Pyramid-BERT: Reducing Complexity via Successive Core-set based Token Selection

Xin Huang
Amazon AWS, USA
xinxh@amazon.com

Ashish Khetan
Amazon AWS, USA
khetan@amazon.com

Rene Bidart
University of Waterloo, Canada
rbbidart@uwaterloo.ca

Zohar Karnin
Amazon AWS, Israel
zkarnin@amazon.com

Abstract

Transformer-based language models such as BERT (Devlin et al., 2018) have achieved the state-of-the-art performance on various NLP tasks, but are computationally prohibitive. A recent line of works use various heuristics to successively shorten sequence length while transforming tokens through encoders, in tasks such as classification and ranking that require a single token embedding for prediction. We present a novel solution to this problem, called Pyramid-BERT where we replace previously used heuristics with a *core-set* based token selection method justified by theoretical results. The core-set based token selection technique allows us to avoid expensive pre-training, gives a space-efficient fine tuning, and thus makes it suitable to handle longer sequence lengths. We provide extensive experiments establishing advantages of pyramid BERT over several baselines and existing works on the GLUE benchmarks and Long Range Arena (Tay et al., 2020) datasets.

1 Introduction

Transformers (Vaswani et al., 2017) have gradually become a key component for many state-of-the-art natural language representation models. A recent Transformer based model BERT (Devlin et al., 2018), and its variations, achieved the state-of-the-art results on various natural language processing tasks, including machine translation (Wang et al., 2019a; Liu et al., 2020), question-answering (Devlin et al., 2018; Yang et al., 2019), text classification (Goyal et al., 2020; Xu et al., 2019), semantic role labeling (Strubell et al., 2018), and so on. However, it takes substantial computational resources to pre-train, fine-tune, or infer such models. The complexity of Transformers is mainly due to a pipeline of encoders, each of which contains a multi-head self-attention layer. The self-attention operation scales quadratically with the input sequence length, which is a bottleneck especially for long-sequence data.

Given this challenge, intensive efforts have been focused on compressing and accelerating Transformers to reduce the cost of pre-training and fine-tuning. This work is particularly inspired by the sequence-level NLP tasks such as text classification and ranking. The state-of-the-art Transformer models for these tasks utilize a single embedding from the top encoder layer, such as the CLS token, for prediction. Under such regime, retaining full-length sequence till the last encoder creates unnecessary complexity. We follow a line of works detailed in Section 2 that aim to gradually reduce the sequence length in the pipeline of encoders. On a high level, the existing works have two components: *Select* : a mechanism in charge of reducing the sequence length, either by pruning or pooling, *Train-Select* : a training or fine-tuning procedure dedicated to this mechanism.

Our main contribution is a novel solution for *Select* , motivated from the following observation: As we consider the token representations in top layers, they have increasing redundancy among themselves. We provide a quantitative study demonstrating this in Figure 1. A collection of tokens with high redundancy can intuitively be represented by a small *core-set*, composed of a subset of the tokens. Inspired by this, our solution for *Select* is based on the idea of *core-sets*. We provide a theoretically motivated approach that becomes more effective as the redundancy in the representation increases. This concept separates our work from previous art that provide heuristic techniques for sequence length reduction, or approaches that require expensive training. All of these can in fact be shown to fail in toy examples with high redundancy, e.g. the same representation duplicated multiple times.

For *Train-Select* there is some variety in previous art. Some require a full pre-training procedure, and others require a fine-tuning procedure that works

on the full uncompressed model, meaning one that keeps all tokens until the final encoder layer. The high quality of our solution to *Select* allows us to simply avoid this additional training phase altogether. The result of this simplification is quite impactful. We obtain a speedup and memory reduction not only to the inference but to the training process itself. This makes it possible to use standard hardware (and training scripts) in the training procedure even for very long sequences.

In Section 6 we provide an empirical comparison of our technique with SOTA alternatives and show that it is superior in eliminating redundancy among the tokens, and thus greatly improves the final classification accuracy. In particular, our experiments on the GLUE benchmarks show that our method achieves up to 3-3.5X inference speedup while maintaining an accuracy (mean value over all GLUE datasets) drop of 1.5%, whereas the best baseline suffers a 2.5% drop. We show that our method can be combined with long-text Transformers such as *Big Bird* (Zaheer et al., 2020) and *Performers* (Choromanski et al., 2020) to further alleviate the quadratic space complexity of the self-attention mechanism. Empirical experiments on the Long Range Arena (LRA) (Tay et al., 2020) show that our model achieves a better trade-off between space complexity reduction and accuracy in comparison to its competitors. In particular, when working with Performers and reducing the space complexity by 70%, while baselines suffer a drop in accuracy of 4% or more, our technique actually improves the accuracy as it acts as a regularizer.

Concluding, our paper provides a novel, theoretically justified technique for sequence length reduction, achieving a speedup and memory reduction for both training and inference of transformers, while suffering significantly less in terms of predictive performance when compared to other existing techniques. Our methods are vetted via thorough empirical studies comparing it against SOTA methods and examining its different components.

2 Related Works

There have been a number of interesting attempts, that were aimed at model compression for Transformers, which can be broadly categorized into three directions. The first line of work focus on the redundancy of model parameters. Structure pruning (McCarley, 2019; Michel et al., 2019; Voita et al., 2019; Wang et al., 2019b; Fan et al., 2019; Wu et al., 2021a), which removes coherent groups of weights to preserve the original structure of the network, is one common strategy. In addition, various types of distillation techniques (Sanh et al., 2019; Sun et al., 2019; Jiao et al., 2019; Wang et al., 2020b) have been proposed to remove encoders by training a compact Transformer to reproduce the output of a larger one. Other strategies include weight quantization (Bhandare et al., 2019; Zafrir et al., 2019; Shen et al., 2020; Fan et al., 2020) and weight sharing (Dehghani et al., 2018; Lan et al., 2019). The second line of work focus on reducing the quadratic operation of the self-attention matrices. The quadratic time and space complexity of the attention mechanism with respect to the input sequence serves the main efficiency bottleneck of Transformers, and thus is prohibitively expensive for training long-sequence data. One popular approach is to sparsify the self-attention operation by restricting each token to only attend a subset of tokens (Child et al., 2019; Kitaev et al., 2020; Ye et al., 2019; Qiu et al., 2019; Ainslie et al., 2020; Zaheer et al., 2020; Beltagy et al., 2020). In particular, the most recent *Big Bird* (Zaheer et al., 2020) and *Longformer* (Beltagy et al., 2020) introduce sparse models which scale linearly with the input sequence. Another popular approach (Wang et al., 2020a; Choromanski et al., 2020) is to approximate the self-attention to reduce its quadratic complexity to linear, where the most recent *Performers* (Choromanski et al., 2020) provides an unbiased linear estimation of the attention matrices.

The third line, which our work lies in, focus on the redundancy in maintaining a full-length sequence of token-level representation across all encoders (Dai et al., 2020; Pietruszka et al., 2020; Ye et al., 2021; Kim and Cho, 2020). This work is particularly inspired by the sequence-level NLP tasks such as text classification and ranking, where the state-of-the-art approach utilizes a single embedding from the top encoder layer of a Transformer, such as the CLS token, for prediction. Under such regime, retaining the full-length sequence till the last encoder creates unnecessary complexity. Dai et al. (2020) applied a simplest strided mean pooling to each sliding window of the sequence to gradually compress tokens. The paper proposed a specialized pre-training procedure for this process that achieves a good balance between speedup and accuracy drop. In comparison we aim to avoid the costly pre-training step and focus on a method that

requires only a lightweight fine-tuning. Wu et al. (2021b) define a *centroid attention* operator that given centroids, computes their embeddings via an attention mechanism. These centroids are chosen either as a random subset of the original tokens, or via strided mean pooling. This results in a similar technique to that of Dai et al. (2020), in terms of having a naive sequence length reduction method. Pietruszka et al. (2020) provide a variant of length 2 strided mean pooling where instead of taking the unweighted average of each pair, they provide learnable weights via a differentiable linear function. In our experiment we did not compare our methods with (Wu et al., 2021b; Pietruszka et al., 2020) since both worked on a limited (non-standard) collection of datasets and at the time of writing this paper, did not provide code allowing us to reproduce their result. Since our paper is focused on techniques to select a subset of tokens and these papers use either random sampling or pooling, we do not believe a thorough comparison is required. Ye et al. (2021) proposed a reinforcement-learning based technique to rank the tokens, thereby allowing it to remove the least important ones. This RL policy must be trained in an expensive process that requires the full network structure. Since we focus on methods to improve both training and inference, we do not include it in our experiments. Vision Transformers (Pan et al., 2021; Heo et al., 2021), which apply various types of pooling techniques to reduce input length, are specially designed for image data and thus non-trivial to compare with our method. Early exiting approaches (Xin et al., 2020; Zhou et al., 2020), which allow samples to exit early based on redundancy, are orthogonal to our technique. Goyal et al. (2020) developed an attention based mechanism to progressively eliminate tokens in the intermediate encoders in the fine-tuning, while maintaining the classification accuracy.

3 Pyramid BERT

Background. A Transformer model, e.g. BERT, takes a sequence of tokens as input for each input sentence. The sequence of tokens consists of a CLS token followed by the tokens generated by tokenizing the input sentence. For batch processing, an appropriate sequence length N is chosen, and shorter input sentences are padded to achieve an uniform length N. The embedding layer E embeds each token into a vector of real numbers of a fixed dimension. For each input sentence, the token embeddings are transformed through a pipeline of encoders and the self-attention mechanism. Each encoder takes all the N embeddings as input, and outputs updated N embeddings of the same dimension. The time and space complexity of the self-attention scales quadratically with the input sequence length N.

Motivation. The state-of-the-art BERT utilizes only the CLS token from the top encoder layer for tasks such as classification and ranking. A natural question is: do we need to propagate all the token embeddings through the entire pipeline of encoders when only the top layer CLS embedding is used for prediction? In general, yes, since the self-attention transforms all the embeddings together, updating each one by capturing information from all the others. However, if two or more tokens are exact duplicates of each other, ignoring the positional embedding, then one can easily remove the duplicates from the input and modify the self-attention appropriately to get the same CLS embedding at the top layer, and hence the same prediction. This would reduce the number of FLOPs carried out in each self-attention layer.

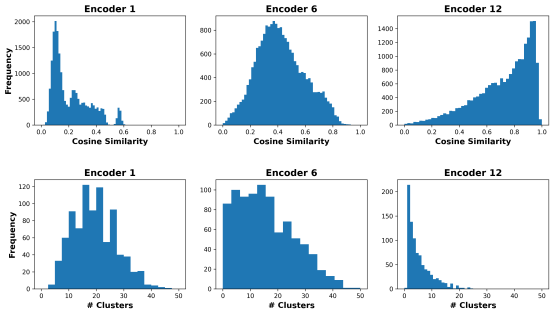

Figure 1: SST-2 *dev* set: Histogram of (a) similarity between CLS and all the other tokens (top row) (b) the number of clusters returned by DBSCAN (bottom row), over all the inputs at encoder 1, 6, and 12.

In general, input sentences do not contain duplicate tokens. However, a preliminary study of token embeddings show that as the embeddings propagate through the pipeline of encoders, they become more similar to the CLS token, Figure 1 (top row). A deeper investigation shows that they also become more similar with each other and form clusters among themselves, Figure 1 (bottom row).

In this work, we exploit these observations, and present pyramid BERT, a novel BERT architecture, that reduces computational and space complexity of fine-tuning and inference of BERT while incurring minimal performance degradation. The pyra-

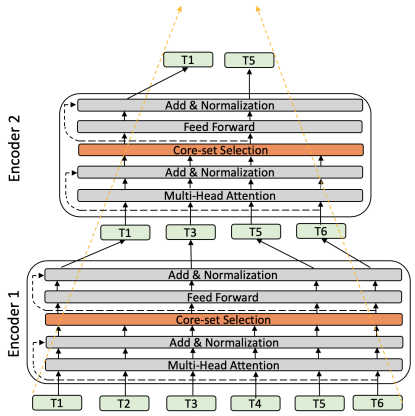

Figure 2: Illustration of Pyramid-BERT.

mid BERT works for all the downstream tasks that only use the top layer CLS token for prediction, such as classification and ranking.

Architecture. The pyramid BERT successively selects subsets of tokens in each encoder layer to propagate them to the next encoder. An illustration of pyramid BERT is shown in Figure 2. It involves two main components over BERT: (a) A sequence-length configuration: a monotonically decreasing sequence $\boldsymbol{\ell} = (\ell_1, \ell_2, \cdots, \ell_L)$ that specifies the number of tokens to retain in each of the L encoders, for all the input examples. (b) A core-set based token selection methodology which in each j-th encoder, given ℓ_{j-1} token embeddings from the $(j-1)$-th encoder selects a subset of it of size ℓ_j to propagate them to the next encoder. Note that the rest of pyramid BERT architecture is same as the BERT, and it has exactly the same number of parameters as the BERT.

In Section 4, we provide a theoretical derivation of the core-set based token selection methodology by minimizing an upper bound of successive token selection loss. To computationally perform the core-set based token selection, we provide a greedy k-center algorithm in Section 5.1. In Section 5.2, we present a simple yet effective approach to select the sequence-length configuration $\boldsymbol{\ell}$ for a desired time/space complexity reduction.

4 Coreset Based Token Selection

Problem Definition. We are interested in a C class classification problem defined over a compact space $\mathcal{X}$ and a label space $\mathcal{Y} = \{1, 2, \cdots, C\}$. We consider a loss function $\mathcal{L}_{\mathbf{w}}(\cdot, \cdot) : \mathcal{X} \times \mathcal{Y} \to \mathcal{R}$ which is parametrized over the hypothesis class $(\mathbf{w})$, the parameters of the transformer network (e.g. BERT), and a set of training data points sampled i.i.d. over the space $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ as $\{\mathbf{x}_i, y_i\}_{i\in[n]} \sim p_{\mathcal{Z}}$ where $[n] = \{1, 2, \cdots, n\}$.

Let $\mathcal{T}$ denote a token selection algorithm. For an example input $\mathbf{x}$, let the input and output embeddings of the token selection algorithm $\mathcal{T}$ at encoder j be $\widetilde{S}_j$ and S_j respectively. The size of the two sets are $|\widetilde{S}_j| = \ell_{j-1}$, and $|S_j| = \ell_j$. In particular, at each encoder j, the algorithm $\mathcal{T}$ selects ℓ_j embeddings of the input set $\widetilde{S}_j$ as the output set S_j, and eliminates the remaining ℓ_{j-1} - ℓ_j embeddings in $\widetilde{S}_j$. Let $\widetilde{\mathcal{S}} = \{\widetilde{S}_j\}_{j\in[L]}$, and $\mathcal{S} = \{S_j\}_{j\in[L]}$. Given the underlying BERT parameters $\mathbf{w}$, the sequence length configuration $\boldsymbol{\ell}$, and the classification loss $\mathcal{L}_{\mathbf{w}}$, pyramid BERT solves the token selection problem by minimizing the population risk as follows:

$$\min_{\{\mathcal{S}: S_j \subseteq \widetilde{S}_j, |S_j| \le \ell_j\}_{j\in[L]}} \mathbb{E}_{\mathbf{x},y\sim p_{\mathcal{Z}}} \left[\mathcal{L}_{\mathbf{w}}(\mathbf{x}, y, \widetilde{\mathcal{S}}, \mathcal{S})\right]. \tag{1}$$

Method. In order to design an optimal token selection algorithm $\mathcal{T}$, we consider the following upper bound of the token selection loss defined in (1):

$$\underbrace{\mathbb{E}_{\mathbf{x},y\sim p_{\mathcal{Z}}}[\mathcal{L}(\mathbf{x}, y, \widetilde{\mathcal{S}}, \mathcal{S})]}_{\text{pyramid BERT population risk}} \le \underbrace{\frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(\mathbf{x}_i, y_i)}_{\text{BERT training error}} + \underbrace{\left|\mathbb{E}_{\mathbf{x},y\sim p_{\mathcal{Z}}}[\mathcal{L}(\mathbf{x}, y, \widetilde{\mathcal{S}}, \mathcal{S})] - \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(\mathbf{x}_i, y_i, \widetilde{\mathcal{S}}_i, \mathcal{S}_i)\right|}_{\text{pyramid BERT generalization error}} + \underbrace{\frac{1}{n}\sum_{i=1}^{n} \left|\mathcal{L}(\mathbf{x}_i, y_i) - \mathcal{L}(\mathbf{x}_i, y_i, \widetilde{\mathcal{S}}_i, \mathcal{S}_i)\right|}_{\text{pyramid BERT token selection loss}}. \tag{2}$$

For ease of notation, we write $\mathcal{L}_{\mathbf{w}}$ as $\mathcal{L}$. The above bound follows immediately from the triangle inequality. For the first two terms in the above bound: the BERT training error is a constant for fixed parameters $\mathbf{w}$, and the generalization error of models like BERT is known to be small (Hao et al., 2019; Jakubovitz et al., 2019). Therefore, we redefine the token selection problem, Equation 1, to minimize the third term, the pyramid BERT token selection loss

$$\frac{1}{n}\sum_{i=1}^{n} \min_{\{\mathcal{S}_i: |(S_{ij})| \le \ell_j\}} \left|\mathcal{L}(\mathbf{x}_i, y_i) - \mathcal{L}(\mathbf{x}_i, y_i, \widetilde{\mathcal{S}}_i, \mathcal{S}_i)\right|. \tag{3}$$

To solve Equation 3, we first optimize a slightly different token selection algorithm $\mathcal{T}^*$ for a model

pyramid* BERT. In pyramid* the embedding sequence length is not reduced across the encoder layers. Let $\widetilde{S}_j^*$ and S_j^* denote the set of input and output embeddings of $\mathcal{T}^*$, where the size of the two sets is equal to the input sequence length N, $|\widetilde{S}_j^*| = |S_j^*| = N$. Given an input set $\widetilde{S}_j^*$ in encoder j, the algorithm $\mathcal{T}^*$ first selects a subset of it of size ℓ_j, which is exactly same as S_j selected by $\mathcal{T}$ in pyramid BERT. Next, instead of eliminating the remaining $N - \ell_j$ embeddings in $\widetilde{S}_j^*$ as is done by $\mathcal{T}$, the $\mathcal{T}^*$ replaces them with their nearest embedding in S_j to form the output set S_j^*. The unique embeddings of the output set S_j^* of pyramid* BERT are exactly same as the embeddings of output set S_j of pyramid BERT, that is $\mathrm{unique}(\mathrm{S_j^*}) = \mathrm{S_j}$. We make the following observation.

Remark 1 *A pyramid* BERT can be reduced to a pyramid BERT with a weighted self attention network. The weighted self attention network weighs attention scores according to the duplicity of the tokens in the embeddings of pyramid* BERT.*

Given the above remark, we optimize the token selection problem for pyramid* BERT. In the theorem below, we give an upper bound for the token selection loss stated in (3) for pyramid* BERT. The proof relies on λ-Lipschitz continuity of the encoder and classification layers. A function f is λ-Lipschitz continuity if, $\|f(x) - f(x')\| \leq \lambda\|x - x'\|$, for all $x, x' \in \mathrm{domain(f)}$.

Theorem 1 *If the classification layer is λ_C-Lipschitz, and for all $j \in [L]$, the encoder E_j is λ_j-Lipschitz continuous for all its parameters, and $\mathcal{T}^*$ is a token selection algorithm such that the unique elements of the output embedding set $\mathrm{unique}(\mathrm{S_j^*})$ is a δ-cover of the input embedding set $\widetilde{S}_j^*$, and the $N - \ell_j$ remaining elements in $\widetilde{S}_j^* \setminus S_j^*$ are replaced in S_j^* by their nearest elements in $\mathrm{unique}(\mathrm{S_j^*})$, then for all i such that $\mathbf{x}_i$ is bounded, the following holds:*

$$\left|\mathcal{L}(\mathbf{x}_i, y_i) - \mathcal{L}(\mathbf{x}_i, y_i, \widetilde{\mathcal{S}}_i^*, \mathcal{S}_i^*)\right| \leq \delta\lambda_C \sum_{j=1}^{L} \left((N - \ell_j) \prod_{a=j}^{L} \lambda_a \right). \quad (4)$$

We visualize the concept of δ-cover in Figure 3. The set of red points (i.e., token embeddings in our case) with radius δ covers the entire set of points. Theorem 1 suggests that we can bound the token selection loss of algorithm $\mathcal{T}^*$ for pyramid* BERT with the δ-cover core-set token selection. The loss goes to zero as the covering radius δ goes to zero. A proof of the theorem is given in Appendix A.2.

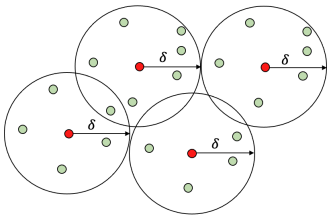

Figure 3: Illustration of δ cover core-set.

From Remark 1, the optimal choice of token selection for pyramid BERT is same as the optimal choice of unique tokens selected in pyramid* BERT, up to weighing of self-attention. However, in numerical experiments we found that fine-tuning pyramid BERT with the core-set based token selection performs better than weighing the self-attention. The δ-cover core-set token selection problem is equivalent to the k-Center problem (also called min-max facility location problem) (Wolf, 2011). We explain how we solve the k-Center problem using a greedy approximation algorithm in §5.1.

5 Pyramid BERT: Algorithm

Given a pre-trained BERT, we create a fine-tuned pyramid BERT as follows. For the core-set selection module shown in Figure 2, we implement a k-Center-greedy-batch-m algorithm, Section 5.1, to approximately select the core-set of embeddings. We fine-tune all the trainable parameters of the pyramid BERT for different choices of sequence-length configurations $\boldsymbol{\ell}$, according to approach given in Section 5.2. We select the optimal configuration $\boldsymbol{\ell}$ satisfying the required inference speed-up or the space complexity reduction. The selected optimal choice of $\boldsymbol{\ell}$ is kept fixed during inference. We note that our practical implementation of pyramid BERT, proposed here, is not exactly the same as the theoretical token selection algorithm analyzed in the previous section. In Appendix A.1, we justify the differences between the two.

5.1 Token Selection Algorithm

The δ-cover core-set problem is equivalent to k-Center problem which is NP-Hard. However, it is possible to obtain a $2 \times \mathrm{OPT}$ solution of k-Center using a greedy approach (Cook et al., 2009). The greedy approach selects the core-set of size k one-by-one, making it un-parallelizable, and hence runs

Algorithm 1: k-Center-greedy-batch-m

Data: Input set $\widetilde{S}$, the number of centers to add per iteration m.
Result: Output set S, with $|S| = k$.
Initialize $S = \{\text{CLS embedding}\}$
while $|S| < k$ **do**
  $M = \{\}$
  while $|M| \leq m$ **do**
    $s = \arg\max_{u \in \widetilde{S} \setminus S} \min_{v \in S} \textit{distance}(u, v)$
    $M = M \cup \{s\}$
  end
  $S = S \cup M$
end
return S

slow on GPUs. For pyramid BERT, we developed a parallelizable version of this greedy approach, Algorithm 1, which selects m centers at a time.

5.2 Sequence-length Configuration

For the sequence-length configuration $\boldsymbol{\ell}$, we restrict the sequences to be exponentially decaying. Specifically, a valid sequence is determined by two parameters. The target pruning ratio $0 < p < 1$, and the index of the layer after which we stop reducing the sequence length $1 \leq i_{\text{prune-upto}} \leq L$. The sequence lengths are defined as

$$l_j = \left\lceil N \cdot p^{\frac{\min(j, i_{\text{prune-upto}})}{i_{\text{prune-upto}}}} \right\rceil, j = 0, 1, \cdots, L, \quad (5)$$

where $j = 0$ corresponds to the input layer. We found that this strategy provides a good balance between the need to reduce the length quickly while retaining information. It involves hyperparameter tuning with two HPs: $p, i_{\text{prune-upto}}$, and allows for an efficient training procedure. In Appendix B we provide a study comparing this restriction to possible alternatives showing its advantages. In addition we discuss how it compares to recent approaches (Goyal et al., 2020; Ye et al., 2021).

6 Experiments

We plug-in our *core-set* based token selection method and the other competitive methods into encoder layers of a backbone Transformer, and after fine-tuning evaluate their performance on a wide range of natural language classification tasks. Specifically, we conduct the evaluations on two popular benchmarks: (1) the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018), and (2) the Long Rang Arena (LRA), a collection of challenging long range context tasks (Tay et al., 2020). For the backbone Transformer, we choose $\text{BERT}_{\text{Base}}$ (Devlin et al., 2018) for the GLUE benchmarks, and two state-of-the-art long-text Transformers *Big Bird* (Zaheer et al., 2020) and *Performers* (Choromanski et al., 2020) for the LRA. For dataset statistics such as the number of classes and input sequence length, and the details of the backbone Transformers, see Appendix C.

For sequence-length configurations, we generate a set $\mathcal{F}$ of 30 sequence-length configurations using Equation 5, the details of which are listed in Table 9 Appendix C. The high level idea is to cover multiple tradeoffs between efficiency and accuracy.

For each token selection method, we show the predictive performance for $1.5X$, $2X$, $3X$, and $3.5X$ speedup. Similarly, we show the predictive performance for 30% and 70% space complexity reductions of the attention layer. The reason we consider the space reduction for the attention layer alone is that its quadratic complexity serves the main bottleneck for long sequences. For details on how to get the performance at different speedup and mathematical formula for computing speedup and space complexity reduction, see Appendix C.

6.1 Baseline Methods

We compare our method with following five methods: (1) *Attention-select* (**Att**): An attention based mechanism from Goyal et al. (2020). (2) *Average-pool* (**Pool**): A strided mean pooling applied to each sliding window of the sequence (Dai et al., 2020).(3) *First-k-select* (**1st**): Selects the first k tokens, a strategy often considered with long documents. (4) *Input-first-k-select* (**1st-I**): Selects the first k tokens, but rather than gradually reducing the sequence length in the encoder layers, performs a single truncation directly on the input. (5) *Random-select* (**Rand**): Selects a random subset of tokens. For all of the methods (including ours), the CLS token is always retained during the token selection.

For our *core-set* based token selection, we try 6 values of $m \in \{1, \lceil 0.1k \rceil, \lceil 0.2k \rceil, \lceil 0.3k \rceil, \lceil 0.4k \rceil, (k-1)\}$ where $k = l_j$, for $j = 1, 2, \cdots, L$. In particular, $m = k - 1$ selects all $k - 1$ tokens (centers) in one iteration given the first selected token is always the CLS token. And $m = 1$ selects one token (center) per iteration which corresponds to the most fine-grained but also the slowest token selection strategy. We denote the strategy of $m = k - 1$ as *Coreset-select-k-1* (**CS-k-1**), and the others as

Coreset-select-x where x $\in \{1, 0.1, 0.2, 0.3, 0.4\}$. We use *Coreset-select-opt* (**CS-opt**) to represent the best value from the *Coreset-select-k-1* and *Coreset-select-x*. In what follows, in tables presenting results we refer to the methods by their shortened (bold) names.

6.2 Implementation

To fairly evaluate our method against the baselines, we use the same set of hyperparameters for all the methods, for a given dataset. For details, see Appendix D. The code for Pyramid-BERT is made available as a supplementary material with the submission. The training and inference jobs are run separately on a NVIDIA Tesla V100 GPU machine and a Intel Xeon Platinum 8000 series CPU machine respectively. All the accuracy and speedup scores are averaged over 20 trials.

6.3 Results on GLUE benchmarks

We first examine the trade-off between accuracy and speedup. The accuracy results for $3X$ and $1.5X$ speedup are summarized in Table 1 and 2 respectively. The results for $3.5X$ and $2X$ speedup are given in the Table 11 and 13 in Appendix E. We observe that as the speedup increases the gap between our *Coreset-select-opt* and its competitors becomes large, where for $3X$ speedup, *Coreset-select-opt* outperforms the second best method *Attention-select* by 1% accuracy in average and beats the standard baselines by 2% or more. The *Average-pool* performs the worst in average across the GLUE benchmarks, specially on the COLA dataset. For detailed justification, see Appendix E. To better understand the performance of *Coreset-select-opt* with different values of m, an ablation study is shown in Section 7. For mild speedup of $1.5X$, we note that all methods (except *Average-pool*) suffer only a small loss in accuracy and our method suffers no loss. A similar situation occurs when viewing the tradeoff between space complexity and accuracy, where we provide results for a memory reduction of 70% and 30% in the Tables 3 and 16 (in § E).

6.4 Results on Long Range Arena

We show results on the following three datasets of LRA benchmark: (1) byte-level text classification using real-world data (IMDB), (2) Pathfinder task (long range spatial dependency problem), and (3) image classification on sequences of pixels converted from CIFAR-10.

Dataset	1st-I	1st	Rand	Pool	Att	CS-k-1	CS-opt	$\text{BERT}_{\text{Base}}$
STS-B	86.4	86.4	86.8	81.6	**87.0**	**87.0**	**87.0**	87.9
MRPC	81.4	80.9	83.2	83.9	84.6	86.2	**86.9**	87.3
SST-2	83.8	84.4	85.6	85.2	86.0	87.3	**89.6**	92.4
QNLI	84.8	84.4	86.4	84.1	86.8	**87.8**	**87.8**	90.9
COLA	49.7	49.7	49.5	3.0	51.1	51.7	**52.8**	53.3
RTE	63.5	63.5	62.1	59.2	63.4	**63.7**	**63.7**	65.8
MNLI_M	77.8	76.7	81.4	75.4	**82.5**	82.4	**82.5**	84.0
MNLI_MM	75.9	75.6	78.7	76.7	**82.7**	82.6	**82.7**	84.6
QQP	80.8	80.4	87.0	79.4	**87.3**	**87.3**	**87.3**	87.5
Mean	76.0	76.1	77.9	69.6	79.0	79.6	**80.0**	81.5

Table 1: GLUE *dev* performance at $3X$ speedup. Here and everywhere else, F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, Matthew's correlations are reported for COLA, and accuracy scores are reported for the other tasks. Each value is averaged over 20 trials. Larger values indicates better performance.

Dataset	1st-I	1st	Rand	Pool	Att	CS-k-1	CS-opt	$\text{BERT}_{\text{Base}}$
STS-B	**87.9**	**87.9**	87.8	87.8	**87.9**	87.7	**87.9**	87.9
MRPC	86.8	86.4	87.2	87.0	87.1	86.9	**87.3**	87.3
SST-2	92.1	91.5	91.9	90.3	92.3	**92.4**	**92.4**	92.4
QNLI	90.8	90.8	90.8	90.2	90.7	**90.9**	**90.9**	90.9
COLA	53.0	52.7	53.1	25.6	53.2	**53.3**	**53.3**	53.3
RTE	65.6	65.2	**65.7**	61.5	**65.7**	65.4	65.6	65.8
MNLI_M	**84.0**	83.8	83.9	80.9	**84.0**	**84.0**	**84.0**	84.0
MNLI_MM	84.1	84.0	83.9	84.0	84.5	**84.6**	**84.6**	84.6
QQP	87.1	86.9	87.4	85.7	87.4	**87.5**	**87.5**	87.5
Mean	81.3	81.0	81.3	76.8	81.4	81.4	**81.5**	81.5

Table 2: GLUE *dev* performance at $1.5X$ speedup.

For baselines, we include *First-k-select* and *Random-select* methods, but fail to include *Attention-select*. *Attention-select* requires a full attention matrix for selecting tokens which is not available in *Big Bird* (Zaheer et al., 2020) and *Performers* (Choromanski et al., 2020). In addition, the Transformers including *Big Bird* and *Performers* in LRA have shallow architectures because of no pre-training: the default number of encoders for text classification, path finder, and image classification datasets are four, four, and one, respectively. Thus, for both baselines and our method, we only reduce sequence length in the *input layer*, which is before the first encoder. For the sequence-length configurations, see Appendix C.2. For *Average-pool*, due to its worst performance on the GLUE benchmarks and the shallow architectures of the models in LRA, we exclude it from the baselines.

Dataset	1st-I	1st	Rand	Pool	Att	CS-k-1	CS-opt	$\text{BERT}_{\text{Base}}$
STS-B	85.3	85.1	85.6	78.7	85.4	86.5	**86.7**	87.9
MRPC	81.3	81.5	83.3	83.1	84.3	86.0	**86.6**	87.3
SST-2	83.3	84.6	84.9	85.1	87.2	87.6	**87.7**	92.4
QNLI	84.6	84.3	85.1	84.0	86.4	86.6	**86.5**	90.9
COLA	49.0	49.0	48.4	0.0	50.9	51.0	**52.3**	53.3
RTE	62.1	62.0	61.8	59.8	62.7	**63.6**	**63.6**	65.8
MNLI_M	76.9	76.3	79.0	75.2	80.5	80.9	**81.0**	84.0
MNLI_MM	74.9	74.5	79.3	76.3	80.7	81.6	**81.8**	84.6
QQP	80.6	80.0	86.6	82.9	87.0	87.2	**87.3**	87.5
Mean	75.3	75.3	77.1	69.5	78.3	79.0	**79.3**	81.5

Table 3: GLUE *dev* performance at 70% space complexity reduction.

Big Bird					
Dataset	1st	Rand	CS-k-1	CS-opt	Trans.-no-prune
CIFAR-10	26.9	39.4	38.6	**43.3**	40.9
PATHFINDER-32	55.6	69.9	69.3	**71.7**	73.5
IMDB (BYTE-LEVEL)	57.9	59.6	59.1	**61.4**	63.8
Mean	46.8	56.3	55.7	**58.8**	59.4
Performers					
CIFAR-10	26.9	41.5	39.8	**45.5**	42.9
PATHFINDER-32	52.4	58.2	61.5	**67.7**	66.2
IMDB (BYTE-LEVEL)	59.9	59.9	59.7	**62.8**	64.3
Mean	46.4	53.2	53.7	**58.7**	57.8

Table 4: LRA *test* set performances at 70% space complexity reduction for *Big Bird* (top) and *Performers* (bottom) as the backbone Transformer. Here and everywhere else, accuracy scores are reported for all three tasks. Each value is averaged over 20 trials. Larger values indicates better performance.

MNLI_M							
Seq.len.config.		CS-1		CS-0.5		CS-k-1	
$i_{\text{prune-upto}}$	p	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup
1	1.5	**78.3**	2.5X	77.9	2.9X	77.9	3.6X
2	0.2	**81.0**	1.9X	80.7	2.3X	80.6	2.6X
3	0.3	**82.9**	1.7X	82.8	1.9X	82.8	2.1X
4	0.4	**83.6**	1.7X	83.5	1.7X	83.4	1.8X
$\text{BERT}_{\text{Base}}$		84.0	–	84.0	–	84.0	–
MNLI_MM							
1	1.5	**79.2**	2.4X	78.3	2.9X	77.9	2.9X
2	0.2	**81.5**	1.7X	81.3	2.2X	81.2	2.2X
3	0.3	**83.5**	1.2X	83.4	1.7X	83.1	1.7X
4	0.4	84.0	1.1X	**84.1**	1.4X	84.0	1.4X
$\text{BERT}_{\text{Base}}$		84.6	–	84.6	–	84.6	–

Table 5: *dev* set performance comparisons for *Coreset-select* with $m \in \{1, \lceil 0.5k \rceil, k-1\}$. A fixed set of four sequence-length configurations are specified based on $i_{\text{prune-upto}}$ and p.

The results of accuracy scores for space complexity reduction at 70% and 30% are presented in Table 4 and Table 18 (in Appendix E), respectively. The *Coreset-select-opt* here represents the *Coreset-select* with $m = 1$ because of its superior performance over other $m \in \{\lceil 0.1k \rceil\}, \lceil 0.2k \rceil, \lceil 0.3k \rceil, \lceil 0.4k \rceil\}$.

We observe a similar pattern as discussed in GLUE benchmark evaluations: at high space complexity reduction 70%, *Coreset-select-opt* significantly outperforms its competitors *First-k-select* and *Random-select* by 12% and 2.5% in average for *Big Bird* (12.3% and 5.5% in average for *Performers*). Moreover, on CIFAR-10, our *Coreset-select-opt* is even better than the *Big Bird* and *Performers* without any sequence reduction with accuracy gain 2.4% and 2.6%, respectively (similarly for *Performers* on PATHFINDER-32). On the other hand, different from the GLUE evaluations, *Coreset-select-k-1* does not show any significant advantages over the baseline methods. Our conjecture is that the input in the LRA datasets contain too many noisy or low level information which is not helpful for predicting the target. For an example, each pixel of an image (CIFAR-10) or a character in the byte-level text classification represents a token as the input. Our *Coreset-select* based strategy with $m = 1$ does the most fine-grained token-level selection than its baselines and thus filter out the noisy information. Note, we do not include accuracy and speedup tables because of insignificant gains observed in speedup due to the shallow architectures of Transformers in LRA.

7 Ablation Studies

We conduct four ablation studies to better study pyramid-BERT: (1) Performance comparisons for *Coreset-select* with different values of m, the number of centers to add per iteration. (2) Justification on the token importance measured by the *Coreset-select*. (3) Comparison of applying *Coreset-select* at both fine-tuning and inference versus at only inference to justify the necessity of fine-tuning in selecting tokens. (4) Exploring the position to plug-in the *Coreset-select* in the encoder.

The result for the first ablation study is presented in Table 5 We can see that *Coreset-select* with $m = 1$ gives the best performance but with the smallest speedup for MNLI-M/MM datasets.

Next, we conduct a study to validate the importance of tokens measured by our *Coreset-select* strategy. We consider a trained BERT that has been fine-tuned on a downstream dataset without any sequence length reduction. During inference, given a encoder j and input example consist of a sequence of tokens, we eliminate the k-th most important token measured by the *core-set* selection method with $1 \leq k \leq L$, and obtain a classification output (prediction label). The classification outputs for all input examples are then compared with those generated by BERT without any sequence length reduction. The comparison between the two classification outputs is measured by the mutual information (Shannon, 2001). Larger mutual information indicates more similarity between the two classification outputs. The importance score is specifically the order of tokens (centers) added by the *core-set* selection method. For a batch of size m tokens that are added at the same time, their importance is determined by the maximum distance between the token and its nearest centers that have already been added. The expectation is that the

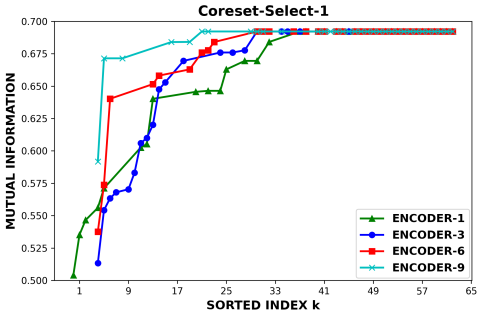

Figure 4: Demonstration of token importance measured by the *Coreset-select-1* based token selection method on SST-2 *dev* set.

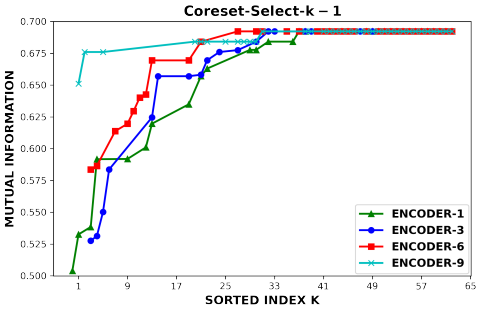

Figure 5: Demonstration of token importance measured by the *Coreset-select-k-1* based token selection method on SST-2 *dev* set.

importance of tokens is negatively correlated with the corresponded mutual information. For an example, reducing the least important token for each input example should make the classification outputs have the least difference from those generated without reducing any token, and thus result in the largest mutual information.

A result on SST-2 dataset is presented in Figure 4. The input sequence length L is set as 64, and thus $k \in [1, 64]$. The encoder index j takes value $\{1, 3, 6, 9\}$. Since the target variable of SST-2 is a relatively balanced binary value, the largest mutual information, which corresponds to no difference between the classification outputs, is $ln(2) \approx 0.69$. For the token selection method, we choose *Coreset-select* with $m = 1$. The pattern shown in the figure aligns with our expectation that the importance of tokens is negatively correlated with the mutual information. Same pattern is observed for *Coreset-select-k-1*. See Figure 5.

Next, we study the difference between applying *Coreset-select* at both fine-tuning and inference, and at only inference. Table 6 justifies the necessary of fine-tuning in sequence length reduction.

	Only Infer	Fine-tune & Infer
QNLI	68.2	**85.9**
SST-2	81.5	**87.2**
COLA	47.9	**50.5**
MRPC	83.5	**85.8**
STS-B	84.6	**86.5**
MNLI-M	75.0	**80.6**
MNLI-MM	74.8	**81.4**

Table 6: Comparison of token selection at inference only versus at both fine-tuning and inference. The particular sequence-length configuration is generated using Equation 5 with $i_{\text{prune-upto}}$ as 3, p as 0.2, and the input sequence length N is 128.

	Middle	End
MRPC	**85.8**	84.0
RTE	**63.4**	60.4
MNLI-M	**80.7**	79.9
MNLI-MM	**81.4**	80.1

Table 7: Comparison between placing the *Coreset-select* method in the middle (right after the attention layer) and at the end of the encoder layer. The particular sequence-length configuration is generated using Equation 5 with $i_{\text{prune-upto}}$ as 3, p as 0.2, and the input sequence length N as 128.

Finally, we study the position to insert the *Coreset-select* method in the encoder. Two choices of position have been considered. The first option is to plug-in the token selection method right after the attention layer and before the feed-forward network, and the second option is to place it at the end of the encoder. The results is shown in Table 7. The experiment shows that placing the token selection method right after the attention layer gives better performance than placing it at the end of encoder.

8 Conclusion

We provide pyramid-BERT, a theoretically justified technique for sequence length reduction, achieving speedup and memory reduction for both training and inference of transformers, while incurring significantly less accuracy drop than competitive methods. However, this technique can be applied only for classification and ranking tasks which use single embedding from the top layer for prediction. Also, our token selection approach requires fine-tuning the network. An interesting future study would be to eliminate the need of fine-tuning.

References

Joshua Ainslie, Santiago Ontañón, Chris Alberti, Philip Pham, Anirudh Ravula, and Sumit Sanghai. 2020. Etc: encoding long and structured data in transformers. *arXiv e-prints*, pages arXiv–2004.

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

Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. 2019. Efficient 8-bit quantization of transformer neural machine language translation model. *arXiv preprint arXiv:1906.00532*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2020. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.

William J Cook, William Cunningham, William Pulleyblank, and A Schrijver. 2009. Combinatorial optimization. *Oberwolfach Reports*, 5(4):2875–2942.

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

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2018. Universal transformers. *arXiv preprint arXiv:1807.03819*.

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

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231.

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

Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. 2020. Training with quantization noise for extreme model compression. *arXiv preprint arXiv:2004.07320*.

Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. 2020. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *International Conference on Machine Learning*, pages 3690–3699. PMLR.

Yaru Hao, Li Dong, Furu Wei, and Ke Xu. 2019. Visualizing and understanding the effectiveness of bert. *arXiv preprint arXiv:1908.05620*.

Byeongho Heo, Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Junsuk Choe, and Seong Joon Oh. 2021. Rethinking spatial dimensions of vision transformers. *arXiv preprint arXiv:2103.16302*.

Daniel Jakubovitz, Raja Giryes, and Miguel RD Rodrigues. 2019. Generalization error in deep learning. In *Compressed Sensing and Its Applications*, pages 153–193. Springer.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*.

Gyuwan Kim and Kyunghyun Cho. 2020. Length-adaptive transformer: Train once with length drop, use anytime with search. *arXiv preprint arXiv:2010.07003*.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

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

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742.

J Scott McCarley. 2019. Pruning a bert-based question answering model. *arXiv preprint arXiv:1910.06360*, 142.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *arXiv preprint arXiv:1905.10650*.

Zizheng Pan, Bohan Zhuang, Jing Liu, Haoyu He, and Jianfei Cai. 2021. Scalable visual transformers with hierarchical pooling. *arXiv preprint arXiv:2103.10619*.

Michał Pietruszka, Łukasz Borchmann, and Filip Gralinski. 2020. Sparsifying transformer models with differentiable representation pooling. *arXiv e-prints*, pages arXiv–2009.

Jiezhong Qiu, Hao Ma, Omer Levy, Scott Wen-tau Yih, Sinong Wang, and Jie Tang. 2019. Blockwise self-attention for long document understanding. *arXiv preprint arXiv:1911.02972*.

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*.

Claude Elwood Shannon. 2001. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. *arXiv preprint arXiv:1804.08199*.

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

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2020. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*.

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

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F Wong, and Lidia S Chao. 2019a. Learning deep transformer models for machine translation. *arXiv preprint arXiv:1906.01787*.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. 2020a. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020b. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *arXiv preprint arXiv:2002.10957*.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2019b. Structured pruning of large language models. *arXiv preprint arXiv:1910.04732*.

Gert W Wolf. 2011. Facility location: concepts, models, algorithms and case studies. series: Contributions to management science: edited by zanjirani farahani, reza and hekmatfar, masoud, heidelberg, germany, physica-verlag, 2009, 549 pp., isbn 978-3-7908-2150-5 (hardprint), 978-3-7908-2151-2 (electronic).

Hongqiu Wu, Hai Zhao, and Min Zhang. 2021a. Not all attention is all you need. *arXiv preprint arXiv:2104.04692*.

Lemeng Wu, Xingchao Liu, and Qiang Liu. 2021b. Centroid transformers: Learning to abstract with attention. *arXiv preprint arXiv:2102.08606*.

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

Hu Xu, Bing Liu, Lei Shu, and Philip S Yu. 2019. Bert post-training for review reading comprehension and aspect-based sentiment analysis. *arXiv preprint arXiv:1904.02232*.

Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. 2019. End-to-end open-domain question answering with bertserini. *arXiv preprint arXiv:1902.01718*.

Deming Ye, Yankai Lin, Yufei Huang, and Maosong Sun. 2021. Tr-bert: Dynamic token reduction for accelerating bert inference. *arXiv preprint arXiv:2105.11618*.

Zihao Ye, Qipeng Guo, Quan Gan, Xipeng Qiu, and Zheng Zhang. 2019. Bp-transformer: Modelling long-range context via binary partitioning. *arXiv preprint arXiv:1911.04070*.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. *arXiv preprint arXiv:1910.06188*.

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

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. Bert loses patience: Fast and robust inference with early exit. *arXiv preprint arXiv:2006.04152*.

A Appendix: Theory

A.1 Difference between the Theoretical and Practical Algorithm

There are two main aspects in which our practical implementation of pyramid BERT token selection differ from its theoretical one, for which we have provided guarantees in Theorem 1. Below we explain the differences and justify them.

1. In the theoretical implementation of pyramid BERT token selection, for which we have provided guarantees in Theorem 1, the original BERT parameters, w, are kept fixed, and the self-attention is weighted according to the duplicity of the tokens in the corresponding pyramid* BERT. Whereas in the practical implementation we remove the weighing of self-attention and offset it by fine-tuning the original BERT parameters w. We make this deviation as we found that fine-tuning BERT not only obviates the need of weighing self-attention but also reduces the performance degradation incurred due to token selection. We note that it is intractable to analyze this deviation in theory as BERT fine-tuning is a non-convex optimization.

2. In the theoretical implementation of pyramid BERT, a δ-cover core-set of tokens is selected. The δ-cover core-set token selection is equivalent to the k-Center problem. The theoretical guarantees assume that we can get the optimal solution of the k-Center problem. However, the k-Center problem is NP-hard and a best known algorithm of it k-Center-greedy gives a $2 \times \mathrm{OPT}$ solution. In our practical implementation, we go a step beyond the greedy approach and propose a parallelizable version of the k-Center-greedy algorithm that takes in an additional hyper-parameter m. The hyper-parameter m sets the level of parallelization and the choice of $m = 1$ reduces it to the original k-Center-greedy. In the numerical experiments, we report the accuracy for the best pyramid BERT by optimizing over the hyper-parameter m.

We note that despite the above mentioned differences of our practical algorithm from the theoretical one, the theoretical guarantees achieved in Theorem 1 do justify the approach of core-set based token selection. The theorem establishes that the token selection loss of the pyramid BERT is bounded by the δ-cover of the core-set. Informally, if the input sentence comprises near-duplicate tokens, in pyramid BERT, the loss incurred by the token selection method goes to zero.

A.2 Proof of Theorem 1

On a high level, theorem follows from (1) the definition of Lipschitz continuity, (2) the definition of the token selection algorithm $\mathcal{T}^*$ of pyramid* BERT, and (3) the fact that the loss function $\mathcal{L}_{\mathbf{w}}$ comprises a sequence of encoder layers stacked on top of each other.

We introduce two new notations. Let $\mathcal{O}_j$ denote the BERT model up to the output of the j-th encoder, and $\mathcal{E}_j$ denote the BERT network up to the input of the j-th encoder. We assume that the token selection algorithm $\mathcal{T}^*$ operates on the embeddings before they are inputted to the encoder. Also, for ease of notation we omit the subscript i denoting the i-th training example from $(\mathbf{x}_i, y_i)$.

Based on the above notations, for BERT we have $\mathcal{E}_j(\mathbf{x}, y) = \mathcal{O}_{j-1}(\mathbf{x}, y)$. For pyramid* BERT, due to the token selection algorithm $\mathcal{T}^*$, for each j-th encoder layer, we have,

$$\left\| \mathcal{E}_j(\mathbf{x}, y, \widetilde{\mathcal{S}}^*, \mathcal{S}^*) - \mathcal{O}_{j-1}(\mathbf{x}, y, \widetilde{\mathcal{S}}^*, \mathcal{S}^*) \right\| \leq \delta(N - \ell_j)\,. \quad (6)$$

The above equation uses the fact that at most $(N - \ell_j)$ tokens are replaced with their corresponding core-set center token, which is at most δ away from them. The Equation (7) follows immediately from the definition of Lipschitz continuity of the classification layer.

$$\left| \mathcal{L}(\mathbf{x}, y) - \mathcal{L}(\mathbf{x}, y, \widetilde{\mathcal{S}}^*, \mathcal{S}^*) \right| \leq \lambda_C \left\| \mathcal{O}_L(\mathbf{x}, y) - \mathcal{O}_L(\mathbf{x}, y, \widetilde{\mathcal{S}}^*, \mathcal{S}^*) \right\|. \quad (7)$$

The Equation (8) follows from the Lipschitz continuity of the L-th encoder. The Equation (9) follows from Equation (6). The Equation (10) follows from the repeated application of Equation (9) over the next encoder layer. The Equation (11) follows from the repeated application of Equation (9) over the subsequent encoder layers, and the fact that $\mathcal{O}_0(\mathbf{x}, y) = \mathcal{O}_0(\mathbf{x}, y, \widetilde{\mathcal{S}}^*, \mathcal{S}^*)$, as input to the first encoder layer is same for BERT and pyramid* BERT.

$$\left\|\mathcal{O}_L(\mathbf{x},y) - \mathcal{O}_L(\mathbf{x},y,\widetilde{\mathcal{S}}^*,\mathcal{S}^*)\right\|$$

$$\leq \lambda_L \left\|\mathcal{E}_L(\mathbf{x},y) - \mathcal{E}_L(\mathbf{x},y,\widetilde{\mathcal{S}}^*,\mathcal{S}^*)\right\| \tag{8}$$

$$\leq \lambda_L \delta(N-\ell_L)$$

$$+ \lambda_L \left\|\mathcal{O}_{L-1}(\mathbf{x},y) - \mathcal{O}_{L-1}(\mathbf{x},y,\widetilde{\mathcal{S}}^*,\mathcal{S}^*)\right\| \tag{9}$$

$$\leq \lambda_L \delta(N-\ell_L) + \lambda_L\lambda_{L-1}\delta(N-\ell_{L-1})$$

$$+ \lambda_L\lambda_{L-1} \left\|\mathcal{O}_{L-2}(\mathbf{x},y) - \mathcal{O}_{L-2}(\mathbf{x},y,\widetilde{\mathcal{S}}^*,\mathcal{S}^*)\right\| \tag{10}$$

$$\leq \lambda_L \delta(N-\ell_L) + \lambda_L\lambda_{L-1}\delta(N-\ell_{L-1})$$

$$+ \cdots + \Big(\prod_{j=0}^{L-1} \lambda_{L-j}\Big)\delta(N-\ell_1) \tag{11}$$

$$= \delta \sum_{j=1}^{L} \left((N-\ell_j)\prod_{i=j}^{L}\lambda_i\right). \tag{12}$$

The theorem follows by combining Equations (7) and (12).

B Appendix: Evaluation on the Sequence-length Generation Function

We note that most recent approaches such as (Goyal et al., 2020; Ye et al., 2021) try to learn a task-dependent sequence-length configuration with a cost of fine-tuning more than twice on the downstream data, where the first fine-tuning trains a full model without any sequence-length reduction, with additional parameters that often requires delicate tuning. This approach does not help the training process and in fact increase its cost whereas our goal is to improve the training procedure. Furthermore, there is still an amount of HP tuning involved in order to find the right ratio of acceleration (or memory reduction) to accuracy. Our technique involves hyperparameter tuning but with two HPs: $p, i_{\text{prune-upto}}$, and allows for an efficient training procedure.

We conduct an experiment to validate the sequence-length generation function, in comparison to a random method that gives configurations for all encoders. Given a dataset we use the retention generation function in Equation 5 and random method to generate a fix number of sequence-length configurations, separately. Then we apply the same *core-set* based method on the dataset with each sequence-length configuration and compute the statistics of accuracy and speedup for our and random method. We repeat the random method for three times. The number of sequence-length configurations is set as 30, and details of those generated by the Equation 5 is presented in Table 9 in Appendix C.1. The results for SST-2 dataset are shown in Figure 6. We can see that the sequence-length configurations generated by our method provide wider searching range of accuracy and speedups than those generated by the random method.

DATASET	# CLASSES	INPUT SEQUENCE LENGTH (N)
STS-B	–	128
MRPC	2	128
SST-2	2	64
QNLI	2	128
COLA	2	64
RTE	2	256
MNLI-M	3	128
MNLI-MM	3	128
QQP	2	128
CIFAR10	10	1024
PATHFINDER32	2	1024
IMDB (BYTE-LEVEL)	2	1000

Table 8: Dataset statistics for GLUE and LRA benchmarks. STS-B is a regression task and thus does not have classes.

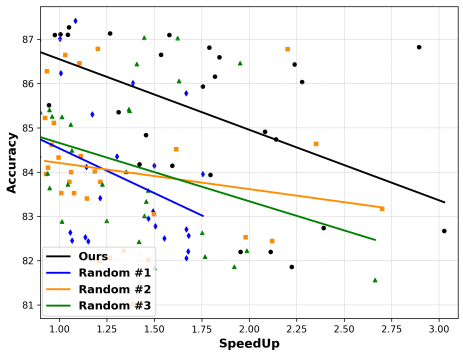

Figure 6: MRPC: Scatter plots of accuracy versus speedup for 30 sequence-length configurations generated by our and random method. Each solid line is from a linear regression model fitted on the scatter points of the corresponded method.

C Appendix: Experiments Setup

Data statistics such as the number of classes and input sequence length are specified in Table 8.

The details of the backbone Transformer used in GLUE(Wang et al., 2018) and LRA (Tay et al., 2020) are presented as follows: For $\text{BERT}_{\text{Base}}$, it was pre-trained on the BooksCorpus and English Wikipedia with $L = 12$ encoders, $A = 12$ self-attention heads per encoder and hidden size $H = 768$. For *Big Bird* (Zaheer et al., 2020) and *Performers* (Choromanski et al., 2020), we follow

the original implementation in LRA (Tay et al., 2020) that models are fine-tuned from scratched on each task without any pre-training.

C.1 GLUE Benchmarks

For experiments on GLUE benchmarks, the sequence-length configurations $\mathcal{F}$ generated by Equation 5 are presented in Table 9. For each dataset and Transformer model with a token selection method, the same set of sequence-length configurations are used.

For each sequence-length configuration in $\mathcal{F}$ a Transformer with a token selection method is applied at both fine-tuning and inference. Next, for each token selection method we select the accuracy scores when there are speedup $1.5X$, $2X$, $3X$, and $3.5X$ over the Transformers without any sequence reduction. To get the accuracy score at the exact speedup X, a linear interpolation is used when necessary. However, for objective evaluation we do not extrapolate the results. For details on how to get accuracy scores at different speedup number, see Figure 8 and 9 in Appendix E. Similarly, we select the accuracy scores when the space complexity reductions for the attention layer are 30% and 70%. The reason why we only focus the space reduction for the attention layer is that its quadratic complexity serves the main efficiency bottleneck for Transformers.

For *Input-first-k-select*, since it does not rely on $\mathcal{F}$ but different truncated input sequence lengths, we specify its configuration as following: For dataest with $N = 256$, we try truncated sequence lengths of $\{240, 224, \cdots, 48, 32, 16\}$. For dataset with $N = 120$, we try truncated sequence lengths of $\{112, 96, 80, 64, 48, 32, 16, 8\}$. And for dataset with $N = 64$, we try truncated sequence lengths of $\{48, 32, 16, 8, 4\}$.

Similarly, *Average-pool* does not rely on $\mathcal{F}$ but the window size and encoder layer(s) to apply the pooling. To get the accuracy scores at various speedups, we try the average pooling with different window sizes of $\{2, 3, 4, 5, 6\}$ on various encoder layer(s). The window size is set equal to the stride size.

The mathematical formulas for speedup and space complexity reduction are presented as below. Consider $\text{BERT}_{\text{Base}}$ as the backbone Transformer for an example, and denote the $\text{BERT}_{\text{Base}}$ with a token selection method as $\text{BERT}_{\text{Token-select}}$. The speedup is computed as $T(\text{BERT}_{\text{Base}}) \,/\, T(\text{BERT}_{\text{Token-select}})$, where $T(\cdot)$ is the time duration in inference. Larger value indicates higher inference speedup for $\text{BERT}_{\text{Token-select}}$ over $\text{BERT}_{\text{Base}}$. The space complexity reduction for the attention layer is computed as $1 - S(\text{BERT}_{\text{Token-select}}) \,/\, S(\text{BERT}_{\text{Base}})$, where $S(\cdot) = \sum_{j=1}^{L} l_j^2 + l_j d, j = 1, 2, \cdots, L$, and l_j and d denotes the number of tokens to select at encoder j and the hidden dimension (dimension of the latent representation), respectively. Larger value indicates higher space complexity reduction for $\text{BERT}_{\text{Token-select}}$ over $\text{BERT}_{\text{Base}}$. For simplicity, the "space complexity reduction" refers to the reduction for the attention layer in the following discussion.

C.2 LRA

We run a set of sequence-length configurations where the number of tokens to select on the input layer is $\{\lceil 0.1 \cdot N \rceil, \lceil 0.2 \cdot N \rceil, \cdots, \lceil 0.9 \cdot N \rceil\}$ and N is the input sequence length. The *Coreset-select-opt* here represents the *core-set* token selection method with $m = 1$ because we observe it gives the best performance than $m \in \{\lceil 0.1 \cdot k \rceil\}, \lceil 0.2 \cdot k \rceil, \lceil 0.3 \cdot k \rceil, \lceil 0.4 \cdot k \rceil\}$. The results of accuracy scores for space complexity reduction at 70% and 30% are presented in Table 4 and 18, respectively. We observe a similar pattern as shown in Section 6.3: at high space complexity reduction 70% in Table 4, *Coreset-select-opt* significantly outperforms its competitors *First-k-select* and *Random-select* by 12% and 2.5% in average for *Big Bird* (12.3% and 5.5% in average for *Performers*). Moreover, on CIFAR-10, our *Coreset-select-opt* is even better than the *Big Bird* and *Performers* without any token selection with accuracy gain 2.4% and 2.6%, respectively. Similarly on PATHFINDER-32, the *Coreset-select-opt* shows 1.5% accuracy gain over the *Performers* without any token selection. On the other hand, different from Section 6.3, *Coreset-select-k-1* does not show any significant advantages over the baseline methods. Our conjecture is that the input in the LRA tasks contain too many noisy or low level information which is not helpful for predicting the target. For an example, each pixel of an image (CIFAR-10) or a character in the byte-level text classification represents a token in the input for the Transformer. Our *core-set* based strategy, especially with $m = 1$, does the most fine-grained token-level selection than its baselines and thus filter out the noisy information. At low space complexity reduction 30%

Input		Output: Retention configurations – Number of tokens to retain at each encoder layer											
The last layer index to apply token selection $i_{\text{prune-upto}}$	The proportion of input tokens p to retain at $i_{\text{prune-upto}}$	layer 1	layer 2	layer 3	layer 4	layer 5	layer 6	layer 7	layer 8	layer 9	layer 10	layer 11	layer 12
2	0.15	49	19	19	19	19	19	19	19	19	19	19	19
3	0.15	68	36	19	19	19	19	19	19	19	19	19	19
4	0.15	79	49	30	19	19	19	19	19	19	19	19	19
3	0.10	59	27	12	12	12	12	12	12	12	12	12	12
4	0.10	71	40	22	12	12	12	12	12	12	12	12	12
3	0.20	74	43	25	25	25	25	25	25	25	25	25	25
4	0.20	85	57	38	25	25	25	25	25	25	25	25	25
3	0.17	70	39	21	21	21	21	21	21	21	21	21	21
3	0.18	72	40	23	23	23	23	23	23	23	23	23	23
3	0.19	73	42	24	24	24	24	24	24	24	24	24	24
3	0.22	77	46	28	28	28	28	28	28	28	28	28	28
3	0.25	80	50	32	32	32	32	32	32	32	32	32	32
1	0.25	32	32	32	32	32	32	32	32	32	32	32	32
2	0.25	64	32	32	32	32	32	32	32	32	32	32	32
3	0.25	80	50	32	32	32	32	32	32	32	32	32	32
5	0.25	97	73	55	42	32	32	32	32	32	32	32	32
9	0.25	109	94	80	69	59	50	43	37	32	32	32	32
11	0.25	112	99	87	77	68	60	52	46	41	36	32	32
1	0.50	64	64	64	64	64	64	64	64	64	64	64	64
2	0.50	90	64	64	64	64	64	64	64	64	64	64	64
3	0.50	101	80	64	64	64	64	64	64	64	64	64	64
5	0.50	111	97	84	73	64	64	64	64	64	64	64	64
9	0.50	118	109	101	94	87	80	74	69	64	64	64	64
11	0.50	120	112	105	99	93	87	82	77	72	68	64	64
1	0.75	96	96	96	96	96	96	96	96	96	96	96	96
2	0.75	110	96	96	96	96	96	96	96	96	96	96	96
3	0.75	116	105	96	96	96	96	96	96	96	96	96	96
7	0.75	122	117	113	108	104	100	96	96	96	96	96	96
9	0.75	123	120	116	112	109	105	102	99	96	96	96	96
11	0.75	124	121	118	115	112	109	106	103	101	98	96	96

Table 9: Sequence-length configurations $\mathcal{F}$ for experiments on GLUE benchmarks. Each sequence-length configuration is determinied by the configuration generation function (Equation 5) which requires two hyperparameters "the last layer index to apply token selection $i_{\text{prune-upto}}$" and "the proportion of input tokens p to retain at $l_{\text{prune-upto}}$". The input sequence N is set as 128.

in Table 18, the advantages for *Coreset-select-opt* over its baselines becomes smaller, which matches our expectations in the mild sequence-length reduction regime. Note, we do not include accuracy and speedup tables because of insignificant gains observed in speedup due to the shallow architectures of Transformers in LRA and the usage of the slowest *coreset* based method with $m = 1$.

D Appendix: Hyperparameters

To fairly evaluate our method, we do not tune any hyperparameters for both GLUE and LRA benchmarks, i.e., the same set of hyperparameters are used for each dataset across every competing method. For GLUE benchmarks, the learning rate and number of epochs are set as $2e^{-5}$ and 3, respectively. The batch size in training is set as 48 for all except MNLI-M/mm and RTE, which are set as 32 and 16, respectively. The batch size in inference is uniformly set as 128. For LRA (Tay et al., 2020), we follow the exact settings for the task-specific hyperparameters and models configurations provided in its official github repository, except reducing the number of encoders in the backbone Transformer (Zaheer et al., 2020; Choromanski et al., 2020) for certain tasks to allow more efficient learning. The details of the hyperparameters and model configurations are presented in Table 10. Note, the learning rate, number of epochs, batch size are all consistent with the default settings in LRA for both *Big Birds* (Zaheer et al., 2020) and *Performers* (Choromanski et al., 2020). For the rest of model configurations, see (Tay et al., 2020).

E Appendix: Experimental Results

First, the GLUE *dev* performance at $3.5X$, $3X$, $2X$, and $1.5X$ speedup are shown in Table 11, 12, 13, and 14. The same conclusion as discussed in Section 6.3 is reached. Similarly, the GLUE *dev* performance at 70% and 30% space complexity reduction are presented in the Table 15 and 16. In particular, the baseline *Average-pool* performs the worst in average across the GLUE benchmarks. This aligns with the proposed method from Dai et al. (2020) that a pre-training step is necessary to make the simple pooling method perform competitive. Especially for COLA dataset, *Average-pool* shows the matthews correlation coefficients at most 25.6 when the speedup is higher than $1.5X$. At 30% space reduction in Table 16, the

	Big Bird			*Performers*		
Hyperparameters	Cifar 10	Path Finder 32	Text Classification	Cifar 10	Path Finder 32	Text Classification
Learning Rate	$5e^{-4}$	$5e^{-4}$	$2.5e^{-2}$	$5e^{-4}$	$3e^{-4}$	$2.5e^{-2}$
# of Epoch	200	200	625	200	200	625
Train & Infer Batch Size	256	512	32	256	512	32
# of Layers	1	4	1	1	4	1
# of Heads	4	2	4	8	8	4
Embedding Dimension	128	64	256	128	32	256
Query/Key/Value Dimension	64	32	256	64	16	256
Feedforward Network Dimension	128	64	1024	128	32	1024
Block Size (specific to *Big Birds*)	8	8	64	–	–	–

Table 10: LRA hyperparameters and model configurations for *Big Bird* (left) and *Performers* (right). Note. The learning rate, number of epoch, and the batch size are all consistent with default settings in LRA (Tay et al., 2020). For the rest of model configurations, please see (Tay et al., 2020).

pooling method shows a relatively high matthews correlation coefficient of 44.6, which is still the worst among all the other methods. We conjecture that the poor performance comes from the following reasons: (1) The pooling method, when applied at a encoder, significantly reduces the sequence length at least by half (as the least sliding window size is 2). Additionally, the COLA dataset has the shortest sequence length in average in the GLUE benchmarks, making the reduced sequence length from the pooling even shorter. This fails the model to learn any useful pattern from the data. For the sequence length distributions of GLUE benchmarks, see Figure 7. (2) The reason the pooling method shows a relatively high matthews correlation coefficient at 30% space reduction in Table 16 is that the pruning scenario corresponds to a "mild" one where the pooling is applied near the top encoder layer (after 8th encoder) and the corresponded speedup is significantly less than $1.5X$. We also observed that the SST-2, which has the second shortest sequence length in average in the GLUE benchmarks, does not suffer similarly as the COLA for the pooling method. We conjecture that this is due to the task of sentiment analysis for SST-2 is much easier than that of judging the grammatical correctness of a sentence for COLA. In summary, the *Average-pool* method merges the consecutive tokens together, according to the window length. However, in English text usually it is not the case that the consecutive tokens are similar and can be merged into one without significant loss in information.

Second, we demonstrate how to generate the accuracy and inference speedup table as shown in Table 1, 2, 11, and 13. A demonstration for dataset SST-2 and MRPC are presented in Figure 8 and 9, respectively. Specifically, for each

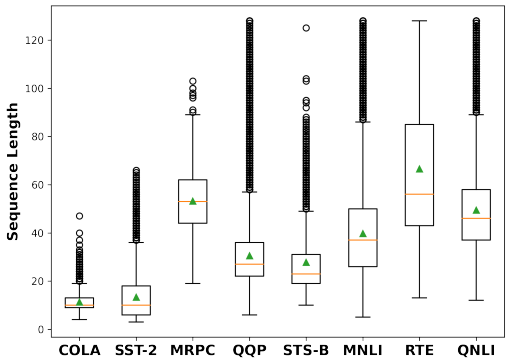

Figure 7: Sequence length distributions of GLUE benchmarks. For each dataset, the triangle mark and solid line represent the mean and median, respectively.

sequence-length configuration shown in Table 9, we fine-tune a Transformer model with a token selection method on a GLUE *train* set. Then we compute accuracy score and inference speedup number on its *dev* set. A scatter plot of accuracy vs. speedup is made where each point corresponds to a sequence-length configuration. Next, among the scatter plot we find a pareto curve where accuracy scores at speedup $1.5X$, $2X$, $3X$, $3.5X$ are obtained. A linear interpolation is applied based on the pareto curve when necessary. However, we do not extrapolate the results for objective evaluation purpose. For *Coreset-select-opt*, given a sequence-length configuration we choose the model that has the best accuracy score for $m \in \{\lceil 0.1 \cdot k \rceil, \lceil 0.2 \cdot k \rceil, \lceil 0.3 \cdot k \rceil, \lceil 0.4 \cdot k \rceil\}$.

Third, for LRA, the results of accuracy scores for space complexity reduction at 70% and 30% are presented in Table 17 and 18. At low space complexity reduction 30%, the advantages for *Coreset-select-opt* over its baselines becomes

smaller, which matches our expectations in the mild sequence-length reduction regime.

F Appendix: Experimental Details for Figure 1 in Section 3

A preliminary study of token embeddings show that as the embeddings propagate through the pipeline of encoders, they become more and more similar to the CLS token, Figure 1 (top row). A deeper investigation shows that they also become more and more similar with each other and form clusters among themselves, Figure 1 (bottom row).

To obtain Figure 1, we examine token embeddings of the SST-2 *dev* set after fine-tuning a $\text{BERT}_{\text{Base}}$ on its *train* set. For Figure 1 (top row), we compute the histogram of cosine similarity between the embedding of CLS and that of all the other tokens in encoder 1, 6, and 12, respectively. For Figure 1 (bottom row), we apply DBSCAN (Ester et al., 1996) to cluster the embeddings of tokens in encoder 1, 6, and 12 respectively. For the hyperparameters of DBSCAN, the maximum distance ϵ of two points in a cluster, the minimum number of points required to form a cluster are set as 0.2 and 1. The distance metric is cosine dissimilarity.

METHOD	STS-B	MRPC	SST-2	QNLI	COLA	RTE	MNLI_M	MNLI_MM	QQP	MEAN
Input-First-K-Select	80.6 ± 0.2	–	–	80.5 ± 0.1	47.9 ± 0.5	–	75.9 ± 0.1	74.5 ± 0.1	–	71.9
First-K-Select	80.3 ± 0.2	–	–	80.8 ± 0.1	47.9 ± 0.5	–	75.5 ± 0.1	74.7 ± 0.1	–	71.8
Random-Select	85.7 ± 0.2	–	–	86.2 ± 0.1	48.9 ± 0.5	–	80.2 ± 0.1	78.1 ± 0.1	–	75.8
Average-Pool	77.6 ± 0.1	–	–	84.0 ± 0.1	0.0 ± 0.0	–	73.3 ± 0.1	74.7 ± 0.0	–	61.9
Attention-Select	85.3 ± 0.1	–	–	86.7 ± 0.1	51.0 ± 0.4	–	81.1 ± 0.1	81.4 ± 0.1	–	77.1
Coreset-Select-CLS (ours)	$\mathbf{86.6} \pm 0.1$	–	–	$\mathbf{87.5} \pm 0.1$	51.1 ± 0.3	–	$\mathbf{82.2} \pm 0.1$	$\mathbf{82.4} \pm 0.1$	–	77.9
Coreset-Select-Opt (ours)	$\mathbf{86.6} \pm 0.1$	–	–	$\mathbf{87.5} \pm 0.1$	$\mathbf{52.7} \pm 0.3$	–	$\mathbf{82.2} \pm 0.1$	$\mathbf{82.4} \pm 0.1$	–	**78.3**
$\text{BERT}_{\text{Base}}$	87.9 ± 0.2	–	–	90.9 ± 0.1	53.3 ± 0.5	–	84.0 ± 0.1	84.6 ± 0.1	–	80.2

Table 11: GLUE *dev* performance at $3.5X$ speedup. Each score with standard deviation is averaged over 20 trials. Larger score indicates better performance. The best score among each token selection method is embolden for each column. Note, missing values indicate that no accuracy score is observed for $3.5X$ speedup.

METHOD	STS-B	MRPC	SST-2	QNLI	COLA	RTE	MNLI-M	MNLI-MM	QQP	MEAN
Input-First-K-Select	86.4 ± 0.1	81.4 ± 0.2	83.8 ± 0.2	84.8 ± 0.0	49.7 ± 0.5	63.5 ± 0.5	77.8 ± 0.1	75.9 ± 0.0	80.8 ± 0.1	76.0
First-K-Select	86.4 ± 0.1	80.9 ± 0.2	84.4 ± 0.2	84.4 ± 0.1	49.7 ± 0.5	63.5 ± 0.5	76.7 ± 0.1	75.6 ± 0.1	80.4 ± 0.0	76.1
Random-Select	86.8 ± 0.1	83.2 ± 0.3	85.6 ± 0.3	86.4 ± 0.1	49.5 ± 0.5	62.1 ± 0.8	81.4 ± 0.1	78.7 ± 0.1	87.0 ± 0.1	77.9
Average-Pool	81.6 ± 0.1	83.9 ± 0.3	85.2 ± 0.2	84.1 ± 0.1	3.0 ± 0.6	59.2 ± 0.5	75.4 ± 0.1	76.7 ± 0.1	79.4 ± 0.0	69.6
Attention-Select	$\mathbf{87.0} \pm 0.1$	84.6 ± 0.3	86.0 ± 0.2	86.8 ± 0.1	51.1 ± 0.4	63.4 ± 0.6	$\mathbf{82.5} \pm 0.1$	$\mathbf{82.7} \pm 0.1$	$\mathbf{87.3} \pm 0.1$	79.0
Coreset-Select-CLS (ours)	$\mathbf{87.0} \pm 0.1$	86.2 ± 0.1	87.3 ± 0.1	$\mathbf{87.8} \pm 0.1$	51.7 ± 0.3	$\mathbf{63.7} \pm 0.5$	82.4 ± 0.1	82.6 ± 0.1	$\mathbf{87.3} \pm 0.0$	79.6
Coreset-Select-Opt (ours)	$\mathbf{87.0} \pm 0.1$	$\mathbf{86.9} \pm 0.2$	$\mathbf{89.6} \pm 0.1$	$\mathbf{87.8} \pm 0.1$	$\mathbf{52.8} \pm 0.3$	$\mathbf{63.7} \pm 0.5$	$\mathbf{82.5} \pm 0.1$	$\mathbf{82.7} \pm 0.1$	$\mathbf{87.3} \pm 0.0$	**80.0**
$\text{BERT}_{\text{Base}}$	87.9 ± 0.2	87.3 ± 0.2	92.4 ± 0.1	90.9 ± 0.1	53.3 ± 0.5	65.8 ± 0.5	84.0 ± 0.1	84.6 ± 0.1	87.5 ± 0.1	81.5

Table 12: GLUE *dev* performance at $3X$ speedup. Each score with standard deviation is averaged over 20 trials. Larger score indicates better performance. The best score among each token selection method is embolden for each column.

METHOD	STS-B	MRPC	SST-2	QNLI	COLA	RTE	MNLI-M	MNLI-MM	QQP	MEAN
Input-First-K-Select	87.8 ± 0.2	82.2 ± 0.2	91.8 ± 0.1	$\mathbf{90.4} \pm 0.1$	51.9 ± 0.5	65.2 ± 0.5	81.9 ± 0.1	82.2 ± 0.1	85.4 ± 0.1	79.9
First-K-Select	87.8 ± 0.2	82.1 ± 0.2	91.2 ± 0.1	$\mathbf{90.4} \pm 0.1$	51.9 ± 0.5	64.9 ± 0.5	81.9 ± 0.1	82.2 ± 0.1	84.9 ± 0.0	79.7
Random-Select	87.8 ± 0.1	86.6 ± 0.3	90.0 ± 0.2	88.9 ± 0.1	52.6 ± 0.5	64.9 ± 0.5	83.6 ± 0.1	80.3 ± 0.1	87.2 ± 0.1	80.2
Average-Pool	87.8 ± 0.2	85.8 ± 0.3	88.6 ± 0.1	86.3 ± 0.1	11.2 ± 0.7	60.2 ± 0.6	76.3 ± 0.0	82.9 ± 0.0	82.1 ± 0.0	73.4
Attention-Select	87.8 ± 0.1	86.8 ± 0.2	91.0 ± 0.1	90.4 ± 0.1	$\mathbf{53.2} \pm 0.5$	$\mathbf{65.7} \pm 0.6$	83.6 ± 0.1	84.1 ± 0.1	87.3 ± 0.1	81.1
Coreset-Select-CLS (ours)	87.6 ± 0.1	86.6 ± 0.2	90.1 ± 0.1	89.4 ± 0.1	$\mathbf{53.2} \pm 0.5$	64.1 ± 0.5	$\mathbf{83.9} \pm 0.1$	$\mathbf{84.3} \pm 0.1$	$\mathbf{87.5} \pm 0.1$	80.7
Coreset-Select-Opt (ours)	$\mathbf{87.9} \pm 0.1$	$\mathbf{87.2} \pm 0.1$	$\mathbf{92.4} \pm 0.1$	89.4 ± 0.1	$\mathbf{53.2} \pm 0.5$	64.9 ± 0.5	$\mathbf{83.9} \pm 0.1$	$\mathbf{84.3} \pm 0.1$	$\mathbf{87.5} \pm 0.0$	**81.2**
$\text{BERT}_{\text{Base}}$	87.9 ± 0.2	87.3 ± 0.2	92.4 ± 0.1	90.9 ± 0.1	53.3 ± 0.5	65.8 ± 0.5	84.0 ± 0.1	84.6 ± 0.1	87.5 ± 0.1	81.5

Table 13: GLUE *dev* performance at $2X$ speedup. Each score with standard deviation is averaged over 20 trials. Larger score indicates better performance. The best score among each token selection method is embolden for each column.

METHOD	STS-B	MRPC	SST-2	QNLI	COLA	RTE	MNLI-M	MNLI-MM	QQP	MEAN
Input-First-K-Select	$\mathbf{87.9} \pm 0.2$	86.8 ± 0.2	92.1 ± 0.1	90.8 ± 0.1	53.0 ± 0.5	65.6 ± 0.5	$\mathbf{84.0} \pm 0.1$	84.1 ± 0.1	87.1 ± 0.1	81.3
First-K-Select	$\mathbf{87.9} \pm 0.2$	86.4 ± 0.2	91.5 ± 0.1	90.8 ± 0.1	52.7 ± 0.4	65.2 ± 0.5	83.8 ± 0.1	84.0 ± 0.1	86.9 ± 0.0	81.0
Random-Select	87.8 ± 0.1	87.2 ± 0.2	91.9 ± 0.2	90.8 ± 0.1	53.1 ± 0.4	$\mathbf{65.7} \pm 0.5$	83.9 ± 0.1	83.9 ± 0.1	87.4 ± 0.1	81.3
Average-Pool	87.8 ± 0.1	87.0 ± 0.2	90.3 ± 0.1	90.2 ± 0.1	25.6 ± 0.7	61.5 ± 0.6	80.9 ± 0.1	84.0 ± 0.0	85.7 ± 0.0	76.8
Attention-Select	$\mathbf{87.9} \pm 0.1$	87.1 ± 0.1	92.3 ± 0.1	90.7 ± 0.1	53.2 ± 0.5	$\mathbf{65.7} \pm 0.5$	$\mathbf{84.0} \pm 0.1$	84.5 ± 0.1	87.4 ± 0.1	81.4
Coreset-Select-CLS (ours)	87.7 ± 0.1	86.9 ± 0.2	$\mathbf{92.4} \pm 0.1$	$\mathbf{90.9} \pm 0.1$	$\mathbf{53.3} \pm 0.5$	65.4 ± 0.3	$\mathbf{84.0} \pm 0.0$	$\mathbf{84.6} \pm 0.1$	$\mathbf{87.5} \pm 0.1$	81.4
Coreset-Select-Opt (ours)	$\mathbf{87.9} \pm 0.1$	$\mathbf{87.3} \pm 0.2$	$\mathbf{92.4} \pm 0.1$	$\mathbf{90.9} \pm 0.1$	$\mathbf{53.3} \pm 0.5$	65.6 ± 0.5	$\mathbf{84.0} \pm 0.0$	$\mathbf{84.6} \pm 0.1$	$\mathbf{87.5} \pm 0.0$	**81.5**
$\text{BERT}_{\text{Base}}$	87.9 ± 0.2	87.3 ± 0.2	92.4 ± 0.1	90.9 ± 0.1	53.3 ± 0.5	65.8 ± 0.5	84.0 ± 0.1	84.6 ± 0.1	87.5 ± 0.1	81.5

Table 14: GLUE *dev* performance at $1.5X$ speedup. Each score with standard deviation is averaged over 20 trials. Larger score indicates better performance. The best score among each token selection method is embolden for each column.

METHOD	STS-B	MRPC	SST-2	QNLI	COLA	RTE	MNLI-M	MNLI-MM	QQP	MEAN
Input-first-k-select	85.3 ± 0.1	81.3 ± 0.2	83.3 ± 0.2	84.6 ± 0.0	49.0 ± 0.4	62.1 ± 0.5	76.9 ± 0.1	74.9 ± 0.1	80.6 ± 0.1	75.3
First-k-select	85.1 ± 0.1	81.5 ± 0.2	84.6 ± 0.2	84.3 ± 0.1	49.0 ± 0.4	62.0 ± 0.5	76.3 ± 0.1	74.5 ± 0.1	80.0 ± 0.0	75.3
Random-select	85.6 ± 0.2	83.3 ± 0.3	84.9 ± 0.2	85.1 ± 0.1	48.4 ± 0.4	61.8 ± 0.5	79.0 ± 0.1	79.3 ± 0.1	86.6 ± 0.1	77.1
Average-Pool	78.7 ± 0.1	83.1 ± 0.2	85.1 ± 0.2	84.0 ± 0.1	0.0 ± 0.0	59.8 ± 0.6	75.2 ± 0.1	76.3 ± 0.1	82.9 ± 0.1	69.5
Attention-select	85.4 ± 0.1	84.3 ± 0.2	87.2 ± 0.1	86.4 ± 0.1	50.9 ± 0.4	62.7 ± 0.4	80.5 ± 0.1	80.7 ± 0.1	87.0 ± 0.1	78.3
Coreset-Select-CLS (ours)	86.5 ± 0.1	86.0 ± 0.1	87.6 ± 0.1	86.6 ± 0.1	51.0 ± 0.4	$\mathbf{63.6} \pm 0.4$	80.9 ± 0.1	81.6 ± 0.1	87.2 ± 0.1	79.0
Coreset-Select-Opt (ours)	$\mathbf{86.7} \pm 0.1$	$\mathbf{86.6} \pm 0.2$	$\mathbf{87.7} \pm 0.1$	$\mathbf{86.5} \pm 0.1$	$\mathbf{52.3} \pm 0.4$	$\mathbf{63.6} \pm 0.4$	$\mathbf{81.0} \pm 0.1$	$\mathbf{81.8} \pm 0.1$	$\mathbf{87.3} \pm 0.0$	**79.3**
$\text{BERT}_{\text{Base}}$	87.9 ± 0.2	87.3 ± 0.2	92.4 ± 0.1	90.9 ± 0.1	53.3 ± 0.5	65.8 ± 0.5	84.0 ± 0.1	84.6 ± 0.1	87.5 ± 0.1	81.5

Table 15: GLUE *dev* performance at at 70% space complexity reduction. Each score with standard deviation is averaged over 20 trials. Larger score indicates better performance. The best score among each token selection method is embolden for each column.

METHOD	STS-B	MRPC	SST-2	QNLI	COLA	RTE	MNLI-M	MNLI-MM	QQP	MEAN
Input-first-k-select	$\mathbf{87.9} \pm 0.2$	87.2 ± 0.2	91.8 ± 0.1	90.8 ± 0.1	53.0 ± 0.5	65.2 ± 0.5	$\mathbf{84.0} \pm 0.1$	$\mathbf{84.3} \pm 0.1$	87.1 ± 0.2	81.3
First-k-select	$\mathbf{87.9} \pm 0.2$	87.2 ± 0.1	91.3 ± 0.1	90.9 ± 0.1	53.0 ± 0.5	65.2 ± 0.5	$\mathbf{84.0} \pm 0.1$	$\mathbf{84.3} \pm 0.1$	87.4 ± 0.1	81.2
Random-select	87.8 ± 0.1	87.1 ± 0.2	92.1 ± 0.1	$\mathbf{90.9} \pm 0.2$	53.1 ± 0.4	$\mathbf{65.7} \pm 0.5$	83.9 ± 0.1	84.4 ± 0.1	87.4 ± 0.2	81.4
Average-Pool	87.8 ± 0.1	87.2 ± 0.1	91.7 ± 0.1	89.2 ± 0.1	44.6 ± 0.5	64.1 ± 0.5	82.7 ± 0.1	83.7 ± 0.1	86.2 ± 0.1	79.7
Attention-select	$\mathbf{87.9} \pm 0.1$	87.1 ± 0.1	92.3 ± 0.1	90.7 ± 0.1	$\mathbf{53.2} \pm 0.4$	$\mathbf{65.7} \pm 0.5$	83.9 ± 0.1	84.4 ± 0.1	$\mathbf{87.5} \pm 0.2$	81.4
Coreset-Select-CLS (ours)	87.7 ± 0.1	$\mathbf{87.3} \pm 0.2$	$\mathbf{92.4} \pm 0.1$	$\mathbf{90.9} \pm 0.1$	$\mathbf{53.2} \pm 0.4$	65.5 ± 0.3	$\mathbf{84.0} \pm 0.1$	$\mathbf{84.3} \pm 0.1$	$\mathbf{87.5} \pm 0.1$	81.4
Coreset-Select-Opt (ours)	$\mathbf{87.9} \pm 0.1$	$\mathbf{87.3} \pm 0.2$	$\mathbf{92.4} \pm 0.1$	$\mathbf{90.9} \pm 0.1$	$\mathbf{53.2} \pm 0.4$	65.6 ± 0.5	$\mathbf{84.0} \pm 0.1$	$\mathbf{84.3} \pm 0.1$	$\mathbf{87.5} \pm 0.1$	**81.5**
$\text{BERT}_{\text{Base}}$	87.9 ± 0.2	87.3 ± 0.2	92.4 ± 0.1	90.9 ± 0.1	53.3 ± 0.5	65.8 ± 0.5	84.0 ± 0.1	84.6 ± 0.1	87.5 ± 0.1	81.5

Table 16: GLUE *dev* performance at at 30% space complexity reduction. Each score with standard deviation is averaged over 20 trials. Larger score indicates better performance. The best score among each token selection method is embolden for each column.

	BIG BIRD				*PERFORMERS*			
METHOD	CIFAR-10	PATHFINDER-32	IMDB (BYTE-LEVEL)	MEAN	CIFAR-10	PATHFINDER-32	IMDB (BYTE-LEVEL)	MEAN
First-k-select	26.9 ± 0.2	55.6 ± 0.2	57.9 ± 0.1	46.8	26.9 ± 0.1	52.4 ± 0.2	59.9 ± 0.1	46.4
Random-select	39.4 ± 0.1	69.9 ± 0.2	59.6 ± 0.1	56.3	41.5 ± 0.2	58.2 ± 0.1	59.9 ± 0.1	53.2
Coreset-Select-CLS (ours)	38.6 ± 0.1	69.3 ± 0.2	59.1 ± 0.1	55.7	39.8 ± 0.1	61.5 ± 0.2	59.7 ± 0.1	53.7
Coreset-Select-Opt (ours)	$\mathbf{43.3} \pm 0.2$	$\mathbf{71.7} \pm 0.1$	$\mathbf{61.4} \pm 0.2$	**58.8**	$\mathbf{45.5} \pm 0.1$	$\mathbf{67.7} \pm 0.1$	$\mathbf{62.8} \pm 0.1$	**58.7**
Trans-No-Prune	40.9 ± 0.1	73.5 ± 0.2	63.8 ± 0.1	59.4	42.9 ± 0.1	66.2 ± 0.2	64.3 ± 0.1	57.8

Table 17: LRA *test* set performances at 70% space complexity reduction for *Big Bird* (left) and *Performers* (right) as the backbone Transformer. Each score with standard deviation is averaged over 20 trials. Larger score indicates better performance. The best score among each token selection method is embolden for each column.

	BIG BIRD				*PERFORMERS*			
METHOD	CIFAR-10	PATHFINDER-32	IMDB (BYTE-LEVEL)	MEAN	CIFAR-10	PATHFINDER-32	IMDB (BYTE-LEVEL)	MEAN
First-k-select	37.2 ± 0.1	69.7 ± 0.2	$\mathbf{62.8} \pm 0.1$	56.6	37.6 ± 0.1	64.6 ± 0.2	63.0 ± 0.1	55.1
Random-select	41.1 ± 0.2	70.0 ± 0.2	62.5 ± 0.1	57.9	43.2 ± 0.1	65.6 ± 0.2	63.3 ± 0.1	57.4
Coreset-Select-CLS (ours)	40.4 ± 0.2	$\mathbf{72.3} \pm 0.2$	62.2 ± 0.1	58.3	41.9 ± 0.2	66.6 ± 0.1	63.1 ± 0.1	57.2
Coreset-Select-Opt (ours)	$\mathbf{43.7} \pm 0.1$	$\mathbf{72.3} \pm 0.2$	62.7 ± 0.1	**59.6**	$\mathbf{46.1} \pm 0.1$	$\mathbf{68.5} \pm 0.2$	$\mathbf{63.9} \pm 0.1$	**59.5**
Trans-No-Prune	40.9 ± 0.1	73.5 ± 0.2	63.8 ± 0.1	59.4	42.9 ± 0.1	66.2 ± 0.2	64.3 ± 0.1	57.8

Table 18: LRA *test* set performances at 30% space complexity reduction for *Big Bird* (left) and *Performers* (right) as the backbone Transformer. Each score with standard deviation is averaged over 20 trials. Larger score indicates better performance. The best score among each token selection method is embolden for each column.

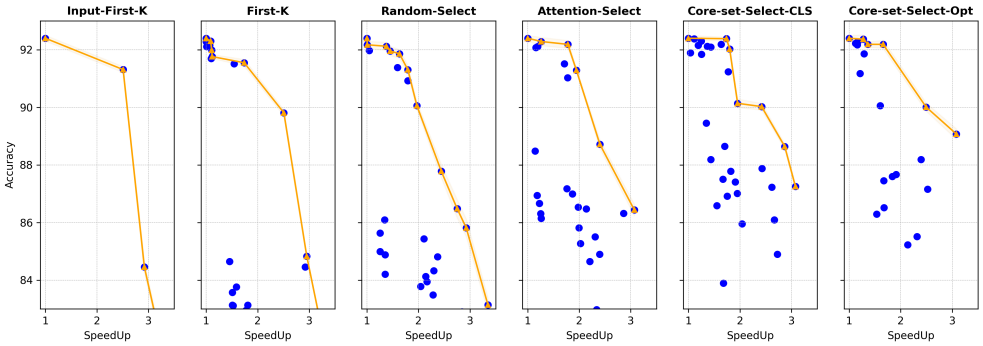

Figure 8: SST-2: Pareto curves and scatter plots for accuracy vs. inference speedup trade-off. Each point corresponds to a sequence-length configuration in Table 9. The pareto curves are used to generate accuracy vs. speedup Table 1, 2, 11, 13.

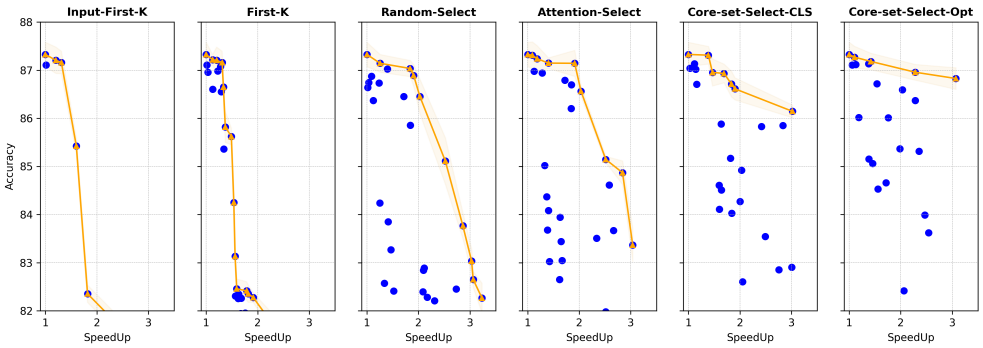

Figure 9: MRPC: Pareto curves and scatter plots for accuracy vs. inference speedup trade-off. Each point corresponds to a sequence-length configuration in Table 9. The pareto curves are used to generate accuracy vs. speedup Table 1, 2, 11, 13.