Vcc: Scaling Transformers to 128K Tokens or More by Prioritizing Important Tokens

Zhanpeng Zeng
University of Wisconsin, Madison
zzeng38@wisc.edu

Cole Hawkins
AWS AI
colehawk@amazon.com

Mingyi Hong
University of Minnesota, Minneapolis
mhong@umn.edu

Aston Zhang
AWS AI
astonz@amazon.com

Nikolaos Pappas
AWS AI
nppappa@amazon.com

Vikas Singh
University of Wisconsin, Madison
vsingh@biostat.wisc.edu

Shuai Zheng
AWS AI
shzheng@amazon.com

Abstract

Transformer models are foundational to natural language processing (NLP) and computer vision. Despite various recent works devoted to reducing the quadratic cost of such models (as a function of the sequence length n), dealing with ultra long sequences efficiently (e.g., with more than 16K tokens) remains challenging. Applications such as answering questions based on an entire book or summarizing a scientific article are inefficient or infeasible. In this paper, we propose to significantly reduce the dependency of a Transformer model's complexity on n, by compressing the input into a representation whose size r is *independent* of n at each layer. Specifically, by exploiting the fact that in many tasks, only a small subset of special tokens (we call VIP-tokens) are most relevant to the final prediction, we propose a VIP-token centric compression (Vcc) scheme which selectively compresses the input sequence based on their impact on approximating the representation of these VIP-tokens. Compared with competitive baselines, the proposed algorithm not only is efficient (achieving more than $3\times$ efficiency improvement compared to baselines on 4K and 16K lengths), but also achieves competitive or better performance on a large number of tasks. Further, we show that our algorithm can be scaled to 128K tokens (or more) while consistently offering accuracy improvement.

1 Introduction

The Transformer [28] is a foundational model for natural language processing (NLP) and computer vision. It has shown remarkable performance across NLP applications including machine translation [28], language inference [8], and summarization [12]. Transformers have also been successfully applied to various visual recognition tasks and achieve impressive results [9, 2, 36]. Unfortunately, the runtime/memory needs of Transformers involve an unfavorable dependence on the input length sequence. Assuming that l is the number of layers in the network, n is the input sequence length, and d is the model dimension, the complexity of a Transformer is $\mathcal{O}(ln^2d + lnd^2)$. The first and second terms pertaining to computing the multi-head self-attention and feed-forward layers, respectively, two core modules in a Transformer architecture.

Preprint. Under review.

The direct use of Transformers in ultra-long sequence applications is difficult. To deal with this setting, many current NLP models make use of strategies such as truncation to ensure that the input sentence length is at most 512, e.g., BERT, T5, and other Transformer-based language models [31, 20, 24]. Unfortunately, such a truncation, and other related strategies, inevitably results in loss of accuracy, the extent of which can vary from one task/dataset to another. Therefore, reducing this quadratic dependence on the input sequence length is a key focus of many proposals, this body of work is often called efficient self-attention. These developments are important milestones, and they have reduced the quadratic dependency on n to linear. Consequently, many Transformer models can now process samples with sequence lengths up to 4K (or 16K at most). In recent weeks, some reports of newer models being able to handle much longer sequences have appeared.

Rationale. It is natural to ask whether the ability to process longer sequences via linear self-attention is worth the trouble. The short answer is yes. In fact, even with weaker attention mechanisms [1, 33, 12], improved accuracy has been reported on long sequence tasks. So, what is stopping us from harvesting even stronger gains in accuracy by feeding even longer sequences to such models? It turns out that models such as Longformer [1] and Big Bird [33] become slow and consume an excessive amount of memory as the sequence length keeps increasing. See Fig. 1 for illustration. The linear dependence on n dominates the costs for sequences much longer than 4K (or 16K) tokens. It is obvious that any model must raster through the full sequence length at least once, and incur an $\mathcal{O}(n)$ cost. But to endow the models the ability to learn ultra-long range dependency, we need to lower the cost to be sub-linear or even independent of n. This goal is indeed aspirational, but what we describe in this paper is a concrete step forward – based on certain task-specific assumptions which appear to generally hold, we outline a formulation that works and delivers the expected improvements.

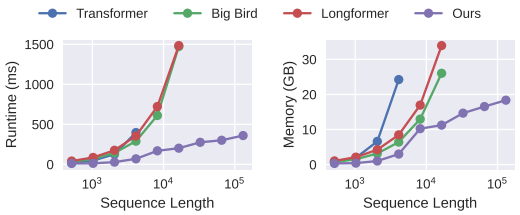

Figure 1: Model efficiency of processing one sequence on an A100 as sequence length increases (note logarithm x axis).

(1) Focus on what we need for a task: VIP-token centric compression (Vcc). It appears that our requirement outlined above would benefit from a mature body of literature on sub-linear algorithms. This is true to some extent, but we find that using generic sketching based ideas directly yields unsatisfactory performance. Part of the reason is the peculiar nature of processing/transformations that the tokens undergo in a Transformer module – a general analysis of information loss for common sketching schemes turns out to be difficult. Instead, we approach it backwards. We argue that in many tasks where Transformers are effective, only a small subset of tokens (which we refer to as VIP-tokens) are relevant to the final output (and accuracy) of a Transformer. If these tokens had been identified somehow, we could preserve this information in its entirety and only incur a moderate loss in performance. Now, *conditioned* on *these* specific VIP-tokens, an aggressive compression on the other tokens, can serve to reduce (and in some cases, fully recover) the loss in performance while dramatically decreasing the sequence length. This compression must leverage information regarding the VIP-tokens, with the goal of improving the approximation of the representation of the VIP-tokens. In other words, a high-fidelity approximation of the entire sequence is unnecessary. Once this "selectively compressed" input passes through a Transformer layer, the output sequence is decompressed to the original full sequence allowing the subsequent layers to access the full sequence.

(2) A specialized data structure for efficient compression/decompression. A secondary, but nonetheless important practical issue, is avoiding the linear dependence on n when compressing/decompressing the input/output sequences internally in the network. Ignoring this problem severely impacts efficiency. We describe a simple but specialized data structure to maintain the hidden states of the intermediate layers, where the compression can be easily accessed from the data structure, and decompression can be performed only by updating the data structure. Therefore, the sequence is never fully materialized in intermediate layers.

Practical contributions. Apart from the algorithmic modules described above, we show that despite an aggressive compression of the input sequences, we can achieve better or competitive performance on a long list of long sequence experiments. Compared to baselines, we obtain significantly better runtime and memory efficiency. Our paper shows that it is now practical to run Transformer models on 128K token sequence lengths, with consistent performance benefits.

2 Preliminaries

We briefly review the Transformer layer and define notations/simplifications. We use **BOLD** upper-case letters to denote matrices, **bold** lower case letters for vectors, and regular lower case letters for scalars. We also summarize related work on efficient Transformers.

2.1 Transformers (and Efficient Transformers)

Let us first check the input/output relation of a Transformer. Fix n to be the sequence length and let d be the embedding dimension. Define an embedding matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ which gives the n feature vector inputs for a Transformer block. The output of this Transformer block, $\mathbf{X}_{new}$, is defined as

$$\mathbf{X}_{new} = \beta(\alpha(\mathbf{X}, \mathbf{X}) + \mathbf{X}) + \alpha(\mathbf{X}, \mathbf{X}) + \mathbf{X} \tag{1}$$

where $\alpha(\cdot, \cdot)$ is called multi-head attention (MHA) and $\beta(\cdot)$ is a two-layer feed-forward network. Layer normalizations are omitted to reduce clutter. Let the inputs to $\alpha(\cdot, \cdot)$ be $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{n \times d}$. MHA is defined as:

$$\alpha(\mathbf{Q}, \mathbf{K}) := \text{cat}_{i=1}^{i=h} \left[\text{softmax}(\mathbf{Q}\mathbf{W}_{Q,i}\mathbf{W}_{K,i}^{\top}\mathbf{K}^{\top})\mathbf{K}\mathbf{W}_{V,i}\right] \mathbf{W} \tag{2}$$

where h is the number of attention heads (a hyper-parameter), $\{\mathbf{W}_{Q,i}, \mathbf{W}_{K,i}, \mathbf{W}_{V,i}\}$ are trainable projections, and the 'cat' operation concatenates the outputs of multiple self-attention modules. We omit the biases just for notational simplicity. For ease of discussion, let us further simplify the above notation by assuming that the number of self-attention modules is $h = 1$, and suppress $\mathbf{W}_{Q,1}, \mathbf{W}_{K,1}, \mathbf{W}_{V,1}, \mathbf{W}$ as well as the normalization in softmax: they will *still* be estimated within the model but are tangential to the description of our idea which leaves this module unchanged. With these simplifications the $\alpha(\cdot, \cdot)$ can be expressed as:

$$\alpha(\mathbf{Q}, \mathbf{K}) := \exp(\mathbf{Q}\mathbf{K}^{\top})\mathbf{K}. \tag{3}$$

Let $\gamma(\cdot)$ be a placeholder for all heavy computations in the Transformer layer above:

$$\gamma(\mathbf{X}) := \beta(\alpha(\mathbf{X}, \mathbf{X}) + \mathbf{X}) + \alpha(\mathbf{X}, \mathbf{X}). \tag{4}$$

We can verify that the output of a Transformer block (parameters are suppressed to reduce clutter) is,

$$\mathbf{X}_{new} = \gamma(\mathbf{X}) + \mathbf{X}. \tag{5}$$

A full Transformer model consists of many such identical layers: the input of each layer is the output $\mathbf{X}_{new}$ from the previous block. We may check that the overall complexity is $\mathcal{O}(ln^2d + lnd^2)$.

Efficient Transformers. A variety of efficient self-attention methods are available to reduce the $\mathcal{O}(ln^2d)$ cost. While this literature is growing rapidly, we list a few models commonly used in the community noting that this list is not exhaustive. Performer [4], Random Feature Attention [23], and Nyströmformer [30] propose different low rank approximations of the self-attention matrices. Longformer [1] and Big Bird [33] describe global + local sparse attention. Reformer [15] and Yoso [35] exploit locality sensitive hashing for approximating the self-attention matrix. MRA attention [34] takes a multi-resolution view of the self-attention matrix and proposes an approach to progressively refine the approximation of self-attention matrices.

Efficient Self-Attention does not scale well to ultra-long sequences. As briefly noted earlier, the existing self-attention mechanisms often reduce the complexity from $\mathcal{O}(ln^2d)$ to $\mathcal{O}(lndm)$ where m is a model specific hyper-parameter for each method. So far, most experiments report sequence lengths of up to 4K, with some exceptions [1, 33, 12]. Beyond 4K, the linear dependence on n for both $\mathcal{O}(lndm)$ and $\mathcal{O}(lnd^2)$ terms makes the cost prohibitive, especially for large models. For example, although LongT5 [12] manages to train a model on sequence lengths of up to 16K tokens with an efficient self-attention and shows promising results for longer sequences, it is slower and needs a sizable amount of compute (for example, see Fig. 1).

Other alternatives for sequence compression? Compressing input sequences for efficiency reasons in Transformers is not a new idea, and alternatives indeed exist. For example, [6] and [14] propose pyramid Transformer variants that progressively compress the sequence of hidden states, as the layers grow deeper. This is accomplished using pooling or a core-set based token selection. Separately, [21] proposes adaptively compressing the sequence based on the predicted semantic boundaries within

the sequence. There are **three** key differences with our approach. First, all methods listed above are *task agnostic*. They seek compressed/smaller representations to represent the *original* sequence well. Our formulation places no emphasis on representing the original sequence, as long as information pertinent to the VIP-tokens is preserved as much as possible. Second, once these methods compress the sequence in the initial stages, the residual information is lost (for the deeper layers). Our entire approach is predicated on avoiding this loss – we maintain access to the full sequence at each layer (via residual connection at least). Lastly, these ideas often involve an n^2 dependence on the sequence length in the initial stages of their formulation, making long sequence experiments problematic.

3 VIP-Token Centric Compression (Vcc)

Our main goal is to reduce the dependency on n (but not by modifying self-attention calculations as in efficient transformers). To do this, we describe a scheme that compresses the input sequence of a Transformer layer and decompresses the output sequence, resulting in a model whose complexity is $\mathcal{O}(lrd^2 + lr^2d + lr\log(n_c)d + lrn_pd + nd)$. Here, r is a model hyperparameter for the size of the compressed sequence, n_p is the size of VIP-tokens described shortly, and n_c is the size of non-VIP/remaining tokens. So, we have $n_p + n_c = n$ and assume $n_p \ll r \ll n$. The full complexity analysis is described in the Appendix.

Parsing the complexity term: Let us unpack the term to assess its relevance. The first two terms $\mathcal{O}(lrd^2 + lr^2d)$ give the cost for a Transformer, while the remaining terms are the overhead of compression and decompression. The term $\mathcal{O}(lr\log(n_c)d + lrn_pd)$ is the overhead of compression and updating our data structure at each layer. The $\mathcal{O}(nd)$ term pertains to pre-processing involving converting the hidden states to our data structure and post-processing to recover the hidden states from the data structure. Note that unlike the dependence on n for vanilla Transformers, this $\mathcal{O}(nd)$ is incurred only at the input/output stage of the Transformer, but **not** at any intermediate layers.

High level design choices. We use the *standard* Transformer layers with a *standard* feed-forward network (which results in d^2 in the first term) and *standard* quadratic cost self-attention (which gives the r^2 factor in the second term). Why? These choices help isolate the effect of incorporating their efficient counterparts. The proposed algorithm operates on the *input/output of each Transformer layer* leaving the Transformer module itself unchanged. Therefore, our goals are distinct from the literature investigating efficient self-attentions and efficient feed-forward networks. This is because one can replace these two vanilla modules with any other efficient alternatives to further reduce the r^2 and d^2 terms directly. We note that despite these quadratic terms, our approach is significantly faster compared to baselines, described in §4.

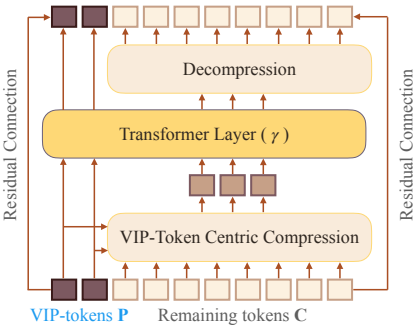

Figure 2: Diagram that illustrates a Transformer layer with VIP-token centric sequence compression.

We will first describe our general idea, as shown in Fig. 2, which uses VIP-tokens to guide the compression/decompression of the input/output of a Transformer layer so that it only needs to process the compressed sequence (§3.1, §3.2). Then, we will discuss an instantiation of the compression process, by adapting a multi-resolution analysis technique (§3.3). Lastly, we will introduce a data structure which allows more efficient compression/decompression (§3.4).

3.1 Elevating the Importance of a Few Tokens: VIP-Tokens

Let us start with the simplest compression, which identifies a linear transformation $\mathbf{S} \in \mathbb{R}^{r\times n}$ which acts on the input, resulting in a smaller representation $\mathbf{SX} \in \mathbb{R}^{r\times d}$. Of course, a smaller r implies that more information about $\mathbf{X}$ is lost. But we find that in many tasks, only the embedding representations of *a few* tokens drives the final prediction: we refer to these tokens as *VIP-tokens*.

Examples of VIP-tokens: Observe that only the embedding outputs of masked tokens in masked language modeling [8] and the CLS token in sequence classification [8, 9] are/is used for prediction. In question answering, only the questions and possible answers associated with the questions are used for prediction. It is important to note that the masked tokens, CLS tokens, and question tokens are **(1)**

defined by the tasks and **(2)** *known* to the model (although the embedding representation of these tokens are unknown). These VIP-tokens can be viewed as a task or question that is given to the model. The model can process the sequence with a specific goal in mind so that the model can skip/skim less relevant segments. Our general principle involves choosing *a set of tokens* as the VIP-tokens that **(1)** are important to the specific task goals and **(2)** easily pre-identifiable by the user.

Caveats. Not all important tokens can be pre-identified. For example, the tokens in the correct answer span in answer span prediction are also important to the specific goals, but are difficult to pre-identify, so only the question tokens (and not the answer tokens) are used as VIP-tokens. We assume that any other tokens that is relevant for prediction should have high dependency with these VIP-tokens. For example, the answer tokens should have high dependency (in self-attention) to the question tokens.

VIP-tokens occupy the front seats. VIP-tokens can typically occur anywhere within a sequence. But we can re-order the input sequence as well as the positional encodings such that VIP-tokens are always at the *head of sequence* to make analysis/implementation easier. This is possible since Transformer is permutation invariant when permuting positional encodings (such as positional embedding or positional IDs) along with tokens. This re-ordering is performed only once for the input of the entire Transformer model, then the token outputs generated by the model are rearranged to their original positions. With this layout, let $\mathbf{P} \in \mathbb{R}^{n_p \times d}$ be the VIP-tokens (indicated by blue) and $\mathbf{C} \in \mathbb{R}^{n_c \times d}$ be the non-VIP/remaining tokens, then $\mathbf{X}$ can be expressed as

$$X = \begin{bmatrix} \mathbf{P} \\ \mathbf{C} \end{bmatrix} \tag{6}$$

From the above discussion, it is clear that one needs to make sure that after compressing the input tokens X, the VIP-tokens must still stay (more or less) the same, and the compression matrix S must be *VIP-token dependent*. We hypothesize that such *VIP-token dependent* compression matrices require a much smaller dimension r, compared to *VIP-token agnostic* compression matrices.

3.2 VIP-Token Centric Compression (Vcc): An Initial Proposal

For a Transformer layer, let $\mathbf{X}$ denote its input matrix. Express the output of this layer as follows:

$$\mathbf{X}_{new} = \mathbf{S}^{\dagger}\gamma(\mathbf{S}\mathbf{X}) + \mathbf{X} \tag{7}$$

where $\mathbf{S} \in \mathbb{R}^{r \times n}$ is a matrix compressing $\mathbf{X}$ to a smaller representation and $\mathbf{S}^{\dagger}$ is the pseudo inverse for decompression. With the layout in (6), we can write (7) as

$$\begin{bmatrix} \mathbf{P}_{new} \\ \mathbf{C}_{new} \end{bmatrix} = \mathbf{S}^{\dagger}\gamma(\mathbf{S}\begin{bmatrix} \mathbf{P} \\ \mathbf{C} \end{bmatrix}) + \begin{bmatrix} \mathbf{P} \\ \mathbf{C} \end{bmatrix} \tag{8}$$

where $\mathbf{P}_{new}$ and $\mathbf{C}_{new}$ are the new embeddings for $\mathbf{P}$ and $\mathbf{C}$.

Always reserve seats for VIP-tokens. What is a useful structure of $\mathbf{S}$? Since $\mathbf{P}_{new}$ is the embedding output for the VIP-tokens $\mathbf{P}$, we want them to be fully preserved. To achieve this, we impose the following structure on $\mathbf{S}$ and $\mathbf{S}^{\dagger}$:

$$\mathbf{S} = \begin{bmatrix} \mathbf{I}_{n_p \times n_p} & 0 \\ 0 & \mathbf{S}_c \end{bmatrix} \qquad \mathbf{S}^{\dagger} = \begin{bmatrix} \mathbf{I}_{n_p \times n_p} & 0 \\ 0 & \mathbf{S}_c^{\dagger} \end{bmatrix}. \tag{9}$$

The rearrangement simply says that we will avoid compressing $\mathbf{P}$? But rewriting it this way helps us easily unpack (8) to check the desired functionality of $\mathbf{S}_c$.

Prioritize information of VIP-tokens. Our goal is to ensure $\mathbf{P}_{new}$ generated from the compressed sequence in (8) will be similar to its counterpart from the uncompressed sequence. Let us check (8) using the compression matrix S defined in (9) first. We see that

$$\mathbf{S}^{\dagger}\gamma(\mathbf{S}\mathbf{X}) = \begin{bmatrix} \mathbf{I}_{n_p \times n_p} & 0 \\ 0 & \mathbf{S}_c^{\dagger} \end{bmatrix} \gamma(\begin{bmatrix} \mathbf{P} \\ \mathbf{S}_c\mathbf{C} \end{bmatrix}) = \begin{bmatrix} \beta(\alpha(\mathbf{P}, \mathbf{S}\mathbf{X}) + \mathbf{P}) + \alpha(\mathbf{P}, \mathbf{S}\mathbf{X}) \\ \mathbf{S}_c^{\dagger}\beta(\alpha(\mathbf{S}_c\mathbf{C}, \mathbf{S}\mathbf{X}) + \mathbf{S}_c\mathbf{C}) + \mathbf{S}_c^{\dagger}\alpha(\mathbf{S}_c\mathbf{C}, \mathbf{S}\mathbf{X}) \end{bmatrix}. \tag{10}$$

The orange color identifies terms where $\mathbf{P}_{new}$ interacts with other compression-related terms $\mathbf{C}$ and/or $\mathbf{S}_c$. We primarily care about $\mathbf{P}_{new}$ in (8), so the first (orange) row in (10) is the main concern. We see that $\mathbf{P}_{new}$ only depends on the compressed sequence $\mathbf{S}\mathbf{X}$ via $\alpha(\mathbf{P}, \mathbf{S}\mathbf{X})$. We can further unpack,

$$\alpha(\mathbf{P}, \mathbf{S}\mathbf{X}) = \exp(\mathbf{P}\mathbf{X}^{\top}\mathbf{S}^{\top})\mathbf{S}\mathbf{X} = \exp(\mathbf{P}\mathbf{P}^{\top})\mathbf{P} + \exp(\mathbf{P}\mathbf{C}^{\top}\mathbf{S}_c^{\top})\mathbf{S}_c\mathbf{C}. \tag{11}$$

Again, $\alpha(\mathbf{P}, \mathbf{SX})$ depends on $\mathbf{C}$ and $\mathbf{S}_c$ via the second (orange) term. This helps us focus on the key term that matters: $\exp(\mathbf{P}\mathbf{C}^\top\mathbf{S}_c^\top)\mathbf{S}_c\mathbf{C}$. As long as $\mathbf{S}_c$ is such that the following approximation is good,

$$\exp(\mathbf{P}\mathbf{C}^\top\mathbf{S}_c^\top)\mathbf{S}_c \approx \exp(\mathbf{P}\mathbf{C}^\top), \tag{12}$$

we will obtain a good approximation of $\mathbf{P}_{new}$. Our remaining task is to outline a scheme of finding a compression $\mathbf{S}_c$ such that this criterion can be assured.

3.3 An Instantiation: Multi-Resolution Compression

What should be the mechanics of our compression such that (12) holds? In general, to get $\mathbf{S}_c$, we can use any sensible data driven sketching idea which minimizes the error of (12) incrementally (or in a coarse to fine manner). Doing so efficiently needs more work; we describe the high level intuition below and the low-level details are provided in the appendix.

High level idea. Ideally, an efficient scheme for constructing $\mathbf{S}_c$ should operate as follows. If some regions of the sequence $\mathbf{C}$ have a negligible impact on (12) (via the orange terms above), the procedure should compress the regions aggressively. If other regions are identified to have a higher impact on (12) (again due to the orange terms above), the procedure should scan these regions more carefully for a more delicate compression. This suggests that procedurally a coarse-to-fine strategy may work but integrating it within Transformer pipelines is a bit involved. For example, multi-resolution analysis does help in approximating self-attention matrices in Transformers [34], but requires specialized CUDA kernels. One reason is that the formulation in [34] cannot be easily written as matrix operations similar to (12). Nonetheless, it turns out that the analogous form for 1D can be expressed as a matrix operation (and gives a strategy for obtaining the $\mathbf{S}_c$) for compressing $\mathbf{C}$. To summarize, while [34] offers a multi-resolution view of the self-attention matrix, our scheme is best thought of as emulating the multi-resolution strategy, but to compress the sequence itself in (12).

We use a truncated linear combination of a set or "basis" (to be introduced shortly) to approximate the rows of $\exp(\mathbf{P}\mathbf{C}^\top)$. One can view $\mathbf{S}_c$ as a truncated row space wavelet transform. Note that the approximation depends on the signal being explained by these basis elements and how many such bases we have. We call these bases as components defined as $\mathbf{b}_x^s \in \mathbb{R}^{n_c}$ are

$$[\mathbf{b}_x^s]_i := \begin{cases} \frac{1}{s} & \text{if } sx - s < i \leq sx \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

for $s \in \{2^0, 2^1, 2^2, \cdots, n_c\}$ assuming n_c is a power of 2 and $x \in \{1, 2, \cdots, n_c/s\}$. Here, s and x represent the scaling and translation of the component, respectively to dilate/expand it or move it around (similar to wavelets) and $[\cdot]_i$ refers to the i-th entry/row of the input vector/matrix.

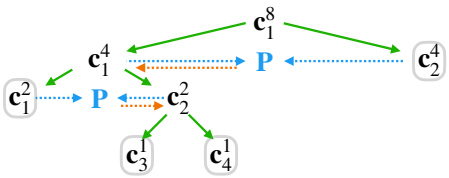

Figure 3: Illustration of multi-resolution compression. Green arrow: node is refined to higher resolution. Blue arrow: compare attention score between node and VIP-tokens. Orange arrow: node is selected to be refined by VIP-tokens. A set $\mathcal{J} = \{\mathbf{b}_1^2, \mathbf{b}_3^1, \mathbf{b}_4^1, \mathbf{b}_2^4\}$ is constructed according to the leaf (circled) nodes.

Procedurally, our scheme starts with the heaviest compression and progressively decompresses certain segments of $\mathbf{C}$ guided by the VIP-tokens $\mathbf{P}$. This high level scheme is illustrated in Fig. 3. The node $\mathbf{c}_x^s$ in Fig. 3 is defined as

$$\mathbf{c}_x^s := \mathbf{b}_x^s\mathbf{C} \tag{14}$$

for all scaling s and translation x, which represents a s-length segment of non-VIP tokens $\mathbf{C}$. In the end, we obtain the desired compression described by a set $\mathcal{J}$. The rows of $\mathbf{S}_c$ are the components $\mathbf{b}_x^s \in \mathcal{J}$, and the rows of $\mathbf{S}_c\mathbf{C}$ are the associated $\mathbf{c}_x^s$, which are the leaf nodes of the illustrated tree 3. The compression $\mathbf{S}_c$ obtained via this scheme has some desirable properties: **(1)** Each column of $\mathbf{S}_c$ contains exactly one nonzero entry. **(2)** The pseudo-inverse of $\mathbf{S}_c$ is simply $\mathbf{S}_c^\dagger = \mathbf{S}_c^\top\mathbf{D}$ where $\mathbf{D}$ is a diagonal scaling matrix such that each nonzero entry of $\mathbf{S}_c^\dagger$ is 1. **(3)** If the VIP-tokens $\mathbf{P}$ has high attention weights for some rows of $\mathbf{C}$, then corresponding row in $\mathbf{C}$ will be approximated with higher frequencies (less compressed). The technical details of this algorithm are less relevant for our overall approach, but for interested readers, the derivation and description is discussed in the Appendix.

How good is this approximation? The output $\mathbf{P}_{new}$ is well approximated, by design, since the approximation preserves the relevant frequency components of the subset of rows of $\mathbf{C}$ that have a high impact on the output $\mathbf{P}_{new}$. Further, the output in $\mathbf{C}_{new}$ corresponding to the subset of rows

of $\mathbf{C}$ that have a higher dependency with the VIP-tokens will have a better approximation than the remaining rows of $\mathbf{C}$, as desired. This property is useful since some tokens with unknown locations but high dependency with the VIP-tokens can also relevant to the final prediction of a Transformer model in some tasks. The candidate answer tokens in question answering tasks is one example, and our construction ensures that they will be approximated well too.

3.4 Efficient Data Structure for Compression

By employing the procedure in §3.3 illustrated in Fig. 3, we can find an $\mathbf{S}_c$ with an $\mathcal{O}(n_c d + r n_p d)$ cost at each layer. The main cost $\mathcal{O}(n_c d)$ is the overall cost of computing $\mathbf{c}_x^s$ for all scaling s and translation x. We find that this cost could introduce a large runtime overhead. Further, note that in (8), if we actually perform the inverse transform $\mathbf{S}^\dagger$ for decompression, the cost is $\mathcal{O}(n_c d)$, which is undesirable. As a solution, we now describe a data structure $\mathcal{T}(\mathbf{C})$ for storing $\mathbf{C}$ which enables efficient compression/decompression and reduces the $\mathcal{O}(n_c d)$ cost to $\mathcal{O}(r \log(n_c) d)$. Whenever $\mathbf{C}$ is needed, it can be reconstructed exactly and efficiently from $\mathcal{T}(\mathbf{C})$. This data structure is only possible due to the specific structure of $\mathbf{S}_c$ constructed in §3.3. Specifically, we introduce a tree structure cache storing $\mathbf{c}_1^{n_c}$ and $\Delta\mathbf{c}_x^s$ defined as

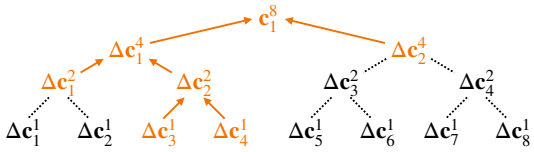

Figure 4: Proposed tree structure cache $\mathcal{T}(\mathbf{C})$

$$\Delta\mathbf{c}_x^s := \mathbf{c}_{\lceil x/2 \rceil}^{2s} - \mathbf{c}_x^s \tag{15}$$

for every scaling $s \neq n_c$ and translation x. Then, for our reconstruction, any $\mathbf{c}_x^s$ defined in (14) can be retrieved in $\mathcal{O}(\log(n_c) d)$ cost via recursion

$$\mathbf{c}_x^s = \mathbf{c}_{\lceil x/2 \rceil}^{2s} - \Delta\mathbf{c}_x^s = \mathbf{c}_{\lceil x/4 \rceil}^{4s} - \Delta\mathbf{c}_{\lceil x/2 \rceil}^{2s} - \Delta\mathbf{c}_x^s = \cdots \tag{16}$$

The benefit of this data structure is that we can change some nodes in $\mathcal{T}(\mathbf{C})$ and so the updated version is $\mathcal{T}(\mathbf{C}_{new})$, but this only needs $\mathcal{O}(r)$ updates.

An example. We show a four level cache for $n_c = 8$ in Fig. 4. If $\mathcal{J} = \{\mathbf{b}_1^2, \mathbf{b}_3^1, \mathbf{b}_4^1, \mathbf{b}_2^4\}$ as shown in Fig. 3, then, for example, via arithmetic manipulations (more details in Appendix), we have:

$$(\mathbf{c}_{new})_1^1 - \mathbf{c}_1^1 = (\mathbf{c}_{new})_2^1 - \mathbf{c}_2^1 = (\mathbf{c}_{new})_1^2 - \mathbf{c}_1^2 \tag{17}$$

We use notation $(\mathbf{c}_{new})_x^s$ and $\Delta(\mathbf{c}_{new})_x^s$ to denote the counterparts of (14) and (15) when using $\mathbf{C}_{new}$ instead of $\mathbf{C}$. Then, we can verify that $\Delta(\mathbf{c}_{new})_1^1, \Delta(\mathbf{c}_{new})_2^1$ in $\mathcal{T}(\mathbf{C}_{new})$ stays the same as $\Delta\mathbf{c}_1^1, \Delta\mathbf{c}_2^1$ in $\mathcal{T}(\mathbf{C})$ and thus do not need to be updated:

$$\begin{aligned} \Delta(\mathbf{c}_{new})_1^1 &= (\mathbf{c}_{new})_1^2 - (\mathbf{c}_{new})_1^1 = \mathbf{c}_1^2 - \mathbf{c}_1^1 = \Delta\mathbf{c}_1^1 \\ \Delta(\mathbf{c}_{new})_2^1 &= (\mathbf{c}_{new})_1^2 - (\mathbf{c}_{new})_2^1 = \mathbf{c}_1^2 - \mathbf{c}_2^1 = \Delta\mathbf{c}_2^1 \end{aligned} \tag{18}$$

With similar logic, we can verify that only the colored nodes in Fig. 4 will be updated. In summary, for each $\mathbf{b}_x^s \in \mathcal{J}$, only the node $\Delta(\mathbf{c}_{new})_x^s$ and its ancestor nodes in $\mathcal{T}(\mathbf{c}_{new})$ must be updated. Via some calculations, the number of updates is $\mathcal{O}(r)$, so the complexity of modifying $\mathcal{T}(\mathbf{C})$ to $\mathcal{T}(\mathbf{C}_{new})$ is $\mathcal{O}(rd)$. The detailed algorithm is described in Appendix.

By maintaining this data structure, we never need to materialize the entire $\mathbf{C}$ or $\mathbf{C}_{new}$ in any intermediate layer, but instead we use (16) to construct the rows of $\mathbf{S}_c\mathbf{C}$ and perform updates to $\mathcal{T}(\mathbf{C})$ to obtain $\mathbf{C}_{new}$ (represented as $\mathcal{T}(\mathbf{C}_{new})$) at each intermediate layer. At the output of a Transformer, $\mathbf{C}_{new}$ is materialized from $\mathcal{T}(\mathbf{C}_{new})$ at a $\mathcal{O}(n_c d)$ cost via the recursion (16).

4 Experiments

We perform a broad set of experiments to empirically evaluate the performance of our proposed compression. We evaluate our method on both encoder-only and encoder-decoder architecture types. We compare our method with baselines on a large list of question answering and summarization tasks, where we found long sequences occur most frequently. Then, we study the model performance of scaling to ultra long sequences enabled by our method. Since efficiency is the focus of the efficient baselines and our work, we include runtime efficiency (of a single sequence) in millisecond at each table. All training hyperparameters and statistics of datasets are reported in Appendix.

In our discussion, the ratio of resolution between parent node and child nodes (Fig. 3) is 2 for simplicity, but in practice, we allow this ratio to be larger to reduce the tree depth. For ease of implementation, we restrict $\mathcal{J} = \{\mathbf{b}_x^s : s \in \{1, s_0\}\}$ for a pre-defined s_0 to have exactly two resolutions for our experiments. Further, we found a few layers of standard Transformer layers to preprocess tokens helps the performance. Therefore, in the initial stage of a Transformer, we segment input sequence into multiple segments of 512 length. For each segment, we use vanilla computation in the first 4 layers (for base models and 6 layers for larger models) of a Transformer. Then, for the remaining layers, segments are concatenated back into one sequence and processed using our proposed compression. There is *no communication* among any segments, so the initial stage is used just for getting a reasonable representation for the compression to operate on. To simplify the implementation, we only use the proposed compression in the encoder, and use the vanilla computation in the decoder.

Table 1: Dev set results for encoder-only models.

Method	Size	Length	HotpotQA			QuALITY		WikiHop	
			Time	EM	F1	Time	Accuracy	Time	Accuracy
RoBERTa	base	512	19.9	35.1	44.9	21.2	39.0	19.6	67.6
RoBERTa	base	4K	422.3	62.2	76.1	403.2	39.5	414.1	75.2
Big Bird	base	4K	297.9	59.5	73.2	307.0	38.5	293.3	74.5
Longformer	base	4K	371.0	59.9	73.6	368.0	27.9	369.7	74.3
Ours	base	4K	114.6	60.9	74.6	126.4	39.6	108.0	75.9
Ours*	base	4K	114.6	61.4	75.0	125.7	39.5	108.0	76.1
Ours*	large	4K	285.8	66.7	80.0	390.8	41.8	394.3	79.6

Encoder-Only Models. For encoder-only architecture, we compare our method with RoBERTa [20] and two strong baselines: Longformer [1] and Big Bird [33]. We first pretrain a standard RoBERTa model using masked language modeling task, then for each method, we do continuous pretraining from the pretrained RoBERTa checkpoint to expend the positional embeddings to 4K length and adjust model parameters to adapt any approximations used in Longformer, Big Bird, and our method. We verify that our proposed method can be integrated into a pretrained Transformer with some continuous pretraining. However, we note that the amount of reduction in log perplexity for our method (-0.114) during pre-training is much larger than Longformer (-0.017) and Big Bird (-0.025) from 50K steps to 250K steps. The continuous pretraining for these baselines might have saturated since only the self-attention is approximated while our method might require more pretraining to adjust the parameters for more aggressive approximation. Therefore, we run a larger scale pretraining for our method and the downstream results are shown in Tab. 1 and Fig. 5, denoted with *.

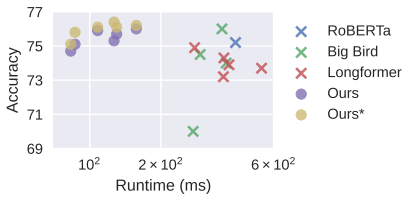

Figure 5: Model runtime vs WikiHop dev accuracy when using different model specific hyperparameters

We use HotpotQA [32], QuALITY [22], and WikiHop [29] to assess the language models. HotpotQA is an answer span extraction task, while QuALITY and WikiHop are multi-choice question answering tasks. We set questions and multi-choice answers (for QuALITY and WikiHop) as VIP-tokens.

As shown in Tab. 1, we verify that our method is consistently better compared to Longformer and Big Bird. Our method obtains better accuracy in QuALITY and WikiHop compared to 4K length RoBERTa model, but it is a bit worse than 4k length RoBERTa model on HotpotQA. More pretraining helps close the gap. We also use WikiHop to experiment with method specific hyperparameters (such as block size in Big Bird, window size in Longformer, and compression size r in our method). As shown in Fig. 5, our runtime efficiency frontier is consistently better than the baselines. The key **takeaway** is that our method has a much better runtime efficiency than baselines that have the same sequence length without sacrificing its model performance. Further, we note that our method can be scaled to larger models for accuracy improvement.

Encoder-Decoder Models. For encoder-decoder model, we compare our method with T5 [24], LongT5 [12], and LED [1]. We directly use the public pretrained checkpoints for baselines. The pretrained models for our method are obtained by doing a continuous pretraining from the public T5 checkpoints using T5's pretraining task [24]. We note that LED-base has 6 encoder layers and 6 decoder layers compared to 12 encoder layers and 12 decoder layers in base models of other methods, its performance is usually lower. As a result, we only include LED in a few tasks.

We use HotpotQA [32], WikiHop [29], CNN/Dailymail [26], MediaSum [38], Arxiv [5], and GovReport [13], SummScreenFD [3], QMSum [37], NarrativeQA [17], Qasper [7], QuALITY [22], ContractNLI [19] from SCROLLS benchmark [27] to assess the language models. For question answering tasks, we set questions and multi-choice answers (for QuALITY and WikiHop) as VIP-tokens

Table 2: Dev set results for encoder-decoder models. The left / right values of runtime columns are the runtime for the entire model / the encoder.

Method	Size	# Param	Length	WikiHop			HotpotQA			CNN/Dailymail				MediaSum			
				Runtime	EM	F1	Runtime	EM	F1	Runtime	R-1	R-2	R-L	Runtime	R-1	R-2	R-L
T5	base	223M	512	25.7 / 20.5	66.7	69.1	26.3 / 20.5	34.1	44.4	40.0 / 20.5	43.3	20.5	40.4	39.9 / 20.5	30.7	14.5	28.1
T5	base	223M	4K	594.3 / 553.7	76.2	78.1	594.3 / 550.6	64.2	77.5	614.4 / 549.4	43.8	20.9	41.0	613.5 / 552.9	34.9	17.2	31.9
LongT5	base	248M	4K	270.7 / 233.9	72.7	74.8	271.3 / 233.7	62.3	75.7	291.6 / 234.9	43.3	20.6	40.5	287.3 / 229.5	34.9	17.3	32.0
LED	base	162M	4K	236.6 / 222.9	70.0	72.4	237.4 / 222.9	55.1	67.9	249.4 / 221.8	43.3	20.0	40.5	- / -	-	-	-
Ours	base	223M	4K	181.7 / 148.1	76.7	78.4	155.4 / 127.4	64.5	77.7	195.8 / 139.9	43.6	20.7	40.7	196.7 / 140.2	34.8	17.3	31.9
T5	large	738M	512	83.5 / 67.0	69.1	71.4	84.1 / 67.0	36.9	47.8	124.6 / 67.0	43.8	20.7	40.9	124.5 / 67.0	31.9	15.5	29.1
T5	large	738M	4K	1738.7 / 1601.0	79.1	80.7	1598.1 / 1598.1	68.0	81.3	1824.8 / 1600.4	44.3	21.0	41.4	- / -	-	-	-
Ours	large	738M	4K	561.4 / 460.6	79.0	80.6	485.3 / 382.8	67.8	81.0	608.1 / 433.8	44.4	21.4	41.5	609.7 / 434.4	35.8	18.2	32.8
Ours	3b	3B	4K	1821.5 / 1441.2	80.8	82.3	1547.7 / 1197.1	70.2	83.2	1930.7 / 1364.8	44.8	21.5	41.9	1930.7 / 1364.8	36.3	18.5	33.3

Method	Size	# Param	Length	Qasper			QuALITY			Arxiv				SummScreenFD			
				Runtime	EM	F1	Runtime	EM	F1	Runtime	R-1	R-2	R-L	Runtime	R-1	R-2	R-L
T5	base	223M	512	31.8 / 20.5	10.8	16.4	29.3 / 20.5	33.6	47.3	59.0 / 20.5	28.9	8.6	25.6	59.1 / 20.5	27.0	4.8	23.5
T5	base	223M	4K	608.2 / 551.7	13.2	29.1	596.3 / 551.2	34.7	47.4	645.4 / 549.1	44.4	18.4	39.9	647.9 / 551.1	31.6	6.8	27.6
LongT5	base	248M	16K	1628.5 / 1421.3	16.2	33.4	1633.1 / 1439.7	35.8	48.5	1699.7 / 1370.4	48.5	21.7	43.7	1763.4 / 1427.8	33.1	7.3	28.5
LED	base	162M	16K	- / -	-	-	- / -	-	-	1055.8 / 923.6	47.8	20.6	43.2	- / -	-	-	-
Ours	base	223M	16K	538.3 / 391.6	16.0	30.8	557.1 / 419.2	36.5	48.7	672.8 / 392.1	48.5	21.4	43.9	670.5 / 390.9	33.1	7.3	28.6
T5	large	738M	512	101.9 / 66.4	11.3	17.0	95.8 / 67.1	35.3	49.0	182.2 / 67.1	30.5	9.1	27.1	180.9 / 66.5	28.3	4.9	24.9
T5	large	738M	4K	- / -	-	-	1760.5 / 1596.4	37.8	50.5	1901.5 / 1598.8	46.0	19.4	41.4	- / -	-	-	-
Ours	large	738M	16K	1679.6 / 1120.2	16.3	33.7	1753.6 / 1210.7	40.3	52.5	1959.1 / 1111.0	49.5	22.2	44.7	1957.1 / 1109.2	34.3	7.6	29.6
Ours	3b	3B	16K	6165.4 / 4637.3	19.0	38.2	6398.8 / 4962.7	45.2	56.0	7676.3 / 4642.2	49.8	22.4	45.0	7641.5 / 4631.3	34.7	7.8	30.1

Method	Size	# Param	Length	ContractNLI			NarrativeQA			GovReport				QMSum			
				Runtime	EM	F1	Runtime	EM	F1	Runtime	R-1	R-2	R-L	Runtime	R-1	R-2	R-L
T5	base	223M	512	24.0 / 20.5	73.5	73.5	26.8 / 20.5	2.0	11.3	59.1 / 20.5	40.5	14.8	38.2	43.5 / 20.5	30.2	8.0	26.5
T5	base	223M	4K	579.0 / 551.6	86.8	86.8	593.4 / 547.6	3.8	13.3	648.3 / 551.5	54.0	25.2	51.4	620.2 / 551.5	31.1	8.2	27.4
LongT5	base	248M	16K	1564.2 / 1462.5	85.1	85.1	1541.7 / 1370.2	5.2	15.6	1726.4 / 1387.7	55.8	27.9	53.2	1721.4 / 1450.7	35.7	11.7	31.4
Ours	base	223M	16K	484.2 / 393.1	87.0	87.0	518.2 / 394.4	5.0	15.8	674.0 / 391.6	55.2	27.1	52.6	623.1 / 396.5	31.8	8.8	27.9
T5	large	738M	512	78.1 / 67.1	74.3	74.3	- / -	-	-	180.9 / 67.0	43.3	16.2	41.1	136.4 / 67.1	31.7	8.1	27.6
T5	large	738M	4K	1702.4 / 1601.2	87.2	87.2	- / -	-	-	- / -	-	-	-	- / -	-	-	-
Ours	large	738M	16K	1440.6 / 1122.6	87.8	87.8	1551.7 / 1133.9	6.6	18.7	1955.5 / 1113.8	56.3	28.0	53.8	1816.4 / 1134.6	34.8	10.4	30.7
Ours	3b	3B	16K	5850.2 / 4665.9	88.5	88.5	6055.4 / 4659.4	8.2	21.2	7668.2 / 4642.7	56.9	28.5	54.3	7146.7 / 4655.6	35.7	10.9	31.1

in our method. For query-based summarization, such as QMSum, we use the query as VIP-tokens in our method. For general summartization tasks, we prepend a "summarize:" in each instance and use it as VIP-tokens in our method. Our method achieves matching or better performance in most tasks compared to T5, LongT5, and LED with much higher efficiency (see Tab. 2). Further, the performance of our method monotonically increases as the model size increases, so our method can be scaled to larger models.

Scaling to Longer Sequences. The prior experiments limit the sequence length to at most 4K or 16K since the baselines can only be scaled up to these sequence lengths. However, our method can be scaled to much longer sequences. We note that NarrativeQA [17] is an ideal testbed as shown in dataset statistics in Appendix. The results are shown in Tab. 3. The left / middle / right values of runtime column are for the entire model / the encoder / the last 8 layers (out of 12 layers) that uses our compression. The performance monotonically increases as sequence length increases. We note that for sequence length 64K, the performance of model with $s_0 = 64$ is lower than the model with $s_0 = 16$. We suspect that since the results are finetuned from the same model that is pretrained with $s_0 = 16$, the large gap between the two different s_0's might have negative impact on finetuning performance. Nevertheless, the performance is still higher than 32K length models.

Table 3: Dev results of NarrativeQA on base model when scaling sequence length from 16K to 128K. s_0 is defined in second paragraph of §4.

Length	Runtime (ms)	s_0	r	EM	F1
16K	518.2 / 394.4 / 162.4	16	2419	5.9	16.6
32K	946.8 / 671.6 / 212.6	32	2791	6.6	17.5
32K	1027.9 / 751.0 / 298.0	16	3443	6.4	17.5
64K	1848.7 / 1177.2 / 254.8	64	2977	7.2	18.4
64K	2244.8 / 1574.2 / 659.4	16	5491	7.5	19.3
128K	6267.8 / 5125.9 / 1902.2	16	9587	8.0	19.6

Why focus on 4K - 128K lengths? We believe that the computation required by full Transformers at processing shorter sequences is not an efficiency bottleneck. As a result, we did not profile the performance of our method for smaller length sequences, since the standard Transformers are sufficiently fast in this case. Further, while our model can be applied to shorter sequences, we suspect that for shorter sequences, there might be less irrelevant information for VIP-tokens. So compressing the irrelevant information will not offer a meaningful speed up. This is a limitation of our method as the compression works better when there is more compressible information. We have only pushed

the sequence lengths to 128K since this length was sufficient to cover a majority of sequence lengths encountered in long sequence tasks (for example, our model is able to process an entire book at once).

Limitations. The motivation of our method is based on the assumption that in many tasks, a subset of tokens are disproportionately responsible for the model prediction, and the remainder of the tokens may play a role but less critical in comparison. Our method is designed to excel at specifically such tasks by selectively locating relevant information in the sequence for given VIP-tokens. As the experiments show, this choice is effective in many cases but this behavior is not universal. In some settings, an embedding is pre-computed which must then serve multiple tasks concurrently, e.g., both text retrieval and natural language inference. In this case, if we do not know the tasks beforehand, and so VIP-token selection cannot be meaningfully performed.

5 Conclusions

We propose a VIP-token centric sequence compression method to compress/decompress the input/output sequences of Transformer layers thereby reducing the complexity dependency on the sequence length n without sacrificing the model accuracy. Specifically, we construct compression with the goal of reducing the impact of this step on the output obtained from a small subset of important tokens. Our empirical evaluation shows that our method can be directly incorporated into existing pretrained models with some additional training. Also, it often has much higher efficiency compared to baselines with the same sequence length while offering better or competitive model accuracy. For future work, we believe that extending our method to the decoder of the encoder-decoder models might further boost the efficiency of Transformers while maintaining similar model performance.

References

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

[2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020.

[3] Mingda Chen, Zewei Chu, Sam Wiseman, and Kevin Gimpel. SummScreen: A dataset for abstractive screenplay summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8602–8615, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[4] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations (ICLR)*, 2021.

[5] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. A discourse-aware attention model for abstractive summarization of long documents. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 2018.

[6] Zihang Dai, Guokun Lai, Yiming Yang, and Quoc Le. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 4271–4282. Curran Associates, Inc., 2020.

[7] Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4599–4610, Online, June 2021. Association for Computational Linguistics.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[10] Alexander Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1074–1084, Florence, Italy, July 2019. Association for Computational Linguistics.

[11] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[12] Mandy Guo, Joshua Ainslie, David Uthus, Santiago Ontanon, Jianmo Ni, Yun-Hsuan Sung, and Yinfei Yang. LongT5: Efficient text-to-text transformer for long sequences. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 724–736, Seattle, United States, July 2022. Association for Computational Linguistics.

[13] Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1419–1436, Online, June 2021. Association for Computational Linguistics.

[14] Xin Huang, Ashish Khetan, Rene Bidart, and Zohar Karnin. Pyramid-BERT: Reducing complexity via successive core-set based token selection. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8798–8817, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[15] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations (ICLR)*, 2020.

[16] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *Proceedings of the 15th European Conference on Machine Learning*, ECML'04, page 217–226, Berlin, Heidelberg, 2004. Springer-Verlag.

[17] Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The NarrativeQA reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.

[18] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of Machine Translation Summit X: Papers*, pages 79–86, Phuket, Thailand, September 13-15 2005.

[19] Yuta Koreeda and Christopher Manning. ContractNLI: A dataset for document-level natural language inference for contracts. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 1907–1919, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[20] 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. *arXiv preprint arXiv:1907.11692*, 2019.

[21] Piotr Nawrot, Jan Chorowski, Adrian Łańcucki, and Edoardo M. Ponti. Efficient transformers with dynamic token pooling, 2022.

[22] Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, and Samuel Bowman. QuALITY: Question answering with long input texts, yes! In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5336–5358, Seattle, United States, July 2022. Association for Computational Linguistics.

[23] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. Random feature attention. In *International Conference on Learning Representations (ICLR)*, 2021.

[24] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[25] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*, 2019.

[26] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[27] Uri Shaham, Elad Segal, Maor Ivgi, Avia Efrat, Ori Yoran, Adi Haviv, Ankit Gupta, Wenhan Xiong, Mor Geva, Jonathan Berant, and Omer Levy. SCROLLS: Standardized comparison over long language sequences. In *EMNLP*, 2022.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[29] Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. Constructing datasets for multi-hop reading comprehension across documents. *Transactions of the Association for Computational Linguistics*, 6:287–302, 2018.

[30] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[31] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[32] 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 *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[33] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[34] Zhanpeng Zeng, Sourav Pal, Jeffery Kline, Glenn M Fung, and Vikas Singh. Multi resolution analysis (MRA) for approximate self-attention. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 25955–25972. PMLR, 17–23 Jul 2022.

[35] Zhanpeng Zeng, Yunyang Xiong, Sathya Ravi, Shailesh Acharya, Glenn M Fung, and Vikas Singh. You only sample (almost) once: Linear cost self-attention via bernoulli sampling. In *International Conference on Machine Learning (ICML)*, 2021.

[36] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip HS Torr, et al. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. *arXiv preprint arXiv:2012.15840*, 2020.

[37] Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. QMSum: A new benchmark for query-based multi-domain meeting summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5905–5921, Online, June 2021. Association for Computational Linguistics.

[38] Chenguang Zhu, Yang Liu, Jie Mei, and Michael Zeng. Mediasum: A large-scale media interview dataset for dialogue summarization. *arXiv preprint arXiv:2103.06410*, 2021.

6 Appendix

6.1 Definition of Notations

Table 1: Major notations used

Notation	Description
l	number of layers of a Transformer model
n	number of tokens of a input sequence
d	model embedding dimension
n_p	number of VIP-tokens
n_c	number of non-VIP/remaining tokens, so $n_p + n_c = n$
r	length of a compressed sequence
$\alpha(\cdot,\cdot)$	multi-head attention taking two inputs for query embeddings and key/value embeddings
$\beta(\cdot)$	two-layer feed-forward network
$\gamma(\cdot)$	function representing all heavy computation of a Transformer layer
$\mathbf{X}$	embedding matrix representing a input sequence
$\mathbf{P}$	embedding matrix representing the VIP-tokens
$\mathbf{C}$	embedding matrix representing the non-VIP/remaining tokens
$\mathbf{X}_{new}$	updated embedding matrix of a input sequence, the output of a Transformer layer
$\mathbf{P}_{new}$	updated embedding matrix representing the VIP-tokens
$\mathbf{C}_{new}$	updated embedding matrix representing the non-VIP/remaining tokens
$\mathbf{S}$	compression matrix
$\mathbf{S}_c$	compression submatrix for the non-VIP/remaining tokens
$\mathbf{S}^\dagger$	decompression matrix
$\mathbf{S}_c^\dagger$	decompression submatrix for the non-VIP/remaining tokens
$\mathcal{J}$	subset of components that is constructed via Alg. 1 and used for compression
$\mathbf{b}_x^s$	components used for 1-D wavelet transform of scaling s and translation x
$\mathbf{c}_x^s$	result of applying $\mathbf{b}_x^s$ to the non-VIP/remaining tokens, represents a local average of $\mathbf{C}$ over support region of $\mathbf{b}_x^s$
$(\mathbf{c}_{new})_x^s$	result of applying $\mathbf{b}_x^s$ to the non-VIP/remaining tokens, represents a local average of $\mathbf{C}_{new}$ over support region of $\mathbf{b}_x^s$
$\mathcal{T}(\cdot)$	data structure for storing the input sequence
$\Delta\mathbf{c}_x^s$	state stored in $\mathcal{T}(\mathbf{C})$ defined as $\Delta\mathbf{c}_x^s := \mathbf{c}_{\lceil x/2\rceil}^{2s} - \mathbf{c}_x^s$
$\Delta(\mathbf{c}_{new})_x^s$	state stored in $\mathcal{T}(\mathbf{C}_{new})$ defined as $\Delta(\mathbf{c}_{new})_x^s := (\mathbf{c}_{new})_{\lceil x/2\rceil}^{2s} - (\mathbf{c}_{new})_x^s$
$[\cdot]_i$	i-th entry/row of the input vector/matrix
$[\cdot]_{i,j}$	(i,j)-th entry of the input matrix

We provide a table 1 of notations that are used for more than once so that the readers can refer to their definition easily.

6.2 Details of Multi-Resolution Compression

We describe the omitted technical details of a modified formulation of [34] to construct $\mathbf{S}_c$ satisfying good approximation of

$$\exp(\mathbf{P}\mathbf{C}^\top\mathbf{S}_c^\top)\mathbf{S}_c \approx \exp(\mathbf{P}\mathbf{C}^\top). \tag{1}$$

Before diving into the technical details of constructing $\mathbf{S}_c$, we introduce some notations and tools that will be used later. We use $[\cdot]_i$ to refer the i-th entry/row of the input vector/matrix and $[\cdot]_{i,j}$ to refer the (i,j)-th entry of the input matrix. We use bold uppercase letters to denote matrices, bold lower case letters to denote vectors, and regular lower case letters to denote scalars.

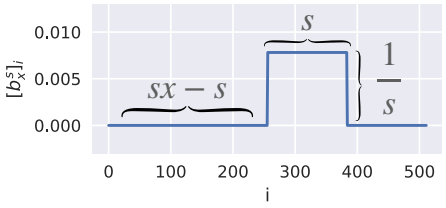

Figure 1: Visualization of B_x^s for some scaling s and translation x. The y axis for different plots are not the same.

6.2.1 Basic Problem Setup

Let $\mathbf{b}_x^s \in \mathbb{R}^{n_c}$ be a multi-resolution component defined as

$$[\mathbf{b}_x^s]_i := \begin{cases} \frac{1}{s} & \text{if } sx - s < i \le sx \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

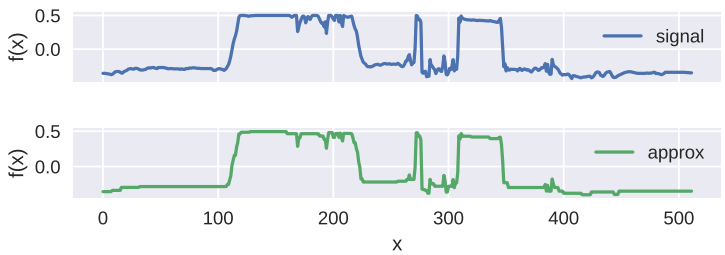

Figure 2: An example of approximating an 1D signal using a truncated wavelet transform with components defined in (2). It uses a set $\mathcal{J}$ of size 79 to represent a signal in $\mathbb{R}^{512}$.

for $s \in \{2^0, 2^1, 2^2, \cdots, n_c\}$ assuming n_c is a power of 2. Here, s and x represent the scaling and translation of the component, respectively. Fig. 1 is the visualization of $\mathbf{b}_x^s$. Any 1D signal $\mathbf{f} \in \mathbb{R}^{n_c}$ can be represented as a linear combination of $\mathbf{b}_x^s$:

$$\mathbf{f} = \sum_{s,x} c_x^s \mathbf{b}_x^s \tag{3}$$

where c_x^s are the coefficients for the linear combination. For a signal with multi-resolution structure (that is, signal has high frequency in some regions and has low frequency in other regions), we can find an approximation $\hat{f}^*$ that can be expressed as a *sparse* linear combination where most coefficients are zeros, as shown in Fig. 2.

$$\mathbf{f} \approx \hat{\mathbf{f}}^* := \sum_{\mathbf{b}_x^s \in \mathcal{J}} c_x^s \mathbf{b}_x^s \tag{4}$$

We denote $\mathcal{J}$ as the set of major components $\mathbf{b}_x^s$ corresponding to the large coefficients, that is, $\mathcal{J} := \{\mathbf{b}_x^s \mid |c_x^s| \text{ being large}\}$. Since the set of all possible $\mathbf{b}_x^s$ is an over-complete dictionary, there are multiple possible linear combinations. To reduce the search space of the best set $\mathcal{J}$, we place a mild restriction on the set $\mathcal{J}$:

$$\sum_{\mathbf{b}_x^s \in \mathcal{J}} s\mathbf{b}_x^s = \mathbf{1} \qquad \langle \mathbf{b}_x^s, \mathbf{b}_{x'}^{s'} \rangle = 0 \quad \forall \mathbf{b}_x^s, \mathbf{b}_{x'}^{s'} \in \mathcal{J}, \mathbf{b}_x^s \neq \mathbf{b}_{x'}^{s'} \tag{5}$$

The conditions state that each entry of signal $\mathbf{f}$ is included in the support region of exactly one component in $\mathcal{J}$. With these tools, we will first describe the approximation when $\mathcal{J}$ is given, then discuss how the approximation connects the set $\mathcal{J}$ to our target $\mathbf{S}_c$ and $\mathbf{S}_c\mathbf{C}$. Finally, we will discuss how to construct this $\mathcal{J}$.

6.2.2 Plugging Our Problem into the Setup

A recent result shows that the self-attention matrix $\exp(\mathbf{X}\mathbf{X}^\top)$ has the multi-resolution structure discussed above [34]. Since $\exp(\mathbf{P}\mathbf{C}^\top)$ is a sub-matrix of $\exp(\mathbf{X}\mathbf{X}^\top)$, we conjecture that the multi-resolution structure also holds in $\exp(\mathbf{P}\mathbf{C}^\top)$. As a result, we can find a sparse combination of $\mathbf{b}_x^s$ to represent rows of $\exp(\mathbf{P}\mathbf{C}^\top)$.

Claim 6.1. *Given the set $\mathcal{J}$ satisfying restriction* (5)*, we can define an approximation of the i-th row of $\exp(\mathbf{P}\mathbf{C}^\top)$ similar to* (4) *as illustrated in Fig. 2*

$$\left[\widehat{\exp(\mathbf{P}\mathbf{C}^\top)}^*\right]_i := \sum_{\mathbf{b}_x^s \in \mathcal{J}} c_x^s \mathbf{b}_x^s \tag{6}$$

where c_x^s is the optimal solution that minimizes

$$\left\| \left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_i - \left[\widehat{\exp(\mathbf{P}\mathbf{C}^\top)}^*\right]_i \right\|_2^2 \tag{7}$$

Then, the approximation can be written as:

$$\left[\widehat{\exp(\mathbf{P}\mathbf{C}^\top)}^*\right]_{i,j} = \langle \left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_i, \mathbf{b}_x^s \rangle \tag{8}$$

where $\mathbf{b}_x^s \in \mathcal{J}$ is the component that is supported on j (a.k.a. $[\mathbf{b}_x^s]_j \neq 0$ and there is exactly one $\mathbf{b}_x^s \in \mathcal{J}$ satisfy this condition due to restriction (5)*).*

Proof. If $\mathcal{J}$ is given, let $\mathbf{B} \in \mathbb{R}^{n_c \times |\mathcal{J}|}$ be a matrix whose columns are elements $\mathbf{b}_x^s \in \mathcal{J}$ and let $\mathbf{c} \in \mathbb{R}^{|\mathcal{J}|}$ be a vector whose entries are the corresponding c_x^s:

$$\begin{aligned} \mathbf{B} &:= \begin{bmatrix} \mathbf{b}_{x_1}^{s_1} & \mathbf{b}_{x_2}^{s_2} & \cdots & \mathbf{b}_{x_{|\mathcal{J}|}}^{s_{|\mathcal{J}|}} \end{bmatrix} \\ \mathbf{c} &:= \begin{bmatrix} c_{x_1}^{s_1} & c_{x_2}^{s_2} & \cdots & c_{x_{|\mathcal{J}|}}^{s_{|\mathcal{J}|}} \end{bmatrix}^\top \end{aligned} \tag{9}$$

then the approximation can be expressed as

$$\left[\widehat{\exp(\mathbf{P}\mathbf{C}^\top)}^*\right]_i = \sum_{\mathbf{b}_x^s \in \mathcal{J}} c_x^s \mathbf{b}_x^s = \mathbf{B}\mathbf{c} \tag{10}$$

If we solve for

$$\mathbf{c} := \arg\min_\beta \left\| \left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_i - \mathbf{B}\beta \right\| \tag{11}$$

then

$$\mathbf{c} = (\mathbf{B}^\top\mathbf{B})^{-1}\mathbf{B}^\top \left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_i \tag{12}$$

Due to the restriction (5), the columns of $\mathbf{B}$ are orthogonal, so $\mathbf{B}^\top\mathbf{B}$ is a diagonal matrix:

$$\mathbf{B}^\top\mathbf{B} = \begin{bmatrix} 1/s_1 & & & \\ & 1/s_2 & & \\ & & \cdots & \\ & & & 1/s_{|\mathcal{J}|} \end{bmatrix} \tag{13}$$

We can also write down $\mathbf{B}^\top \left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_i$

$$\mathbf{B}^\top \left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_i = \begin{bmatrix} \langle \left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_i, \mathbf{b}_{x_1}^{s_1} \rangle \\ \langle \left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_i, \mathbf{b}_{x_2}^{s_2} \rangle \\ \cdots \\ \langle \left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_i, \mathbf{b}_{x_{|\mathcal{J}|}}^{s_{|\mathcal{J}|}} \rangle \end{bmatrix} \tag{14}$$

Putting everything together, we have

$$\mathbf{B}\mathbf{c} = \begin{bmatrix} s_1\mathbf{b}_{x_1}^{s_1} & s_2\mathbf{b}_{x_2}^{s_2} & \cdots & s_{|\mathcal{J}|}\mathbf{b}_{x_{|\mathcal{J}|}}^{s_{|\mathcal{J}|}} \end{bmatrix} \begin{bmatrix} \langle \left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_i, \mathbf{b}_{x_1}^{s_1} \rangle \\ \langle \left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_i, \mathbf{b}_{x_2}^{s_2} \rangle \\ \cdots \\ \langle \left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_i, \mathbf{b}_{x_{|\mathcal{J}|}}^{s_{|\mathcal{J}|}} \rangle \end{bmatrix} \tag{15}$$

We note that $s\mathbf{b}_x^s$ simply re-scale the entry of $\mathbf{b}_x^s$ such that any non-zero entry becomes 1. Then, let us consider j-th entry of $\mathbf{B}\mathbf{c}$. Due to the restriction (5), we have exactly one $\mathbf{b}_x^s \in \mathcal{J}$ whose support region contains j, so the j-th row of the first matrix at the right hand side of (15) contains exactly a 1 and the remaining entries are 0. Therefore, we have

$$\left[\widehat{\exp(\mathbf{P}\mathbf{C}^\top)}^*\right]_{i,j} = [\mathbf{B}\mathbf{c}]_j = \langle \left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_i, \mathbf{b}_x^s \rangle \tag{16}$$

where $\mathbf{b}_x^s \in \mathcal{J}$ is the component that is supported on j, which concludes our proof.

□

6.2.3 Efficient Approximation

We note that computing (8) for all j would require access to the entire $\left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_i$. We exploit the same strategy as described in [34], so the exponential of inner product is used as an approximation to inner product of exponential.

$$\left[\widehat{\exp(\mathbf{P}\mathbf{C}^\top)}\right]_{i,j} := \exp(\langle \left[\mathbf{P}\mathbf{C}^\top\right]_i, \mathbf{b}_x^s \rangle) \tag{17}$$

We note that $\langle \left[\mathbf{P}\mathbf{C}^\top\right]_i, \mathbf{b}_x^s \rangle$ is the local average of $\left[\mathbf{P}\mathbf{C}^\top\right]_i$ over the support region of $\mathbf{b}_x^s$.

By using some arithmetic manipulations, (17) can be efficiently computed

$$\begin{aligned} \exp(\langle [\mathbf{P}\mathbf{C}^\top]_i, \mathbf{b}_x^s \rangle) &= \exp(\frac{1}{s} \sum_{[\mathbf{b}_x^s]_j \neq 0} \left[\mathbf{P}\mathbf{C}^\top\right]_{i,j}) = \exp(\frac{1}{s} \sum_{(\mathbf{b}_x^s)_j \neq 0} \langle [\mathbf{P}]_i, [\mathbf{C}]_j \rangle) \\ &= \exp(\langle [\mathbf{P}]_i, \frac{1}{s} \sum_{[\mathbf{b}_x^s]_j \neq 0} [\mathbf{C}]_j \rangle) = \exp(\langle [\mathbf{P}]_i, \mathbf{c}_x^s \rangle) \end{aligned} \tag{18}$$

where $\mathbf{c}_x^s$ is defined as

$$\mathbf{c}_x^s := \mathbf{b}_x^s \mathbf{C} \tag{19}$$

and can be efficiently computed via

$$\mathbf{c}_x^s = \frac{1}{2}\mathbf{c}_{2x-1}^{s/2} + \frac{1}{2}\mathbf{c}_{2x}^{s/2} \qquad \mathbf{c}_x^1 = [\mathbf{C}]_x \tag{20}$$

We note that $\mathbf{c}_x^s$ a local average of the rows of $\mathbf{C}$ over support region of $\mathbf{b}_x^s$.

Claim 6.2. *Given the set $\mathcal{J}$ satisfying restriction* (5)*, let $\mathbf{S}_c$ be be a matrix whose rows are elements $\mathbf{b}_x^s \in \mathcal{J}$*

$$\mathbf{S}_c = \begin{bmatrix} \mathbf{b}_{x_1}^{s_1} \\ \mathbf{b}_{x_2}^{s_2} \\ \cdots \\ \mathbf{b}_{x_{|\mathcal{J}|}}^{s_{|\mathcal{J}|}} \end{bmatrix} \qquad \mathbf{D} = \begin{bmatrix} s_1 & & & \\ & s_2 & & \\ & & \cdots & \\ & & & s_{|\mathcal{J}|} \end{bmatrix} \tag{21}$$

Then,

$$\exp(\mathbf{P}\mathbf{C}^\top \mathbf{S}_c^\top)\mathbf{D}\mathbf{S}_c = \widehat{\exp(\mathbf{P}\mathbf{C}^\top)} \tag{22}$$

where $\widehat{\exp(\mathbf{P}\mathbf{C}^\top)}$ is defined as (17)*.*

Proof. Consider i-th row of $\exp(\mathbf{P}\mathbf{C}^\top \mathbf{S}_c^\top)$,

$$\begin{aligned} \left[\exp(\mathbf{P}(\mathbf{S}_c\mathbf{C})^\top)\right]_i &= \exp([\mathbf{P}]_i (\mathbf{S}_c\mathbf{C})^\top) \\ &= \exp([\mathbf{P}]_i \begin{bmatrix} \mathbf{c}_{x_1}^{s_1} & \mathbf{c}_{x_2}^{s_2} & \cdots & \mathbf{c}_{x_{|\mathcal{J}|}}^{s_{|\mathcal{J}|}} \end{bmatrix}) \\ &= \begin{bmatrix} \exp(\langle [\mathbf{P}]_i, \mathbf{c}_{x_1}^{s_1} \rangle) & \cdots & \exp(\langle [\mathbf{P}]_i, \mathbf{c}_{x_{|\mathcal{J}|}}^{s_{|\mathcal{J}|}} \rangle) \end{bmatrix} \end{aligned} \tag{23}$$

Then, we have

$$\left[\exp(\mathbf{P}(\mathbf{S}_c\mathbf{C})^\top)\mathbf{D}\mathbf{S}_c\right]_i = \begin{bmatrix} \exp(\langle [\mathbf{P}]_i, \mathbf{c}_{x_1}^{s_1} \rangle) & \cdots & \exp(\langle [\mathbf{P}]_i, \mathbf{c}_{x_{|\mathcal{J}|}}^{s_{|\mathcal{J}|}} \rangle) \end{bmatrix} \begin{bmatrix} s_1 \mathbf{b}_{x_1}^{s_1} \\ \cdots \\ s_{|\mathcal{J}|} \mathbf{b}_{x_{|\mathcal{J}|}}^{s_{|\mathcal{J}|}} \end{bmatrix} \tag{24}$$

We note that $s\mathbf{b}_x^s$ simply re-scales the entry of $\mathbf{b}_x^s$ such that any non-zero entry becomes 1. Then, let us consider j-th entry of $\left[\exp(\mathbf{P}\mathbf{C}^\top \mathbf{S}_c^\top)\mathbf{D}\mathbf{S}_c\right]_i$. Due to the restriction (5), we have exactly one $\mathbf{b}_x^s \in \mathcal{J}$ whose support region contains j, so the j-th column of the second matrix in the right hand side of (24) contains exactly a 1 and the remaining entries are 0. Therefore, we have

$$\left[\exp(\mathbf{P}(\mathbf{S}_c\mathbf{C})^\top)\mathbf{D}\mathbf{S}_c\right]_{i,j} = \exp(\langle [\mathbf{P}]_i, \mathbf{c}_x^s \rangle) = \exp(\langle [\mathbf{P}\mathbf{C}^\top]_i, \mathbf{b}_x^s \rangle) = \left[\widehat{\exp(\mathbf{P}\mathbf{C}^\top)}\right]_{i,j} \tag{25}$$

where $\mathbf{b}_x^s \in \mathcal{J}$ is the component that is supported on j. The second equality is based on (18).

□

Claim 6.3. *If $\mathbf{S}_c$ and $\mathbf{D}$ are defined as Claim 6.2, the pseudo inverse of $\mathbf{S}_c$ is simply $\mathbf{S}_c^\dagger = \mathbf{S}_c^\top \mathbf{D}$, so each row of $S^\dagger$ contain exactly a* 1*.*

Proof. Since each row of $\mathbf{S}_c$ is some $\mathbf{b}_x^s \in \mathcal{J}$, due to the restriction (5), for $i \neq j$,

$$\begin{aligned} \left[\mathbf{S}_c\mathbf{S}_c^\top\mathbf{D}\right]_{i,i} &= \langle [\mathbf{S}_c]_i, [\mathbf{D}\mathbf{S}_c]_i \rangle = \langle \mathbf{b}_{x_i}^{s_i}, s_i \mathbf{b}_{x_i}^{s_i} \rangle = s_i \frac{1}{s_i} = 1 \\ \left[\mathbf{S}_c\mathbf{S}_c^\top\mathbf{D}\right]_{i,j} &= \langle [\mathbf{S}_c]_i, [\mathbf{D}\mathbf{S}_c]_j \rangle = \langle \mathbf{b}_{x_i}^{s_i}, s_j \mathbf{b}_{x_i}^{s_j} \rangle = s_j 0 = 0 \end{aligned} \tag{26}$$

As a result, $\mathbf{S}_c\mathbf{S}_c^\top\mathbf{D} = \mathbf{I}$. Further, $\mathbf{S}_c^\top\mathbf{D}\mathbf{S}_c$ is a symmetric matrix. So, all Moore-Penrose conditions are verified. $\mathbf{S}_c^\dagger = \mathbf{S}_c^\top\mathbf{D}$. The second statement follows from the restriction (5) and the fact that $s\mathbf{b}_x^s$ simply re-scales the entry of $\mathbf{b}_x^s$ such that any non-zero entry becomes 1.

□

At the end, the approximation

$$\widehat{\exp(\mathbf{P}\mathbf{C}^\top)} = \exp(\mathbf{P}\mathbf{C}^\top \mathbf{S}_c^\top)\mathbf{D}\mathbf{S}_c \approx \exp(\mathbf{P}\mathbf{C}^\top) \tag{27}$$

does not look exactly as (1), but we can insert a simple diagonal matrix $\mathbf{D}$ to the formulation (1) and make the whole thing work.

6.2.4 How to Construct $\mathcal{J}$?

The derivation so far assume access to $\mathcal{J}$, but in practice, we have no knowledge of $\mathcal{J}$ and need to construct $\mathcal{J}$ that leads to good approximation. With the approximation scheme in place, we can now analyze the approximation error, which will be leveraged later to find a reasonable set of components $\mathcal{J}$. The approximation error of i-th row of $\exp(\mathbf{P}\mathbf{C}^\top)$ can be expressed as

$$\begin{aligned}
\mathcal{E}_i &:= \left\| \left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_i - \left[\widehat{\exp(\mathbf{P}\mathbf{C}^\top)}\right]_i \right\|_F^2 \\
&= \sum_j \left(\left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_{i,j} - \left[\widehat{\exp(\mathbf{P}\mathbf{C}^\top)}\right]_{i,j}\right)^2 \\
&= \sum_{\mathbf{b}_x^s \in \mathcal{J}} \sum_{[\mathbf{b}_x^s]_j \neq 0} \left(\left[\exp(\mathbf{P}\mathbf{C}^\top)\right]_{i,j} - \exp(\langle [\mathbf{P}\mathbf{C}^\top]_i, \mathbf{b}_x^s \rangle)\right)^2 \\
&= \sum_{B_x^s \in \mathcal{J}} \exp(\langle [\mathbf{P}\mathbf{C}^\top]_i, \mathbf{b}_x^s \rangle) \sum_{(B_x^s)_j \neq 0} \left(\exp(\left[\mathbf{P}\mathbf{C}^\top\right]_{i,j} - \langle [\mathbf{P}\mathbf{C}^\top]_i, \mathbf{b}_x^s \rangle) - 1\right)^2 \\
&= \sum_{B_x^s \in \mathcal{J}} \exp(\langle [\mathbf{P}]_i, \mathbf{c}_x^s \rangle) \sum_{(B_x^s)_j \neq 0} \left(\exp(\left[\mathbf{P}\mathbf{C}^\top\right]_{i,j} - \langle [\mathbf{P}\mathbf{C}^\top]_i, \mathbf{b}_x^s \rangle) - 1\right)^2
\end{aligned} \tag{28}$$

The approximation error is governed by two components multiply together: attention score between $[\mathbf{P}]_i$ and the local average $\mathbf{c}_x^s$ of $\mathbf{C}$ over support region of $\mathbf{b}_x^s$ and the amount of deviation of $\left[\mathbf{P}\mathbf{C}^\top\right]_{i,j}$ from its local average $\langle [\mathbf{P}\mathbf{C}^\top]_i, \mathbf{b}_x^s \rangle$.

It is reasonable to assume that this deviation is smaller if s is smaller. When $s = 1$, the deviation is simply zero. Therefore, this actually suggests a simple heuristic for selecting $\mathcal{J}$: when $\langle [\mathbf{P}]_i, \mathbf{c}_x^s \rangle$ is large, we should approximate the support region of $\mathbf{b}_x^s$ with higher resolution. This heuristic describes the selection criteria for one row of $\exp(\mathbf{P}\mathbf{C}^\top)$, for multiple rows of $\exp(\mathbf{P}\mathbf{C}^\top)$, we simply use

$$\mu_x^s = \sum_{i=1}^{n_p} \exp(\langle [\mathbf{P}]_i, \mathbf{c}_x^s \rangle) \tag{29}$$

as selection criteria since $\mathcal{J}$ is shared by all VIP-tokens.

Algorithm 1 Constructing $\mathcal{J}$

Input: VIP-tokens $\mathbf{P}$ and $\mathbf{c}_x^s$ for all s and x
Input: Budgets $m^1, m^2, m^4, \cdots, m^{n_c}$ for each s
Initialize empty $\mathcal{J}$
Set $s \leftarrow n_c$
Compute μ_1^s (29) and add $\mathbf{b}_1^s$ (2) to $\mathcal{J}$
for $s \leftarrow n_c/2, n_c/4, \cdots, 1$ **do**
  Pop m^s elements $\mathbf{b}_x^{2s}$ with the largest μ_x^{2s} (29)
  for each $\mathbf{b}_x^{2s}$ **do**
    Compute μ_{2x-1}^s, μ_{2x}^s (29) and add $\mathbf{b}_{2x-1}^s, \mathbf{b}_{2x}^s$ (2) to $\mathcal{J}$
  end for
end for
Output: $\mathcal{J}$

The construction of $\mathcal{J}$ is described in Alg. 1. The budgets $m^1, m^2, m^4, \cdots, m^{n_c}$ required by Alg. 1 is used determine the number of components at each resolution that will be added to $\mathcal{J}$. Specifically, there are $2m^s - m^{2s}$ number of components $\mathbf{b}_x^s$ for scaling $s \neq n_c$ based on simple calculations. We choose budgets such that the final size of $\mathcal{J}$ is $r - n_p$ to make the length of compressed sequence to be r.

6.2.5 How Good is This Approximation?

At high level, the compression $\mathbf{S}_c$ performs more compression on tokens that are not relevant to the VIP-tokens and less compression to tokens that are important to the VIP-tokens. We will discuss it in more details. Since each row of $S^\dagger$ contain exactly a 1 as stated in Claim 6.3, $S^\dagger$ can commute with β, so in summary, we can write the approximation of the computation of a Transformer layer as

$$\begin{aligned}
\alpha(\mathbf{P}, \mathbf{S}\mathbf{X}) &= \exp(\mathbf{P}\mathbf{P}^\top)\mathbf{P} + \exp(\mathbf{P}\mathbf{C}^\top\mathbf{S}_c^\top)\mathbf{D}\mathbf{S}_c\mathbf{C} \\
\mathbf{S}_c^\dagger \alpha(\mathbf{S}_c\mathbf{C}, \mathbf{S}\mathbf{X}) &= \mathbf{S}_c^\dagger \exp(\mathbf{S}_c\mathbf{C}\mathbf{P}^\top)\mathbf{P} + \mathbf{S}_c^\dagger \exp(\mathbf{S}_c\mathbf{C}\mathbf{C}^\top\mathbf{S}_c^\top)\mathbf{D}\mathbf{S}_c\mathbf{C} \\
\begin{bmatrix} \mathbf{P}_{new} \\ \mathbf{C}_{new} \end{bmatrix} &= \begin{bmatrix} \beta(\alpha(\mathbf{P}, \mathbf{S}\mathbf{X}) + \mathbf{P}) + \alpha(\mathbf{P}, \mathbf{S}\mathbf{X}) \\ \beta(\mathbf{S}_c^\dagger \alpha(\mathbf{S}_c\mathbf{C}, \mathbf{S}\mathbf{X}) + \mathbf{S}_c^\dagger\mathbf{S}_c\mathbf{C}) + \mathbf{S}_c^\dagger \alpha(\mathbf{S}_c\mathbf{C}, \mathbf{S}\mathbf{X}) \end{bmatrix} + \begin{bmatrix} \mathbf{P} \\ \mathbf{C} \end{bmatrix}
\end{aligned} \tag{30}$$

Note that $\mathbf{D}$ is added as discussed in (27).

There are four main approximation components (purple) in (30). Taking the fact that $\mathbf{D}\mathbf{S}_c = (\mathbf{S}_c^\dagger)^\top$, all of these approximations are row or column space multi-resolution approximations governed by $\mathbf{S}_c$ matrix. High attention weight implies higher dependency, and the procedure in Alg. 1 refines regions with large attention weights with higher resolutions. Therefore, the token embedding in $\mathbf{C}$ that have higher dependency to $\mathbf{P}$ are better approximated. The output $\mathbf{P}_{new}$ is well approximated by design since the approximation preserves the higher frequency components of the subset of rows of $\mathbf{C}$ that has high impact on the output $\mathbf{P}_{new}$. Further, the output in $\mathbf{C}_{new}$ corresponding to the subset of rows of $\mathbf{C}$ that have higher dependency with the VIP-tokens will have better approximation than the remaining rows of $\mathbf{C}$. This property addresses the issue that some tokens with unknown locations are also relevant to the final prediction of a Transformer in some tasks. For example, in question answering tasks, candidate answers are usually expected to have large dependency with question tokens (VIP-tokens), so they are approximated well as well. This approximation property is exactly what we need.

6.2.6 Relation to [34] that Inspires Multi-Resolution Compression

Our work and [34] can be viewed as operating at slightly different levels of abstractions. While [34] tries to approximate self-attention computation efficiently, our paper proposes a general framework for performing a VIP-token centric compression on the sequence to efficiently handle extremely long sequences (the self-attention module remains completely unchanged). Our VIP-token centric compression involves a number of steps described in the main text. But one of the key steps involves constructing a compression matrix $\mathbf{S}_c$ which has some desirable properties, namely satisfying (1) which we elaborate further below.

Algorithm 2 One layer computation with $\mathcal{T}(\mathbf{C})$

Input: VIP-tokens $\mathbf{P}$ and cache $\mathcal{T}(\mathbf{C})$
Use Algo. 1 to construct $\mathcal{J}$ but use (35) to retrieve $\mathbf{c}_x^s$ from $\mathcal{T}(\mathbf{C})$
Construct $\mathbf{S}_c, \mathbf{S}_c\mathbf{C}$ associated with $\mathcal{J}$ using Claim 6.2
Compute $\mathbf{P}_{new} = \beta(\alpha(\mathbf{P}, \mathbf{S}\mathbf{X}) + \mathbf{P}) + \alpha(\mathbf{P}, \mathbf{S}\mathbf{X}) + \mathbf{P}$
Set $\mathcal{T}(\mathbf{C}_{new}) \leftarrow \mathcal{T}(\mathbf{C})$
for $s \leftarrow 1, 2, 4, \cdots n_c/2$ **do**
  for $\mathbf{b}_x^s \in \mathcal{J}$ **do**
    Locate row location $\mathbf{b}_x^s$ in $\mathbf{S}_c$, refer the index as j
    Compute $(\mathbf{c}_{new})_x^s = [\beta(\alpha(\mathbf{S}_c\mathbf{C}, \mathbf{S}\mathbf{X}) + \mathbf{S}_c\mathbf{C}) + \alpha(\mathbf{S}_c\mathbf{C}, \mathbf{S}\mathbf{X})]_j + \mathbf{c}_x^s$
    Mark $(\mathbf{c}_{new})_x^s$ dirty
  end for
  for dirty $(\mathbf{c}_{new})_x^s$ **do**
    Compute $(\mathbf{c}_{new})_{\lceil x/2 \rceil}^{2s}$ and update $(\mathbf{c}_{new})_x^s$
    Mark $(\mathbf{c}_{new})_{\lceil x/2 \rceil}^{2s}$ dirty
  end for
end for
Update $(\mathbf{c}_{new})_1^{n_c}$
Output: VIP-tokens $\mathbf{P}_{new}$ and cache $\mathcal{T}(\mathbf{C}_{new})$

Note that for equation (1), we need a matrix $\mathbf{S}_c$ such that the approximated attention matrix involving $\mathbf{P}$ and $\mathbf{C}$ is similar to the true attention matrix involving $\mathbf{P}$ and $\mathbf{C}$. This is precisely where the general idea of [34] can be used. But the formulation in [34] cannot be applied directly in its original form since it cannot give us $\mathbf{S}_c$. Why? One reason is that the formulation in [34] cannot be easily written as matrix operations similar to equation (1). This may be a reason why [34] has to use custom CUDA kernels in their implementation. Nonetheless, the properties of [34] are useful. So we derive the analogous form but for 1D instead: this 1D case is expressed as applying a matrix (this is the $\mathbf{S}_c$ we are looking for) to the signal $\mathbf{C}$.

One bonus of this modification is that it also removes the need for custom CUDA kernels. At a high level, [34] offers a multi-resolution view of the self-attention matrices, and our modified version is best thought of as a similar multi-resolution view of the sequence itself. But we can also substitute in a different means of obtaining $\mathbf{S}_c$ (which could simply be a sketching matrix). Finally, we note that a naive implementation of the resulting modification still requires a $O(n_c d)$ cost due to the computation of $\mathbf{c}_x^s$ for all possible scaling s and translation x. There is a similar cost in [34] (second paragraph in section 4.4 in [34]). The data structure we propose reduces this cost.

6.3 Details of Proposed Data Structure

In section, we describe some omitted technical details of the proposed data structure $\mathcal{T}(\cdot)$.

Why $(\mathbf{c}_{new})_1^1 - \mathbf{c}_1^1 = (\mathbf{c}_{new})_2^1 - \mathbf{c}_2^1 = (\mathbf{c}_{new})_1^2 - \mathbf{c}_1^2$ if $\mathbf{b}_1^2 \in \mathcal{J}$?

Claim 6.4. *Given the set $\mathcal{J}$ satisfying restriction* (5)*, $\mathbf{b}_x^s \in \mathcal{J}$, then $(\mathbf{c}_{new})_x^s - \mathbf{c}_x^s = (\mathbf{c}_{new})_{x'}^{s'} - \mathbf{c}_{x'}^{s'}$ for all $\mathbf{b}_{x'}^{s'}$ satisfying the support of $\mathbf{b}_{x'}^{s'}$ is contained in the support of $\mathbf{b}_x^s$.*

Proof. To simplify the notations a bit, without loss of generality, we assume $x = 1$. Then, for $i \leq s$, consider $(\mathbf{c}_{new})_i^1$:

$$\begin{aligned}(\mathbf{c}_{new})_i^1 = [\mathbf{C}_{new}]_i &= \left[\mathbf{S}_c^\dagger \beta(\alpha(\mathbf{S}_c\mathbf{C}, \mathbf{S}\mathbf{X}) + \mathbf{S}_c\mathbf{C}) + \mathbf{S}_c^\dagger \alpha(\mathbf{S}_c\mathbf{C}, \mathbf{S}\mathbf{X})\right]_i + [\mathbf{C}]_i \\ &= \left[\mathbf{S}_c^\dagger\right]_i \beta(\alpha(\mathbf{S}_c\mathbf{C}, \mathbf{S}\mathbf{X}) + \mathbf{S}_c\mathbf{C}) + \left[\mathbf{S}_c^\dagger\right]_i \alpha(\mathbf{S}_c\mathbf{C}, \mathbf{S}\mathbf{X}) + \mathbf{c}_i^1\end{aligned} \tag{31}$$

By Claim 6.3, $\mathbf{S}_c^\dagger = \mathbf{S}_c^\top \mathbf{D}$ and i-th row of $\mathbf{S}_c^\dagger$ contains exactly a 1. The column that contains 1 in the i-th row of $\mathbf{S}_c^\dagger$ is exactly $s\mathbf{b}_1^s$ since i is contained in the support of exactly one components in $\mathcal{J}$ due to the restriction (5). Denote this column index as j, then

$$(\mathbf{c}_{new})_i^1 = [\beta(\alpha(\mathbf{S}_c\mathbf{C}, \mathbf{S}\mathbf{X}) + \mathbf{S}_c\mathbf{C})]_j + [\alpha(\mathbf{S}_c\mathbf{C}, \mathbf{S}\mathbf{X})]_j + \mathbf{c}_i^1 \tag{32}$$

Note that this holds for all $i \leq s$. As a result, for $i, i' \leq s$,

$$(\mathbf{c}_{new})_i^1 - \mathbf{c}_i^1 = [\beta(\alpha(\mathbf{S}_c\mathbf{C}, \mathbf{S}\mathbf{X}) + \mathbf{S}_c\mathbf{C})]_j + [\alpha(\mathbf{S}_c\mathbf{C}, \mathbf{S}\mathbf{X})]_j = (\mathbf{c}_{new})_{i'}^1 - \mathbf{c}_{i'}^1 \tag{33}$$

Then,

$$\begin{aligned}(\mathbf{c}_{new})_{\lceil i/2 \rceil}^2 - \mathbf{c}_{\lceil i/2 \rceil}^2 &= \frac{1}{2}(\mathbf{c}_{new})_{2\lceil i/2 \rceil - 1}^1 + \frac{1}{2}(\mathbf{c}_{new})_{2\lceil i/2 \rceil}^1 - \frac{1}{2}\mathbf{c}_{2\lceil i/2 \rceil - 1}^1 - \frac{1}{2}\mathbf{c}_{2\lceil i/2 \rceil}^1 \\ &= \frac{1}{2}((\mathbf{c}_{new})_{2\lceil i/2 \rceil - 1}^1 - \mathbf{c}_{2\lceil i/2 \rceil - 1}^1) + \frac{1}{2}((\mathbf{c}_{new})_{2\lceil i/2 \rceil}^1 - \mathbf{c}_{2\lceil i/2 \rceil}^1) \\ &= (\mathbf{c}_{new})_{2\lceil i/2 \rceil - 1}^1 - \mathbf{c}_{2\lceil i/2 \rceil - 1}^1\end{aligned} \tag{34}$$

The rest follows from induction.

□

6.3.1 Algorithm for Making $\mathcal{T}(\mathbf{C})$ into $\mathcal{T}(\mathbf{C}_{new})$

In this section, we describe the exact algorithm to update $\mathcal{T}(\mathbf{C})$ into $\mathcal{T}(\mathbf{C}_{new})$. The pseudo code is described in Alg. 2 where $\mathbf{c}_x^s$ is computed via

$$\mathbf{c}_x^s = \mathbf{c}_{\lceil x/2 \rceil}^{2s} - \Delta\mathbf{c}_x^s = \mathbf{c}_{\lceil x/4 \rceil}^{4s} - \Delta\mathbf{c}_{\lceil x/2 \rceil}^{2s} - \Delta\mathbf{c}_x^s = \cdots \tag{35}$$

We use the term "dirty" in Alg. 2 to indicate the node needs to be handled due to node updates. This term is commonly used in computer cache implementations to indicate that the data of a specific location has been updated and needs to be accounted for.

6.4 Complexity Analysis

In this section, we will discuss the detailed complexity analysis of our proposed method. The overall complexity of our proposed method is $\mathcal{O}(lrd^2 + lr^2d + lr\log(n_c)d + lrn_pd + nd)$ when using the proposed efficient data structure.

6.4.1 Preparing Input Sequence to $\mathcal{T}(\mathbf{C})$: $\mathcal{O}(nd)$

At the first layer, we need to permute the rows of $\mathbf{X}$ into $[\mathbf{P}; \mathbf{C}]$, which takes $\mathcal{O}(nd)$ cost. Then, we process $\mathbf{C}$ into $\mathcal{T}(\mathbf{C})$ by computing $\mathbf{c}_x^s$ defined in (20) sequentially from $\mathbf{c}_x^2$ for all x, $\mathbf{c}_x^4$ for all x, ..., $\mathbf{c}_1^{n_c}$ and then computing $\Delta\mathbf{c}_x^s$ for all s and x. The amount of cost is the same as the number of nodes in the tree $\mathcal{T}(\mathbf{C})$, so the cost is $\mathcal{O}(n_cd)$. Note that $n_c < n$, so the overall complexity of the above operations is $\mathcal{O}(nd)$.

6.4.2 Constructing $\mathcal{J}, \mathbf{S}_c, \mathbf{S}_c\mathbf{C}$: $\mathcal{O}(lr\log(n_c)d + lrn_pd)$

We can analyze the complexity of constructing $\mathcal{J}$ using Algo. 1. There is only one possible $\mu_x^{n_c}$. Then at scale s for $s \neq n_c$, there are $2m^s$ number of μ_x^s being computed since there are 2 components $\mathbf{b}_{x'}^s$ for each $\mathbf{b}_x^{2s}$. As a result, we need to compute $\mathcal{O}(1 + \sum_s 2m^s) = \mathcal{O}(\sum_s m^s)$ number of μ_x^s. When $\mathbf{c}_x^s$ is given, the cost of computing a μ_x^s is $\mathcal{O}(n_pd)$, so the overall cost of constructing $\mathcal{J}$

is $\mathcal{O}((1+\sum_{i=1}^{k} 2m_i)n_p d)$. Further, at scale s for $s \neq n_c$, the size of $\mathcal{J}$ is increased by m^s, so the size of $\mathcal{J}$ is $\mathcal{O}(\sum_s m^s)$. Since $\mathbf{S}_c \in \mathbb{R}^{(r-n_p)\times n}$ and $|\mathcal{J}| = r - n_p$ as discussed in §6.2.4, $\mathcal{O}(r-n_p) = \mathcal{O}(\sum_s m^s)$. We use $\mathcal{O}(r)$ for simplicity instead of $\mathcal{O}(r-n_p)$. As a result, the overall cost of constructing $\mathcal{J}$ is $\mathcal{O}(rn_p d)$.

The above cost assumes $\mathbf{c}_x^s$ is given. If we compute $\mathbf{c}_x^s$ using (20) for all s and x, the cost will be $\mathcal{O}(n_c d)$ as analyzed in §6.4.1. However, if we employ the proposed data structure, each $\mathbf{c}_x^s$ can be retrieved in at most $\mathcal{O}(\log(n_c)d)$ by recursively computing (35). Since we need to retrieve $\mathcal{O}(r) = \mathcal{O}(\sum_s m^s)$ number of $\mathbf{c}_x^s$, the complexity of computing necessary $\mathbf{c}_x^s$ is $\mathcal{O}(r\log(n_c)d)$.

As a result, the complexity of constructing $\mathcal{J}$ is $\mathcal{O}(rn_p d + r\log(n_c)d)$ at each layer. When summing the cost over all layers, the complexity is $\mathcal{O}(lrn_p d + lr\log(n_c)d)$.

By Claim 6.2, the rows of $\mathbf{S}_c$ and $\mathbf{S}_c\mathbf{C}$ are simply the $\mathbf{b}_x^s \in \mathcal{J}$ and the corresponding $\mathbf{c}_x^s$, which are already computed during the construction of $\mathcal{J}$, so we essentially can get these $\mathbf{S}_c$ and $\mathbf{S}_c\mathbf{C}$ for free.

6.4.3 Feeding Compressed Sequence into a Transformer Layer: $\mathcal{O}(lrd^2 + lr^2 d)$

At each layer, we need to compute

$$\mathbf{P}_{new} = \beta(\alpha(\mathbf{P}, \mathbf{SX}) + \mathbf{P}) + \alpha(\mathbf{P}, \mathbf{SX}) + \mathbf{P} \tag{36}$$

and the result of

$$\beta(\alpha(\mathbf{S}_c\mathbf{C}, \mathbf{SX}) + \mathbf{S}_c\mathbf{C}) + \alpha(\mathbf{S}_c\mathbf{C}, \mathbf{SX}) \tag{37}$$

for updating $\mathcal{T}(\mathbf{C})$. This is the part of a Transformer layer that requires heavy computation. It can be verified that the complexity of a Transformer layer is $\mathcal{O}(nd^2 + n^2 d)$ for a input sequence of length n. Now a compressed sequence of length r is fed into a Transformer layer, the cost is simply $\mathcal{O}(lrd^2 + lr^2 d)$. We note that there is an additional re-scaling to plug $\mathbf{D}$ into $\exp(\mathbf{P}\mathbf{C}^\top\mathbf{S}_c^\top)\mathbf{D}\mathbf{S}_c$ during multi-head attention computation discussed in 27. However, the additional cost of applying $\mathbf{D}$ is $\mathcal{O}(rd)$, which does not change the complexity. When summing the cost of all layers, the overall complexity is $\mathcal{O}(lrd^2 + lr^2 d)$.

6.4.4 Updating $\mathcal{T}(\mathbf{C})$ into $\mathcal{T}(\mathbf{C}_{new})$: $\mathcal{O}(lrd)$

Once (37) is computed, we need to change $\mathcal{T}(\mathbf{C})$ into $\mathcal{T}(\mathbf{C}_{new})$. The cost of change $\mathcal{T}(\mathbf{C})$ into $\mathcal{T}(\mathbf{C}_{new})$ is $\mathcal{O}(rd)$. Specifically, let us take a look at the first three iterations:

(1) At the first iteration, there are $\mathcal{O}(m^1)$ number of $(\mathbf{c}_{new})_x^1$ to be computed at the first inner for loop, and there are $\mathcal{O}(m^1)$ number of $(\mathbf{c}_{new})_x^1$ to be updated in the second inner for loop. Additional $\mathcal{O}(\frac{m^1}{2})$ number of $(\mathbf{c}_{new})_{\lceil x/2 \rceil}^2$ are masked dirty.

(2) At the second iteration, there are $\mathcal{O}(m^2)$ number of $(\mathbf{c}_{new})_x^2$ to be computed at the first inner for loop, and there are $\mathcal{O}(m^2 + \frac{m^1}{2})$ number of $(\mathbf{c}_{new})_x^2$ to be updated in the second inner for loop. The second term is due to the dirty $(\mathbf{c}_{new})_{\lceil x/2 \rceil}^2$ from the first iteration. Additional $\mathcal{O}(\frac{m^2}{2} + \frac{m^1}{4})$ number of $(\mathbf{c}_{new})_{\lceil x/2 \rceil}^4$ are masked dirty.

(3) At the third iteration, there are $\mathcal{O}(m^4)$ number of $(\mathbf{c}_{new})_x^4$ to be computed at the first inner for loop, and there are $\mathcal{O}(m^4 + \frac{m^2}{2} + \frac{m^1}{4})$ number of $(\mathbf{c}_{new})_x^4$ to be updated in the second inner for loop. The second and third term is due to the dirty $(\mathbf{c}_{new})_{\lceil x/2) \rceil}^{s_4}$ from the second iteration. Additional $\mathcal{O}(\frac{m^4}{2} + \frac{m^2}{4} + \frac{m^1}{8})$ number of $(\mathbf{c}_{new})_{\lceil x/2 \rceil}^8$ are masked dirty.

It becomes apparent that if we sum over the number of computes of $(\mathbf{c}_{new})_x^s$ and updates of $(\mathbf{c}_{new})_x^s$, the total number is $\mathcal{O}(\sum_s m^s + \sum_s \sum_{j=1}^{\log(s)} \frac{m^s}{2^j}) = \mathcal{O}(\sum_s m^s + \sum_s m^s) = \mathcal{O}(r)$. Since each compute and update takes $\mathcal{O}(d)$ cost, the overall complexity of changing $\mathcal{T}(\mathbf{C})$ into $\mathcal{T}(\mathbf{C}_{new})$ is $\mathcal{O}(rd)$. When summing the cost of all layers, the overall complexity is $\mathcal{O}(lrd)$.

6.4.5 Materializing C_{new} from $\mathcal{T}(\mathbf{C}_{new})$ at the Last Layer: $\mathcal{O}(nd)$

At the output of the last layer, we can compute $(\mathbf{c}_{new})_x^s$ sequentially from $(\mathbf{c}_{new})_1^{n_c}$, $(\mathbf{c}_{new})_x^{n_c/2}, \cdots,$ $(\mathbf{c}_{new})_x^1$ for all x from $\mathcal{T}(\mathbf{C}_{new})$ using (35) at $\mathcal{O}(d + 2d + 4d + \cdots + n_c d) = \mathcal{O}(n_c d) = \mathcal{O}(nd)$

cost. Then, $[\mathbf{C}_{new}]_i = c_i^1$ is materialized from $\mathcal{T}(\mathbf{C}_{new})$. Lastly, undoing the permutation so that $[\mathbf{P}_{new}; \mathbf{C}_{new}]$ are re-ordered to the original positions has a complexity of $\mathcal{O}(nd)$. As a result, the overall complexity is $\mathcal{O}(nd)$.

6.4.6 Overall Complexity

In summary, the overall complexity of our method is

$$\mathcal{O}(lrd^2 + lr^2d + lr\log(n_c)d + lrn_pd + nd) \tag{38}$$

6.5 More on Limitations

To reduce the complexity of implementations, the method is proposed for the encoder module of the Transformer that assumes full access to the entire sequence. The proposed compression can be extended to approximate the computation in the decoder, but it requires more implementation efforts, so we leave it for the future work. We briefly describe two possible options to do so. **(1)** We can use the input tokens of the decoder as VIP-tokens to compress the representations of context sequence generated by the encoder before Cross Attention computation to reduce the cost of Cross Attention. **(2)** Auto-regressive decoding operates using Causal Attention at each step. This Causal Attention operation requires memory and computation that is linear in the length of the prefix. We can keep the same Causal Attention VIP-token (the representation of the token currently being generated) and apply our method to compress the representations of the previously generated tokens. This reduces the linear complexity of the Causal Attention operation to sublinear. This is useful for reducing the cost of inference, but it is not clear how to apply this compression during training auto-regressive decoding.

6.6 Experiments

Table 2: Length statistics of each dataset. The values are the percentiles of number of tokens for the specific tokenizers. For T5 tokenizer, the left value of is for sequence lengths of encoder input, and the right value is for sequence lengths of decoder input.

	RoBERTa			T5				
Percentile	HotpotQA	QuALITY	WikiHop	WikiHop	HotpotQA	Qasper	QuALITY	ContractNLI
75th	1535	7603	2204	2399 / 6	1692 / 6	7029 / 29	7747 / 17	2991 / 4
95th	1928	8495	3861	4206 / 9	2129 / 10	10920 / 71	8603 / 28	5061 / 4

	T5							
Percentile	NarrativeQA	CNN/Dailymail	MediaSum	Arxiv	SummScreenFD	GovReport	QMSum	MultiNews
75th	90482 / 10	1242 / 87	2621 / 29	13477 / 364	12119 / 188	13304 / 811	19988 / 110	3032 / 379
95th	260533 / 18	1946 / 130	5061 / 64	26024 / 759	16722 / 330	23795 / 983	31749 / 162	6676 / 468

We run all experiments on NVIDIA A100 GPUs. All code is implemented using the standard PyTorch framework. No custom CUDA kernels are needed. As a result, it can be easily deployed to other platforms or ML frameworks. We will publish all code and checkpoints necessary for reproducibility concurrently with the paper publication.

Table 3: Dev set results for encoder-only models fintuning on HotpotQA, QuALITY, and WikiHop.

Method	Size	Length	HotpotQA			QuALITY		WikiHop	
			Runtime	EM	F1	Runtime	Accuracy	Runtime	Accuracy
RoBERTa	base	512	19.9	35.1	44.9	21.2	39.0	19.6	67.6
RoBERTa	base	4k	422.3	62.2	76.1	403.2	39.5	414.1	75.2
Big Bird	base	4k	297.9	59.5	73.2	307.0	38.5	293.3	74.5
Longformer	base	4k	371.0	59.9	73.6	368.0	27.9	369.7	74.3
Ours	base	4k	114.6	60.9	74.6	126.4	39.6	108.0	75.9
Ours-150k	base	4k	114.6	60.7	74.1	126.4	39.4	108.0	76.1

6.6.1 Pretraining

We use a filtered The Pile dataset [11] for all pretrainings. Since we are using public pretrained tokenizers, we want to enable the distribution of pretraining corpus aligns well with the distribution of corpus used to create the tokenizers. As a result, we use tokens per byte as a proxy for alignment of distributions and filter out PubMed Central, ArXiv, Github, StackExchange, DM Mathematics [25],

Ubuntu IRC, EuroParl [18], YoutubeSubtitles, and Enron Emails [16] components, which have tokens per byte greater than 0.3. Then, the remaining corpus of The Pile dataset is used for pretraining.

For encoder-only models, we pretrain RoBERTa for 750K steps. A batch consists of 8,192 sequences of 512 length. The masking ratio for masked language modeling (MLM) is 15%. Then, 4K length models are continuously pretrained from the RoBERTa checkpoints for 300k steps. The positional embeddings are extended by duplicating the pretrained 512 positional embedding multiple times. For 4K length RoBERTa, Longformer, and Big Bird, the batch size is 64, and the masking ratio is 15%. With 15% masking ratio, there are roughly 616 masked tokens scattering in the sequences. We find that using 616 scattered masked tokens as VIP tokens for 4,096 length sequences might not be indicative for VIP-token centric compression, so we use masking ratio 7.5% and batch size 128 for our method. The number of masked tokens per sequence is reduced, and the number of total masked token predictions remains the same during pretraining. We note that with larger batch size, the wall clock pretraining runtime for our method is still smaller than baselines. In case that anyone is interested, we also show downstream finetuning on our method pretrained on the same number of tokens but fewer number of masked token predictions in Tab. 3 and Fig. 3, denoted as Ours-150k. The accuracy is consistent with our model pretrained on 300k steps. For the larger scale pretraining denoted with *, we pretrain our method for 250K steps with batch size 512 and masking ratio 7.5%.

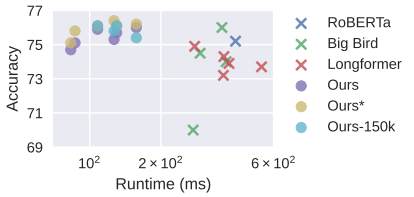

Figure 3: Model runtime vs WikiHop dev accuracy when using different model specific hyperparameters

For encoder-decoder architecture of our method, we do continuous pretraining from the public checkpoints of T5 for 250K steps with batch size 256 using the masked span prediction. Since each masked span (consists of multiple tokens) is replaced by a single special token, when using masking ratio is 15%, the number of special tokens in a sequence is not too large, we keep masking ratio 15 % unchanged.

6.6.2 Downstream Finetuning

The statistics of the sequence lengths of instances in each dataset are summarized in Tab. 2. The hyperparameters of all experiments are summarized in Tab 5. When there are multiple values in an entry, it means we perform a hyperparameter search on these values. The amount of search is determined by the size of datasets. If a dataset is relatively large, we only search the learning rate. If a dataset is small, we include batch size and the number of epochs in search. For all tasks, if the sequence lengths are longer than the model length m, the sequences will be truncated and only the first m tokens will be used. For encoder-decoder models, we use greedy decoding in sequence generations for simplicity. The maximal decoder output length, specified in Tab. 5, is set such that the maximal length covers the output lengths of more than 99% of instances. When the length of covering 99% of instances is greater than 512, we just set the maximal decoder output length to 512. Additionally, we show one extra experiment on MultiNews [10] in Tab. 4, which is not in the main text due to space limit.

Table 4: Dev results for encoder-decoder models on MultiNews.

Method	Size	# Param	Length	MultiNews			
				Runtime	R-1	R-2	R-L
T5	base	223M	512	59.2 / 20.5	42.5	15.3	39.0
T5	base	223M	4K	651.2 / 551.8	46.4	18.2	42.6
LongT5	base	248M	8K	721.7 / 550.6	46.7	18.3	42.9
LED	base	162M	8K	526.5 / 454.2	46.6	17.8	42.7
Ours	base	223M	8K	377.0 / 224.6	46.4	18.1	42.7
T5	large	738M	512	180.8 / 67.0	43.4	15.6	39.8
Ours	large	738M	8K	1140.3 / 651.5	48.2	19.2	44.2
Ours	3b	3B	8K	4094.5 / 2696.0	48.9	19.4	44.7

6.7 Practical Questions

Why is the performance of our method is better than standard models?

Our method is an approximation of the standard models, which should be inferior to the standard models, but in some cases, the performance of our method is better than standard models. We believe the reason is that the correct inductive bias improves the performance for tasks with limited amounts of data. Our approach is forced to compress irrelevant information and the attention is carried out on the compressed sequences, but in standard model with standard attention, each token has access to

Table 5: Hyperparameters for all experiments.

LM	Encoder-Only			Encoder-Decoder				
Task	HotpotQA	QuALITY	WikiHop	WikiHop	HotpotQA	CNN/Dailymail	MediaSum	Qasper
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam
Weight Decay	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
LR Decay	Linear	Linear	Linear	Linear	Linear	Linear	Linear	Linear
Precision	FP16	FP16	FP16	BF16	BF16	BF16	BF16	BF16
Batch Size	32	16	32	32	32	32	32	{16, 32}
Learning Rate	{3e-5, 5e-5}	{3e-5, 5e-5}	{3e-5, 5e-5}	{1e-4, 3e-4}	{1e-4, 3e-4}	{1e-4, 3e-4}	{1e-4, 3e-4}	{1e-4, 3e-4}
Epochs	10	{10, 20}	10	10	10	10	10	{10, 20}
Warmup Steps	1000	200	1000	1000	1000	1000	1000	200
Max Output Length	-	-	-	32	40	256	256	128

LM	Encoder-Decoder							
Task	QuALITY	ContractNLI	NarrativeQA	Arxiv	SummScreenFD	GovReport	QMSum	MultiNews
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam
Weight Decay	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
LR Decay	Linear	Linear	Linear	Linear	Linear	Linear	Linear	Linear
Precision	BF16	BF16	BF16	BF16	BF16	BF16	BF16	BF16
Batch Size	{16, 32}	{16, 32}	32	32	{16, 32}	{16, 32}	{16, 32}	32
Learning Rate	{1e-4, 3e-4}	{1e-4, 3e-4}	{1e-4, 3e-4}	{1e-4, 3e-4}	{1e-4, 3e-4}	{1e-4, 3e-4}	{1e-4, 3e-4}	{1e-4, 3e-4}
Epochs	{10, 20}	{10, 20}	5	{10, 20}	{10, 20}	{10, 20}	{10, 20}	10
Warmup Steps	200	1000	1000	1000	200	1000	100	1000
Max Output Length	90	4	47	512	512	512	310	512

the entire sequence, which enables a larger degree of freedom. As a result, more training data might be required for the model to learn the correct pattern or bias.