**Federal state autonomous educational institution for higher education**
**«Moscow institute of physics and technology**
**(National Research University)»**

***On the rights of a manuscript***

**Arij Al Adel**

# INVESTIGATION OF TRANSFORMER OPTIONS FOR VARIOUS LONG DOCUMENTS PROCESSING TASKS

Specialty 2.3.5 - «Mathematical and software support for computers, complexes, and computer networks»

**Synopsis**
submitted in requirements for the degree of
candidate of technical sciences

**Supervisor:**
PhD of physical and mathematical sciences
Valentin Malykh

Dolgoprudny - 2024

**Федеральное государственное автономное образовательное учреждение высшего образования «Московский физико-технический институт (национальный исследовательский университет)»**

***На правах рукописи***

**Ариж Аль Адел**

# ИССЛЕДОВАНИЕ ВАРИАНТОВ ТРАНСФОРМЕРА ДЛЯ РАЗЛИЧНЫХ ЗАДАЧ ОБРАБОТКИ ДЛИННЫХ ДОКУМЕНТОВ

Специальность 2.3.5 – «Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей»

Диссертация на соискание учёной степени
кандидата технических наук

**Научный руководитель:**
кандидат физико-математических наук
Малых Валентин Андреевич

Долгопрудный - 2023

The dissertation was approved at ..... - name of the institute.

| | |
|---|---|
| Scientific supervisor: | PhD of physical and mathematical sciences, |
| | _____ |
| Leading organization: | name of the organization |

The defence of the dissertation will be held on date and place at the meeting of the dissertation council ФРКТ(number) at the Moscow institute of physics and technology, located at 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russian Federation.

The dissertation can be found in the library and on the website of Moscow institute of physics and technology: https://mipt.ru/.....etc

The work was submitted on month day, year to the Attestation committee of the Federal State Autonomous Educational Institution of Higher Education “Moscow institute of physics and technology (national research university)” for consideration by the dissertation committee for the candidate of science and doctor of science degrees in accordance with etc need to ensure what to write here

## General description of the work

The transformers have been limited to finite input lengths since their proposal [19]. This is because attention in transformers needs to attend to every token in the input. Research efforts since then have been spent to avoid this bottleneck by simplifying transformer structure [21] or using new technical tricks to mitigate the overwhelming computational cost.

### Scientific Actuality of the Research

Transformers have state-of-the-art results on most machine-learning tasks. The attention mechanism in the transformer is one of the main components of the transformer. Different attention variants were proposed for different tasks, leading to different transformer models.

The main bottleneck for transformers to solve NLP tasks is attention complexity. The longer the text input, the more computational cost is needed.

Processing long documents is an NLP research trend nowadays for many reasons in different areas. Most real-life domains such as research, education, media, jurisdiction [15], finance, health, etc. tend to produce medium to long documents. For example, for a researcher to switch to a new research field, he needs to read a tremendous body of research papers. Systems that help him find quick answers to specific questions need to be able to process long scientific papers to produce an answer on demand.

Generally, pre-training and finetuning transformers are expensive because attention needs to compute the similarity between each token in the input. This is quadratic computation cost. According to the previously mentioned bottleneck, this cost increases as long as the length of input increases. That is why more structure investigation is needed to benefit from the transformer's power and use it in processing long documents. Hence, we need to tune the transformer and get its benefit in processing long documents using new transformer structural modifications, and this is what this thesis topic is about.

### The goal of the dissertation

The main goal of this dissertation is to suggest and study different structural modifications of the encoder-decoder model to process longer input. These modifications are different combinations of attention, masking, position bias, and additional memory to the transformer to process long text inputs and check the proposed techniques on tasks such as translation, masked language modeling, question answering, and summarization tasks.

To achieve this goal, it is necessary to solve the following **Tasks**:

1. Systematically study and review publications on the application of attention mechanism

in transformer for different tasks concentrating on long text inputs to suggest a new way of processing long input;

2. Propose and implement new transformer structures using different masking patterns, position bias algorithms, additional memory tokens to inputs, and new attention schema so the model can process long inputs;

3. Apply the proposed model designs to four specific tasks: translation, masked language modeling, question answering with context, and summarization;

4. Based on the conducted experiments on given tasks, analyze the performance of the different implemented designs;

5. Propose new encoder-decoder architecture modifications based on t5 architecture for processing input longer than standard input length;

6. Propose new methods for the additional memory capacity of the encoder-decoder transformer;

7. Develop related methods of masking and relative positional encoding for the new additional modifications on the encoder-decoder transformer;

8. Investigate the effectiveness of the resulting modifications on the problems of translation, mask language modeling, question answering with long-range context, and summarization of long inputs;

9. Develop and publish in the public domain the implementation of the proposed structural modifications and results with analysis.

**Scientific Novelty**

This dissertation introduces a transformer with additional memory for processing long documents. This model and its variants were applied for various tasks on long text input like translation, question answering, and summarization. This work is different from previous works in:

1. Propose new encoder-decoder architecture structural modifications based on t5 architecture for processing input longer than standard input length;

2. An investigation of the effectiveness of these modifications applied to translation, mask language modeling, question answering with long-range context, and summarization of long inputs tasks.

**Theoretical value of the work in the dissertation** The theoretical value of the work in the dissertation lies in the comprehensive exploration of the proposed variants of transformer modifications for processing long input.

The thesis overcomes existing attention issues (limited transformer input length) by suggesting new structural modifications to the encoder-decoder transformer structure. These new modifications led to breaking attention barriers for processing long-range inputs.

Furthermore, this thesis presents a new usage for the internal global tokens called memory tokens, proper masking technique, and proper usage for the original relative positional encoding to relate chunked input with related memory slots.

**Practical value of the work in the dissertation** The proposed modifications were applied to different tasks using different data sets. They resulted in consuming long-range input, information interchange, and data compression. Models trained as part of the dissertation work are made publicly available.

**Statements to be defended**

- The developed encoder-decoder architecture supplied with a sentence selector for in-context-translation improves the translation quality;
- The proposed T5 modifications for MLM in separate input chunks have proven the ability to process longer inputs by introducing a memory system for information inter-flow between chunks, where the memory slots are the only connection between the chunks;
- The proposed architecture is based on additional trainable parameters, masking attention, and pre-trained weights reusing has proven its efficiency in scaling input length for summarization tasks compared with T5 and SLED models.

**Presentations and Validation of the Research Results**

The main findings and contributions of the dissertation were presented and discussed in three conferences:

- Engineering and Telecommunications Conference, MIPT University, Moscow, Russia, November 24-25, 2021.
- NEUROINFORMATICS-2022 International Conference on Artificial Neural Networks
- NEUROINFORMATICS-2023 International Conference on Artificial Neural Networks

**Publications**

1. **Al Adel A.**, Burtsev M.S. (2021). Memory transformer with hierarchical attention for long document processing. 2021 International Conference Engineering and Telecommunication (En&T), 1-7. Url: `https://ieeexplore.ieee.org/document/9681776`

2. **Al Adel, A.** (2022). Global Memory Transformer for Processing Long Documents. In Advances in Neural Computation, Machine Learning, and Cognitive Research VI. NEUROINFORMATICS 2022. Studies in Computational Intelligence, vol 1064. Springer, Cham. https://doi.org/10.1007/978-3-031-19032-2_36

3. **Al Adel, A.** (2023). SAMDIT: Systematic Study of Adding Memory to Divided Input in the Transformer to Process Long Documents. In: Kryzhanovsky, B., Dunin-Barkowski, W., Redko, V., Tiumentsev, Y., Klimov, V. (eds) Advances in Neural Computation, Machine Learning, and Cognitive Research VII. NEUROINFORMATICS 2023. Studies in Computational Intelligence, vol 1120. Springer, Cham. https://doi.org/10.1007/978-3-031-44865-2_10

**The author's contribution** in the works with co-authors is planning all the experiments, studying, implementing, and experimenting with the model in all stages, code implementation, training models, and baselines into analyzing the results and publication.

**Thesis structure and content**: The dissertation is divided into seven chapters:
**Introduction** delves into natural language processing, starting with its definition and ending with its main applications, such as translation, question-answering systems, masked language processing, and summarization. It gives an overview of each task and the related work for long inputs.
**Chapter 1** starts with a glimpse of natural language processing, starting from definition and ending with the main applications in translation, question-answering systems, masked language processing, and summarization. It displays an overview of each task and its related work for long inputs.
**Chapter 2** is dedicated to the evolution of the model and its mathematical representation throughout the design process. It introduces the preliminary design of the model used in the translation experiments. The section also covers the various modification variants used for subsequent tasks. Additionally, this section provides a detailed description of the model components and their mathematical representation as follows.

**Model design for translation task** The first design version used in the translation tasks is depicted in figure 1

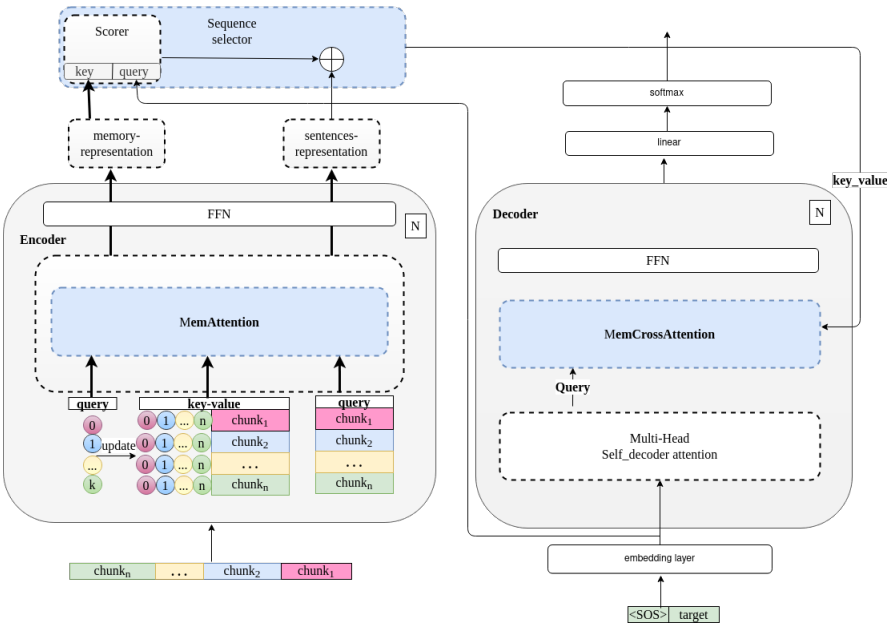


Figure 1: The overall view of the model structure used for the translation task.

**Model design for MLM and QA tasks**: For masked language modeling and question-answering tasks, several modifications were made to the model design, Fig. 2 shows the first modification. This modification suggests feeding the selector with a query resulting from the decoder's self-attention.

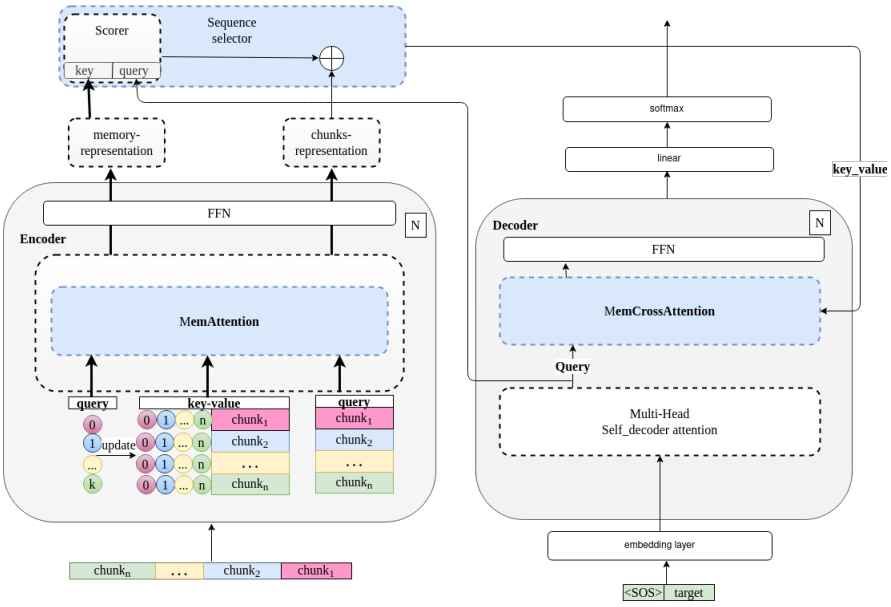


Figure 2: The first modified model structure used for masked language modeling and QA tasks.

### Model design variants for QA and MLM tasks

The computational cost of the selector was reduced in the second modification, as shown in Fig. 3

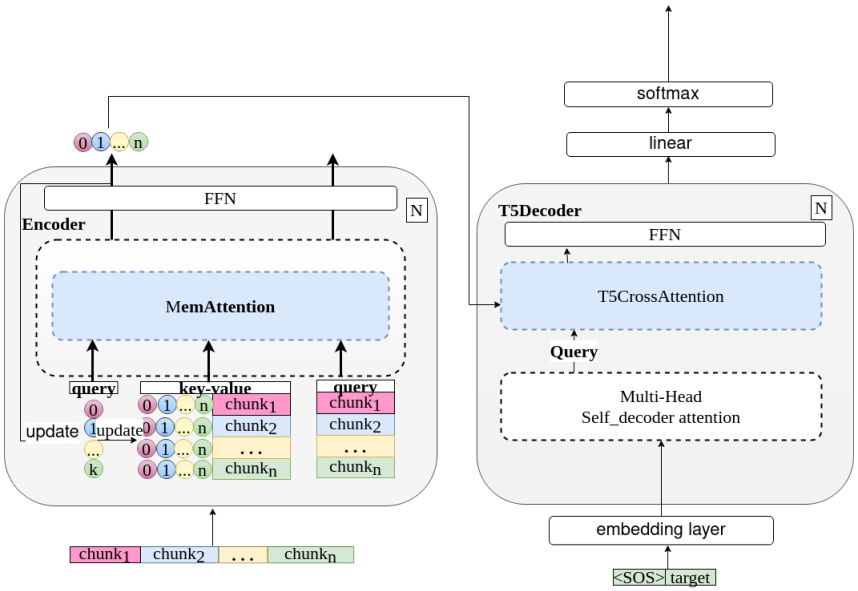


Figure 3: The second modified model structure used for masked language modeling and QA tasks.

The third modification was also to mitigate the computational cost that resulted from adding the selector and to check the importance of the MemAttention. This modification is reflected in figure 4

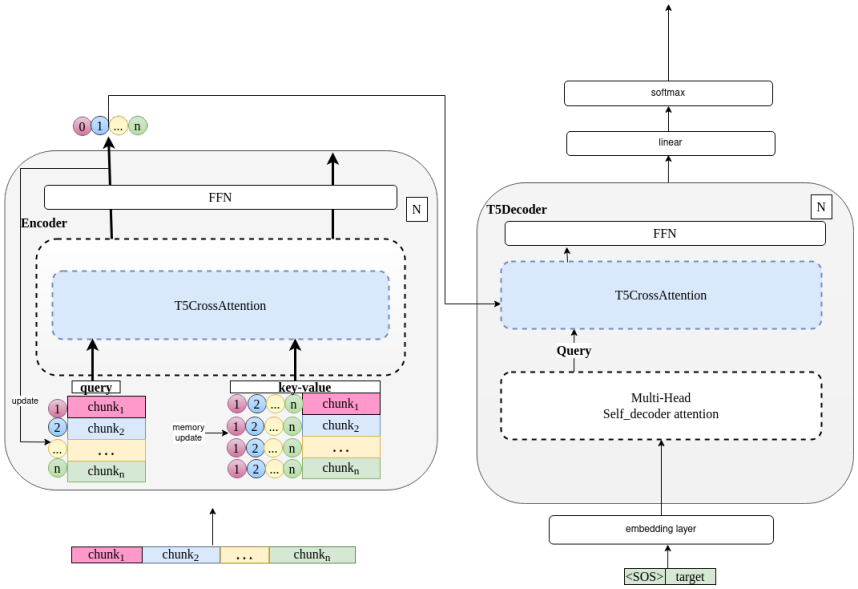


Figure 4: The third modified model structure used for masked language modeling and QA tasks.

**The mathematical representation of the translation model** This section introduces the mathematical representation of the model displayed in figure 1.

- **Encoder**: Each encoder layer in a transformer model consists of two sub-layers, namely

a T5MemAttention layer and a feed-forward layer. These two layers differ from the layers present in the T5 transformer. The process of encoding involves the following steps: The encoder input could either be a single sequence or a source sequence with (n) previous sequences as context. In the latter case, the model needs to learn which of these sequences is the source and which ones are from the context. It also determines whether the context sequences can aid in improving the translation or not. If the encoder input is divided into multiple sequences, then each sequence will have its slot memory.

Each memory slot consists of one or many memory tokens. Memory tokens are simply T5 tokenizer's special tokens. Slot memory is added to each sequence inside the encoder and applies attention between the memory slots and the sequences as follows: every slot has access to all other slots, and only to token representations of related sequences, and every token in a sequence has access to all tokens of the same sequence (inner sequence attention) and all slots with noting that sequences are encoded in parallel.

At the end of the encoding process, we will have two hidden states: memory slots hidden states and their related sequences hidden states. For positional encoding, we pay attention to encoding each slot memory with its sequence. No positional encoding between sequences was applied. This is the memory slot's mission.

- **Positional encoding**: There was no positional encoding between consequent input sequences. T5 default positional encoding was used instead in a way that take into consideration the position between each memory slot and its related sequence. We left the mission of positional encoding between sequences to the memory slots and sequence selector.

- **Multi-head encoder cross attention**:

  We call this attention T5MemAttention. It takes as input a combination of $X^{mem+sent}$, which represents each memory slot along with its related sentence. This input is split into two separate queries within our attention. The input also consists of $X_c^{mem+sent}$ of dimension $d_k$ for each head. This is the sentence prefixed by all memory slots of all sentences from the same input sample in the same order in the sample sentences. It is used to represent the keys and values in T5MemAttention.

  Inside T5MemAttention, the memory slot is separated from its related sentence to get two queries, keeping the key and value without any change and then calculating attention for each memory slot and related sentence independently. The output of the attention layer is:

$$att = LN(X^{mem+sent}) + T5MemAttention(X^{mem+sent}, X_c^{mem+sent}, X_c^{mem+sent}), (2)$$

- **Feed forward network**: After attention, we need to apply FFN as a traditional transformer as follows:

$$H = LN(X) + FFN(X), \qquad (3)$$

to remember X here is the output of the previous attention layer which is the concatenation between memory slot representation and the related sentence representation; $X = A = [A^{mem}_{mems+sent}; A^{sent}_{mems+sent}] \in R^{(c+L)*d_{model}}$ the dimension of the hidden layer of FFN is $d_{ff} = 2048$

- **Sequence selector**: The output of the encoder is separated into memory slots representation $H^{mem}$ and related sentence representation $H^{sent}$, to recall, before encoding we have reshaped the memory representation sentence representation input embedding, so after encoding we return the original batched shapes; batched memory representation after encoding $\in R^{b*k*c*d_{model}}$. The batched sentences representation is $\in R^{b*k*L*d_{model}}$; b is the batch size, k is the number of input sentences including the source and context, c is the number of memory tokens in the memory slot and $L$ is the length of each sentence in the input. The inputs of the sentence selector are three; query comes from the decoder(model output), key memory representations from the encoder, and sentences representations from the encoder. To choose the next token to be generated from one of the source-context sequences, we need to calculate the weighted score between the decoder output and memory slots resulting from the encoder. We calculate the score between the decoder and memory slots. Then, each sequence representation is multiplied by its score. The weighted representations are summed up to get one vector representation of the source size for each token in the target. This process can be expressed as:

$$S = softmax(Query * Key^T) \qquad (4)$$

where $Query = query * W^q; W^q \in R^{d_{model}*(c\_d_{model})}, c\_d_{model} = c * d_{model}$, $Key = key * W^k; W^k \in R^{d_{model}*d_{model}}$.

We note that the query is simply the embedded target as follows:

$$query = target * W_{emb}; target \in R^{vocab\_size}, W_{emb} \in R^{vocab\_size*d_{model}} \qquad (1)$$

This means that every target token has $k$ scores for each sentence in the corresponding input. Then we multiply each sentence representation with its score. Finally, the weighted

sentences representations are summed up to get one vector $\in R^{L*d_{model}}$ for each target token.

$$Z = \sum_{1<=i<=k} Sent_i * S_i, \qquad (5)$$

where $Sent_i = Sent_i * W^{sent}; W^{sent} \in R^{d_{model}*d_{model}}$

- **Decoder** Since the key-value vectors are not coming directly from the encoder but from the sequence selector which has one more additional dimension than the encoder output; this leads to having one vector for each target token, unlike the transformer which uses just one memory representation to generate all target tokens. Consequently, cross-attention in the decoder has been modified to handle this issue.
  - Decoder Multi-head self attention: The same as T5 self-attention
  - Multi-head encoder-decoder cross attention: This is the bottleneck of our model and is the future study topic in this thesis. Apply attention between sequence selector output and decoder self-attention output. We can express it as follows:

$$MHA(Q, K, V) = diagonal(concat(head_1, .., head_h)W^O), \qquad (6)$$

where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$.
$W_i^Q, W_i^K, W_i^V \in R^{d_{model} \times d_{head}}$ are the parameter matrices of linear transformation, $Q \in R^{T \times d_{head}}$ for each $head_i$ and it is the previous layer output in the decoder. K and V are the sequence selector output $\in R^{T \times L \times d_{head}}$ for each head. $W^O \in R^{d_{model}*d_{model}}$.

**The mathematical representation of the used model for MLM and QA Task** The mathematical representation of the modification is reflected on query 1 that is fed into the selector as follows:

$$query = DSA(target * W_{emb}); target \in R^{vocab\_size}, W_{emb} \in R^{vocab\_size*d_{model}} \qquad (2)$$

So, the query here resulted from applying caused self-attention in the decoder(DSA-decoder self-attention) on the embedded target.

**Model variants that were used for the MLM and QA Task:** In the mathematical representation for the first variant, which is depicted in Fig. 3, the sequence selector was

deleted, and the concatenation of memory slots was used as a compressed representation of related chunks as the key and value in the T5 encoder-decoder attention.

The key value representation of the encoder-decoder attention is given as a concatenation of memory representations of all related sequenced chunks in the batch after encoding:

$$K\_V = concat[H_{mem_1}; H_{mem_2}, ..., H_{mem_n}] \tag{3}$$

In the mathematical representation for the second variant which is depicted in Fig. 4 the sequence selector was deleted and the concatenation of memory slots was used as a compressed representation of related chunks as a key-value in the T5 encoder-decoder attention as in the first variant. The small modification that was added here to the first variant is replacing the T5MemAttention in the encoder with T5cross attention with added t5 positional bias.

**Chapter 3** This chapter covers the application of the proposed model on translation task. The task is to translate of a sentence combined with three previous sentences as a context. In this section a comparison between the context-aware translation and the context-agnostic translation for document-level translation. For this mission, a new T5 structural modification was proposed and investigated. The model is detailed in Fig. 1.

**Machine translation** is to translate text from the human source language to another human target language. The first version of the transformer was for context-agnostic translation tasks (sentence-level translation). Transformer was trained on a tremendous amount of sentence-level translation, but during the translation of long documents, translation happens for each sentence independently, which results in ignoring the contextual relations between tokens in sentences[22].

In this section, the **task** was to propose a model structure based on the encoder-decoder transformer, which can be used simultaneously for context-agnostic and context-aware translation. The context-aware translation was formulated as follows:

Given a corpus of tokens $\mathbf{x} = (x_1, x_2, ..., x_n)$ splitting it into $K$ equal segments where the last segment is the source that requires a response, and the previous $k-1$ segments represent the context. This formulation is suitable for translation tasks, question-answering tasks with the context, and reading comprehension tasks. **Dataset** The used data set is the same data set used by [20]. This data set was derived from OpenSubtitles2018 corpus [13]. This data set is for English and Russian translation tasks. After processing the steps mentioned in the paper appendix [20] on the OpenSubtitles2018 corpus. They got 6 million training instances from the resulting data for context-agnostic translation. From these 6 million sentences, there were

1.5 million sentences that have context. Each record in the context-aware data set consists of four consequent sentences for each language. $(s_1, ..., s_4), (s_2, ..., s_5), ..., (s_{n_3}, ..., s_n)$ from a group of consecutive sentences $s_1, s_2, ..., s_n$. Three context sentences were used to form the context-aware data set. The resulting context-aware data set has 1.5m sentences and has the context of three sentences. Two sub-sets of 10k instances were randomly chosen for development and testing sets.

**chunking approach** The main idea of the chunking approach is to chunk the long text after tokenization into $n$ equal chunks. For the translation tasks, since the data set was based on individual sentences and the context of these sentences in the case of presence the context, the data set was used for building the input into the model first. This data set gives the same length for each sentence in the source or the target. In this case, all sentences are padded or truncated to be the same length. In context-aware translation, the target is the translation of the last (fourth) sentence in the source sentence. Considering the previous consequent sentences are the context. Inside the model. The input in case sentence with its previous context sentences is divided equally into four chunks. Each chunk is one complete tokenized sentence. Each chunk is provided internally with its related memory slot as, explained in Fig. 5 in the encoder. These memory slots have their trainable weights.

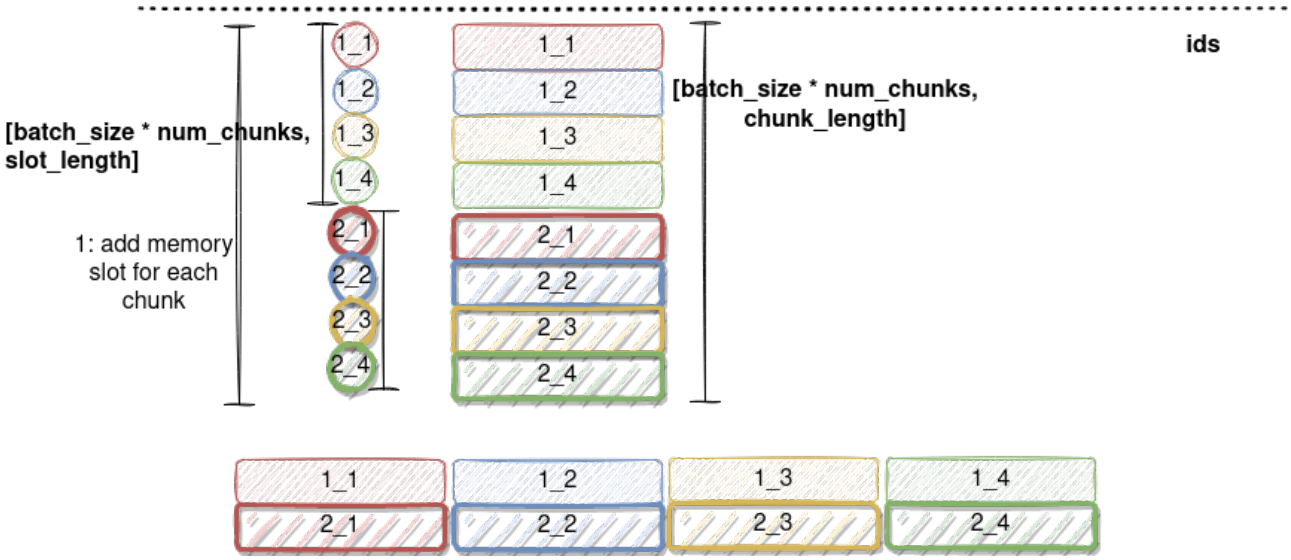


Figure 5: One memory slot for each chunk.

**T5MemAttention Design** In this section, the detailed description of the T5memAttention was initially modified from T5 Attention. Figure 6 displays an overview of the proposed attention design. The T5MemAttention layer in the encoder receives composed inputs, query(memory slot and its related chunk), key, and value (all memory slots of the long chunked input and the same chunk in the query). Inside this layer, the query is split into two separate queries.

Next, figures 7 and 8 detailed the design of the implementation of the attention. The only difference between them is using a linear layer of the attention output. In figure 7, just one linear layer is used for the attention output after concatenating the memory slot with its related chunk. In Fig. 8, two linear layers are used for the memory slot and its related chunk. For the rest of the T5MemAttention design, after separating the memory slot and related chunk, we calculate the weights as in T5Attention. After that, we concatenate the memory and chunk weights to calculate and add the relative positional encoding as calculated bias. Relative positional encoding was calculated similarly to positional encoding in T5 but included memory slot positions in addition to relating each chunk with its memory slot. After masking and adding the calculated bias, we separate the representations of memory and chunk to calculate the attention for each one. For applying the linear output layer, it could be as in Fig. 7 or in figure 8

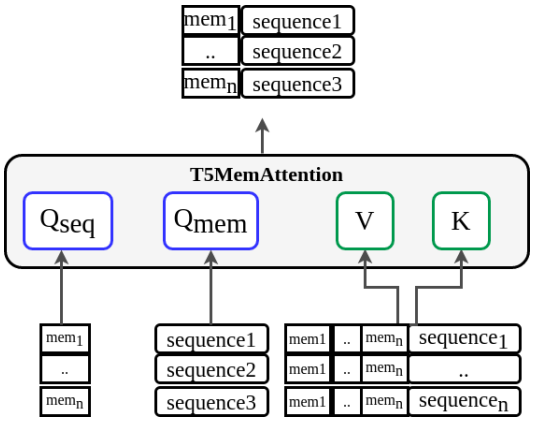


Figure 6: The T5MemAttention design inside the encoder of the proposed model.

This gives us two design variants. We need to experiment with the two variants to see which is better. Not just that, for the feed-forward layer, we also had two variants. Either use just one feed-forward layer for the output of the T5MemAttention as in figure 9a or a separate feed-forward layer for memory representation and its related chunk as in figure 9b. This put us in front of two additional design variants. In this case, we have four design variants to study.

**Design variants for translation task**

- v1: uses one linear layer for the output Fig. 7 inside the T5MemAttention to handle both memory slots and related sequences but two separated Feed Forward Fig. 9b after MemAttention; one for memory slots and the other for sequences representations.
- v2: uses one linear layer for the output Fig. 7 inside the T5MemAttention to handle both memory slots and related sequences and one Feed Forward Fig. 9a after T5MemAttention; for both memory slots and the other for sequences representations.

- v3: uses two linear layers for the output of T5MemAttention Fig. 8 to handle memory slots and related sequences independently, and two separated Feed Forward Fig. 9b after T5MemAttention; one for memory slots and the other for sequences representations, I should mention that this was the main proposed design of the model.

- v4: uses two linear layers Fig. 8 in T5MemAttention and just one feed forward Fig. 9a after attention.

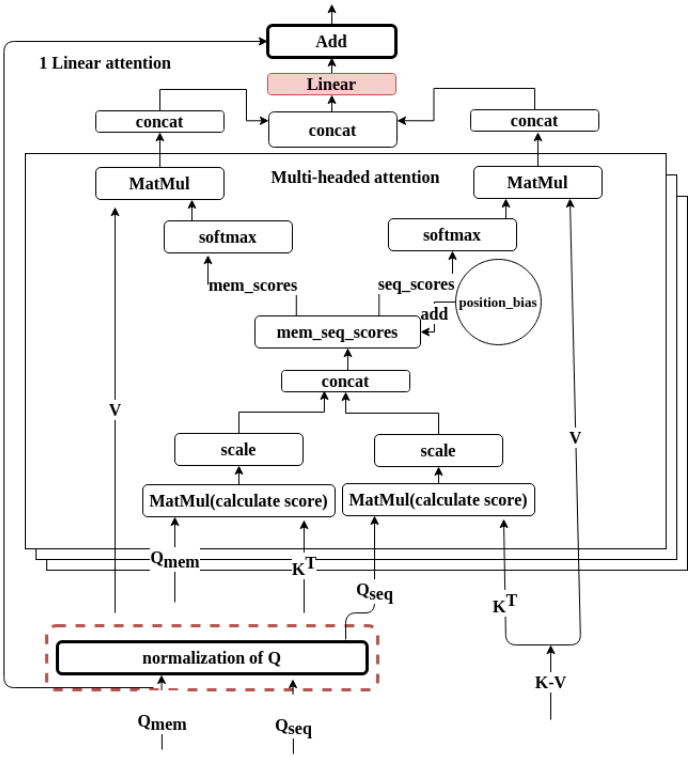


Figure 7: The internal design of the T5MemAttention uses the shared linear output for the memory slot and its related chunk concatenation.

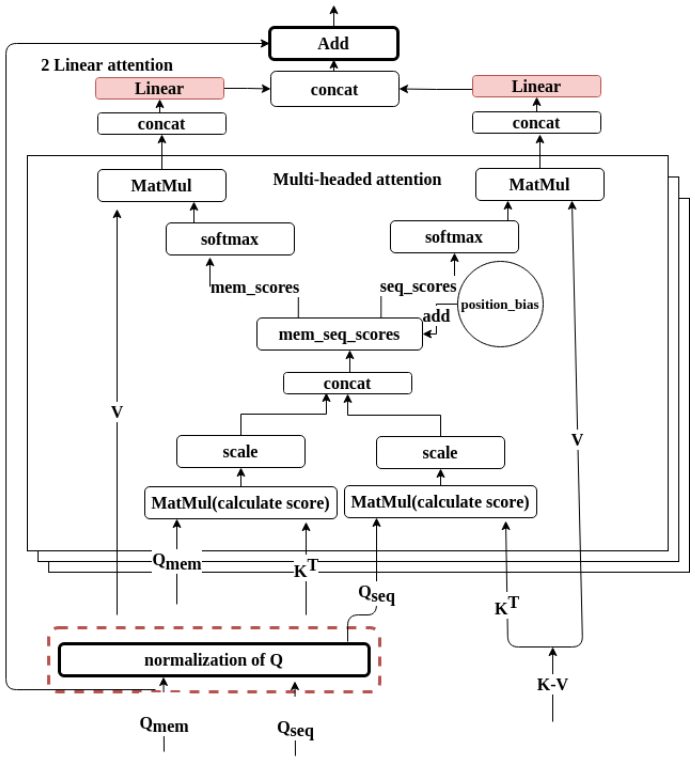


Figure 8: The internal design of the T5MemAttention uses separate linear output for the memory slot and its related chunk.

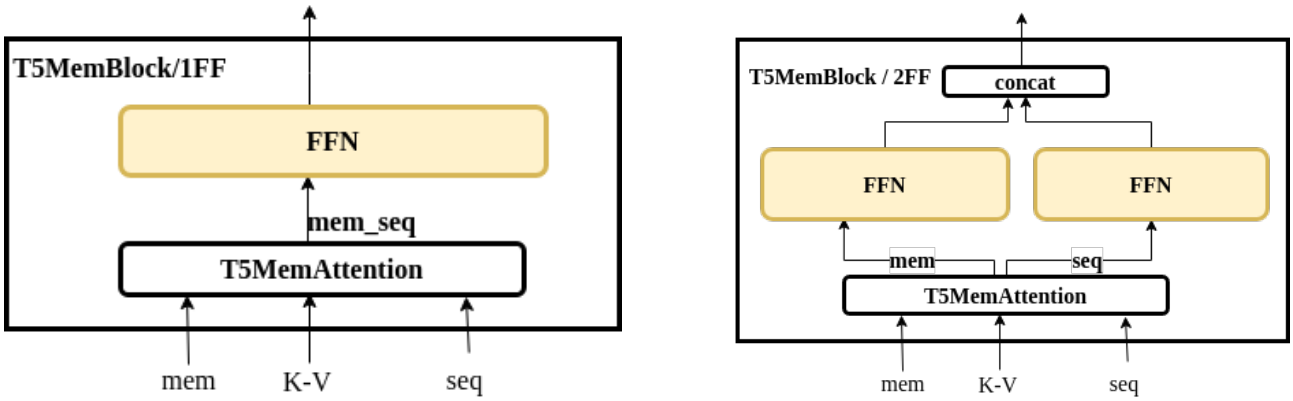


Figure 9: Figure (a) uses one feed-forward for memory and chunk representations concatenation, and figure (b) uses a separate feed-forward for memory and related chunks.

**Experiments setups** This section outlines our setup of experimental. SentencePiece tokenizer was trained on training data files [1]for both languages Russian and English [12] and was used to encode text as WordPiece tokens with source and target vocabularies of 32128 tokens for all experiments. We use a batch size of 160. length of all source sentences, and the target is 100

[1]Data used for training and pretraining model and tokenizer can be found: `https://www.dropbox.com/s/5drjpx07541eqstacl19_good_translation_wrong_in_context.zip?dl=0`

tokens. Adam optimizer was used with fixed learning rate $lr = 0.00005$ with no weight decay or warm-up. $d_{model} = 512, d_k = 512/8 = 64, h = 8, num\_layers = 2, d_{ff} = 2048$ If a source has one or more contexts then the input will be divided into the given number of sentences and rearranged in the shape shown in the encoder **Fig.** 1 for each sample in the batch. In our experiments, three context sequences were used as context. These are the settings for all experiments if it was not declared about different settings separately. For inference, the greedy search was used.

**Experiments results and conclusion**

- **Sentence level translation** Table 1 displays the results of the experiments using the four versions. The used data set is the context-agnostic data set for sentence-level translation [2]. Reference baseline for translation task is the T5 transformer.

| variant | train loss | val loss | test loss | val BLEU | test BLEU |
|---|---|---|---|---|---|
| $T5(baseline)$ | 1.455 | 1.596 | 2.349 | 26.12 | 25.648 |
| $v1$ | 1.466 | 1.632 | 1.634 | 26.34 | **26.725** |
| $v2$ | 1.472 | 1.642 | 1.639 | 26.09 | 26.437 |
| $v3$ | 1.465 | 1.629 | 1.621 | **26.5** | 26.706 |
| $v4$ | 1.476 | 1.636 | 1.639 | 26.34 | 26.354 |

[a] sacreBLEU was used `https://github.com/mjpost/sacrebleu`

Table 1: Loss and blue scores on Agnostic aware data set.

- **Context level translation**

  Table 2 displays the results of training the model from scratch on the context-aware data set. As shown, the model has a low BLEU score for all versions.

| variant | train loss | val loss | test loss | val BLEU | test BLEU |
|---|---|---|---|---|---|
| $v1$ | 2.93 | 3.859 | 3.8 | 1.533 | 1.019 |
| $v2$ | 2.946 | 3.86 | 3.837 | 1.944 | 0.882 |
| $v3$ | 2.94 | 3.787 | 3.753 | 1.597 | 1.543 |
| $v4$ | 2.908 | 3.823 | 3.806 | 1.698 | 1.291 |

Table 2: Loss and blue scores on Context-aware data set before finetuning, training the context-aware from scratch.

[2] `https://www.dropbox.com/s/5drjpx07541eqst/acl19_good_translation_wrong_in_context.zip?dl=0`

For fine-tuning the model has trained on context agnostic corpus on context-aware corpus. We use transfer learning by using the trained model itself to train the context-aware. This is possible because the same model can be used for processing one or more chunks using the chunking approach explained in the section. Using weights of the model trained on sentence-level translation to train the model on context-aware translation data set we noticed that the model got better BLEU scores for all versions. Results are displayed in table 3.

| variant | train loss | val loss | test loss | val BLEU | test BLEU |
|---|---|---|---|---|---|
| $v1$ | 1.322 | 1.678 | 1.665 | 26.12 | **26.615** |
| $v2$ | 1.374 | 1.7 | 1.685 | 26.03 | 26.120 |
| $v3$ | 1.33 | 1.698 | 1.667 | 25.88 | 26.356 |
| $v4$ | 1.303 | 1.656 | 1.652 | **26.64** | 26.582 |

Table 3: Loss and blue scores on Context-aware data set after finetuning.

There is a noticeable improvement from training from scratch on context-aware into fine-tuning after training on context-agnostic corpus.

**Discussion** We suggested using a transformer-based model to cope with attention limitation by adding another layer of hierarchical attention over document chunks. Specifically, we introduced the attention over memory slots related to the document chunks. Applying attention to memory slots to choose the proper chunk for the next generation step is cheaper than using the representation of the related chunks. Training the model on a context-agnostic data set and then tuning the model on a context-aware data set helped to get more precise translation and promoted translating context-aware sentences. Results showed superior BLEU scores for sentence-level translation for all model variants compared with the T5 transformer baseline, and high BLEU scores for context-aware translation.

**Chapter 4** presents the model that was used for masked language modeling and question-answering tasks, obstacles, solutions, results, discussion of results, limitations, and future work. In this section, we abandon the encoder output usage, replacing it with selector output[1] as shown in Fig. 4. This brings two advantages: First, much more focus and attention on the source and target tokens during generation, and second, depending on the most relevant part of the input to generate the token, the model pays attention to the overall input during generation. The main difference between our model and models[5, 2] is that they used sparse attention where chunking the input is happening inside the attention layer. Input chunking in our model is already done in the encoder before using the attention layer inside the model,

which makes it easier to implement. We have two main tasks; we use MLM pre-training task with the same objective used in [17] and use question answering finetuning task on HotpotQA.

This study does not aim to benchmark on any of the mentioned data sets in this chapter but to verify the ability of the proposed model to handle chunks as if they were one chunk compared with the base model as a step forward to increase the input length. Our work follows the line of works that added general memory to the transformer input [5, 1, 6] and uses T5 as a baseline to compare results and as a base structure to implement all the modifications.

**Data sets and metrics**

The used **data sets** for this chapter are:

- **For pre-training**: Since pre-training is a very costly process and for initial results and comparison, there was a need to choose a medium size data set for pre-training on mask language modeling for this purpose; wikitext-103-raw-v1 [3].

- **For the question-answering task**, the HotpotQA data set was used. HotpotQA is a challenging data set. It was introduced to encourage systems to learn more complex reasoning, where the evidence pieces to answer a question are scattered among different documents.

  HotpotQA consists of two parts full-wiki data set and a distractor data set. The two data sets are identical in structure. Gold paragraphs, i.e., the paragraphs containing the support facts, are included in the context paragraphs of the distractor validation data set. In the full-wiki validation data set, these gold paragraphs are not included in the context paragraphs. For this reason, the full-wiki data set is more challenging. I have used a distractor data set. HotpotQA contains the features: ['id', 'question', 'answer', 'type', 'level', 'supporting_facts', 'context'] and has 90447 training records and 7405 validation records. An example of data and data structure is displayed in table 4

[3] wikitext-103-v1-1 can be found here: `https://huggingface.co/datasets/wikitext#wikitext-103-v1-1`

| Column | Value |
|---|---|
| *id* | '5a7a06935542990198eaf050' |
| *question* | "Which magazine was started first Arthur's Magazine or First for Women?" |
| *answer* | "Arthur's Magazine" |
| *context* | ['Radio City is India's first private FM radio station and was started on 3 July 2001.', ' It broadcasts on 91.1 (earlier 91.0 in most cities) megahertz from Mumbai...] |
| *type* | 'comparison' |
| *level* | 'medium' |
| *Supporting_facts* | ["Arthur's Magazine", 'First for Women'] |

Table 4: Example of HotpotQA data set records.

Each context consists of a list of ten paragraphs. Each paragraph is a list of sentences. For each paragraph, all sentences were merged into one text. Then, all paragraphs were merged to form one context for each question.

To feed inputs to the T5 and T5MemModel, a data set class was built to tokenize question and context in the following way question_ids </s> context_ids </s> and target as target_ids< /s >. If the model uses more than one chunk, the input ids in the encoder are divided into the specified number of chunks without any positional encoding for the chunks.[1].

**Results of pre-training task MLM** We compared the results of the T5Mem model with T5(baseline) with two main input lengths: a) input length 128 table 5 model uses just one chunk and b) input length 512 table 6 either as one chunk or divided into four chunks each chunk of 128 length. As displayed in the table. 5 T5Mem model surpasses the baseline.

Table 5: Experimental results on train and dev set of wikitext-103-raw-v1 data set for Masked language modeling task using just one chunk as input length 128. The best results for the proposed model are highlighted.

| **Expriment name** | **Train loss** | **Valid loss** | Perplexity |
|---|---|---|---|
| **T5_basline_128** | 3.709 | 3.455 | 32.265 |
| **T5Mem_MLM_1_chunk_128** | **3.122** | **3.417** | **22.78** |

Table 6 T5Mem model surpasses the baseline for both variants using just one or four chunks. However, using just one chunk results are still better than four chunks. This may return to the loss of full representation because of chunking in the encoder part. For the final goal to process longer text, having more than one chunk is acceptable since the final result is still comparable or competitive as we can compare line 5 and line 1 in the table 6. If we continue training the

pre-trained model on length 128 using length 512 even for baseline (t5) or our model, then this will not give a better result than training the model from scratch using the 512 length. This can be noticed by comparing line 1, which refers to training from scratch, and line 2, which refers to the continued training for another 100 epochs using longer input for baseline, and lines 4 and 5 for the proposed model. As a result, increasing input length while pre-training did not bring any good, and using a lower chunk number with shorter lengths was more efficient than longer ones. As a result, The proposed model using four chunks or one chunk outperformed the baseline for 128 input length and 512 input length as one chunk or 512 input length divided into four chunks, each chunk of 128 lengths as seen in tables 5 and 6.

Table 6: Experimental results on train and dev set of wikitext-103-raw-v1 data set for Masked language modeling task using input length 512. Input length for all models is 512 length as one chunk or divided into four chunks of 128 chunk length. 128_to_512 means continuing pre-training the model using 512 input length after training it for 100 epochs using input length 128 since the model is resilient to the input length.

| **Experiment name** | **Train loss** | **Valid loss** | Perplexity |
|---|---|---|---|
| **T5_basline_512** | 4.327 | 4.204 | 66.94 |
| **T5_128_to_512** | 4.772 | 4.589 | 98.46 |
| **T5Mem_MLM_1_chunk_512** | **4.101** | **4.186** | **60.422** |
| **T5Mem_MLM_1_to_4_chunks**[a] | 4.735 | 4.5147 | 91.35 |
| **T5Mem_MLM_4_chunks**[a] | **4.281** | **4.184** | **65.66** |

[a] Model has been trained once.

**Results of fine-tuning proposed model on HotpotQA**

For fine-tuning the baseline and the proposed model on the HotpotQA data set. In table 7, the model did not outperform the baseline when using four chunks. This can be interpreted as follows: the main idea of the selector component in the model is to pay more attention to one chunk than to other chunks. The answer could be scattered in many chunks, so concentrating the attention on just one chunk can be the reason for this degradation. This is why it worked well for the translation task but not the question-answering task.

Table 7: Experimental results of fine-tuning the proposed model on HotpotQA data set(distractor part) using input length as 512 even as one chunk(T5 takes input as a chunk of 512 lengths) or 512 input length is divided into four chunks. Lines in grey use Adam Optimizer. All experiments were pretrained and fine-tuned using a constant learning rate.

| **Experiment name** | **Train loss** | **Valid loss** | **Test loss** | **exact match** | **f1** | **recall** | precission |
|---|---|---|---|---|---|---|---|
| **T5_hp_AF_const_512** | 0.757 | 3.231 | 3.231 | 17.008 | 24.619 | 24.995 | 25.825 |
| **T5_hp_Adm_const_512** | 0.755 | 3.285 | 3.285 | 16.819 | 24.175 | 24.522 | 25.279 |
| **T5Mem_hp_4_chunks_AF_const** | 1.696 | 4.024 | 4.024 | 8.727 | 14.007 | 13.977 | 14.894 |
| **T5Mem_hp_4_chunks_Adm_const** | 2.416 | 4.150 | 4.150 | 4.090 | 5.507 | 5.480 | 5.781 |

From here, the idea of dropping the selector to try other design variants comes to use.

The following subsection displays the model variants without a selector component results.

### Results of modified variants on MLM task

Results of experiments on these two new variants compared with the previous results presented in the tables 8 and 9. To remember the new proposed design variants are:

- T5MemWsWMA T5memory model **w**ithout **s**elector component as explained in chapter 2 and displayed in Fig. 3
- T5MemWsWMA T5memory model **w**ithout **s**elector component **w**ithout **m**emory **a**ttention as explained in chapter 2 and displayed in figure 4

For clarity, in the following tables, after the name of the model, we have the suffix $(\_2mem.AF, linear)$, which means $(\_(n)mem.$ name of the optimizer, name of the scheduler). Where $n$ number of memory tokens in each memory slot, $n = 2$, that means that the memory slot is composed of 2 memory tokens.

Table 8: Experimental MLM task results using just two layers - training for 100 epochs input length is 512 for T5. For the T5Mem model and its variants. The input length 512 is divided into four chunks. optimizers(Adafactor "AF" and AdamW "Adm"), the used scheduler is **linear**. All models use two memory tokens for each chunk as a slot, except for the T5MemWsWMA_1mem model, which utilizes one memory token. Lines in grey use Adam Optimizer. The best results for the Adafactor optimizer are underlined. The best results for Adam Optimizer are bold.

| Model | Train loss. | Valid loss. | val_acc. | val_PPL | Test loss. | test_acc. | test_PPL |
|---|---|---|---|---|---|---|---|
| T5. AF,linear[1] | 4.327 | 4.196 | 30.060 | 66.394 | 4.163 | 30.218 | 64.289 |
| T5. Adm,linear | 4.368 | 4.258 | 27.605 | 70.648 | 4.234 | 27.675 | 68.991 |
| T5Mem. AF, linear[2,3] | 4.286 | 4.171 | 29.775 | 64.769 | 4.140 | 29.808 | 62.816 |
| T5Mem Adm,linear[2] | 4.422 | 4.301 | 27.620 | 73.754 | 4.280 | 27.919 | 72.258 |
| T5MemWs_2mem. AF,linear | 4.193 | 4.093 | 30.090 | 59.979 | 4.091 | 30.891 | 59.779 |
| T5MemWs_2mem. Adm,linear | 4.247 | 4.143 | **30.210** | **62.998** | 4.113 | **30.397** | **61.158** |
| T5MemWsWMA_1mem. AF,linear | 4.149 | 4.050 | 31.145 | 57.417 | 3.999 | 31.355 | 54.589 |
| T5MemWsWMA_1mem. Adm,linear | 4.307 | 4.202 | 28.316 | 66.873 | 4.176 | 28.316 | 65.117 |
| T5MemWsWMA_2mem. AF,linear | 4.196 | 4.069 | 31.010 | 58.497 | 4.0217 | 31.262 | 55.813 |
| T5MemWsWMA_2mem. Adm,linear | 4.266 | 4.161 | 29.985 | 64.111 | 4.136 | 30.103 | 62.548 |

[1] T5. AF, linear refers to the model T5_baseline_512. The name in the table is to make it easy to compare in this table.
[2] The Model has been trained once.
[3] T5Mem. AF, linear refers to the model T5Mem_MLM_4_chunks. The name was changed in the table to make it easy to compare.

Table 9: Experimental MLM task results using just two layers - training for 100 epochs input length is 512 for T5. For the T5Mem model and its variants. The input length 512 is divided into four chunks. optimizers(Adafactor"AF" and AdamW "Ad"), the used scheduler is constant. All models use two memory tokens for each chunk as a slot, except for T5MemWsWMA_1mem, which uses one memory token. Lines in grey use Adam Optimizer. The best results for the Adafactor optimizer are underlined. The best results for Adam Optimizer are bold. For all best results, the baseline is superior.

| Model | Train loss. | Valid loss. | val_acc. | val_PPL | Test loss. | test_acc. | test_PPL |
|---|---|---|---|---|---|---|---|
| T5. AF,const | 2.486 | 2.308 | 59.225 | 10.0595 | 2.265 | 60.150 | 9.634 |
| T5. Adm,const | 2.495 | 2.309 | 59.59 | 10.060 | 2.269 | 60.523 | 9.673 |
| T5Mem. AF,const[a] | 3.094 | 2.892 | 52.47 | 18.020 | 2.864 | 52.742 | 17.533 |
| T5Mem. Adm,const[a] | 3.304 | 3.078 | 47.77 | 21.720 | 3.757 | 33.659 | 42.822 |
| T5MemWs_2mem. AF,const | 3.126 | 2.889 | 50.87 | 17.990 | 2.865 | 51.175 | 17.555 |
| T5MemWs_2mem. Adm,const | 3.117 | 2.886 | **51.005** | **17.916** | 2.864 | **51.213** | **17.531** |
| T5MemWsWMA_1mem. AF,const | 3.369 | 3.157 | 44.805 | 23.495 | 3.126 | 45.216 | 22.789 |
| T5MemWsWMA_1mem. Adm,const | 3.328 | 3.104 | 47.275 | 22.290 | 3.076 | 47.656 | 21.677 |
| T5MemWsWMA_2mem. AF,const | 3.280 | 3.086 | 47.585 | 21.541 | 3.055 | 47.961 | 21.225 |
| T5MemWsWMA_2mem. Adm,const | 3.277 | 3.085 | 47.465 | 21.88 | 3.0535 | 47.806 | 21.195 |

[a] Model has been trained once.

As we can see from Table 9, using a constant scheduler during the pre-training stage has a significant impact on the pre-training process. The baseline model outperforms all the proposed structural modifications. Moreover, the results show that MemAttention outperforms T5Cross attention, suggesting that MemAttention is superior to T5Attention for experiments with Adafactor optimizer. Using separate parameters for the memory slot and its related chunk helped to get this improvement. Appending just one memory token for each chunk in the T5MemWS_WMA_1mem model, we noticed that the quality of results was decreased using the Adam optimizer. This emphasizes that increasing memory slot capacity can improve the results. Baseline, which uses Adam optimizer and constant learning rate, gives better accuracy but not better perplexity compared with itself. For our model, T5Mem and all its modifications using Adam optimizer with a constant learning rate led to worse results regarding both accuracy and perplexity. Table 10 displays the results of the finetuned pre-trained models from table 8 and 9 on question answering task using HotpotQA data set. The same tendency of using a constant learning rate to improve the results was kept in the finetning stage. Still, the baseline is better than the proposed model, but both the baseline and the proposed model in all design variants start tackling the task. Interestingly, the results with dropped selector and MemAttention used were the best compared with all other model variants except for (T5Mem. AF, const) in the fine-tuning stage. This was the best among all model variant results. Using a constant learning rate gives better results on all experiments. Fine-tuning results again display preference MemAttention on T5cross attention and the role of compressed representation in auto-regressive response generation. The length used in these experiments was not enough to know whether feeding the model with longer inputs would be better. This is possible in future work. In the result, we understand that the selector does the mission using a linear learning rate and surpasses the baseline while the performance is on par during fine-tuning with the baseline. Using a constant learning rate enables the models to handle the finetuning task, the baseline is better for all the experiments that used the constant schedule. Using compressed representation for extractive responses is possible and promising. This can mitigate the expensive computational cost of cross-attention in the decoder while processing long documents.

**Results of modified model variants on HotpotQA data set**

Table 10: Experimental results of fine-tuning all pre-trained models and modified models on HotpotQA data set(distractor part) using input length as 512 even as one chunk(T5 takes input as a chunk of 512 lengths) or 512 input length is divided into four chunks. Lines in grey use Adam Optimizer. All experiments were pretrained and fine-tuned using a constant learning rate.

| Experiment name | Train loss | Valid loss | Test loss | exact match | f1 | recall | precission |
|---|---|---|---|---|---|---|---|
| **T5_hp_AF_const_512** | 0.757 | 3.231 | 3.231 | 17.008 | 24.619 | 24.995 | 25.825 |
| **T5_hp_Adm_const_512** | 0.755 | 3.285 | 3.285 | 16.819 | 24.175 | 24.522 | 25.279 |
| **T5Mem_hp_4_chunks_AF_const** | 1.696 | 4.024 | 4.024 | 8.727 | 14.007 | 13.977 | 14.894 |
| **T5Mem_hp_4_chunks_Adm_const** | 2.416 | 4.150 | 4.150 | 4.090 | 5.507 | 5.480 | 5.781 |
| **T5MemWS2mem_hp_4_chunks_AF_const** | 1.227 | 4.090 | 4.090 | 7.930 | 14.227 | 14.190 | 15.190 |
| **T5MemWS2mem_hp_4_chunks_Adm_const** | 1.236 | 4.133 | 4.133 | 8.140 | 14.797 | 14.369 | 15.372 |
| **T5MemWSWMA1mem_hp_4_chunks_AF_const** | 1.296 | 4.052 | 4.052 | 7.876 | 13.905 | 13.911 | 14.904 |
| **T5MemWSWMA1mem_hp_4_chunks_Adm_const** | 1.112 | 4.331 | 4.331 | 6.965 | 12.657 | 12.610 | 13.587 |
| **T5MemWSWMA2mem_hp_4_chunks_AF_const** | 1.233 | 4.217 | 4.218 | 7.843 | 13.964 | 13.966 | 14.87 |
| **T5MemWSWMA2mem_hp_4_chunks_Adm_const** | 1.117 | 4.401 | 4.401 | 6.959 | 12.888 | 12.941 | 13.658 |

### Results discussion

This chapter presented a study of our proposed model on two new tasks: Masked language modeling as a pre-training task and a question-answering task using HotpotQA as a fine-tuning task. Experimental results showed good performance of the model on masked language modeling using a linear learning rate. The proposed model with chunked input outperformed T5 as a baseline. This approves previous results on translation tasks where the proposed model overcomes the baseline. The results of the proposed model were on par with the baseline on a fine-tuning task. Both models were not able to manage to solve the finetuning task. There were no preferred optimizer results between two different optimizers: Adam and Adafactor. The proposed model does not show good results for segmented inputs using a segment selector for fine-tuning on HotpotQA using a fixed learning rate. Across the way, using memory slots as a compressed representation of chunks led to better results on fine-tuning tasks than the proposed model but did not beat the baseline T5. This result is important because all previous sequence-to-sequence models still use the long input representation in the decoder part, and it can be studied in future work in encoder-decoder transformers such as T5.

**Chapter 5** dedicated to the model modifications using just the memory added to the encoder with the new masking and relative positional encoding while investigating the effectiveness the these modifications on summarization data sets.

**Summarize long documents** In this section using the memory tokens was in a different way than was used in the previous chapters. It depends on consuming the documents long document by long document simulating the human reading way. Second, inside the model each long document is chunked into chunks dynamically based on fixed chunk length, adding slot

memory that consists of one or more memory tokens, the size of the memory slot is a hyper-parameter and it is a proportion of the chunk length. Then we use masking so each memory slot depends only on itself and related chunk, and each chunk relates to all memory slots of the chunked document and itself. Finally for positional encoding; we adapt the relative positional encoding that was used in the t5 model between the memory slots and the chunks. Summarization task in general is a very challenging task for many reasons especially when the document gets longer; some reasons are related to the models others are related to data sets. Most of the long documents are domain-specific articles such as scientific papers that contain more complex formulas and terminologies [10, 16]. With relation to data sets many studies cover that the nature of summarization data sets plays a very important role and has clear consequences on the summarization results based on summarization method used even extractive, abstractive or hybrid [4, 11, 10, 14, 18].

**Task definition** *Text summarization* is the process of building a fluent, cohesive, and concise summary of a lengthier text document and distilling the content by pulling out the meaningful facts of the document that we need to summarize.

In our era, we witness an information explosion. Professionals in many fields (scientific, medical, economic,..etc.) may consume plenty of text at work or for many purposes, usually in scientific papers form, reports, news ..etc. Usually, they need just a specific part from all that is read. Consequently, not all texts are at the same level of importance, and they need to capture the gist of content related to the main idea in the text they read. This is where NLP can step in, using automatic summarization, making life easier. Reading a summary of a document from previously mentioned areas is much easier than reading the whole document.

Usually, there is a setup called Oracle setup [11, 9] where the gold facts or the crux information is located or specially designed at the beginning of the document or long input. This setup makes using the longer document not meaningful because less meaningful information can be found or not after processing longer input, which consumes more cost. This setup was not used in our experiments.

**Datasets** In this section, we provide a short overview of the data sets used for experiments on the proposed model in this chapter for the summarization task.

- **CNN/Daily Mail** [7]: version 3.0.0, English-language data set. This data set can be used for both abstractive and extractive summarization. It has two fields: article as long text and highlights as two-sentence summary. These articles were written by journalists at CNN between April 2007 and April 2015, and at Daily Mail between June 2010 and

April 2015.

- **SAMSum** [3]: This data set was written by linguists and used for abstractive summarization. It contains about 16k messenger-like conversations with summaries.
- **GovReport** was proposed by [8] for summarizing long documents, this dataset is challenging for many reasons. First, the summary itself is long and scattered over very long document.

| **Data set** | **Number of Instances in Split** | | | **Length of tokenized input** | | | **Length of tokenized target** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **Train** | **Validation** | **Test** | **Mean** | **Median** | **Max** | **Mean** | **Median** | **Max** |
| **SAMSum** | 14732 | 818 | 819 | 148 | 119 | 1153 | 28 | 25 | 94 |
| **GovReport** | 17517 | 973 | 973 | 10305 | 8571 | 324004 | 637 | 658 | 2360 |
| **cnn_dailymail** | 287113 | 13368 | 11490 | 985 | 898 | 5269 | 75 | 70 | 3151 |

Table 11: Statistics for the used summarization data sets. Input length is measured in tokens using a pre-trained t5 tokenizer.

### Results on SAMSum Dataset

As can be seen in table 13 T5mem succeeded in giving better results than baseline considering all metrics except R1 metric and overcome SLED that used block size 256 considering all metrics[4].

| Model | **Validation** | | | | **Test** | | | | **Loss** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | RL | RLsum | R1 | R2 | RL | RLsum | train | validation | test |
| T5-base(base line) | **52.543** | 28.181 | 43.734 | 48.433 | 50.895 | 25.852 | 42.068 | 46.316 | **1.199** | 1.328 | 1.234 |
| SLED_256 | 43.023 | 20.202 | 35.232 | 39.198 | 42.084 | 18.893 | 34.345 | 38.15 | 2.485 | 1.74 | 1.622 |
| T5mem-base_256_32 | 52.092 | 27.855 | 43.46 | 47.994 | 50.394 | 25.904 | 42.032 | 45.97 | 1.201 | 1.332 | 1.241 |
| T5mem-base_256_16 | 51.84 | 27.654 | 43.087 | 47.76 | 51.086 | **26.672** | 42.643 | 46.76 | 1.224 | **1.328** | **1.233** |
| T5mem-base_256_8 | 52.084 | 27.815 | 43.664 | 48.047 | **51.305** | 26.346 | **42.685** | **46.916** | 1.22 | 1.347 | 1.24 |
| T5mem-base_384_8 | 52.503 | **28.23** | **43.853** | **48.507** | 50.996 | 26.002 | 42.251 | 46.456 | 1.203 | 1.331 | 1.236 |

Table 12: Comparing Rouge metrics for the proposed model with T5 as baseline and SLED as another model for processing long documents on SAMSum data set. 256 and 364 represent the block size 32,16,8 represent the number of memory tokens in each slot.

Increasing memory capacity was supposed to give us better results, practically using less memory capacity (8 memory tokens for every 256 tokens) was more efficient computationally and gets better ROUGE scores on the evaluation data set using 384 block size and better scores on test data set using 256 block size [5]. After analyzing the memory content, it was explained

[4]Experiments on SAMSum data set were done using 8 Tesla P100 SXM2 GPU

[5]Results of training models and baseline can be found on wandb cite

that the memory tokens in each memory slot have just one token in all positions for each chunk. According to that, the reason that increasing the memory size was not beneficial was understandable.

**Results on CNN / DailyMail Dataset** The experiments on CNN/Daily Mail were for ten epochs. While the results in the table 13 are for epoch 6 since the models start to overfit after epoch four [6].

| Model | Validation | | | | Test | | | | Loss | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | RL | RLsum | R1 | R2 | RL | RLsum | train | validation | test |
| T5-base (baseline) | 44.074 | 21.421 | 31.154 | 41.024 | 43.21 | 20.647 | 30.566 | 40.102 | 1.262 | 1.409 | 1.443 |
| SLED_256 | 42.196 | 19.99 | 29.865 | 39.14 | - | - | - | - | 1.609 | - | - |
| T5mem-base_8mem_384b | 43.67 | 21.023 | 30.759 | 40.603 | 42.905 | 20.483 | 30.381 | 39.871 | 1.255 | 1.394 | 1.422 |
| T5mem-base_32mem_256b | 43.547 | 26.971 | 30.76 | 40.532 | 42.777 | 20.332 | 30.226 | 39.723 | 1.295 | 1.442 | 1.455 |

Table 13: Analysing Rouge metrics for the proposed model with T5 as baseline and SLED as another model for processing long documents on CNN/Daily Mail data set.

As we can see from the table, our model overcomes SLED as a model for processing long documents, but both our model and SLED did not overcome the baseline t5 under the same conditions. Many notes can be made here since both SLED and our model depend on the fusion of separate encoder representations in the decoder. We infer that using memory slots is important for communication between the independent blocks. SLED model configuration did not include the prefix for the summarization task.

**Results on GovReport**

Results are displayed in the table 14. Since Powerful server was used [7] and for fair comparison, the same long input length was used for all experiments on GovReprt data set[8]. The tokenized input length is 3072, and the tokenized target length is 384. As we see, t5 still overcomes both models SLED and our model. Hence, our model overcomes SLED regarding all metrics. This emphasizes that using the whole length is better than chunking it, whatever it is long. GovReport is a very challenging data set because the input is very long, and the output is long.

[6] More information about results can be found onwandb cite

[7] NVIDIA DGX A100 320GB

[8] Full experiments training and hyperparameters details can be found and displayed on wandb cite

| Model | Validation | | | | Test | | | | Loss | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | RL | RLsum | R1 | R2 | RL | RLsum | train | validation | test |
| T5-base (base line) | 52.796 | 22.655 | 27.96 | 48.866 | 53.35 | 23.328 | 28.614 | 49.578 | 1.773 | 1.751 | 1.808 |
| SLED1024_256 | 38.59 | 14.579 | 25.357 | 33.801 | 38.82 | 15.058 | 25.785 | 34.254 | 2.216 | 1.963 | 1.959 |
| SLED3072_256 | 36.02 | 11.545 | 21.463 | 31.852 | 36.305 | 11.809 | 21.704 | 32.175 | 2.381 | 2.044 | 2.104 |
| T5mem-base | 49.727 | 20.241 | 26.442 | 45.668 | 50.058 | 20.833 | 26.838 | 46.115 | 1.825 | 1.785 | 1.843 |

Table 14: Rouge metrics for the proposed model with T5 as baseline and SLED as another model for processing long documents on the GovReport data set.

Consistent with results on previous data sets, using less memory slot size with longer block length gave better results. This is understandable since the memory slot uses just one fixed token representation in all memory tokens updated during training. Using more memory tokens will not help. Otherwise, these tokens save different compressed representations for each chunk.

**As final thoughts** of this chapter, our main results show that chunking the long input into non-overlapped chunks during encoding allows these chunks to communicate only through the related memory slots. Chunking the long input into equal chunks dynamically allows the computation time of the model to grow linearly with the number of chunks instead of quadratic. This scales well with the input length and gives competitive results on the summarization task. Analyzing memory content reveals that memory tokens with the same positional encoding store identical tokens for each related chunk. Analyzing attention maps showed active interaction between memory tokens and related chunks and active interaction between tokens chunks and most of the memory tokens in all layers. The interaction was from the first layers, where the memory was empty.

**Chapter 6: Conclusion** of the overall thesis, limitations, and future work.
Currently processing long text inputs is a fertile field for research. The tasks within this dissertation were carried out as a part of efforts in this direction to discover new ways and add valuable contributions, focusing on transformer design modification to address transformer input bottlenecks. Design modifications were carried out on the T5 transformer and applied in different stages of development on three main tasks, translation, QA, and summarization tasks. In conclusion, the following theoretical and practical outcomes of overall model modification development stages are:

1. A review of proposed models for processing long documents for different NLP tasks;

2. New usage of the global tokens as memory slots to relate chunked inputs and as compressed representation;

3. Developing a proper masking mechanism and proper usage of the relative positional en-

coding for binding memory slots with related segments(chunks);

The proposed transformer structural modifications for increasing the transformer input length. The outcomes of experiments are of high importance to academic researchers for processing longer contexts. The findings provide different insights for processing longer inputs using transformers on different NLP tasks that require long-range information usage.

**Finally**, there are many recommendations and directions to continue research in this direction. It will be of consequential scientific merit to conduct a fair comparison study between all models directed to processing long-range documents using LLMs and the same data. The field needs this kind of study to decouple the effects of algorithms/models from other factors such as training details, ample computational cost, and big data. In this case, it is worth investigating applying memory in a much more creative way as we did before for the translation task, using separate attention for each memory slot and its related chunk so we can have an initial memory representation for the chunk and then apply attention on the related chunk, and give memory tokens in each slot different positional encoding. This needs experiments with various design variants and investigating the best layer or place to use the memory slots. Research in this direction is extensive. In the future, working on metrics dedicated to measuring the model's predictions over long documents is an appealable direction. Studying metrics was not the crux of the thesis at the current stage. Also, it is worth collecting long-range data for tasks that require long-range information comprehension. This led to better-taking stock of the current research landscape. .

## References


1. Arij Al Adel and Mikhail S. Burtsev. "Memory transformer with hierarchical attention for long document processing". In: *2021 International Conference Engineering and Telecommunication (En&T)* (2021), pp. 1–7.
2. Iz Beltagy, Matthew E. Peters, and Arman Cohan. "Longformer: The Long-Document Transformer". In: *ArXiv* abs/2004.05150 (2020).
3. Bogdan Gliwa et al. "SAMSum Corpus: A Human-annotated Dialogue Dataset for Abstractive Summarization". In: *Proceedings of the 2nd Workshop on New Frontiers in Summarization*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 70–79. DOI: 10.18653/v1/D19-5409. URL: https://www.aclweb.org/anthology/D19-5409.
4. Max Grusky, Mor Naaman, and Yoav Artzi. "Newsroom: A Dataset of 1.3 Million Summaries with Diverse Extractive Strategies". In: *North American Chapter of the Association for Computational Linguistics*. 2018.
5. Mandy Guo et al. "LongT5: Efficient Text-To-Text Transformer for Long Sequences". In: *NAACL-HLT*. 2022.
6. Ankit Gupta and Jonathan Berant. "GMAT: Global Memory Augmentation for Transformers". In: *ArXiv* abs/2006.03274 (2020).
7. Karl Moritz Hermann et al. "Teaching Machines to Read and Comprehend". In: *Advances in Neural Information Processing Systems (NIPS)*. 2015. URL: http://arxiv.org/abs/1506.03340.
8. Luyang Robby Huang et al. "Efficient Attentions for Long Document Summarization". In: *North American Chapter of the Association for Computational Linguistics*. 2021.
9. Maor Ivgi, Uri Shaham, and Jonathan Berant. "Efficient Long-Text Understanding with Short-Text Models". In: *ArXiv* abs/2208.00748 (2022).
10. Huan Yee Koh et al. "An Empirical Survey on Long Document Summarization: Datasets, Models, and Metrics". In: *ACM Computing Surveys* 55 (2022), pp. 1–35.
11. Wojciech Kryscinski et al. "Neural Text Summarization: A Critical Evaluation". In: *Conference on Empirical Methods in Natural Language Processing*. 2019.
12. Taku Kudo and John Richardson. "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing". In: *arXiv preprint arXiv:1808.06226* (2018).
13. Pierre Lison, Jörg Tiedemann, and Milen Kouylekov. "OpenSubtitles2018: Statistical Rescoring of Sentence Alignments in Large, Noisy Parallel Corpora". In: *International Conference on Language Resources and Evaluation*. 2018.

14. Yang Liu and Mirella Lapata. “Text Summarization with Pretrained Encoders”. In: *ArXiv* abs/1908.08345 (2019).
15. Dimitris Mamakas et al. “Processing Long Legal Documents with Pre-trained Transformers: Modding LegalBERT and Longformer”. In: *ArXiv* abs/2211.00974 (2022). URL: https://api.semanticscholar.org/CorpusID:253254835.
16. Potsawee Manakul and Mark John Francis Gales. “Long-Span Summarization via Local Attention and Content Selection”. In: *Annual Meeting of the Association for Computational Linguistics*. 2021.
17. Colin Raffel et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *ArXiv* abs/1910.10683 (2020).
18. A. See, Peter J. Liu, and Christopher D. Manning. “Get To The Point: Summarization with Pointer-Generator Networks”. In: *ArXiv* abs/1704.04368 (2017).
19. Ashish Vaswani et al. “Attention is All you Need”. In: *NIPS*. 2017.
20. Elena Voita, Rico Sennrich, and Ivan Titov. “When a Good Translation is Wrong in Context: Context-Aware Machine Translation Improves on Deixis, Ellipsis, and Lexical Cohesion”. In: *ACL*. 2019.
21. Sinong Wang et al. “Linformer: Self-Attention with Linear Complexity”. In: *ArXiv* abs/2006.04768 (2020).
22. Hongfei Xu et al. “Efficient Context-Aware Neural Machine Translation with Layer-Wise Weighting and Input-Aware Gating”. In: *IJCAI*. 2020.