}_{BST}^{i,y=1} + \mathscr{L}_{BST}^{i,y=0}) \tag{5}$$

Although we add the block-level relevance classification loss in the joint training, we do not actually throw away any blocks because it can skip answer blocks and the QA task training becomes unstable. In this sense, the block-level relevance classification loss can be viewed as a regularization method for the QA training as we force attention heads to better distinguish the answer blocks and non-answer blocks. As we show in the experiment, this regularization effect actually leads to accuracy improvement for the QA task.

### 4.3 Using BST for QA

We now describe how to use the BST model to accelerate the QA task. In the the above joint training process, we add the BSM module in every layer. However, we only augment a specific layer with the BSM module during the inference to save computation and avoid heavy changes to the underlying Transformer model. As such, the layer index for augmenting is a hyper-parameter in our model.

Once the BSM-augmented layer is chosen, we split the input sequence by the block granularity, which is another hyper-parameter in our model. The model skips a set of blocks according to the BSM module results for the following layers. It should be noted that the BST training process does not throw away any blocks because if a relevant block with answer tokens is rejected, the training of original QA task is confused and becomes unstable.

In order to maintain the compatibility to the original Transformer model, we forward the skipped blocks directly to the last layer for the QA classifier. With those design features, BST works as an add-on component to the original Transformer model, and is compatible with many Transformer variant models as well as model compression methods. In specific, we will demonstrate that BST works well with Transformer-based Roberta (Liu et al., 2019), which has a different pre-training objective and sequence encoding, and Albert (Lan et al., 2019), which shares weights among layers for a reduced model size.

We provide an analytical model to demonstrate the speedup potential of BST . Suppose that we insert the BSM module at the layer $l$ out of the total $L$ layers, and a portion of $k$ blocks remain for the following layers. The performance speedup is formulated by Equation 6 if we ignore the computation overhead in the BSM module. In fact, the computation of a single BSM module is much smaller than Transformer layers. For example, if $k = 1/3$ blocks remain after $l/L = 1/3$ layers, we achieve a $1.8\times$ ideal speedup. Similarly, if $k = 1/4$ blocks remain after $l/L = 1/4$ layers, the ideal speedup is $2.29\times$.

$$speedup = \frac{L \cdot N \cdot Layer}{l \cdot N \cdot Layer + (L-l) \cdot N \cdot k \cdot Layer} = \frac{L}{l + (1-k)L} \tag{6}$$

## 5 Experiment

### 5.1 Experimental Setup

**Dataset.** We evaluate our method on 5 extractive QA datasets, including SQuAD 1.1 (Rajpurkar et al., 2016), Natural Questions (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017),

| | QA | | | | BST classifier | |
|---|---|---|---|---|---|---|
| | baseline | | BST | | layer 4 | middle layer |
| | EM | F1 | EM | F1 | F1 | F1 |
| Transformer-based models with BST on SQuAD | | | | | | |
| $\text{BERT}_{\text{base}}$ | 81.14 | 88.52 | 81.38 | 88.69 | 80.99 | 89.37 |
| $\text{BERT}_{\text{large}}$ | 86.53 | 92.82 | 87.12 | 93.20 | 82.48 | 93.32 |
| $\text{Roberta}_{\text{base}}$ | 82.02 | 89.65 | 82.40 | 89.91 | 81.88 | 90.35 |
| $\text{Roberta}_{\text{large}}$ | 86.03 | 92.81 | 86.17 | 92.95 | 78.93 | 86.56 |
| $\text{Albert}_{\text{base}}$ | 82.13 | 89.78 | 82.18 | 89.78 | 84.30 | 90.25 |
| $\text{Albert}_{\text{large}}$ | 84.26 | 91.51 | 84.71 | 91.73 | 78.93 | 93.02 |
| Avg. | 83.68 | 90.85 | 83.99 | 91.04 | 81.38 | 90.48 |
| $\text{BERT}_{\text{base}}$ BST on QA datasets | | | | | | |
| SQuAD | 81.14 | 88.52 | 81.38 | 88.69 | 80.99 | 89.37 |
| NewsQA | 51.45 | 66.57 | 52.28 | 67.43 | 53.74 | 67.02 |
| TriviaQA | 68.99 | 73.78 | 69.04 | 73.91 | 93.96 | 96.20 |
| HotpotQA | 58.76 | 75.56 | 59.55 | 75.93 | 69.67 | 75.36 |
| Natural Questions | 67.00 | 79.00 | 67.47 | 79.15 | 89.32 | 90.54 |
| Avg. | 65.47 | 76.69 | 65.94 | 77.02 | 77.54 | 83.70 |

Table 1: BST training results. (Upper) BST evaluated with different Transformer variant models and model sizes. (Lower) BST evaluated with $\text{BERT}_{\text{base}}$ on 5 QA datasets.

NewsQA (Trischler et al., 2016) and HotpotQA (Yang et al., 2018). The diversity of these datasets such as various passage lengths and different document sources lets us evaluate the general applicability of the proposed BST method. We follow the setting of BERT model to use the structure of Transformer encoder and linear classification layer for all the datasets.

**Model.** As we mentioned earlier, BST works as an plugin module to the oracle Transformer model, and therefore applicable to Transformer-based models. To illustrate this point, we apply our method to three different models including BERT, Roberta (Liu et al., 2019) with a different pre-training objective, and Albert (Lan et al., 2019) with parameter sharing layers. For all three models, we evaluate the base setting with 12 heads and 12 layers and the large setting with 24 layers and 16 heads as described in prior work (Devlin et al., 2018).

**Training Setting.** We implement the proposed method based on open sourced library from Wolf et al. (2019). For each baseline model, we use the released pre-trained checkpoints [1]. We follow the training setting used by Devlin et al. (2018) and Liu et al. (2019) to perform the fine-tuning on the above extractive QA datasets. We initialize the learning rate to $3e-5$ for BERT and Roberta and $5e-5$ for Albert with linear learning rate scheduler. For SQuAD dataset, we apply batch size 16 and maximum sequence length 384. And for the other datasets, we apply batch size 32 and maximum sequence length 512. We perform all the experiments reported with random seed 42. We train a baseline model and BST model with the same setting for two epochs and report accuracies from MRQA task benchmark for comparison. We use four V100 GPUs with 32 GB memory for training and report performance speedup on multiple different hardware platforms.

For the following experiments, we use the block size 32 unless explicitly mentioned. We set the hyper-parameter $\beta$ to 4 for all experiments, and $\alpha$ to 1 except Albert. We use the $\alpha$ value of 0.05 for Albert. In the Albert model, the parameters of transformer layers are shared but BST modules in our method do not share parameters. As such, we decrease the loss from BST to prevent model over-fitting and its impact to the QA task paramters.

**Performance Evaluation.** We measure the performance speedup of our method on the 2.20GHz Intel Xeon(R) Silver 4210 CPU with the batch size of one. On the GPU, the batch size of one could not fully utilize the GPU computation resource so the inference time is bottlenecked at the memory. Our evaluation scenario closely resembles prior work (Wu et al., 2019) that targets the

[1] We use pre-trained language model checkpoints released from `https://huggingface.co/models`

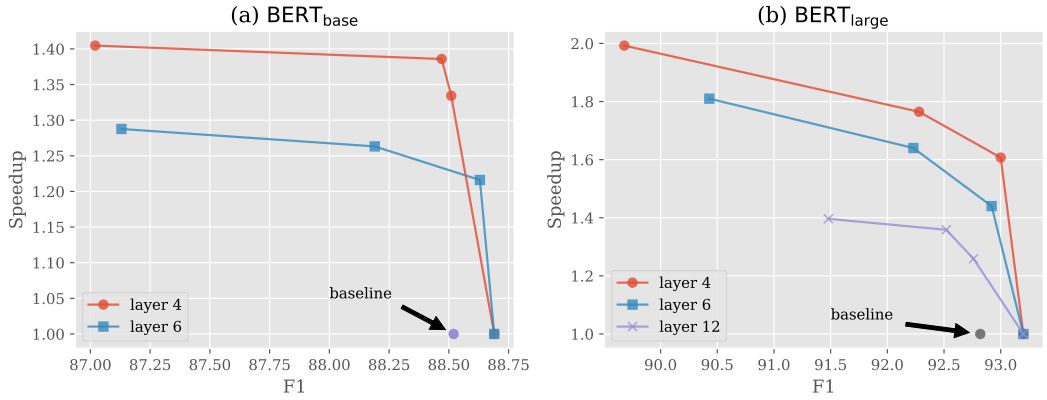


Figure 3: Speedup of BST on BERT$_{base}$ and BERT$_{large}$ augmented at different layers.

mobile application domain. We evaluate BST with different layers and prediction thresholds of BST classifier to explore the trade-off between performance speedup and model accuracy. For example, a lower prediction threshold could lead to more skipped blocks, which means a better performance speedup. On the other hand, it also increases the chance of skipping answer blocks, which hurts the QA task accuracy.

### 5.2 BST as a Regularization Method

We first evaluate BST model as a regularization method to improve the QA task accuracy. Specifically, we compare the accuracy of three baseline models and their BST variants. In their BST versions, the BSM modules only participate the training process, and are removed in the inference task.

The upper half of Tbl. 1 shows the accuracy comparison on SQuAD dataset. By only changing the training process, BST improves the extractive QA accuracy for all baseline models. On average, BST exceeds the baseline by 0.32% in exact match and by 0.19% in F1 score. The accuracy improvement of BST is generally greater on large models. We attribute this to a stronger regularization effect for larger models. The results show the wide applicability of our method to different models.

Tbl. 1 also demonstrates the BST classifier F1 score trained jointly (but not used in this setting) with the baseline models. For simplicity, we only show results of layer 4 and the middle layer (layer 6 for base and 12 for large model). On average, the block-level relevance classifier has a notably high F1 score even at early layers (averaged 81.38%) and even higher scores at the middle layer.

The lower half of Tbl. 1 shows BERT$_{large}$ results on multiple QA datasets. BST outperforms the baseline training objective on all datasets evaluated and exceeds with 0.52% exact match and 0.33% F1 score on average. The results show the wide applicability of our method to different datasets with varying difficulty and complexity. Meanwhile, we also observe a modest correlation between the block-level relevance classifier and the QA task. In other words, the BST classifier tends to be higher on datasets with a higher QA accuracy with the exception of TriviaQA dataset.

### 5.3 QA Task Speedup with BST

We now demonstrate the BST's ability to accelerate the QA task. Fig. 3 demonstrates the performance speedup against F1 score evaluated with BERT$_{base}$ and BERT$_{large}$ model on SQuAD dataset. By tuning the prediction threshold of the BST classifier, we can trade-off between acceleration and accuracy loss. Here we evaluate the BST classifier with $0, 0.5, 0.9, 0.99$ prediction threshold with classifier at layer 4 and the middle layer respectively. On BERT$_{base}$, BST achieves 1.38× speedup with the same accuracy to the baseline. With more aggressive skipping strategy, 1.4× speedup is obtained with minor accuracy loss (less than 1.5%). On BERT$_{large}$, BST achieves 1.6× speedup with minor accuracy improvement and nearly 1.8× speedup with less than 0.5% F1 score degradation.

Generally, the specific layer for inserting the BSM module can be determined by hyper-parameter search according to Eq. 6. As shown in Fig. 3, skipping the irrelevant blocks at layer 4 tends to be

| id | | Update Transformer | Skim Training | BSM | Augment Layer | Block Size | QA | | BST Classifier | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | EM | F1 | layer 4 | layer 6 |
| 1 | Baseline | ✓ | - | - | - | - | 81.14 | 88.52 | - | - |
| 2 | Vanilla BST | ✓ | | ✓ | All | 32 | 81.38 | 88.69 | 80.99 | 89.37 |
| 3 | Freeze Transformer | | | ✓ | All | 32 | 81.14 | 88.52 | 79.60 | 78.41 |
| 4 | Hand-crafted Feature | ✓ | | | All | 32 | 81.21 | 88.59 | 69.41 | 69.51 |
| 5 | Augment One Layer | ✓ | | ✓ | 4 | 32 | 81.20 | 88.63 | 77.26 | - |
| 6 | Skim Traning | ✓ | ✓ | ✓ | 4 | 32 | 79.27 | 86.83 | 84.76 | - |
| 7 | Block Size 1 | ✓ | | ✓ | All | 1 | 81.22 | 88.60 | 75.83 | 85.03 |
| 8 | Block Size 8 | ✓ | | ✓ | All | 8 | 81.25 | 88.63 | 76.12 | 87.04 |
| 9 | Block Size 16 | ✓ | | ✓ | All | 16 | 81.35 | 88.75 | 78.41 | 82.51 |
| 10 | Block Size 64 | ✓ | | ✓ | All | 64 | 81.39 | 88.65 | 87.77 | 90.76 |
| 11 | Block Size 128 | ✓ | | ✓ | All | 128 | 80.90 | 88.33 | 91.36 | 92.79 |

Table 2: Ablation studies of the BST design components of $\text{BERT}_{\text{base}}$ on SQuAD dataset.

better than that at the middle layer, which is layer 6 for $\text{BERT}_{\text{base}}$ and layer 12 for $\text{BERT}_{\text{large}}$. This is because more computation is reduced when skipping at earlier layers and the BST classifier already has quite good prediction accuracy at early layers.

### 5.4 Ablation Study

We compare our BST method with a series ablation of design components to study their individual effect. The experiments are performed based on the same setting as Sec. 5.1. We perform the experiments described in Tbl. 2, which has also the detailed results, and summarize the key finds as follows.

- (3) Instead of joint training as described in Sec. 4.2, we perform a two-step training. We first perform the fine-tuning for the QA task. We then perform the BSM module training with the baseline QA model frozen. In other words, we only use the BST objective and only update the weights in the BSM modules. Therefore, the QA accuracy remains the same as the baseline model, which is lower than the joint training (id 3). Meanwhile, the BST classifier also has a lower accuracy than the joint training especially at layer 6.
- (4) Instead of BSM module, we use the hand-crafted feature in Sec. 3. The resulted block-level relevance classification accuracy is considerably lower than our learned BST model.
- (5) Instead of adding BST module to all layers, we only deploy it into one layer. The experiment result shows that it is beneficial for the model to have BST loss added to every layer.
- (6) We skim blocks during the joint QA-BST training process. Because the mis-skimmed blocks may confuse the QA optimization, this training strategy results in considerable accuracy loss.
- (7-11) We evaluate the accuracy with different block sizes. Specifically when the block size is 1, it is equivalent to skim at the token granularity. Our experimental result shows that the accuracy of BST classifier is better when the block size is larger. On the other hand, a larger block size also leads to less number of blocks and therefore the performance speedup becomes limited on the studied datasets. In this end, we choose the block size of 32 as a design sweet spot.

## 6 Conclusion

In this work, we provide a plug-and-play module BST to Transformer and its variants for efficient QA processing. Our empirical study shows that the attention mechanism in the form of a weighted feature map can provide instructive information for locating the answer span. In fact, we find that an attention map can distinguish between answer and other tokens. Leveraging this insight, we propose to learn the attention in a supervised manner. In effect, BST terminates irrelevant blocks at early layers, significantly reducing the computations. In addition, the proposed BST training objective provides attention mechanism with extra learning signal and improves QA accuracy on all datasets and models we evaluated. With the use of BST module, such distinction is strengthened in a supervised fashion. This idea may be also applicable to other tasks and architectures.

## REFERENCES

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Victor Campos, Brendan Jou, Xavier Giró-i-Nieto, Jordi Torres, and Shih-Fu Chang. Skip RNN: learning to skip state updates in recurrent neural networks. *CoRR*, abs/1708.06834, 2017. URL `http://arxiv.org/abs/1708.06834`.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 276–286, 2019.

Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V Le. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. *arXiv preprint arXiv:2006.03236*, 2020.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

Saurabh Goyal, Anamitra Roy Choudhary, Venkatesan Chakaravarthy, Saurabh ManishRaje, Yogish Sabharwal, and Ashish Verma. Power-bert: Accelerating bert inference for classification tasks. *arXiv preprint arXiv:2001.08950*, 2020.

Cong Guo, Bo Yang Hsueh, Jingwen Leng, Yuxian Qiu, Yue Guan, Zehuan Wang, Xiaoying Jia, Xipeng Li, Minyi Guo, and Yuhao Zhu. Accelerating sparse dnn models without hardware-support via tile-wise sparsity. *arXiv preprint arXiv:2008.13006*, 2020.

Richard HR Hahnloser and H Sebastian Seung. Permitted and forbidden sets in symmetric threshold-linear networks. In *Advances in neural information processing systems*, pp. 217–223, 2001.

Christian Hansen, Casper Hansen, Stephen Alstrup, Jakob Grue Simonsen, and Christina Lioma. Neural speed reading with structural-jump-lstm. In *International Conference on Learning Representations*, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Zhen Huang, Shiyi Xu, Minghao Hu, Xinyi Wang, Jinyan Qiu, Yongquan Fu, Yuncai Zhao, Yuxing Peng, and Changjian Wang. Recent trends in deep learning based open-domain textual question answering systems. *IEEE Access*, 8:94341–94356, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, 2017.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2019.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2019.

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

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

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

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

Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. Neural speed reading via skim-rnn. In *International Conference on Learning Representations*, 2018.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *arXiv e-prints*, pp. arXiv–2009, 2020.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. Newsqa: A machine comprehension dataset. 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Chunpei Wang and Xiaowang Zhang. Q-bert: A bert-based framework for computing sparql similarity in natural language. In *Companion Proceedings of the Web Conference 2020*, pp. 65–66, 2020.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. Lite transformer with long-short range attention. In *International Conference on Learning Representations*, 2019.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*, 2018.

Adams Wei Yu, Hongrae Lee, and Quoc Le. Learning to skim text. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1880–1890, 2017.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*, 2020.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit. *arXiv*, pp. arXiv–2006, 2020.