ReadOnce Transformers: Reusable Representations of Text for Transformers

Shih-Ting Lin[*] **Ashish Sabharwal**[†] **Tushar Khot**[†]

[*] University of Texas, Austin, U.S.A.
[†] Allen Institute for AI, Seattle, U.S.A.

j0717lin@utexas.edu, {ashishs,tushark}@allenai.org

Abstract

We present ReadOnce Transformers, an approach to convert a transformer-based model into one that can build an information-capturing, task-independent, and compressed representation of text. The resulting representation is reusable across different examples and tasks, thereby requiring a document shared across many examples or tasks to only be *read once*. This leads to faster training and evaluation of models. Additionally, we extend standard text-to-text transformer models to Representation+Text-to-text models, and evaluate on multiple downstream tasks: multi-hop QA, abstractive QA, and long-document summarization. Our one-time computed representation results in a 2x-5x speedup compared to standard text-to-text models, while the compression also allows existing language models to handle longer documents without the need for designing new pre-trained models.

1 Introduction

Transformer-based large scale language models (LMs) (Radford et al., 2018; Devlin et al., 2019) are task-independent models that are surprisingly effective when directly fine-tuned on many different end-tasks (Rajpurkar et al., 2016; Wang et al., 2019b,a). However, this approach relies heavily on using end-task supervision to learn to solve two sub-problems simultaneously: extract information[1] from an input document D and solve the end-task (e.g., answer a question about D). This incentivizes LM-based models to learn to extract only task-specific—and even example-specific—information when fine-tuned on the end-task. For example, a Question Answering (QA) model may learn to only extract the answer from D given the input question.

[*]The author's work was primarily done during an internship at the Allen Institute for AI.

[1]By "extract information", we mean implicitly or explicitly compute some representation of the document.

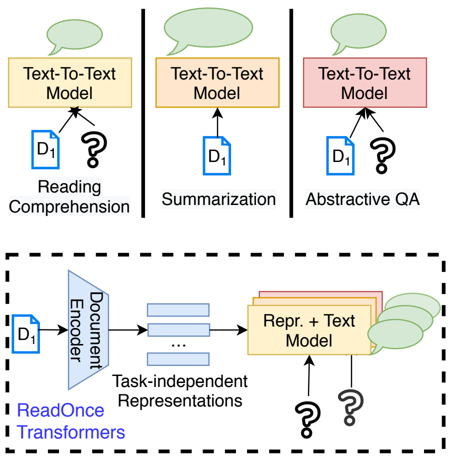

Figure 1: ReadOnce Transformers: Rather than learning to extract information specific to each end-task, we use transformer-based encoders to build task-independent, reusable document representations once, and feed them into various representation+text transformer models trained for end-tasks.

This strategy, while effective on many datasets, is also inefficient. First, it requires model's pre-trained weights to be fine-tuned separately for each end-task, even though the sub-problem of gathering the information content of the input document D is shared across tasks. Second, each D must be re-read from scratch in the context of each example (e.g., once for each question) even when many examples share D. Not only is this computational redundancy undesirable, slow inference can quickly become a bottleneck in deployed, real-time systems if models with billions of parameters must re-read D for every input query.

Inspired by humans' ability to read a document and extract key information from it without having to know the use case in advance, we ask the following question: Can we use transformer-based LMs to *build compressed representations of text* that are example- and task-independent, and hence reusable? Further, can we extend text-to-text transformer architectures to consume such representations in conjunction with text?

Prior representation learning approaches attempt to capture the meaning of sentences into a continuous vector (Conneau et al., 2017; Kiros et al., 2015; Reimers and Gurevych, 2019). While they have been effective on downstream classification tasks, it is unclear whether they can capture the information content of entire paragraphs. Moreover, these approaches focus on building fixed-length representations that are used as the input features for task-specific classifiers. In contrast, our goal is to (a) use transformer-based LMs to build compressed representations that *scale with the document size*, and (b) *combine them with example-specific text inputs* to produce the more general text output.

To this end, we propose an approach to convert any encoder-decoder based transformer LM (such as BART (Lewis et al., 2020)) into a new architecture termed READONCE Transformer, with two key parts: (1) a Document Encoder that reads documents only once to create compressed, information-capturing, reusable representations that we refer to as READONCE Representations (2) a Representation+Text Model that consumes these document representations together with task- and example-specific plain text (e.g., a question) to produce text output (e.g. an answer). To ensure that our compressed representations capture the key facts, we use supervision from two factoid QA datasets, SQuAD (Rajpurkar et al., 2016) and UnsupervisedQA (Lewis et al., 2019) to train READONCE Transformers. To solve an end-task, we only need to compute the READONCE Representations of the documents once and only train the Representation+Text Model to perform the end-task.

Our experiments demonstrate that these representations are more effective at capturing information compared to baseline approaches. Our representations also generalize to other tasks such as multi-hop QA (Yang et al., 2018), abstractive QA (Kociský et al., 2018), and summarization (Narayan et al., 2018). Since READONCE Representations are computed only once, we can train and infer with models 2x-5x faster than standard approaches, with only a marginal drop in accuracy (about 3 F1 points on QA and 4 Rouge-L points on summarization for a 2x speedup). Moreover, the compression ratio parameter K of our representations provides an easy way to trade off computation time with accuracy. Specifically, our analysis suggests that the resulting model has a computation cost of roughly $1/2R + 3/4K^2$ of the base LM, where R is the frequency of document reuse.

Additionally, our compressed representation enables us to efficiently combine information from long (or multiple) documents enabling more accurate long-document summarization (Cohan et al., 2018) without needing costly pre-training of new LMs (Beltagy et al., 2020; Zaheer et al., 2020).

2 Related Work

Representation learning approaches are commonly used to extract fixed-length sentence embeddings (Conneau et al., 2017; Kiros et al., 2015; Wang et al., 2020) from variable-length text inputs. Such fixed length representations have enabled the development of simpler downstream models that do not have to deal with the variable-lengths of textual inputs. However, these representations have mainly been used for simple classification tasks on short input texts (Bowman et al., 2015; Wang et al., 2019b). The word-level representations from RNNs or transformers are also variable-length, but uncompressed. While such representations have been re-used with RNNs (Peters et al., 2018) and are easy to combine with text input, it is not immediately clear how to combine representations from transformers with text, which is what we propose.

Recent work (Reimers and Gurevych, 2019; He et al., 2020; Artetxe and Schwenk, 2019; Karpukhin et al., 2020) has tried building document-embedding using large-scale language models as well. However these fixed-length representations have mostly been built to identify similar documents (Reimers and Gurevych, 2019; Karpukhin et al., 2020) and are not used directly for QA. QuASE (He et al., 2020), also used question-answering supervision for transfer learning but do not produce re-usable representations. Artetxe and Schwenk (2019) learned multi-lingual sentence embeddings that may be able to capture the knowledge present in a sentence but they were designed for BiLSTMs. Some large-scale LMs have been especially designed to handle long documents (Yang et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020) too but need to be pre-trained on large corpora, whereas we can use any pre-trained LM.

Aspects of our work also bears resemblance to domain adaptation (Daume III and Marcu, 2006), transfer learning (Pan and Yang, 2010) and multi-task learning (Caruana, 1993) but focuses on learning information-capturing representations from transformer-based models that has not been explored by prior work. While model distillation (Hinton et al., 2015) can also result in speedups, these techniques are orthogonal and can be easily incorporated in our framework (as we show in our experiments).

3 READONCE Transformers

Our goal in this work is to identify the optimal architecture to extract information-capturing reusable representations. At the same time, we also need to find the optimal architecture to use such representation in conjunction with text inputs. So at a high level (as shown in Fig. 1), we need to develop two systems: (1) A model to compute the representation, Document Encoder and (2) A general model for tasks that can consume vector representations and text, Representation+Text Model. Given the recent success and generality of encoder-decoder models (Radford et al., 2018; Raffel et al., 2020; Lewis et al., 2020), we focus on developing models for such an architecture. We present the potential choices for each model, with the final model used in our system indicated by a *.

3.1 Document Encoder

Given an encoder-decoder model, there are different ways to compute representations for a document d with tokens $\{t_1, \ldots, t_n\}$. We focus on using the output representation generated by the encoder, represented with h_i for each token t_i.

Fixed Length Aggregation. The most common approach is to extract a single representation from a sequence of vector (Kiros et al., 2015; Conneau et al., 2017). While this can be a very compact representation of a document, it tends to be very lossy, especially when dealing with large documents. As a result, these representations are mainly used for classification (Conneau et al., 2017; Reimers and Gurevych, 2019) or retrieval (Karpukhin et al., 2020), and have not been shown to capture the content of the document. E.g, InferSent (Conneau et al., 2017) presented a self-attentive approach to extract sentence embedding using:

$$r = \sum_i U_\theta(h_i) h_i \tag{1}$$

where U_θ is a function that computes a scalar attention over each h_i. To reduce information loss, we extend these models to produce M representation vectors by learning M sets of parameters θ_j for $j \in \{1, \ldots, M\}$, i.e., $r_j = \sum_i U_{\theta_j}(h_i) h_i$ where $U_{\theta_j}(h_i) = e^{\theta_j h_i} / \sum_i e^{\theta_j h_i}$.

Special Token Representations. With the advent of transformer models, another common approach is adding a special `[CLS]` (Radford et al., 2018; Devlin et al., 2019) or `<s>` (Liu et al., 2019) token to the context. The output representation of this special token can then be used as inputs to classifiers and other down-stream models. Again, a single representation can be lossy, so we generate M representations by inserting multiple special tokens. We can dynamically adjust the number of special tokens based on the input length to produce a variable-length representation. To achieve a compression-ratio of $\frac{1}{k}$, we insert $\frac{N}{k}$ special tokens and use their representations.

We consider two ways[2] of inserting special tokens into the context: (1) **Suffix**: Add them at the end of the context[3] (2) **Interleave**: Add them after every k tokens. While the first approach preserves context continuity, the latter might more directly incentivize the model to capture local context.

Sliding Window Aggregation*. We apply the idea of aggregating single-vector representations to generate a variable-length representation. We apply an aggregation function F over sliding windows of size W tokens to capture the local context of the window (akin to CNNs). For a stride length of S, this would result in representation vectors:

$$r_j = F(\{h_{S \cdot j}, \cdots, h_{S \cdot j + W}\}) \tag{2}$$

where $F \in \{\mu, \alpha, \omega\}$ corresponds to mean-pooling, linear weighting (as described in Eqn. (1)), and max-pooling, respectively.

Figure 2 shows how we would compute these representations using a window-size of W=2 with no overlap (i.e. S=2) and the linear weighting function. The resulting READONCE Representations would have $M = N/2$ vectors where N is the number of tokens in the input.

SentenceBERT Baseline. For completeness, we also use an existing transformer-based SentenceBert model (Reimers and Gurevych, 2019)[4] to compute the representation of each sentence in the document. Since the space of these representation might

[2]More complex designs such as special token embeddings, position embeddings, and indicator features are left as future work.

[3]Prefixing special tokens generally worsened performance.

[4]We use the BERT-Large NLI tokens which performed better than the NLI-STSB representations in our experiments

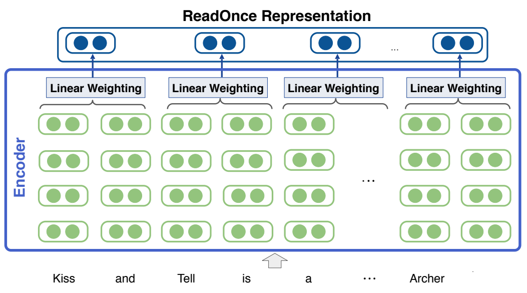

Figure 2: Sliding Window Aggregation approach to extract meaning representations from a transformer-based encoder. Linear weighted sum is used to aggregate each W=2 vectors from the final output layer into a single vector, resulting in the READONCE Representations with N/2 vectors.

be different, we learn a single-layer feedforward network to project the representations into the right space. For fair comparison to models with variable compression ratio k, we also use SentenceBERT representations for a sliding window of k tokens.

3.2 Representation+Text Model

Next, we present our modification to downstream task models to use both text and our generated READONCE Representations. Since most NLP tasks can be re-formulated as a text-to-text problem (Radford et al., 2018; Raffel et al., 2020), we focus on extending text-to-text encoder-decoder models to a (vec+text)-to-text model.

Append to Encoder*. Since the transformer block in an encoder can handle any input length in each layer, one possible approach is to append the representations to the Lth layer of the encoder. This allows the model to focus on parsing the input example text(e.g., question) in the L-1 layers followed by focusing on answering the question in the remaining layers. We show this model in Figure 3 where the encoder only processes the Q tokens of the question for the first L layers. Once the M READONCE Representations are added to the L^{th} layer, all the subsequent layers produce $M + Q$ vectors by attending over both the representations and text. Finally an unmodified decoder produces the output answer.

Modify Transformer Block Attention. Rather than just modifying the input, we consider an alternate approach of modifying the transformer block itself. Similar to PlotMachines (Rashkin et al., 2020), we view the representation as a memory that the self-attention block can attend over (in addition to the input text). We modify the self-attention blocks in both the encoder and the decoder[5] to use two separate attention modules for both of these input types and averages the vectors.[6] With this design, ideally the Representation+Text Model will gain extra capacity to model the interaction between the representation and the input text.

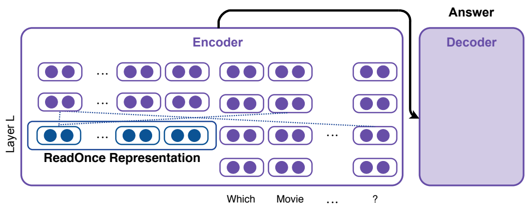

Figure 3: Appending the READONCE Representations to the L^{th} layer of the encoder to extend standard encoder-decoder models to handle text+vector inputs.

3.3 Training READONCE via QA

Given the overall architecture of such a system (shown in Fig. 4), we next focus on training this model to produce READONCE Representations that capture the information present in the document. While prior representation learning models have often focused on classification tasks, we instead use the reading comprehension QA task to ensure this information-capturing property. If a model is able to use just the READONCE Representations to answer the questions grounded in the document, the representations would contain the information needed to answer such questions.

The key question here is: *Which QA datasets are most suitable for training a compact yet information-capturing document representation?*

Low-level semantic QA datasets (Michael et al., 2018; He et al., 2015) don't allow for any compression as the questions require the knowledge about every word in the input sentence. More complex multi-hop QA datasets such as HotpotQA (Yang et al., 2018) are also not appropriate, as they focus on learning to reason in addition to capturing the information. Shallow reading comprehension tasks provide a sweet spot between these two extremes, as extracting key information from the given document is sufficient to answer the questions. Further, unlike semantic QA tasks, the questions only focus on the key facts mentioned in a document, which can be captured in a compressed representation. We

[5]Only modifying the encoder or decoder resulted in slightly lower performance.

[6]See App. A.1 for the detailed formulas.

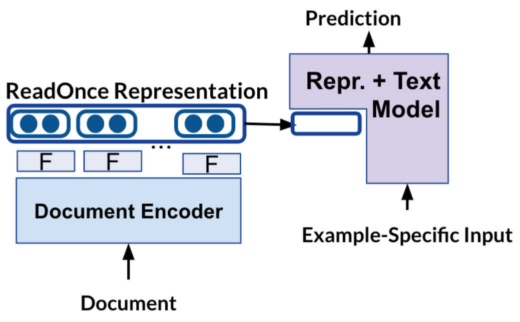

Figure 4: READONCE Transformers architecture. We use aggregated sliding window representations (Fig. 2) as Document Encoder to compute the READONCE Representations. We append these representations to the L^{th} layer of the encoder in our Representation+Text Model (Fig. 3). This end-to-end model is fine-tuned on QA tasks to train the Document Encoder to extract information-capturing representations.

use two such datasets to train our models: SQuAD and Unsupervised QA.

3.4 Downstream Usage of READONCE

To verify the generality of the READONCE Representations, we train models to perform multi-hop reasoning, abstractive QA and summarization using our learned representations. Specifically, we freeze the Document Encoder model and use it to generate the representations for documents. We further fine-tune the Representation+Text Model on the downstream task to produce the output label given the READONCE Representations and any example-specific input.

4 Representation Learning Experiments

We first evaluate the different potential architectural choices for extracting and using document representations discussed in §3.1 and §3.2, respectively. While our main interest is in learning effective representations, we also need to find the optimal Representation+Text Model architecture that can consume the representation.

4.1 Training Setup

We train the entire model on the factoid QA task to ensure that the document representations do capture factual knowledge. We primarily use the SQuAD reading-comprehension dataset (Rajpurkar et al., 2016) containing more than 100,000 crowd-sourced factoid questions. We further augment this dataset with about 500,000 rule-based questions from the UnsupervisedQA (UQA) dataset (Lewis et al., 2019). This increases the size of the training dataset while also introducing question diversity. To avoid these automatically generated questions overwhelming training, we ensure that the same number of questions are selected from both the datasets in each batch (by duplicating SQuAD questions). In the same vein, we evaluate each model based on their performance on the SQuAD task.[7]

Unless otherwise mentioned, we use the BART-Large model in all our experiments, and optimize the model with cross-entropy loss. We set the learning rate to 1e-5 for the weights initialized from the BART model, and to 1e-4 for randomly initialized newly added weights, which is shown beneficial in Peters et al. (2019). For other hyper-parameters, we follow Lewis et al. (2020). We ran all the experiments on RTX 8000 with 48GB GPU memory. All experiments did not use the complete GPU memory, e.g. experim We kept the batch size and gradient accumulation steps constant (both at 8) across different compression ratios.

4.2 Architecture Evaluation

To be able to evaluate the representations, we need to first select the architecture of the model consuming these representations.

4.2.1 Representation+Text Model

We explore the different choices for the Representation+Text Model model discussed in §3.2, assuming the representation is generated by a simple Document Encoder model: Mean aggregation over a Sliding Window with both window size and stride being 8 tokens. The results are shown in Table 1.

Architecture	Design Parameters	SQuAD EM	SQuAD F1
Append	L=1	35.0	52.6
Append*	L=6	**57.4**	**74.5**
Append	L=12	53.5	69.2
ModifyAtt	–	55.3	71.7

Table 1: Comparing BART-based architectures for jointly processing continuous representations and text.

We see that appending READONCE representations too early (L=1) or too late (L=12) in the encoder stack is not as effective as appending about half-way (L=6).[8] We suspect that appending too

[7]The scores on UQA correlate well with the scores on SQuAD, with close to 90 F1 for most models.

[8]We also experimented with L=3 and L=9, and didn't find any significant gains.

early does not allow the model to focus on understanding the question, whereas appending too late does not leave enough room for cross-attention between the question and representations.

Modifying the transformer block to attend over these representations results in a reasonable F1 score on SQuAD, but it is still outperformed by our simple *Append* architecture. Hence, for the rest of this work, we stick to the simpler architecture of appending the representation at the 6^{th} layer, denoted Append(L=6).

4.2.2 Document Encoder

Given the Representation+Text Model model architecture chosen above, we now explore potential Document Encoder architectures to extract READONCE Representations. For a fair comparison, we ensure that all our evaluated representations use, on average across a dataset, the same number of vectors to represent documents. Table 2 presents EM and F1 scores on SQuAD for the various architectural choices discussed in §3.1.

Architecture	Design Parameters	SQuAD EM	SQuAD F1
SlidingWindow*	W=8, S=8, F=α	**58.3**	**74.7**
SlidingWindow	W=8, S=8, F=μ	57.4	74.5
SlidingWindow	W=8, S=8, F=ω	48.8	64.6
SlidingWindow	W=16, S=8, F=α	53.1	69.6
Suffix	M=N/8	44.6	58.7
Interleave	M=N/8	19.0	31.0
SentenceBERT	M=N/8	17.8	29.6
SentenceBERT	M=#sent.	15.2	25.2
FixedLength	M=21	45.6	59.3

Table 2: A comparison of different architectures for extracting continuous representations using BART encoder. Each approach extracts representations of the same length, namely $1/8^{th}$ of the document length either for each document or on average across the dataset. Since SentenceBERT was trained on sentences, we also show results for SentenceBERT(M=#sent) with $N/32$ representations per document on average.

The top 3 rows explore the sliding window architecture with both window size and stride length of 8 (i.e., no overlap between windows), with the three different aggregation functions mentioned earlier. We see that both the mean μ and the learned weighted sum α have comparable performance on this task, and outperform the max-pooling function ω. We also evaluate the impact of increasing the overlap between windows by increasing the window size (not changing the stride length keeps the average number of vectors constant). For the learned weighted sum function, this results in a 5 point F1 drop, possibly due to the aggregation function having to operate over a larger window.[9]

We next evaluate the approaches inspired by prior work where we add special tokens and use the representations of these tokens. For the BART model, we use a newly added `[CLS]` token as our special token. We see from Table 2 that neither appending these tokens at the end nor interleaving them in the input results in representations comparable to the sliding window based approaches.[10] The sliding window representations outperform the pre-trained sentence-based representations from SentenceBERT irrespective of the number of vectors used.[11] Finally, if we fix the representation length to 21 vectors (computed based on the average token length of SQuAD: 163.7), the learned representations are still not as effective.

4.3 Final READONCE Architecture

Based on this set of experiments, we use the sliding window architecture for the Document Encoder with learned weighted sum as the aggregation function, and append these representations to the 6^{th} layer in the final task-dependent Representation+Text Model.

5 Downstream Task Experiments

Next, we evaluate the quality of our representations by using them on three downstream tasks, different from the tasks READONCE Transformers are trained on, demonstrating faster training and inference. We then show the benefit of using our representation when documents are much longer than the token limit of the underlying LM.

5.1 Experimental Setup

Tasks: We consider three end-tasks, extractive QA, summarization, and abstractive QA, to evaluate our system using the following datasets: (1) **HotpotQA** (Yang et al., 2018), a multi-hop reasoning extractive QA dataset. (2) **XSUM** (Narayan et al., 2018), an abstractive News summarization dataset (3) **NarrativeQA** (Kociský et al., 2018), an abstractive QA dataset where answers are not spans

[9] We also compared W=8, S=2 with W=2, S=2 in our early experiments and notice a similar trend—the smaller sliding window performs better.

[10] Special token prefix scored similar to the Suffix model.

[11] Even when the SlidingWindow approach is limited to $M = N/32$ vectors, it achieves a higher F1 score (52.4) than SentenceBERT.

from the input document. More details about these datasets and metrics provided in App. B

Baselines: We compare READONCE Transformers to BART-based QA models *that use the document text directly* to answer the given question. Since these models use text directly without any lossy compression, their score is best viewed as an *upper bound* for any representation-based BART model, including ours. We train the BART model to generate the answer given the entire document and question (we use "Summary" as question for XSUM). In addition to BART-Large, we evaluate two smaller models: BART-Base and DistilBART (Shleifer and Rush, 2020). Since our representations were trained on SQuAD and UQA, we also first fine-tune all our BART models on the same datasets.

READONCE Models: We freeze the parameters of the Document Encoder to generate the representations for all the documents in the datasets. We then use these representations with our Representation+Text Model, which is further fine-tuned on each end-task. To evaluate the impact of our pretraining on QA datasets, we compare our model to the READONCE architecture initialized with the BART model weights, READONCE$_\phi$. To illustrate the architecture-independence of our approach and orthogonality to traditional compression methods, we also train and evaluate READONCE models using the BART-Base and DistilBART models. These models were also first trained on SQuAD +UQA datasets to learn the document representation. See App. C for more details.

Since our Representation+Text Model can handle a variable number of representation vectors, we can change this *compression ratio*, on-the-fly, without having to change the model architecture. Specifically, we can use a stride-length of K in our Document Encoder to generate representations that are $1/K^{th}$ of the input length, and then feed them to a downstream model. By reducing K, we can reduce the compression ratio and improve the model accuracy, at the cost of increased runtime.

Interestingly, we discovered that we don't even need to re-train Document Encoder for each value of K. We can achieve a performance comparable to encoders trained individually for each value of K, by using the Document Encoder trained on $K = 8$ and only varying K during the fine-tuning step.

5.2 Representation Quality

First, we assess the ability of READONCE Representations to capture document information as compared to using the original document text. As shown in Table 3, our framework at K=2 is about 2x faster than BART-Large while being only 3 F1 and 4 Rouge-L points behind this model with full access to the text. This demonstrates that READONCE Representations do capture most of the relevant information in the document. The different compressed models can also result in smaller (DistilBART) or comparable (BART-Base) speed-ups, but (1) our accuracy vs speed trade-off is more easily controllable via K and (2) we can apply our framework on these models to achieve similar speedups.[12]

Architecture	HotpotQA F1 | sec.	Narr.QA R-L | sec.	XSUM R-L | sec.
BART-Large			
READONCE$_\phi$(K=8)	64.8 | 1.0	41.9 | 1.1	32.6 | 1.1
READONCE (K=8)	70.9 | 1.0	55.7 | 1.1	31.9 | 1.1
READONCE (K=2)	77.2 | 2.0	66.7 | 2.1	35.4 | 2.0
UB$_{\text{SQuAD +UQA}}$	80.1 | 4.0	70.5 | 5.7	37.2 | 4.0
BART-Base			
READONCE (K=2)	71.5 | 0.6	61.3 | 0.8	31.6 | 0.9
UB$_{\text{SQuAD +UQA}}$	76.5 | 1.3	65.8 | 1.9	32.2 | 1.5
DistilBART			
READONCE (K=2)	75.5 | 1.5	65.1 | 1.8	33.4 | 1.6
UB$_{\text{SQuAD +UQA}}$	80.5 | 3.7	70.5 | 5.3	36.5 | 3.6

Table 3: Performance of READONCE Transformers on three datasets, vs. corresponding text-to-text transformer models with full access to document text (i.e. **U**pper **B**ounds). We also show the training time (secs) per batch for each model. Our representations are able to reduce the training time compared to the upper-bounds at the cost of small drops in accuracy.

Lastly, we note that the READONCE$_\phi$ system, which simply uses the BART model parameters, is about 6 F1 and 14 Rouge-L points behind our model with learned representations. This shows that our model does utilize the factoid questions to learn to extract meaningful representations — without our training, the representations obtained from the pre-trained models are not as effective.[13]

[12]While more recent LMs can outperform BART (e.g. Pegasus (Zhang et al., 2020) for summarization), we believe similar tradeoffs can be achieved by applying our framework on these newer models.

[13]We also observe drops in score when using the BART model parameters in only the Document Encoder or only the Representation+Text Model.

5.3 Model Efficiency

One key advantage of READONCE Representations is that the model needs to *read* the document only *once*, and can reuse pre-computed representations for multiple examples or even multiple tasks. Specifically, if a document is repeated across R examples (the *replication factor*) and we use a compression ratio of K, our computation cost per question is roughly only $(1/2R + 3/4K^2)$ relative to a baseline seq2seq model (cf. App. C.3 for an analysis). In other words, the higher the replication factor R or the compression ratio K, the higher the speedup achieved via READONCE Representations.

Our model exhibits a speedup of 2x-5x in training time compared to the different BART architectures (Figure 5). Similarly, we observe a 2x-3x speedup in the inference time (as shown in Figure 6), which again plateaus out at K=8.

Note that the time reported for our model includes the cost of reading READONCE Representations from disk as well as some fixed costs. These costs form a larger fraction of the overall time for faster models. Hence, while our speedups do not exactly match up to the theoretical analysis, the empirical trends are as expected: we see larger speedups on the NarrativeQA dataset which has a higher replication factor R. In general, the R value for our datasets (e.g., R=29.7 for NarrativeQA) is within the range of other datasets (e.g., R=9.4 for NewsQA and R=13.9 for DROP). Note that even when R=1 (e.g., XSUM), we observe a speedup due to the compression ratio K.

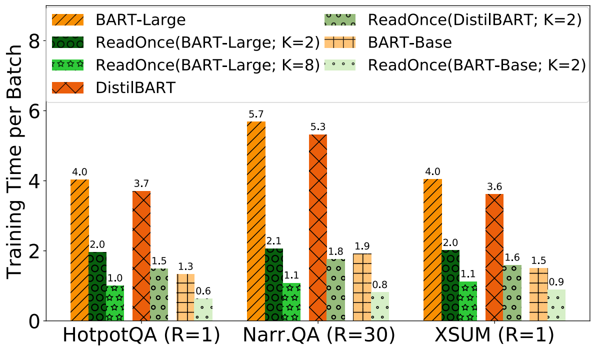

Figure 5: Training time (seconds) per batch. For READONCE models, document representations are pre-computed and cached, resulting in a 2-5x speedup.

5.4 Efficiency-Accuracy Tradeoff

We also perform a more fine-grained analysis of the efficiency-accuracy tradeoff in READONCE by varying the values of the compression ratio K. As shown in Figure 7, across all three of our datasets, as the value of K increases, the model's accuracy goes down due to increased compression but so does the training time. As compared to the upper-bound BART-Large model, we see a large gain in speed when K=2 with diminishing gains as K reaches 8.

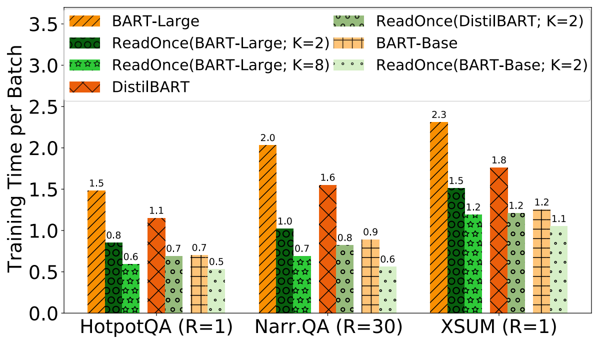

Figure 6: Inference time (seconds) per batch. For READONCE models, document representations are pre-computed and cached, resulting in a 2-3x speedup.

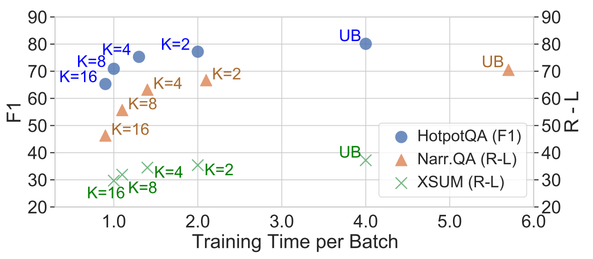

Figure 7: The training time (seconds per batch) vs performance trade-off achieved by the READONCE model with different values of K on our three evaluation tasks. The points annotated with "K=i" indicate the READONCE models and "UB" indicate their corresponding **U**pper **B**ound. All evaluations use the BART-Large model. As K increases, the READONCE model can be trained faster at the cost of accuracy with diminishing gains after K=4.

5.5 Handling Long Documents

Compressing document representations also enables the downstream model to reason over documents longer than its maximum token length limit T. For example, we can compute representations of document chunks with upto T tokens each and concatenate them together. Since these representations do not rely on any position embeddings in Representation+Text Model, theoretically we can use as many representation vectors as needed.

Given GPU memory limits, lets assume we can only accommodate documents upto length T. Given a compression ratio K, we can compute ReadOnce Representations for K such length-T chunks, increasing the capacity of our downstream model to T*K.[14] For simplicity, we ignore the question as it tends to be much shorter than T.

To assess the impact of increased model capacity, we evaluate our learned representations on the long document summarization task PubMed (Cohan et al., 2018).[15] We follow Cohan et al. (2018) and only include the first 4 sections from each document (average length=2270 tokens). We vary the memory budget from T=512 to T=256 and compare our approach to two BART seq2seq baselines: a simple truncation baseline with $T/4$ tokens from each section, and a sliding-window baseline often used in QA models for summarization extended here by concatenating summaries from length-T chunks of the input document. For the ReadOnce Transformer with a compression ratio of K, we can accommodate K*T/4 tokens per section, resulting in a total of T representations from the 4 sections. We choose to obtain these T representations using K/2 chunks from each section, with each chunk containing T/2 tokens.[16]

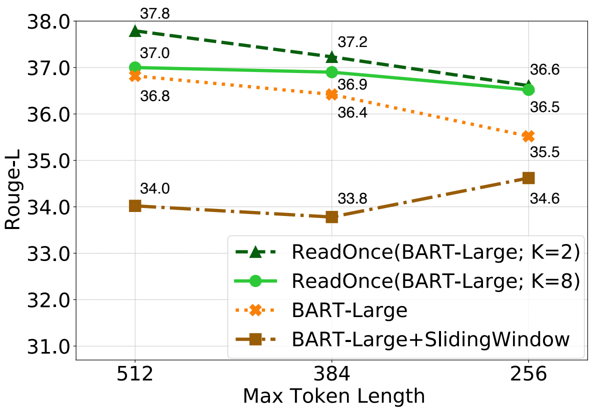

Figure 8: Accuracy of models under different maximum window length assumptions on PubMed dataset. ReadOnce Transformers stay substantially more accurate as the maximum window length decreases.

ROUGE-L scores of these models are depicted in Figure 8. As we reduce T for the underlying transformer model from 512 to 256, the score of the baseline BART model drops to 35.5 ROUGE-L. When used with the sliding window technique, the performance is even worse, likely due to the naive aggregation of the summaries. Our approach, on the other hand, concatenates document representations, allowing the downstream model to build a coherent summary. We see the ROUGE-L score only drops to 36.6 when K=2 (with model capacity dropping from 1024 to 512 tokens) and a much smaller drop from 37.0 to 36.5 when K=8 (with model capacity dropping from 3520 to 1472 tokens). This simulation shows that concatenating ReadOnce Representations is a simple yet effective way to increase the capacity of existing models.

[14]If we allow overlap of O tokens between chunks, the capacity changes to T*K – O*(K–1)

[15]We also evaluate NarrativeQA, see App. C.2

[16]See Appendix C.1 for more details.

6 Conclusion

This work introduced ReadOnce Transformers, a novel approach for using large scale transformer-based language models to both build and consume reusable document representations. Akin to humans' ability to read a document and extract useful information without knowing the end-use, ReadOnce Representations are compact, information-capturing document representations that can be pre-computed once, in a task- and example-independent fashion.

Our results on extractive QA, summarization, and abstractive QA tasks demonstrate that using ReadOnce Representations, in lieu of re-reading document text in the context of every example, results in substantially faster training and inference, at a modest cost in accuracy. The ReadOnce framework also offers an easy way to control the trade off between speed and accuracy (via the compression ratio parameter), and enables the use of standard transformer architectures on long documents beyond the model's token limit.

Identifying the ideal compact document representations in our controlled setting opens up the possibility of efficient open-domain QA, where models retrieve and reason directly over these representations. We leave an exploration of the training of the retrieval function, often with only answer supervision and ideally in an end-to-end setting, to future work.

Acknowledgements

We thank Dirk Groeneveld for providing the output of the Quark system for HotpotQA and the Beaker team for their support with the experiments.

References

M. Artetxe and Holger Schwenk. 2019. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. *TACL*, 7:597–610.

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

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *EMNLP*.

R. Caruana. 1993. Multitask learning: A knowledge-based source of inductive bias. In *ICML*.

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, W. Chang, and Nazli Goharian. 2018. A discourse-aware attention model for abstractive summarization of long documents. In *NAACL-HLT*.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *EMNLP*.

Hal Daume III and Daniel Marcu. 2006. Domain adaptation for statistical classifiers. *Journal of artificial Intelligence research*, 26:101–126.

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

Dirk Groeneveld, Tushar Khot, Ashish Sabharwal, et al. 2020. A simple yet strong pipeline for HotpotQA. In *EMNLP*.

Hangfeng He, Qiang Ning, and Dan Roth. 2020. Quase: Question-answer driven sentence encoding. In *ACL*.

Luheng He, M. Lewis, and Luke Zettlemoyer. 2015. Question-answer driven semantic role labeling: Using natural language to annotate natural language. In *EMNLP*.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *NeurIPS Deep Learning and Representation Learning Workshop*.

V. Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Yu Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *EMNLP*.

Ryan Kiros, Yukun Zhu, Russ R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *NeurIPS*.

Tomás Kociský, Jonathan Schwarz, P. Blunsom, Chris Dyer, K. Hermann, Gábor Melis, and Edward Grefenstette. 2018. The NarrativeQA reading comprehension challenge. *TACL*, 6:317–328.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*.

Patrick Lewis, Ludovic Denoyer, and Sebastian Riedel. 2019. Unsupervised question answering by cloze translation. In *ACL*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Julian Michael, Gabriel Stanovsky, Luheng He, I. Dagan, and Luke Zettlemoyer. 2018. Crowdsourcing question-answer meaning representations. In *NAACL*.

S. Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization. In *EMNLP*.

Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359.

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

Matthew E. Peters, Mark Neumann, Robert L Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. 2019. Knowledge enhanced contextual word representations. In *EMNLP*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding with unsupervised learning. Technical report, OpenAI.

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

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

Hannah Rashkin, A. Çelikyilmaz, Yejin Choi, and Jianfeng Gao. 2020. Plotmachines: Outline-conditioned generation with dynamic plot state tracking. In *EMNLP*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *EMNLP/IJCNLP*.

Sam Shleifer and Alexander M. Rush. 2020. Pre-trained summarization distillation. *ArXiv*, abs/2010.13002.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019a. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. In *NeurIPS*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2019b. GLUE: A Multi-task Benchmark and Analysis Platform for Natural Language Understanding. In *ICLR*.

Shuohang Wang, Yuwei Fang, Siqi Sun, Zhe Gan, Yu Cheng, Jingjing Liu, and Jing Jiang. 2020. Cross-thought for sentence encoder pre-training. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 412–421, Online. Association for Computational Linguistics.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. XLNet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*.

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

Manzil Zaheer, Guru Prashanth Guruganesh, Avi Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Minh Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Mahmoud El Houssieny Ahmed. 2020. Big bird: Transformers for longer sequences. In *NeurIPS*.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. 2020. PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization. In *ICML*.

A Model Details

A.1 The Detailed Architecture of Modified Transformer Block Attention

In this variation of Representation+Text Model, the modified self-attention block uses two separate attention modules for both of these input types and averages the vectors. Specifically, let $\mathbf{H}^L_{\text{enc}}$ be the matrix of hidden states generated from the L$^{\text{th}}$ layer of a standard transformer:

$$\mathbf{H}^L_{\text{enc}} = \text{Attn}(\mathbf{H}^{L-1}_{\text{enc}}, \mathbf{H}^{L-1}_{\text{enc}}, \mathbf{H}^{L-1}_{\text{enc}}) \quad (3)$$

where $\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ is the attention module used in the transformer that takes $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ as the query, key, and value matrices. To take extra READONCE Representations as an input, we instead compute $\mathbf{H}^L_{\text{enc}}$ as:

$$\begin{aligned}\mathbf{H}^L_{\text{enc}} =& (\text{Attn}(\mathbf{H}^{L-1}_{\text{enc}}, \mathbf{H}^{L-1}_{\text{enc}}, \mathbf{H}^{L-1}_{\text{enc}}) + \\ & \text{Attn}'(\mathbf{H}^{L-1}_{\text{enc}}, \mathbf{R}, \mathbf{R}))/2\end{aligned} \quad (4)$$

where $\text{Attn}'(\cdot)$ is a separate attention module to include the READONCE Representations $\mathbf{R}$ in our Representation+Text Model, whose weights are initialized by the corresponding weights in $\text{Attn}(\cdot)$ to speed up the training. For the decoder in the Representation+Text Model, we also compute the hidden states of each layer as per Eqn. 4 so that the model can attend over the extracted document information during the decoding process too.

B Dataset Details

(1) **HotpotQA** (Yang et al., 2018) is a multi-hop reasoning dataset that requires models to aggregate information from two paragraphs to produce the answer (a span from the input paragraphs). We focus on their *distractor* setting where they additionally provide models with 8 distractor paragraphs. For efficiency, we use the output of the Quark system (Groeneveld et al., 2020) which selects up to 512 tokens (including the question) from the input paragraphs. We use the answer EM and F1 scores as the metrics.

(2) **XSUM** (Narayan et al., 2018) is an abstractive News summarization dataset that requires models to generate summaries that go beyond simply extracting key sentences. We use the Rouge-L Summ. score[17] commonly used for summarization datasets, which computes the union-LCS of the longest common subsequences (LCS) between each pair of reference and hypothesis sentences. In contrast, the standard Rouge-L score computes LCS between the reference and hypothesis, treating both of them as one sentence.

(3) **NarrativeQA** (Kociský et al., 2018) is an abstractive QA dataset where answers may not be extractive spans in the input document. Models would need to understand the content of the document to generate such answers. We use the same Rouge-L Summ. score as for the summarization task.[18]

(4) **PubMed** (Cohan et al., 2018) is an abstractive long-document summarization dataset specifically focusing on the scientific publications. The large number of tokens in each document makes it hard for standard Pretrained Transformers to deal with. We use the same Rouge-L Summ. score as for XSUM.

C Experiment Setting for DistilBART

We follow Shleifer and Rush (2020) to obtain our DistilBART model used in §5. Specifically, we first create a student model with 12-layer encoder and 6-layer decoder from BART-Large$_{\text{SQuAD +UQA}}$ using the "Shrink and Fine-Tune" distillation described in Shleifer and Rush (2020), which has been shown to be effective for BART model on summarization tasks. We then further finetune the student model on SQuAD+UQA, and exploit the resulting model as our DistilBART.

C.1 Setup for the Pubmed Dataset in the Long-document Experiment

We follow Cohan et al. (2018) and only include 4 sections from document. After the truncation, the average number of tokens in the documents in this dataset is 2270, with 90% of the documents being under 4253 tokens. To include the information from each section, we evenly distribute the length budget T across the sections. This would mean, for the baseline BART seq2seq model, each section is first truncated to T/4 tokens, then the 4 sections are concatenated as the input. As for READONCE Transformers, we first compute representations for K/2 chunks of each section with length T/2 tokens, then aggregate them as the final READONCE Representations for the input document. In this case, even when K equals 2, we are allowed to include one chunk from each section without exceeding the

[17] https://github.com/google-research/google-research/tree/master/rouge

[18] In our experiments, we did not notice any substantial difference between the simple Rouge-L metric and this summarization-based metric.

length limit. For the BART + SlidingWindow baseline, we concatenate summaries from 16 chunks of length T with 4 chunks from each section.

C.2 Handling Long Documents: Narr.QA

Aside from Pubmed, we also evaluate the ability of READONCE Transformers to handle long documents on the NarrativeQA dataset. The average number of tokens in the documents in this dataset is 668.6, with 90% of the documents being under 981 tokens. The results are depicted in Figure 9.

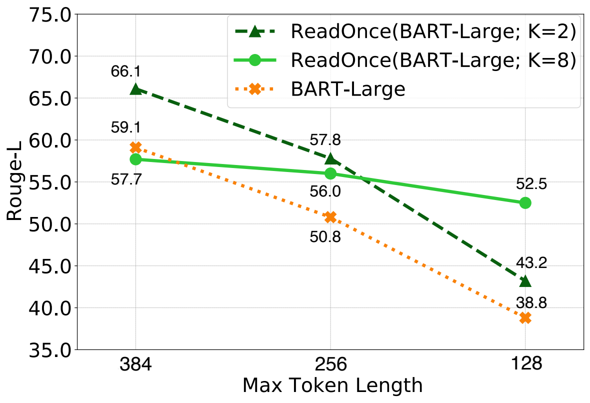

Figure 9: Accuracy of models under different maximum window length assumptions on NarrativeQA dataset. READONCE Transformers stay substantially more accurate as the maximum window length decreases.

As we reduce T for the underlying transformer model from 384 to 128, the score of the baseline BART model drops significantly from 59.1 to 38.8. With K=2, our model consistently outperforms the baseline model but exhibits a similar drop in score as the maximum token limit of this model with T=128 is still only 208 tokens (as per the equation above). On the other hand, with K=8 and T=128, we can handle documents up to 688 tokens, thereby requiring no truncation on 50% of the examples even in this extreme scenario. As a result, we only see a 5.2 point drop in score as T is decreased from 384 to 128. These simulations provide strong evidence of the ability of READONCE Transformers to handle long documents more effectively than standard transformer based models.

C.3 Read-Once Efficiency Gains

Let C denote context length (as #tokens), Q the question length, R the repetition factor (i.e., #questions per context), and K the READONCE compression factor. For an input of N tokens, we treat the computational cost of the encoder (forward or backward pass) as T_eN^2 and that of the decoder as T_dN^2, where T_e, T_d capture the complexity of the encoder and decoder model respectively and N^2 captures the self-attention over the input context. We ignore the self-attention over the decoded text as it is unchanged across models.

For any baseline seq2seq model, the computational cost of processing an example (context+question) can be captured by $(T_e + T_d)(C + Q)^2$. For READONCE Transformers, the cost of computing a context's representation, amortized over the R questions that share this context, is $T_e\frac{C^2}{R}$. Once the compressed context representation is appended to the question encoder at layer L (out of 12), the rest of the encoder computation costs $T_e \cdot \left(\frac{L}{12}Q^2 + \left(1 - \frac{L}{12}\right)\left(\frac{C}{K} + Q\right)^2\right)$. The decoder's computation cost is $T_d \cdot \left(\frac{C}{K} + Q\right)^2$.

When $L = 6$ and $Q \ll \frac{C}{K}$, the net computational cost (without caching) simplifies to $\frac{T_eC^2}{R} + \frac{T_eC^2}{2K^2} + \frac{T_dC^2}{K^2}$. Assuming $T_e \approx T_d = T$, this equals $TC^2\left(\frac{1}{R} + \frac{3}{2K^2}\right)$. In contrast, the baseline model's cost simplifies to $2TC^2$. The efficiency gain of READONCE Transformers over the baseline encoder-decoder model is therefore roughly $\frac{1}{2}\left(1/R + 3/2K^2\right)$.

Additionally, when we use these representations for multiple training runs, inferences, downstream tasks, etc., the cost of computing the fixed representations is basically amortized to a constant term. As a result, over multiple runs, using READONCE Representations now reduces the cost of the building and using models to just $TC^2\frac{3}{2K^2}$. So using these cached representations amortized over multiple epochs/runs, improves the efficiency gains further to $3/4K^2$.