Paper Implementation: *Fine-tune BERT for Extractive Summarization [1]*

An open source implementation for the course Machine Learning of the Master's Degree BDMA at Université Paris-Saclay, CentraleSupélec

Jose Antonio Lorencio Abril
BDMA Student
Université Paris-Saclay, CentraleSupélec
Paris, France
jose-antonio.lorencio-abril@student-cs.fr

Sayyor Yusupov
BDMA Student
Université Paris-Saclay, CentraleSupélec
Paris, France
sayyor.yusupov@student-cs.fr

Professor Tom Dupuis
CEA List
Université Paris-Saclay
Paris, France
tom.dupuis@centralesupelec.fr

***Abstract*—Summarization is a popular NLP task, since the ability to summarize a text is inherently human and important for text understanding. Here, we present a clear explanation and implementation of BERTSUM, which leverages the great language understanding of BERT to perform extractive summaries, i.e., recognizing the most relevant sentences of a given text. In addition, we present a way to process texts longer than the maximum length that BERT can process (512 tokens). For instance, this report is summarized by our tool as:**

'The second approach, $take\_N\_sents$, involved extracting the best N summary sentences from each chunk and joining them together. ROUGE-2: measures the overlap of bigrams (two-word sequences) between the generated text, and the reference text. The tokens are converted to an embedding space. There are two approaches for this task: extractive and abstractive summarization. Summarization is a popular NLP task, since the ability to summarize a text is inherently human and important for text understanding.'

***Index Terms*—NLP, Machine Learning, Summarization, BERT, Transformer**

I. Introduction

In the evolving landscape of Natural Language Processing (NLP), the ability to succinctly summarize textual information has become increasingly paramount. With the exponential growth of digital content, the necessity for automated summarization systems that can distill essential information efficiently and accurately is more pronounced than ever. This project explores the domain of text summarization through the lens of one of the most revolutionary models in NLP: BERT (Bidirectional Encoder Representations from Transformers), specifically its adaptation for extractive summarization, known as BERTSUM.

Our study is based in the understanding that summarization is not just a mere truncation of text but an intricate process of identifying and extracting relevant information. The primary objective of this project is to implement and analyze the BERTSUM model, delving into its unique ability to leverage the contextual prowess of BERT for the purpose of extractive summarization. We aim to demonstrate how BERTSUM extends the capabilities of the original BERT model, making it appropriate for identifying key sentences in a text, thereby generating coherent and concise summaries.

To achieve this, we first provide a comprehensive overview of the underlying principles of BERT and its architectural nuances, setting the foundation for understanding its extension, BERTSUM. We then delve into the specifics of the BERTSUM model, including its architectural modifications and the rationale behind these changes. Our exploration is grounded in practical application, utilizing diverse datasets to train and evaluate the model, and employing standard metrics like ROUGE scores for assessment. Through this project, we seek not only to demonstrate the technical efficacy of BERTSUM but also to contribute to it by providing a more accessible implementation[1][2].

II. Text Summarization

Text summarization is the task of summarizing the important information of an input text, into a shorter, more concise text. There are two approaches for this task: extractive and abstractive summarization.

A. Extractive Summarization

Extractive summarization consists of selecting the most important sentences of an input text, i.e., those sentences that provide more information about the overall message of the text.

For example, for the input sequence *Jose: Hi, there. All good? Sayyor: Indeed.*, we could summarize it extractively with just one sentence as *All good?*, capturing that the text is a greeting.

This task can be understood as a classification task, where each sentence in the text is assigned a logit representing the likelihood of the sentence being included in the summary. More precisely, given a text composed of sentences, $T = (s_1, ..., s_m) \in \mathcal{T}$, where $\mathcal{T}$ is the space of input texts and m

[1]The original implementation is very low level and hard to follow at some points.

[2]Our implementation is accessible from https://github.com/Lorenc1o/NLP_Paper_Summarizer

is the amount of sentences in the text (variable for each text), we train a model, M, that assigns a logit to each sentence, $M(s_1, ..., s_m) = (o_1, ..., o_m)$, with $o_i \in [0, 1]$.

It is in this task that we focus in this project.

B. Abstractive Summarization

Abstractive summarization consists on creating a new text from an input text, in such a way that information is mostly preserved and length is reduced.

Following the previous example, an abstractive summary could be *Jose is greeting Sayyor*.

This is a generative task, which lies out of the scope of this project. However, the model we present can be extended to also perform this complex task [2].

III. BERT

Bidirectional Encoder Representations from Transformers (BERT) [3] is an NLP model developed by Google and which set the ground for many encoder-only transformer architectures. The main capability of the model is understanding the context of a word in a sentence using information from both directions. Its architecture is shown in Figure 1, where we observe how:

- Input text needs to be tokenized. Usually, adding the special tokens [CLS] at the beginning and [SEP] to separate two sentences for the task of Next Sentence Prediction (NSP).
- The tokens are converted to an embedding space. In the beginning, this is just the index of each token in the dictionary.
- Segment embeddings are added. If a [SEP] token is detected, the segment finishes.
- Positional embeddings are added, to take positional information into account.
- A Transformer Encoder model processes this input.
- The output consists of 768-dimensional vectors, one for each input token. These vectors represents the contextual meaning of each input token, taking the rest into account.

BERT was pre-trained for mainly two tasks:

- Masked Language Modeling (MLM): a percentage of the input sequence is masked out, and the model tries to predict the original value of these tokens. This pre-training task lets BERT learn the bidirectional contextual information.
- Next Sequence Prediction (NSP): BERT is presented with two sentences and has to understand whether the second sentence is a logical follow-up of the first one. This pre-training task lets BERT better understand the relationships between different sentences.

It is thanks to these two tasks, and especially leveraging NSP, that BERT can be fine-tuned for extractive summarization.

IV. BERTSUM

BERTSUM is the main attraction of this project, which was presented in [1], as an extension of BERT, designed to enable extractive summarization.

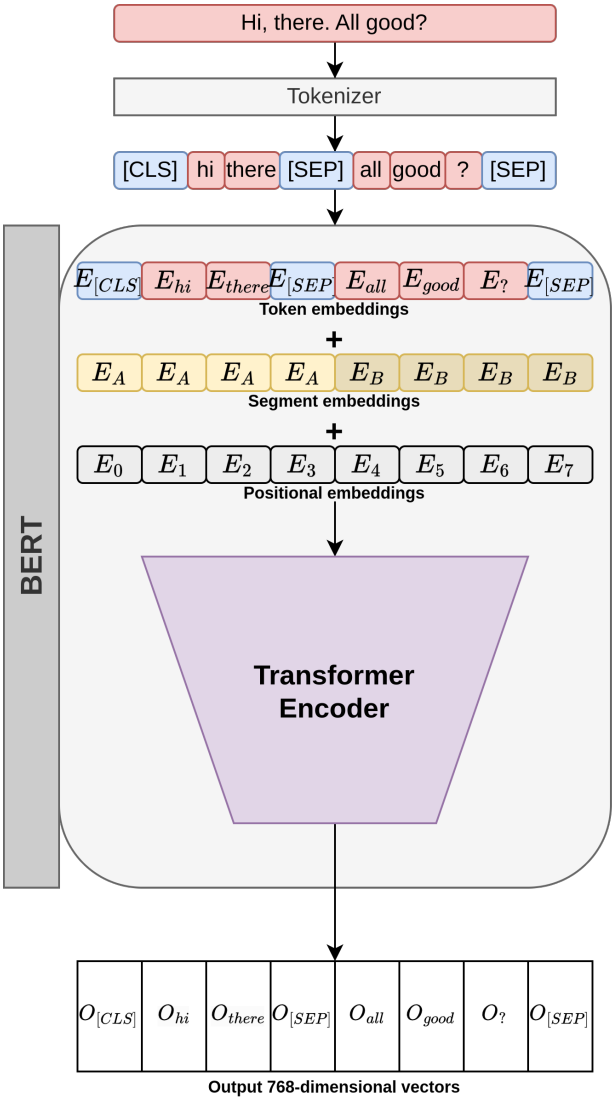


Fig. 1: BERT Architecture

A. BERTSUM Architecture

The new architecture can be observed in Figure 2. The main additions are the following:

- All sentences (and not just two) are separated with a **[SEP] token**. The idea is to enforce the model to add different segment embeddings to each sentence (E_A for odd sentences[3] and E_B for even sentences). This reinforces the contextual information by differentiating adjacent sentences.
- A **[CLS] token** is added at the beginning of each sentence. This is done to add an artificial token to each sentence, giving freedom for the model to learn how to leverage these at the time of fine-tuning. Each [CLS] token serves as a representative token for its associated sentences, and these are, in fact, the only tokens taken into account at the time of classification.

[3]Counting from 1!

- A **binary classifier module** is added after BERT processing of the input, to assign the likelihood of belonging to the summary to each sentence, represented by its associated [CLS] token.

At the end, the N [CLS] tokens with higher logits would be selected to define the extractive summary.

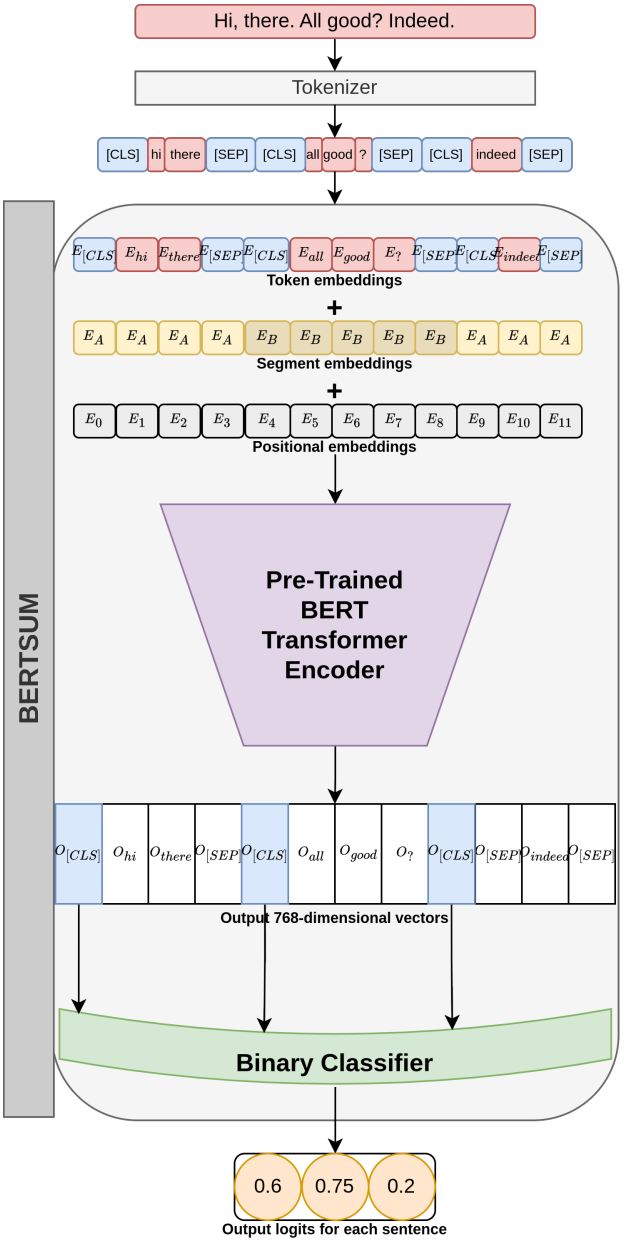
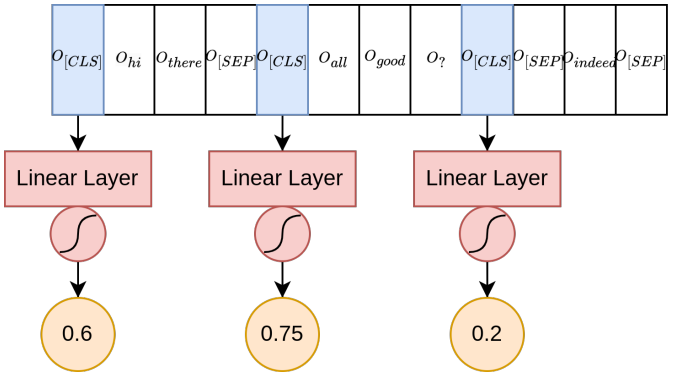


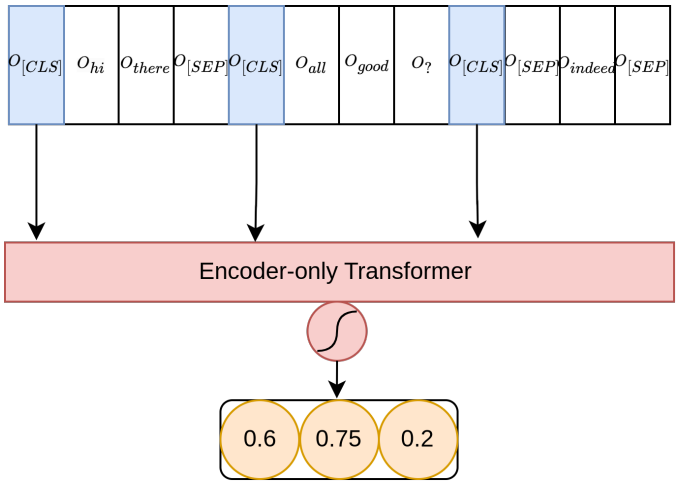
Fig. 2: BERTSUM Architecture

B. The Binary Classifier

The added classifier can be any binary classifier. We test a **simple linear classifier** and, more interestingly, an **encoder-only transformer**, which can leverage the contextual information to arrive to better conclusions than the bare linear classifier. Both approaches are shown in Figure 3, where we can observe how the linear classifier processes each sentence token



(a) Linear classifier



(b) Encoder-only Transformer classifier

Fig. 3: The different binary classifiers

independently, while the transformer classifier can process all of them at once, leveraging the power of transformers to use self-attention matrices to take all the sentence tokens into account for the classification task. In both cases, we normalize with a sigmoid function to create the output logits.

V. Datasets and Evaluation Metrics

A. Datasets

We use two datasets for this project:

- *Dialogsum* [4]: a set of dialogues between two persons, with an associated (abstractive) summary. The dialogues are short, mostly under the maximum token length for BERT (512 tokens).
- *arxiv-summarization* [5]: a set of research papers from arxiv, with the paper full text and its abstract. The papers are long, exceeding several times the maximum token length for BERT. The original dataset contains around 215 thousand articles, but we are training on only 12 thousand of them, due to computational limitations.

B. Evaluation Metrics

The **ROUGE score** (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics used to evaluate automatic summarization of texts, in comparison to reference summaries. These metrics focus on the recall of the generated text. From this family of metrics, we utilize:

- ROUGE-1: measures the overlap of unigrams (single words) between the generated text, and the reference text.
- ROUGE-2: measures the overlap of bigrams (two-word sequences) between the generated text, and the reference text.
- ROUGE-Lsum: ROUGE-L finds the longest common subsequence between the generated and reference texts. ROUGE-Lsum is a variant of this, which aims at the sentence level. It's like applying ROUGE-L to each sentence individually, instead of to the whole text. The scores for all sentences are averaged at the end of the process.

These three metrics were selected because they are widely used to assess summarization tasks.

C. Generating Extractive Summaries from Abstractive Summaries

Since we want to tackle the problem as a classification task, we need to prepare our datasets for this purpose. With this in mind, and following the approach of the original BERTSUM implementation, we use the given abstractive summaries to generate extractive summaries with a greedy approach that takes the n sentences in the text that maximize the ROUGE score when compared to the original abstractive summary. The pseudocode can be read in Listing 1.

```
function generate_oracle_summary(text: str,
    summ: str, n:int):
    sent = split in sentences(text)
    oracle = []
    for i in 1:n:
        scores = [rouge(s, summ) for s in sent
    ]
        selected = select s with higher score
        oracle.append(selected)
    end for

    return oracle
end function
```

Listing 1: Pseudocode for generating extractive summaries.

In the case of very long texts (e.g. $arxiv-summarization$ dataset), we have to consider the limitation of 512 tokens for BERT. There are two possible approaches: increase n or chunk into smaller pieces and compute the oracle summary for each chunk. We opt for the latter option. We explain this approach in more detail as follows.

D. Dealing with Long Texts

During the training phase, we chunk the long text into smaller pieces and regard each chunk as an individual data instance. The pseudocode can be found in Listing 2. In our executions, min_n_tokens were set at 200, to remove too small chunks that are usually the last ones. Thus, the resulting chunks were in the range of 200 and 512. We obtained an average of 16.7 chunks per article. We then generated extractive summaries for each chunk as mentioned above. We selected 3 summary sentences for each chunk.

```
function split_into_chunks(text: str,
    max_n_tokens: int, min_n_tokens: int):
    sents = split in sentences(text)
    initialize a new chunk
    sum_tokens = 0
    for sent in sents:
        sent_tokens = count number of tokens
    in sent + 2
        sum_tokens += sent_tokens
        chunk.append(sent)
        if sum_tokens >= max_n_tokens:
            initialize a new chunk
        elif sum_tokens <= min_n_tokens:
            skip this chunk # too short for
    summary
    end for
end function
```

Listing 2: Pseudocode for generating extractive summaries.

To predict the full article's summary from the predicted summaries of each chunk, we implemented 2 approaches. The first approach, sum_of_sums, involved running BERTSUM again on the summaries of all chunks, that are joined together. The second approach, $take\_N\_sents$, involved extracting the best N summary sentences from each chunk and joining them together. The two approaches are compared in Section VII.

VI. Training

Training is performed with the following configuration:

- Optimizer: Adam with weight decay fix, with starting learning rate $lr = 10^{-5}$.
- Epochs: we train for 100 epochs. However, we noticed how most improvement is obtained in the earlier stages of training, so we also train with just 10 epochs, to compare the results. In contrast, the authors of BERTSUM trained for 50000 epochs. Maybe further gains are obtained at latter stages in the process, but we don't have access to such computing power.
- Batch size: 32.
- Loss function: binary cross-entropy. Each [CLS] token, representing its corresponding sentence, is given a logit value, $z \in [0, 1]$, with 1 meaning the sentence should be included in the summary, and 0 meaning it should not.
- Weights initialization:
  - BERT weights are the pre-trained weights.
  - Classifier weights are initialized randomly.

At the time of training, the classifier is trained from scratch, and BERT is fine-tuned for the given task. The expected behavior is that BERT uses the inserted [CLS] tokens as a *tabula rasa*, i.e., a place where it can put the newly learned information through the training process. This means that, at the beginning, the [CLS] tokens will not have a meaning.

However, when training goes on, BERT will understand that putting the sentence information in these tokens enables the classifier to better classify these tokens as belonging to the summary or not.

We train six different models, belonging to two variants:

- BERTSUM-Dialog: trained on 12460 training instances, with 500 validation instances and 1500 test instances. Both using the linear and transformer classifier are trained for 10 and 100 epochs.
- BERTSUM-Arxiv: trained on around 167000 training instances, with 16700 validation instances and 16700 test instances. These instances, article chunks, were obtained from 10000 training, 1000 validation, and 1000 test articles. Both using the linear and transformer classifiers are trained for 10 epochs.

The evolution of the loss for each training configuration is shown in Figure 4.

We can make several observations:

- Training with DialogSum data seems to provide most gains at the beginning, with steady improvement in the training data but a stagnation for the validation data.
- Training with Arxiv data, however, seems to show steady improvement for both training and validation datasets when using the linear model. This is not as clear as the transformer model. However, computational restrictions did not allow us to perform longer trainings.
- The BCE-Loss with the transformer model is much lower than with the linear model, around 10 times lower.

VII. Experiments and Results

Exp. A. With each of the models, we process its respective test set with it. In addition, we process the test set on which it was not trained on. Notice that in this experiment, the arxiv-dataset is processed as shorter, chunked texts, as obtained from the preprocessing of data. The objective is to assess the generalization capabilities of each model, both in data similar to the one the model was trained on, and in different data. Therefore, we evaluate in two different levels of abstraction the generalization capabilities of the models.

Exp. B. In addition, with each of the models, we process the arxiv papers completely, without preprocessing. Each of the models runs with the two strategies for longer texts, to assess which strategy works better.

For each (input,output) pair, we compute the ROUGE scores as explained in Subsection V-B, and in the end we average these on the full test set. The results are shown in Table I for Experiment A, or visually for each training dataset in Figure 5, and Table II for Experiment B or visually for each predictive strategy in Figure **??**.

We also predicted summaries for our datasets using another BERT-based extractive summarizer [6] for comparison. The model creates word embeddings using BERT and uses K-means to find central sentences. It is accessible online as a python library [7]. The model could be directly on long texts so none of the predictive strategies were used. Additionally, we implemented our sum_of_sums strategy to see if it performs better with less text inputted. We used the same values as the original implementation for the parameters max_n_tokens, min_n_tokens and number of sentences for chunk summaries.

From experiment Exp. A. we can draw the following observations:

- Models trained on the DialogueSum dataset generalize better to unseen data.
- The transformer model performs better in the Arxiv dataset.
- The linear model performs better in the DialogueSum dataset.
- The transformer model, when trained for 100 epochs on the DialogueSum dataset, obtains very poor results in this test set.
- An interesting observation: the linear model trained on DialogueSum for 100 epochs obtains better results in the Arxiv data than the linear model trained on the Arxiv data.
- Regarding the baseline model, we observe how it performs better than our models in the chunked Arxiv dataset, which is expected since this model was trained for summarizing lectures, which are somewhat more similar to scientific articles than to dialogues. Note, however, that some of our models perform better in DialogueSum data.

Regarding experiment Exp. B., we can see that:

- There is a 20% or more decrease in performance from chunks dataset, expected as processing longer texts is a harder task.
- Usually *take_N_sents* is slightly better.
- Arxiv-trained models perform best, but not much difference even though the data is also from the Arxiv dataset. This reinforces our belief that training with DialogSum data leads to better generalization capabilities.
- Transformer models not trained on the data perform slightly worse than the rest. This can be due to the Transformer model needing more epochs to train. Further tests should be performed to draw conclusions on this matter.
- Several of our models perform almost as well as the baseline model in all metrics, which shows that we could obtain decent results with a small number of data. Understandably, the sum_of_sums approach performed slighter worse than the one predicted on the full text as information on the relationship between chunks is missing, however it is not a high difference.

VIII. Conclusion

We have been able to provide a thorough explanation of how BERT can be leveraged to perform extractive summarization, as well as an implementation from scratch, which can be easily understood and used. We have shown how our implementation can obtain decent results with limited computing resources.

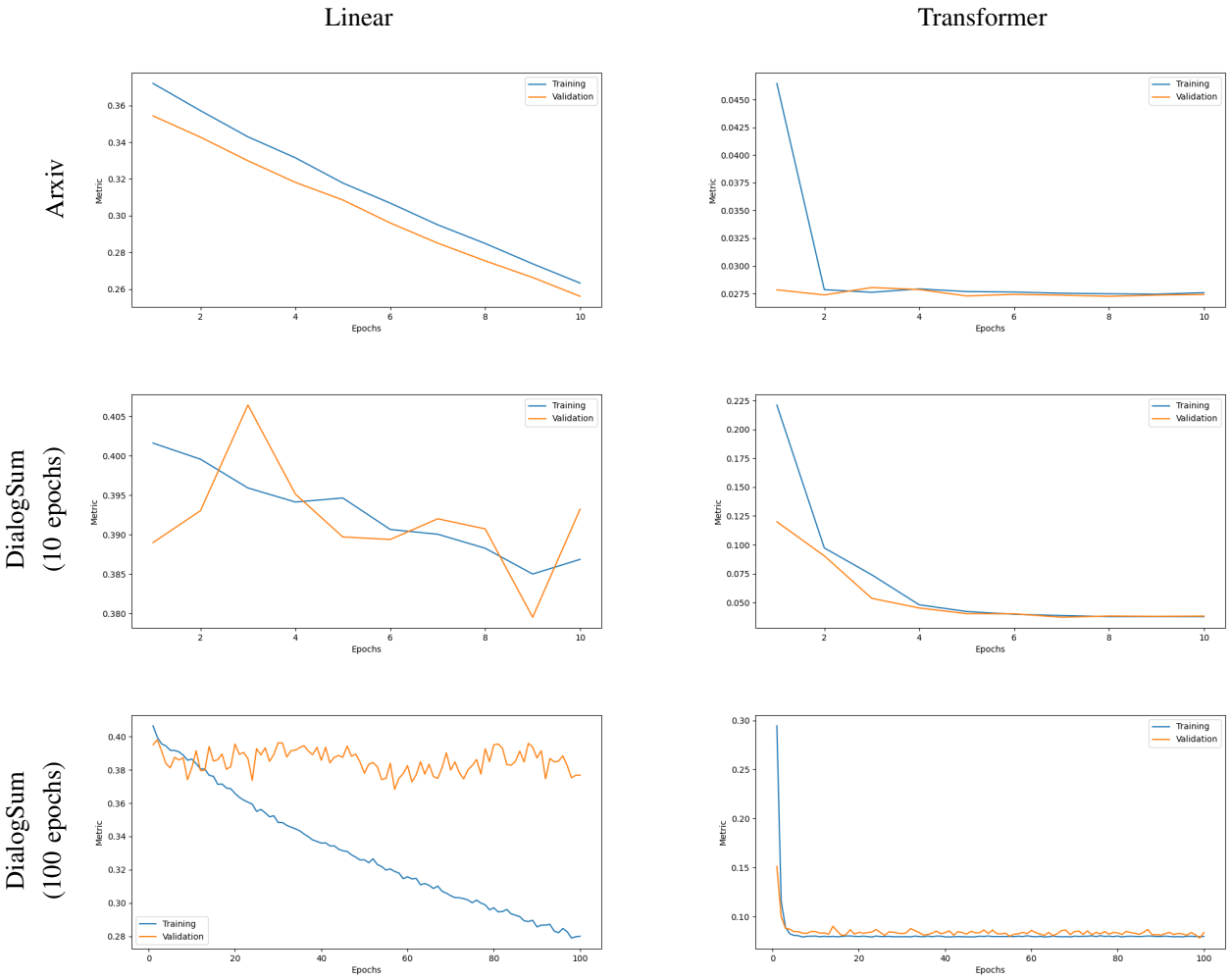
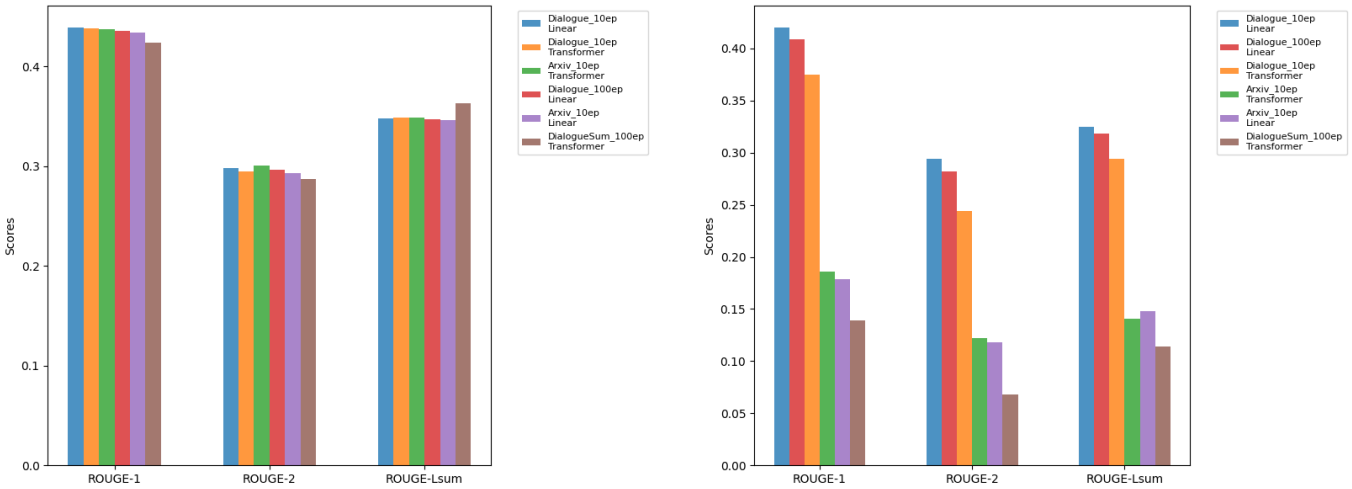


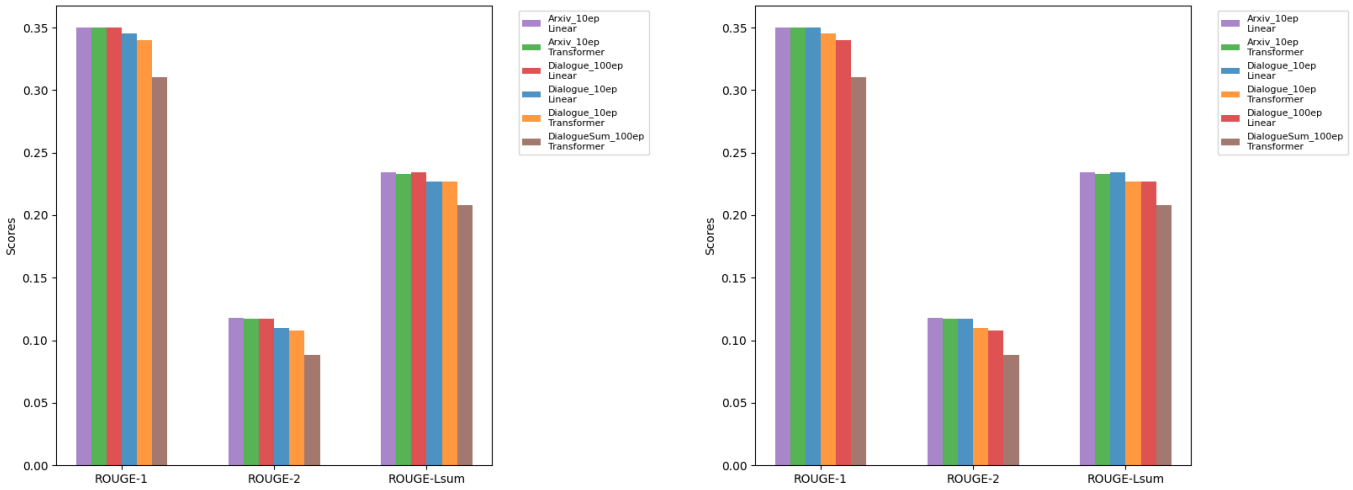
Fig. 4: BCE-Loss evolution during training and validation



(a) Performance of the Models on Arxiv Chunks

(b) Performance of the Models on DialogueSum Testing Dataset

Fig. 5: Experiment Exp. A. results



(a) Performance on full arxiv texts, with sum_of_sums strategy

(b) Performance on full arxiv texts, with take_N_sents strategy

Fig. 6: Experiment Exp. B. results

Trained Dataset # epochs	Model Type	Testing Dataset	ROUGE-1	ROUGE-2	ROUGE-Lsum
Arxiv 10 epochs	Linear	Chunks	0.434	0.293	0.346
		DialogeSum	0.179	0.118	0.148
	Transformer	Chunks	0.437	0.301	0.349
		DialogueSum	0.186	0.122	0.141
DialogueSum 10 epochs	Linear	DialogueSum	0.420	0.294	0.325
		Chunks	0.439	0.298	0.348
	Transformer	DialogueSum	0.375	0.244	0.294
		Chunks	0.438	0.295	0.349
DialogueSum 100 epochs	Linear	DialogueSum	0.409	0.282	0.318
		Chunks	0.436	0.296	0.347
	Transformer	DialogueSum	0.139	0.068	0.114
		Chunks	0.424	0.287	0.363
-	BERT-extractive-summarizer [6]	DialogueSum	0.411	0.290	0.374
		Chunks	0.491	0.351	0.429

TABLE I: Results of our 6 models and a baseline model on the 2 testing datasets.

REFERENCES

1. Liu, Y. Fine-tune BERT for Extractive Summarization. Vol. 2019.
2. Liu, Y. & Lapata, M. Text Summarization with Pretrained Encoders. Vol. 2019.
3. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Vol. 2019.
4. Chen, Y., Liu, Y., Chen, L. & Zhang, Y. *DialogSum: A Real-Life Scenario Dialogue Summarization Dataset* In Proc. of the. *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021* 5062–5074. 2021.
5. Cohan, A. *et al. A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents* In Proc. of the. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)* 615–621. 2018.
6. Miller, D. Leveraging BERT for Extractive Text Summarization on Lectures. Vol. 2019.
7. *bert-extractive-summarizer* https://pypi.org/project/bert-extractive-summarizer (2022).

Trained Dataset # epochs	Model Type	Predicting Strategy	ROUGE-1	ROUGE-2	ROUGE-Lsum
Arxiv 10 epochs	Linear	sum_of_sums	0.350	0.118	0.234
		take_N_sents	0.375	0.132	0.249
	Transformer	sum_of_sums	0.350	0.117	0.233
		take_N_sents	0.352	0.117	0.239
DialogueSum 10 epochs	Linear	sum_of_sums	0.345	0.110	0.227
		take_N_sents	0.318	0.092	0.211
	Transformer	sum_of_sums	0.340	0.108	0.227
		take_N_sents	0.347	0.116	0.237
DialogueSum 100 epochs	Linear	sum_of_sums	0.350	0.117	0.234
		take_N_sents	0.331	0.099	0.221
	Transformer	sum_of_sums	0.310	0.088	0.208
		take_N_sents	0.338	0.106	0.230
	BERT-extractive-summarizer [6]	full text	0.378	0.129	0.254
		sum_of_sums	0.362	0.121	0.245

TABLE II: Performance of Prediction Strategies on Testing Dataset of the Whole Articles