Incremental Transformer: Efficient Encoder for Incremented Text Over MRC and Conversation Tasks

Weisheng Li[1], **Yuechen Wang**[1], **Jiaxin Shi**[3],
Wengang Zhou[1,2], **Houqiang Li**[1,2], **Qi Tian**[3]
University of Science and Technology of China[1]
Institute of Artificial Intelligence, Hefei Comprehensive National Science Center[2]
Huawei Cloud Computing Technologies Co., Ltd.[3]
{li1117heex, wyc9725}@mail.ustc.edu.cn, shijx12@gmail.com,
{zhwg, lihq}@ustc.edu.cn, tian.qi1@huawei.com

Abstract

Some encoder inputs such as conversation histories are frequently extended with short additional inputs like new responses. However, to obtain the real-time encoding of the extended input, existing Transformer-based encoders like BERT have to encode the whole extended input again without utilizing the existing encoding of the original input, which may be prohibitively slow for real-time applications. In this paper, we introduce Incremental Transformer, an efficient encoder dedicated for faster encoding of incremented input. It takes only added input as input but attends to cached representations of original input in lower layers for better performance. By treating questions as additional inputs of a passage, Incremental Transformer can also be applied to accelerate MRC tasks. Experimental results show tiny decline in effectiveness but significant speedup against traditional full encoder across various MRC and multi-turn conversational question answering tasks. With the help from simple distillation-like auxiliary losses, Incremental Transformer achieves a speedup of 6.2x, with a mere 2.2 point accuracy reduction in comparison to RoBERTa-Large on SQuADV1.1. [1]

1 Introduction

Nowadays Transformer (Vaswani et al., 2017)-based large-scale pre-trained models like BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019) are still widely used in various tasks like extractive QA despite the prominent of large language models. However, if an encoded input is extended with short additional text, to obtain the encoding of the extended text, these Transformer encoders must completely re-encode the whole extended input, which may incur substantial inference latency, rendering it impractical for real-time applications. They can neither encode only added input nor update the encoding of the previously encoded input using the existing encoding. Because of the bidirectional information flow in the attention calculation, any changes to the input will affect the encoding of every token in the output. If the existing encoding can be used to encode the extended input, such incremental encoding will be very useful for inputs like browsing histories or conversations, which can be frequently extended by new records or responses.

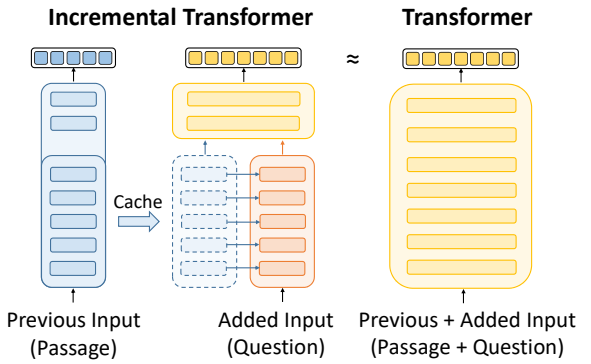

Figure 1: Schematic illustration of Incremental Transformer comparing to standard Transformer. By attending to cached representations of encoded previous input, the model can encode only added input in lower layers, while still generating output of high quality.

Moreover, machine reading comprehension tasks like SQuAD can also benefit from faster incremental encoding by treating questions as an increment of passages. In many cases, passages are available before the arrival of questions. Therefore, the encoding of passages can be pre-computed independently to achieve a faster speed, but this is not feasible for Transformer encoders because they must concatenate and process passages and questions together across all layers.

DeFormer (Cao et al., 2020) is a decomposed Transformer designed to reduce runtime compute for text pair tasks like question answering. It encodes questions and passages independently in

[1]We release our code in https://github.com/li1117heex/IncrementalTransformer.

Proceedings of the 31st International Conference on Computational Linguistics, pages 8819–8829
January 19–24, 2025.

lower layers, then jointly in upper layers like a standard Transformer. This decomposition enables passages to be pre-computed partially, saving most of the runtime computation. However, despite passages being completely available when questions are encoded, DeFormer still encodes questions independently in lower layers without any assistance from passages. Furthermore, we discover that encoding of the question is more negatively affected by the decomposition compared to the passage; refer to the discussion in Section 5. While DeFormer's decomposition can facilitate incremental encoding, but all these above suggest that we can get a better model by employing encoded passage and to aid question encoding in lower layers.

In this paper, we present Incremental Transformer, an efficient text encoder for incremented input leveraging the existing encoding of the previous input. In the lower layers, the model adopts a decomposed encoding pattern to separately encode previous inputs or passages. This enables the pre-calculation or reuse of part of their encoding, improving efficiency. For the encoding of added input or question in lower layers, we substitute self-attention with proposed Incremental Attention, where the attended sequence (key and value) is the whole input sequence. In the case of QA tasks, the attended sequence is the concatenation of question and cached passage hidden states, instead of the question sequence itself. Full self-attention is retained in upper layers to update the encoding output of the previous input.

In contrast to the straightforward decomposed attention in DeFormer, Incremental Transformer extends the attended sequence of question sequence, facilitating information flow from the passage. This assistance is proved crucial to obtain a higher performance without imposing a noticeable increase in computation cost, since the majority of computation occurs in expensive upper layers with quadratic complexity. As a result, Incremental Transformer can be decomposed up to higher layers to achieve a faster speed, while mitigating the performance drop imposed by the decomposition.

Experiments are conducted on several datasets including MRC and conversational QA tasks. Weights of RoBERTa-large are used to initialize our model. Results demonstrate that Incremental Transformer consistently outperforms DeFormer across all datasets while keeping inference costs at a similar level. Remarkably, with only one necessary upper layer, Incremental Transformer achieves a significant speedup of 6.2x, while the loss of effectiveness is only 2.2 points compared to RoBERTa-large on SQuADV1.1 (Rajpurkar et al., 2016) dataset. Furthermore, Incremental Transformer can be applied multiple times to encode a repeatedly growing input. Comparable performance against RoBERTa-large on multi-turn conversational QA dataset QuAC (Choi et al., 2018) shows the model's ability to encode a repeatedly incremented input.

2 Incremental Encoding

In this section, we give definition of incremental encoding and several essential terms. Incremental encoding involves the encoder processing two input texts, firstly T_a and later T_b. T_a has been previously encoded by the same encoder, then T_b is added after T_a, forming a concatenated input (T_a, T_b). In incremental encoding, the model is required to encode (T_a, T_b) efficiently by reusing the calculated encoding or intermediate hidden states of T_a. In other words, an incremental encoder possesses the capability to encode fully or partially encode T_a in advance, without requiring the complete input sequence (T_a, T_b). We refer to T_a, T_b, and (T_a, T_b) as previous input, added input, and extended input respectively.

Incremental encoding offers benefits across various tasks, particularly in scenarios with continuously growing inputs, such as ongoing conversations. Incremental encoding can be applied to MRC or QA tasks as well, in which the concatenation of a passage and a question is encoded to extract the answer. In practical QA applications, passages are typically available in advance. By treating passages as T_a and questions as T_b, passages can be encoded proactively before the arrival of questions, thereby mitigating answering latency through effective reuse. Furthermore, in cases where multiple questions are asked to a single passage, the reuse of representations for the passage allows for a one-time computation, enhancing efficiency. Generally speaking, all tasks featuring (1) real-time encoding of growing input , or (2) part of input available earlier or shared across many samples, can be accelerated by an incremental encoding model, like Incremental Transformer, which will be described in the next section.

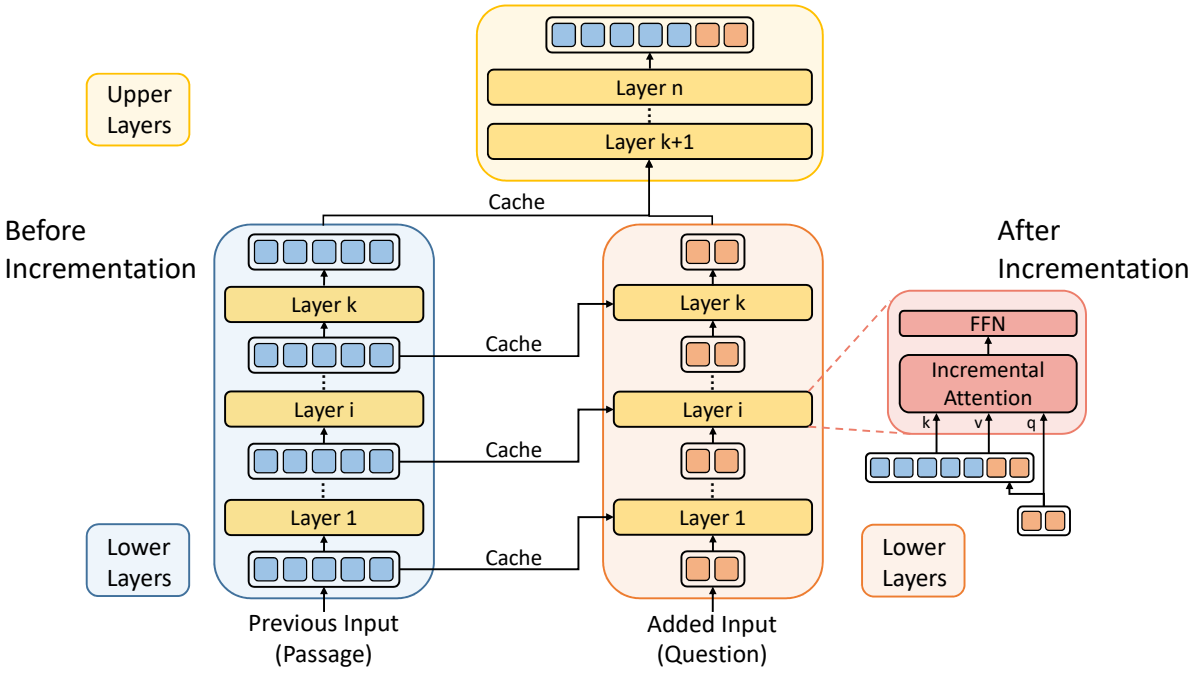

Figure 2: Illustration of the model architecture and attention mechanism of Incremental Transformer. Details of the Incremental Attention module are shown on the right side of this figure. We assume $m = k$ in this figure, in this case Incremental Attention is applied to all lower layers of added input.

3 Model

3.1 Incremental Attention

As discussed in the introduction, passages should be leveraged to assist the encoding of questions in lower layers. Building upon this idea, we propose Incremental Attention for the encoding of the added input T_b in lower layers. It extends the key and value sequence from T_b itself to the whole input (T_a, T_b). In this way, the previous input T_a can also be attended by T_b to assist its encoding. Denote Q_a, K_a, V_a as the query, key, value vector of the previous input, K_b, V_b as the key, value vector of the added input respectively, we compute attention output A_n as follows:

$$A_n = Attention(Q_b, [K_a; K_b], [V_a; V_b])$$
$$= softmax(\frac{Q_b[K_a; K_b]^T}{\sqrt{d_k}})[V_a; V_b]$$

3.2 Incremental Transformer

Layers of Incremental Transformer are split into lower layers and upper layers, then previous and added inputs are encoded separately in lower layers and jointly in upper layers like DeFormer. Incremental Attention is applied in part or all of the lower layers during the encoding of the added input, while standard self-attention is retained in other layers, including shared upper layers and lower layers of the previous input. Denote $L_{i:j}$ and $IncrL_{i:j}$ as the application of a stack of layers from layer i to layer j with common self-attention and Incremental Attention respectively, X_a^i, X_b^i as the i-th layer representation of previous or added input. Given previous and added input T_a, T_b and their embedded sequence X_a, X_b, independent encoding of the previous input in lower layers can be written as:

$$X_a^k = L_{1:k}(X_a)$$

Layer with Incremental Attention takes previous input representation of the same layer as additional input:

$$X_b^i = IncrL_i(X_a^{i-1}, X_b^{i-1})$$

Encoding of added input in lower layers:

$$X_b^{k-m} = L_{1:k-m}(X_b)$$

$$X_b^k = IncrL_{k-m+1:k}(X_b^{k-m}, X_a^{k-m:k-1})$$

Encoding in upper layers can be written as:

$$[X_a^n; X_b^n] = L_{k+1:n}([X_a^k; X_b^k])$$

Where $[X_a^n; X_b^n]$ is the encoding output. k, m are hyper-parameters, k is the number of lower

layers i.e. the separation layer, and m is the number of layers using Incremental Attention. For most experiments below, we set $m = k$, which means that Incremental Transformer Attention is adopted by all lower layers when encoding the added input.

Figure 2 shows the overall structure of Incremental Transformer and how it operates. Before incrementation i.e. the arriving of added input, Incremental Transformer encodes previous input in lower layers, and caches output representations of layers $k - m$ to $k - 1$ for Incremental Attention, plus the representation of k-th layer for encoding in upper layers. After incrementation, Incremental Transformer first encodes added input in lower layers, with assistance from cached representations in layers $k - m$ to $k - 1$, the output is the k-th layer representation of added input. Then the model loads the cached k-th layer representation of previous input, concatenates them, and feeds into upper layers (from $k + 1$ to n) to complete the encoding.

In practice, the key and value projection of previous input representations $K_a = W_k X_a, V_a = W_v X_a$ can be calculated before incrementation as well, then the model caches projected sequence K_a and V_a instead of X_a. In this way, the costly projection module is excluded from post-incrementation computation, allowing Incremental Transformer to run almost as fast as DeFormer. Experiments below follow this procedure.

3.3 Consecutive Incrementation

Inputs like conversation histories can be incremented multiple times. While it is feasible to apply the Incremental Transformer separately for each single incrementation, this approach means we have to encode the extended input using full self-attention again, since it also serves as the previous input in the next incrementation and needs to be encoded earlier. This iterative re-encoding can impose a heavy burden for systems occupied by frequent conversations or questions.

To remove this redundant calculation, instead of re-encode $X_{(a,b)} = [X_a; X_b]$ and re-cache $X^{k-m:k-1}_{(a,b)}$ and $X^{k}_{(a,b)}$, we take $[X^i_a; X^i_b]$, the concatenation of newly computed X^i_b and already cached X^i_a as the updated cache of layer i. In this way, Incremental Transformer can be consecutively applied to encode the multiply incremented input. Every time new input arrives, the model simply concatenates new representations to the existing caches, making these updated caches ready for the next incrementation.

3.4 Computation and Cache Storage Complexity

Here we discuss the complexity of the Incremental Transformer. Note that only computation after incrementation is taken into account. Given previous and added input with length l_a and l_b, computation complexity is $O(l_b(l_a + l_b))$ for each lower layer with Incremental Attention, $O(l^2_b)$ for these without Incremental Attention. However, considering the cost of the upper layers, overall computation complexity is still $O((l_a + l_b)^2)$, the same as DeFormer and full Transformer. Upper layers consume most of the computation, and the complexity difference between these models in lower layers becomes trivial. Nevertheless, with the ability to reduce the number of costly upper layers, Incremental Transformer can still save lots of computation, which is proved by our experiments below.

The cache storage complexity of the Incremental Transformer is $O((2m + 1)l_a)$ for caching key and value sequences of m layers with Incremental Attention, plus the output sequence of k-th layer. It seems a lot, but a Transformer decoder (Vaswani et al., 2017) also needs to cache key and value sequences of encoder output for each decoder layer. In contrast, the storage complexity of the Incremental Transformer is totally acceptable in practice. What's more, Incremental Transformer can preserve most of the performance with a quite small m and the cache storage, as discussed in the ablation study section.

4 Experiments

We use four MRC datasets to evaluate our model, and one additional multi-turn conversational QA dataset QuAC to evaluate it under consecutive incremental encoding. All these datasets are in English and cover different domains. For SQuAD , BoolQ and QuAC datasets, we split 10% of the training set to obtain the real validation set for tuning hyper-parameters, and report metrics on the original validation set. As for RACE which provides a test set, we just tune and report metrics on the validation and the test set as usual.

SQuAD The Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016) is an extractive question answering dataset derived from Wikipedia articles, requiring to extract an answer span from the provided context. SQuAD dataset

Model	SQuADV1.1	SQuADV2	RACE	BoolQ	Speedup
RoBERTa-base	92.1	83.2	75.2	75.8	5.3x
RoBERTa-large (Teacher)	94.6	88.3	84.4	82.7	1.0x
DeFormer$_{16}$	93.1	81.6	81.8	79.0	2.3x
Incremental Transformer$_{16}$	94.2	86.0	83.7	84.7	2.2x
DeFormer$_{18}$	90.6	72.4	75.0	76.7	2.9x
Incremental Transformer$_{18}$	93.6	85.7	83.3	84.9	2.7x
DeFormer$_{20}$	88.5	61.5	71.0	71.4	3.9x
Incremental Transformer$_{20}$	93.4	84.8	83.5	84.8	3.5x
DeFormer$_{23}$	77.6	58.1	56.9	70.9	8.0x
Incremental Transformer$_{23}$	92.4	84.5	83.3	83.9	6.2x

Table 1: Performance of RoBERTa-base, RoBERTa-large, Incremental Transformer and DeFormer each with several different separation layer k. For example, Incremental Transformer$_{23}$ means Incremental Transformer with $k = 23$. The same goes for other models. For SQuADV1.1 and V2.0, we report F1 score; for RACE and BoolQ, we report accuracy. Notice that only computation cost after incrementation is taken into account. Inference latencies and speedups are measured using the averaged running time of the model over the SQuAD test set, with RoBERTa-large model as baseline.

has two versions. In V1.1 the context always contains an answer, whereas in V2.0 some questions are not answerable by the provided context, making the task more challenging.

RACE The ReAding Comprehension from Examinations (RACE) (Lai et al., 2017) is a reading comprehension dataset collected from the English exams designed for middle and high school students in China. For every question, the task is to select one correct answer from four options using provided passage.

BoolQ Boolean Questions (Clark et al., 2019) is a QA task where each example consists of a short passage and a yes/no question about the passage. The questions are provided anonymously and unsolicited by users of the Google search engine, and afterwards paired with a paragraph from a Wikipedia article containing the answer.

QuAC Question Answering in Context (Choi et al., 2018) is a dataset created for simulating information-seeking conversations, where a student asks a series of questions based on a Wikipedia document, and a teacher provides answers in the form of text spans. Its questions are often more open-ended, unanswerable, or only meaningful within the dialog context.

4.1 Implementation Details

We use weights of pre-trained model RoBERTa-Large (Liu et al., 2019) to initialize our model because it is well pre-trained and performs better in various downstream tasks. Because DeFormer uses a different pre-trained model BERT-base, and we can't run DeFormer's open-source code in the Tensorflow framework due to incompatible environments, we reimplemented DeFormer with a simple attention mask in PyTorch (Paszke et al., 2019) framework and conducts experiments on it. Experiment results of DeFormer with BERT-large as backbone are comparable to results reported in the DeFormer paper (see Appendix B).

For these MRC datasets, because Incremental Transformer retains the structure of the full Transformer including positional encoding, decomposed segments should use the positional encoding of the same position as in the original combined input sequence. However, the passage is encoded earlier without the exact length of each question, so it becomes impossible to decide where should its positional encoding begins. To address this issue, we pad or truncate embedded question sequences to unify their length in each dataset. This length is chosen to cover almost all the questions except several extremely long ones. This alignment slightly damages the performance of our model because of these filled padding tokens; nonetheless, our conclusions are not affected.

For the QuAC dataset, following earlier works, we prepend all previous pairs of (question, answer) to the passage and current question to form the current input since every question after the first one depends on the conversation history. Formally, to answer the k^{th} question Q_k, the input sequences of previous and added inputs $X_{a,k}$, $X_{b,k}$ is:

$$X_{a,k} = (P, Q_1, A_1, \cdots, Q_{k-1})$$

$$X_{b,k} = (A_{k-1}, Q_k)$$

Model	QuAC		
	F1	HEQ-Q	HEQ-D
RoBERTa-large (Teacher)	63	57.6	4.6
Incremental Transformer$_{16}$	63.4	57.6	5.3
Incremental Transformer$_{18}$	62.1	56	5.0
Incremental Transformer$_{20}$	61.7	55.3	5.1
Incremental Transformer$_{22}$	59.5	51.1	3.8
Incremental Transformer$_{23}$	50.4	44.5	2.7

Table 2: Performance of RoBERTa-large, Incremental Transformer and DeFormer on dev set of QuAC with different separation layer k.

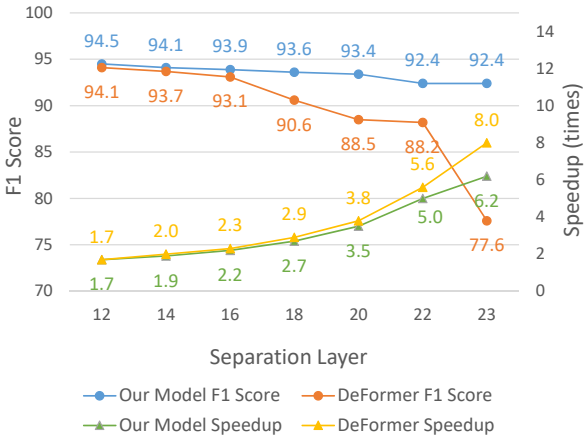

Figure 3: F1 score and speedup of DeFormer and Incremental Transformer with different separation layer k fine-tuned on SQuADV1.1 dataset with auxiliary losses. We provide results of the same experiments with BERT-large as initialization without auxiliary losses in Appendix B to provide a better comparing of two models.

We put the question after conversation history so that $X_{a,k} = [X_{a,k-1}, X_{b,k-1}]$, which means the current input can be constructed through a series of text incrementation and then encoded incrementally. Because inputs are always too long, we extend the length of positional encoding from 512 to 1024.

Following DeFormer, we use two auxiliary losses, Output Representation Similarity Loss $\mathcal{L}_{or}$ and Prediction Similarity Loss $\mathcal{L}_{pd}$, to boost model performance by pushing output representations X_S^n and predicted distributions P_S of Incremental Transformer as student model closer to equivalents X_T^n and P_T of fine-tuned full Transformer as teacher model. They are calculated as:

$$\mathcal{L}_{pd} = CrossEntropy(P_S, P_T)$$

$$\mathcal{L}_{or} = MeanSquaredError(X_S^n, X_T^n)$$

Along with task-specific supervision loss $\mathcal{L}_{task}$, integrated loss $\mathcal{L}$ is calculated as:

$$\mathcal{L} = \alpha\mathcal{L}_{pd} + \beta\mathcal{L}_{or} + \gamma\mathcal{L}_{task}$$

Detail hyper-parameters including passage and question length, batch sizes, learning rates, auxiliary losses coefficientsα, β, γ, and others can be found in the appendix. We use Adam optimizer (Kingma and Ba, 2014) with linear learning rate decay for every experiment. We implement our model based on Huggingface Transformers (Wolf et al., 2020) and PyTorch (Paszke et al., 2019) framework. All main experiments are conducted on 2 NVIDIA Tesla A100 GPUs. It takes roughly 1 to 2 hours to complete one fine-tuning on SQuAD datasets. All these results are from a single run.

4.2 Main Results

Table 1 shows performance metrics of Incremental Transformer and DeFormer on MRC datasets and inference speedups on the SQuADV1.1 test set. To present a complete view, we compare two models on different separation layer k including $k = 16, 18, 20, 23$. For all these settings, Incremental Transformer consistently performs better than DeFormer, and the margin goes larger as k increases. In the meantime, the difference in speedup between two models is rather subtle, because the difference in attention computation in lower layers is covered by much more expensive attention computation in upper layers.

The most noticeable result comes from Incremental Transformer$_{23}$, which achieved a speedup of 6.2x over RoBERTa-large while the F1 score drop is merely 2.2 points. To provide a better comparison, we put RoBERTa-base into the chart. Incremental Transformer$_{23}$ is faster than RoBERTa-base and performs better across all these datasets. $k = 23$ means only one upper layer left, the minimum number needed to update the encoding output of the passage. These results demonstrate the strong effectiveness and flexibility of our model.

Table 2 presents the performance of Incremental Transformer on the QuAC dataset with different separation layer k. Although being incremented several times, our model can still perform comparably against RoBERTa-large even when $k = 20$. These results fully display the potential of Incremental Transformer to be applied consecutively.

4.3 Ablation Study

The choice of separation layer k directly affects the model's effectiveness. In Figure 3, we show

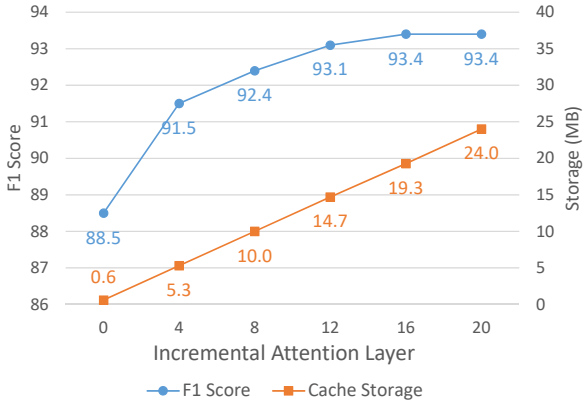

Figure 4: F1 score and cache storage of Incremental Transformer with different numbers of Incremental Attention layer m while $k = 20$. Cache storage is estimated that it takes 0.59MB to store a sequence of 150 tokens with hidden dimension 1024, then multiply $2m + 1$ because $2m + 1$ sequence is cached for Incremental Attention. 150 is the average length of passage for the SQuADV1.1 test set.

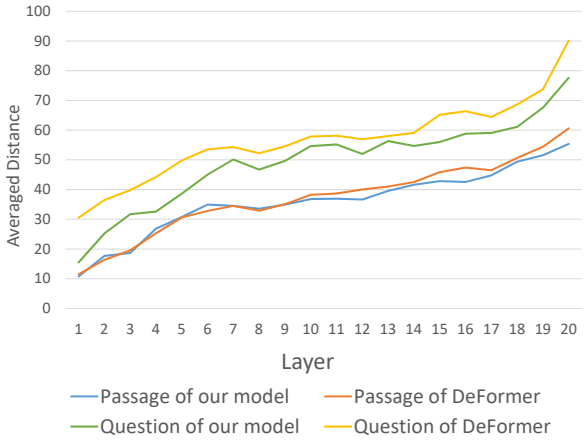

Figure 5: Averaged token distance to fine-tuned RoBERTa-large from both DeFormer$_{20}$ and Incremental Transformer$_{20}$, for both question and passage segments, across all lower layers.

how the F1 score and inference speedup of both DeFormer and Incremental Transformer change as the separation layer changes. In terms of effectiveness, Incremental Transformer performs consistently better than DeFormer, the same as the main results. The inference speed of both models grows like an x-axis reflected $f = 1/x$ function because the computation cost for one upper layer with full self-attention is much more expensive than for one lower layer; while the difference between them is negligible.

For all experiments above, we set $m = k$ to apply Incremental Attention to all lower layers for the encoding of added input. This assistance, however, is less important for layers closer to the input since they encode more locally. We conduct an experiment with different numbers of Incremental Attention layer m, from $m = 0$ (in this case our model becomes the same as DeFormer) to $m = k = 20$, presenting the changes of F1 score and cache storage in Figure 4. F1 score grows faster at the beginning, only 4 layers with Incremental Attention make over half of the total improvement; after that, the growth gradually slows down. Smaller m equals less cache storage and computation required. This experiment shows that Incremental Transformer can preserve most of the improvements over DeFormer while using less cache storage. By adjusting m and k, users can choose the best model with respect to the required inference speed and the cache limitation.

5 Analyses

5.1 Distance of Output Representations

Both DeFormer and Incremental Transformer keep the structure of the full Transformer, with only part of the attention calculation in lower layers removed for decomposition purposes. On the other hand, they can both be initialized with weights of pre-trained model and they both employ outputs of fine-tuned full Transformer as additional supervision. Given that both of them have been trained to mimic the pre-trained model, it's reasonable to use the distance from their hidden layer representations to equivalent representations of fine-tuned full Transformer, to measure how successful their imitations are ; shorter distance and smaller deviation should indicate better performance. We evaluate this deviation using the full test dataset of SQuADV1.1 and calculate the averaged distance as follows:

$$AverageDistance(X, Y) = \frac{\sum_j ||X_j - Y_j||}{l}$$

Where X, Y are two token sequences of equal length l. Masked padding tokens are excluded from this calculation.

5.2 Passage Helps Question in Lower Layers

In QA tasks, the question is much shorter than the passage, whose average sequence length after embedding is only 14.5 in the SQuADv1.1 test set, comparing to the full sequence length of 384 or 512. We suspect that many tokens in the question sequence become harder to be understanded

without reference to the passage sequence, which makes much more damage to the encoding of the question than the passage.

To verify this, we calculate the averaged token distance to fine-tuned RoBERTa-large from both DeFormer$_{20}$ and Incremental Transformer$_{20}$, for both question and passage, at all lower layers. All these results are shown in Figure 5. Results show that question representations deviate further than passages for both models, which is in accordance with our suspicion that question suffers more damage from decomposed encoding.

Comparing the two models, for passage, their behavior is similar, which meets our expectation as they both encode passage independently in lower layers. However, for the question, results show clear differences; with the assistance from cached passage outputs, representations are closer in every layer, enabling Incremental Transformer to produce better performance in downstream tasks.

6 Related Works

To improve the efficiency of Transformer, numerous model compression methods have been investigated.

Knowledge Distillation uses different losses to train a small student model to mimic the behavior of a large teacher model, such behaviors including output distribution, representation, attention matrix, or others. By doing so, the knowledge inside the teacher model is transferred into the student model. Earliest work to apply knowledge distillation on BERT (Devlin et al., 2019) is DistillBERT (Sanh et al., 2019) and TinyBERT (Jiao et al., 2020), followed by MiniLM (Wang et al., 2020) and MobileBERT (Sun et al., 2020), which achieved 4x speed-up and competitive results.

Pruning removes less important parts of the model to achieve a faster speed. Fan et al. (2020) drops some layers, Michel et al. (2019) and Voita et al. (2019) drops redundant attention heads. McCarley et al. (2019) introduces gates to select both heads and feed-forward layers on question answering tasks.

Quantization compresses model parameters of higher precision floating-point to lower precision. Q8BERT (Zafrir et al., 2019) compresses the 32-bit model to 8-bit with negligible performance drop, Q-BERT (Shen et al., 2020) and TernaryBERT (Zhang et al., 2020) even produce a 2-bit model. However, with special hardware needed, it's not easy for other people to use these models.

Compared to these methods, Incremental Transformer accelerate the inference process by beforehand encoding and calculation reusing, without compressing or removing model parameters. As a result, basically all these methods can be applied in combination with Incremental Transformer to achieve a faster speed.

In addition to these approaches above, some methods targeting directly on the reduction of input sequence length. They progressively skip or skim less important tokens or text blocks in each layer, reducing length of hidden states to achieve speedup. POWER-BERT (Goyal et al., 2020) eliminates irrelevant tokens based on self-attention matrix and drop them directly. Length-Adaptive Transformer (Kim and Cho, 2021) improves it by forwarding the inflected tokens to the final output. Incorporating reinforcement learning and reparameterization technique, TR-BERT (Ye et al., 2021) and Transkimmer (Guan et al., 2022a) further enhance the optimization of skim predictor. Block-Skim (Guan et al., 2022b) uses a CNN-based predictor to select and skip irrelevant text blocks of 32 tokens, slightly improves QA models' accuracy on different datasets and achieves 3× speedup on BERT-base model. However, performance of these methods on large size backbones are not presented.

7 Conclusion

In this paper, we present Incremental Transformer, an efficient decomposed encoder for incremented input like conversations or passages with questions in MRC tasks. Decomposed lower layers enable previous input to be encoded in advance or its existing encoding to be directly reused. Above that, its representations are cached and attended in Incremental Attention module during the encoding of added input to assist its encoding. Experiments show significant speedup and tiny decreases in terms of model effectiveness across MRC and conversational QA tasks. This paper shows the strong potential of largely unexplored incremental models. For future works, we will continue to explore the application of incremental encoding in other scenarios including longer inputs, retrieval, multi-tasking, and generative tasks. We also expect to see how these cached sequences can be compressed to save memory. We believe incremental encoding deserves greater attention and deeper exploration.

References

Qingqing Cao, Harsh Trivedi, Aruna Balasubramanian, and Niranjan Balasubramanian. 2020. DeFormer: Decomposing pre-trained transformers for faster question answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4497, Online. Association for Computational Linguistics.

Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wentau Yih, Yejin Choi, Percy Liang, and Luke Zettlemoyer. 2018. QuAC: Question answering in context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2174–2184, Brussels, Belgium. Association for Computational Linguistics.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*.

Saurabh Goyal, Anamitra Roy Choudhury, Saurabh M. Raje, Venkatesan T. Chakaravarthy, Yogish Sabharwal, and Ashish Verma. 2020. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org.

Yue Guan, Zhengyi Li, Jingwen Leng, Zhouhan Lin, and Minyi Guo. 2022a. Transkimmer: Transformer learns to layer-wise skim. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7275–7286, Dublin, Ireland. Association for Computational Linguistics.

Yue Guan, Zhengyi Li, Jingwen Leng, Zhouhan Lin, Minyi Guo, and Yuhao Zhu. 2022b. Block-skim: Efficient question answering for transformer. *Preprint*, arXiv:2112.08560.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

Gyuwan Kim and Kyunghyun Cho. 2021. Length-adaptive transformer: Train once with length drop, use anytime with search. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6501–6511, Online. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint*.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794, Copenhagen, Denmark. Association for Computational Linguistics.

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

J. S. McCarley, Rishav Chakravarti, and Avirup Sil. 2019. Structured pruning of a bert-based question answering model. *arXiv preprint*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

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

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt

Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8815–8821.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2158–2170, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*, volume 33, pages 5776–5788. Curran Associates, Inc.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Deming Ye, Yankai Lin, Yufei Huang, and Maosong Sun. 2021. TR-BERT: Dynamic token reduction for accelerating BERT inference. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5798–5809, Online. Association for Computational Linguistics.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pages 36–39.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. TernaryBERT: Distillation-aware ultra-low bit BERT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 509–521, Online. Association for Computational Linguistics.

A Hyperparameters

In Table 3 we show hyperparameters for fine-tuning both Incremental Transformer and DeFormer models on SQuAD, RACE, BoolQ, and QuAC datasets. Learning rates are chosen from the best of $1e-5, 2e-5, 3e-5, 5e-5$ for every dataset on RoBERTa-large, then batch size is chosen from the best of $8, 16, 32, 64$.

B Results on BERT-large

We have compared Incremental Transformer and DeFormer with RoBERTa-large as backbone model. Here we provide results with BERT-large as backbone in Figure 6 to give a more comprehensive comparison.

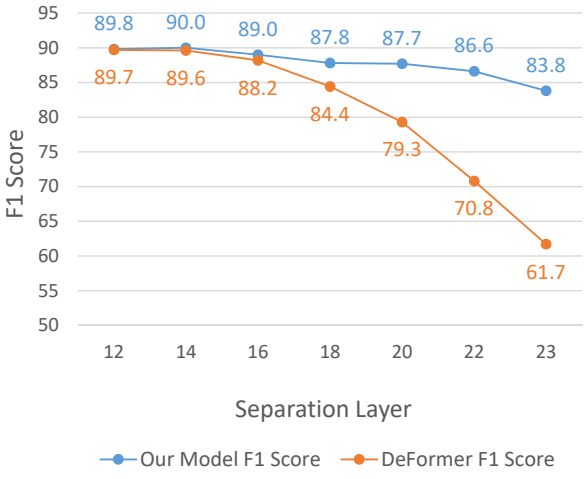

Figure 6: F1 score of DeFormer-BERT-large and Incremental Transformer-BERT-large with different separation layer k fine-tuned on SQuADV1.1 dataset without auxiliary losses.

C Licenses

License for SQuAD and QuAC datasets are CC BY-SA 4.0, and for BoolQ is CC BY-SA 3.0. In the case of RACE, the authors did not provide any license but specified that it could only be used for non-commercial research purposes.

Hyperparam	**SQuAD**	**RACE**	**BoolQ**	**QuAC**
Dropout	0.1	0.1	0.1	0.1
Attention Dropout	0.1	0.1	0.1	0.1
Warmup ratio	0.06	0.06	0.06	0.06
Learning Rate	2e-5	1e-5	1e-5	2e-5
Batch Size	16	8	16	8
Weight Decay	0.01	0.1	0.1	0.01
Max Epochs	2	4	2	2
Learning Rate Decay	Linear	Linear	Linear	Linear
Question Length	50	30	15	—
Passage Length	334	482	369	—
Adam ϵ	1e-8	1e-8	1e-8	1e-8
Adam β_1	0.9	0.9	0.9	0.9
Adam β_2	0.999	0.999	0.999	0.999
α	0.9	0	0	0.6
β	0	0.9	0.9	0
γ	0.1	0.1	0.1	0.4

Table 3: Hyperparameters for fine-tuneing both Incremental Transformer and DeFormer models on SQuAD, RACE, BoolQ, and QuAC datasets.