### VIETNAM GENERAL CONFEDERATION OF LABOR **TON DUC THANG UNIVERSITY FACULTY OF INFORMATION TECHNOLOGY**



**DEEP LEARNING**

**FINAL REPORT**

*FINE-TUNING BART FOR TEXT SUMMARIZATION*

### *Supervisor*: **LE ANH CUONG** *Students*: LIEU DANG KHOA **- 520K0140** VI NGUYEN THAH DAT**- 520C0001** *Class*: **20K50301**

### *Group*: **01** *Year*: **24**

## 

## HO CHI MINH CITY, 2023

## 

# THANK YOU

We would like to take this opportunity to express my gratitude towards Prof. Le Anh Cuong, who had helped us with explaining all the details and requirements carefully as well as provided a large number of references and study materials to use.

Futhermore, we also would like to thank our friends for sharing advices, guides and tips to help us finish the project report quickly. Without our professor and our friends, we would have not been able to finish the report on time

# PROJECT COMPLETED AT TON DUC THANG UNIVERSITY

We hereby declare that this is our own project and is under the guidance of Le Anh Cuong. The research contents and results in this topic are honest and have not been published in any publication before. The data in the tables for analysis, comments and evaluation are collected by the author himself from different sources, clearly stated in the reference section.

In addition, the project also uses a number of comments, assessments as well as data of other authors, other agencies and organizations, with citations and source annotations.

If we find any fraud, we will take full responsibility for the content of my project. Ton Duc Thang University is not related to copyright and copyright violations caused by me (if any).

*Ho Chi Minh, ...........................  
 Author  
 (Sign)*

*Lieu Dang Khoa*

*Vi Nguyen Thanh Dat*

# VERIFICATION AND EVALUATION OF LECTURER

**Verification of guiding lecturer**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Ho Chi Minh, ...........................*

(Sign)

**Evaluation of grading lecturer**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Ho Chi Minh, ...........................*

(Sign)

# 

# SUMMARY

BART (Bidirectional and Auto-Regressive Transformer) is a transformer-based generative language model developed by Facebook AI Research (FAIR), which can be applied to a wide range of natural language processing (NLP) tasks including text generation, summarization, and conversational agents.

BART is based on the transformer architecture and is trained using a combination of masked language modeling and denoising autoencoder objectives. The architecture is designed to facilitate the generation of text that is not only grammatically correct but also semantically coherent and contextually relevant.

One of the key features of BART is its ability to handle both auto-regressive (i.e. predicting one word at a time) and bidirectional (i.e. processing the entire input sequence at once) language modeling, making it suitable for a variety of NLP tasks. During training, BART is trained on both source-to-target and target-to-source directions of the input sequence.

# TABLE OF CONTENT

[THANK YOU 1](#_Toc134382836)

[PROJECT COMPLETED AT TON DUC THANG UNIVERSITY 2](#_Toc134382837)

[VERIFICATION AND EVALUATION OF LECTURER 3](#_Toc134382838)

[SUMMARY 4](#_Toc134382839)

[TABLE OF CONTENT 5](#_Toc134382840)

[1. INTRODUCTION 6](#_Toc134382841)

[1.1. Text summarization](#_Toc134382842)

[1.2. HuggingFace transformer](#_Toc134382842)

[1.3 Dataset](#_Toc134382842)

[2. CODE 9](#_Toc134382846)

[2.1. Installation and connect to hugging face 9](#_Toc134382847)

[2.2. Import library 9](#_Toc134382848)

[2.3. Load dataset 1](#_Toc134382849)0

[2.4. Get the pre-trained BART model 1](#_Toc134382850)1

[2.5. Training model 1](#_Toc134382851)2

[2.6. Test 1](#_Toc134382856)5

# INTRODUCTION

In this report, we provide an overview of text summarization followed by a description of the fine-tuning process used to train the model on text summarization datasets. We then present the results of experiments conducted to evaluate the efficacy of the fine-tuned BART model on various summarization benchmarks.

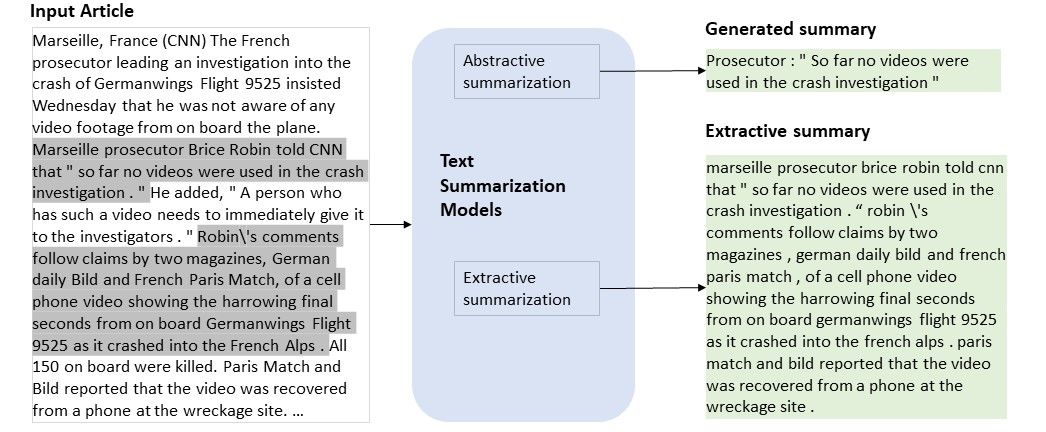
First, we will introduce text summarization models then we will focus on the BART model with the process of fine tuning BART to improve accuracy on text summarization with a dataset of our choice. We will select normal everyday conversations as our main focus for this task, we will choose a random conversation between two people and then fine tune the model to produce a summarization about the conversation.

## 1.1. Text summarization

Text summarization involves creating a shorter summary of lengthy text while retaining important information. It can be done by selecting important sentences (extractive summarization) or generating new ones (abstractive summarization). This task is useful for quickly understanding the key points of a text and has applications in news, meetings, and document summarization.

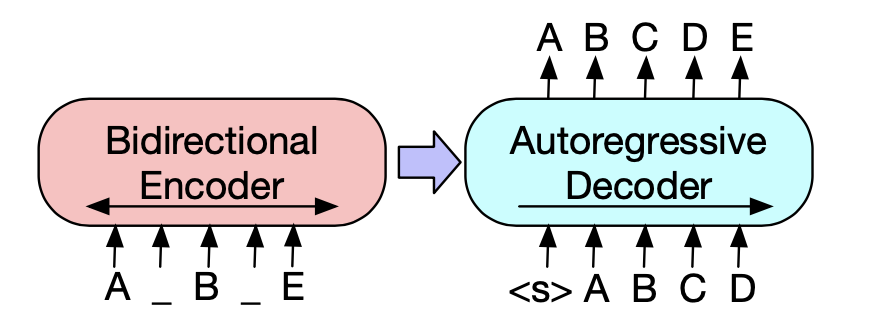
There are two main types of text summarization models: extractive and abstractive. Extractive summarization models select the most important sentences or phrases from the original text and use them to create a summary. These models are relatively simple and often achieve good results, but they have limited ability to generate new information.

Abstractive summarization models, on the other hand, use natural language generation techniques to create a summary that captures the essence of the original text while generating new sentences. These models are more complex and require larger amounts of data to train, but they have the potential to generate higher quality summaries.



In 2017 Google proposes a new architecture called the Transformer in its paper [“Attention Is All You Need”] (<https://arxiv.org/abs/1706.03762>), on which different text generation models are based today, such as GPT-2 and GPT-3, BERT or Transformer XL.

In this post we are going to focus on how to generate summaries with BART, a transformer-based generative language model developed by Facebook AI Research (FAIR),



## 1.2. Hugging face transformer

Huggingface Transformers is a Python library that downloads pre-trained models for tasks like:

* Natural language understanding, such as sentiment analysis
* Natural language generation, such as text generation or text translation.

There are two versions of BART available on the Hugging Face Transformers library: `facebook/bart-base` and `facebook/bart-large`.

* `facebook/bart-base` is the smaller of the two models, with 6 encoder and decoder layers each, and contains a total of 135 million parameters. It is pre-trained on large-scale datasets such as Common Crawl and Wikipedia and fine-tuned on multiple downstream tasks including summarization, translation, and question answering.
* `facebook/bart-large`, on the other hand, is a larger model with 12 encoder and decoder layers each, and contains a total of 406 million parameters. Like `facebook/bart-base`, it is pre-trained on large-scale datasets and fine-tuned on multiple downstream tasks.

There are three main concepts or classes in the library that we will use throughout the post:

* **Tokenizer**: they store the vocabulary of each model and include methods to encode and decode strings in a list of token embeddings indexes that serve as input to the model.
* **Configuration**: they contain the necessary parameters to build a model. They are not required when using a pre-trained model
* **Model**: Pytorch or Keras models to work with the models pre-trained by the library.

**1.3 Dataset**

The SAMSum dataset contains about 16k messenger-like conversations with summaries. Conversations were created and written down by linguists fluent in English. Linguists were asked to create conversations similar to those they write on a daily basis, reflecting the proportion of topics of their real-life messenger convesations. The style and register are diversified - conversations could be informal, semi-formal or formal, they may contain slang words, emoticons and typos.

The created dataset is made of 16369 conversations distributed uniformly into 4 groups based on the number of utterances in con- versations: 3-6, 7-12, 13-18 and 19-30. Each utterance contains the name of the speaker. Most conversations consist of dialogues between two interlocutors (about 75% of all conversations), the rest is between three or more people

The first instance in the training set: {'id': '13818513', 'summary': 'Amanda baked cookies and will bring Jerry some tomorrow.', 'dialogue': "Amanda: I baked cookies. Do you want some?\r\nJerry: Sure!\r\nAmanda: I'll bring you tomorrow :-)"}

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Class** | **Dtype** | **Description** |
| dialogue | Text | string | text of dialogue |
| id | Text | string | id of an example |
| summary | Text | string | human written summary of the dialogue |
|  |  |  |  |

**Data Fields**

* dialogue: text of dialogue.
* summary: human written summary of the dialogue.
* id: unique id of an example.

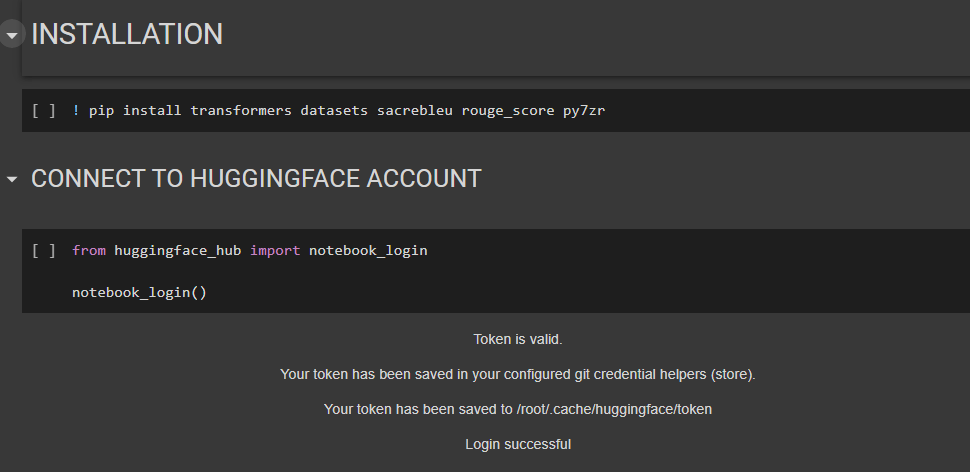
**Data Splits**

* train: 14732 rows
* val: 818 rows
* test: 819 rows

1. **CODE**

**2.1. Installation and connect to hugging face**

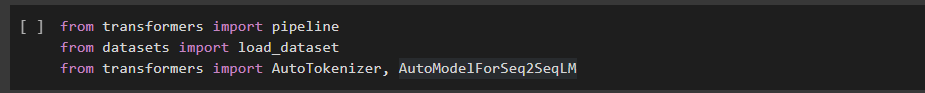
First, we installs five Python packages: `transformers`, `datasets`, `sacrebleu`, ` rouge\_score`, and `py7zr`. These packages provide tools and datasets for natural language processing and machine learning, including pre-trained models, evaluation metrics, and data handling utilities.



**2.2. Import library**

Imports the `pipeline` module from the `transformers` package, which provides an easy-to-use interface for using pre-trained models to perform various NLP tasks.

Imports the `load\_dataset` function from the `datasets` package, which provides a collection of commonly used datasets.

Imports the `AutoTokenizer` and `AutoModelForSeq2SeqLM` classes from the `transformers` package, which are used to create a tokenizer and a pre-trained model, respectively.

Then, import the following libraries:

1. `AutoTokenizer` and `BartTokenizer` from the `transformers` package/module: these are classes that can be used to tokenize input text sequences for training.

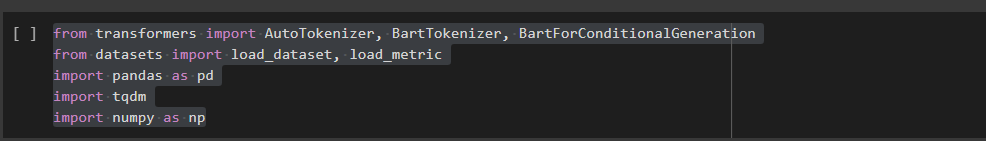
2. `BartForConditionalGeneration` from the `transformers` package/module: this is a BART model that can be fine-tuned to perform various natural language generation tasks including summarization, language translation, and text generation among others.

3. `load\_dataset` and `load\_metric` from the `datasets` package/module: this package offers several publicly available natural language processing datasets as well as Multiple pytorch compatible metrics to evaluate the predictions of NLP models.

4. `pandas` packages/module: provides data analytics tools, for reading and working with data such as table or excel sheets.

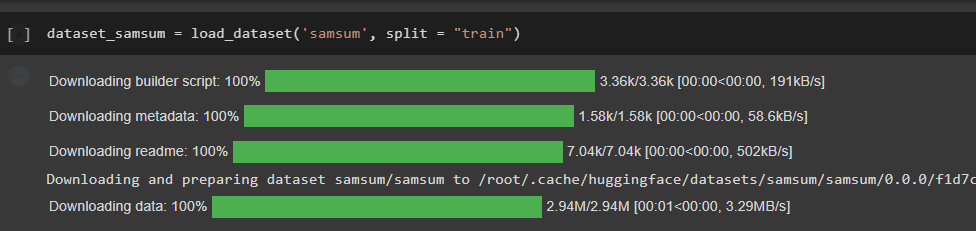
5. `numpy` package/module: provides tools for scientific computing and matrix manipulation.

6. `tqdm` package/module: can be used for displaying a progress bar for tasks that may take long to complete, e.g loading data or training a model.



**2.3. Load dataset**

Then, we will load the dataset, samsum.



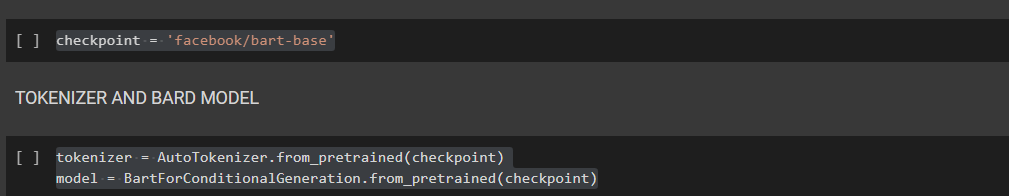
**2.4. Get the pre-trained BART model**

Assigns the model checkpoint string `"facebook/bart-base"` to the variable `checkpoint`.

Loads the tokenizer to the ‘checkpoint’ by calling `AutoTokenizer.from\_pretrained(checkpoint)`.

This creates a tokenizer object that can be used to tokenize input text and convert the tokens to numerical input that can be passed to the model.

Load the pre-trained BART model by calling BartForConditionalGeneration.from\_pretrained(). This create the pre-trained model that can be further refined.

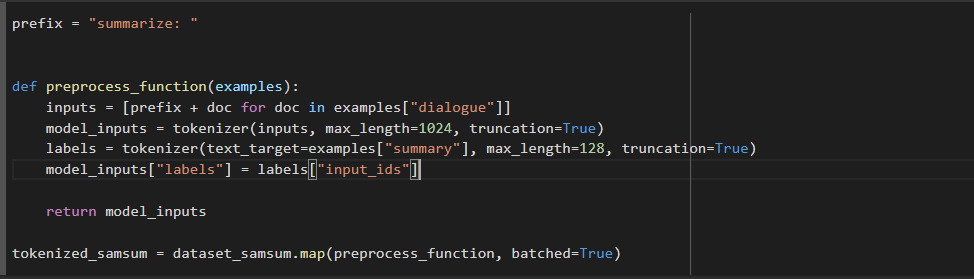


Then, tokenizes the input text and target summaries. This function prefixes each input text with the string `prefix` and then tokenizes the input text and target summary using the `tokenizer` object created earlier in the code.

The `labels` are tokenized by setting the `text\_target` argument to `examples["summary"]` and the maximum sequence length to 128.

The `labels["input\_ids"]` are assigned to the `"labels"` key in the `model\_inputs` dictionary returned by the function/

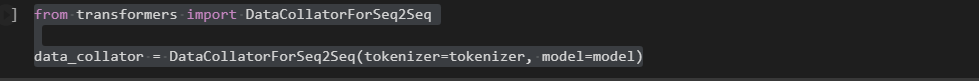
The `tokenized\_samsum` dataset is created by applying the `preprocess\_function` to the `dataset\_samsum` dataset object using the `map()` function with `batched=True`. This applies the `preprocess\_function` to the dataset in batches, which can improve processing speed.



Futhermore, A data collator's job is to take in a batch of examples and format the inputs and labels in the proper way to be fed into the model. For sequence-to-sequence tasks, this means preparing the data in a way that the transformer can generate a summary for the input text.

The `DataCollatorForSeq2Seq` is then instantiated with a `tokenizer` object and a `model` object passed as parameters. The `tokenizer` is used to tokenize the input and target texts, while the `model` is used to determine the maximum output sequence length.

Once instantiated, the resulting `data\_collator` object can be used to format the data for training and evaluation by calling `data\_collator()` on a batch of examples, which typically involves concatenating all the input texts and target summaries into a single tensor.

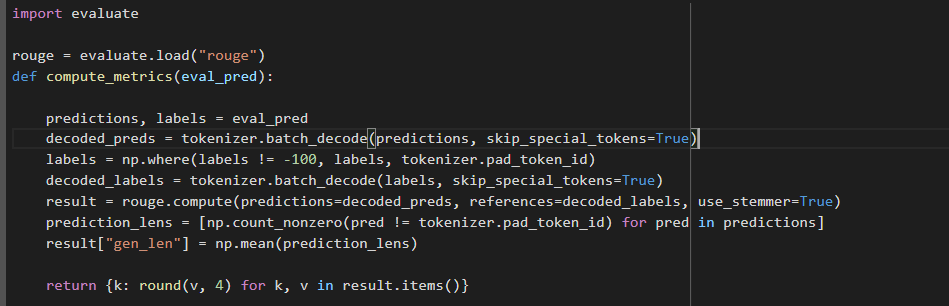


**2.5. Training model**

Before training, defines a function called `compute\_metrics` that takes in an argument called `eval\_pred`. This function is used to compute the evaluation metrics for the BART model during training.

The function first unpacks the `eval\_pred` variable into `predictions` and `labels` using tuple unpacking. The function then decodes the predicted and label sequences using `tokenizer.batch\_decode()` to obtain human-readable summaries. The parameter `skip\_special\_tokens=True` is passed to discard any special tokens, such as padding tokens.

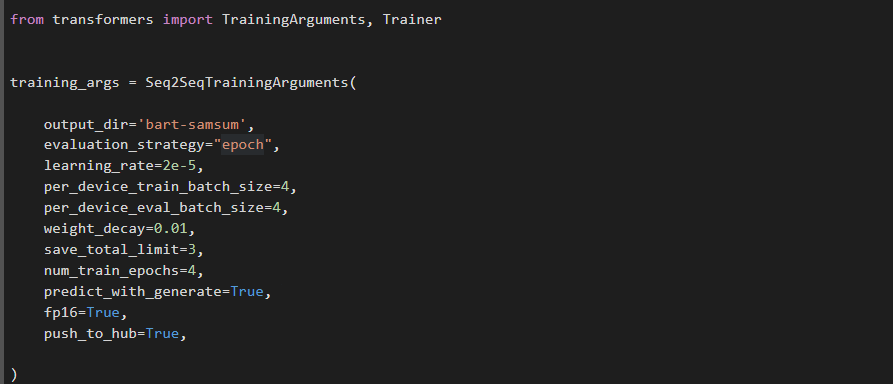
Then, replaces all `-100` values in the `labels` tensor with the `tokenizer.pad\_token\_id` to ensure proper metric computation.. The ROUGE metric measures the similarity between the predicted and target summary texts. The `predictions` and `labels` parameters are the predicted and target sequences, respectively. The `use\_stemmer=True` argument is used to enable word stemming for the metric calculation.The function also computes the average length of the generated summaries and adds it to the results dictionary under the key `"gen\_len"`.

Finally, rounds the result values to four decimal places and returns the resulting dictionary of evaluation metrics. The evaluation metrics are used to monitor the model's performance during training.

Then, defines a training configuration for the BART model using the `TrainingArguments` class from the `transformers` library. The `TrainingArguments` object stores the different settings and hyperparameters used for training the model, such as the learning rate, batch size, number of epochs, and other configurations.

In this code block, the following training configuration options are set:

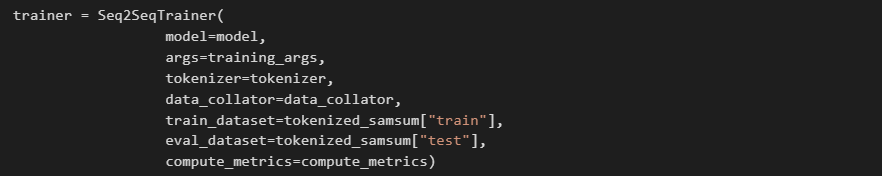
* `output\_dir`: the output directory where the model checkpoints and saved models will be stored.
* `evaluation\_strategy`: specifies how often the model should be evaluated during training. "epoch" means that the model will be evaluated at the end of each epoch.
* `learning\_rate`: the learning rate used for the optimizer during training.
* `per\_device\_train\_batch\_size`: the batch size used for each GPU during training.
* `per\_device\_eval\_batch\_size`: the batch size used for each GPU during evaluation.
* `weight\_decay`: the weight decay used for the optimizer during training.
* `save\_total\_limit`: the maximum number of model checkpoints that will be saved.
* `num\_train\_epochs`: the number of epochs to train the model for.
* `predict\_with\_generate`: indicates whether to use generation during evaluation. If true, the model generates summaries rather than simply predicting the next token.
* `fp16`: indicates whether to use mixed precision training with float16.
* `push\_to\_hub`: indicates whether to push the trained model to the Hugging Face model hub.



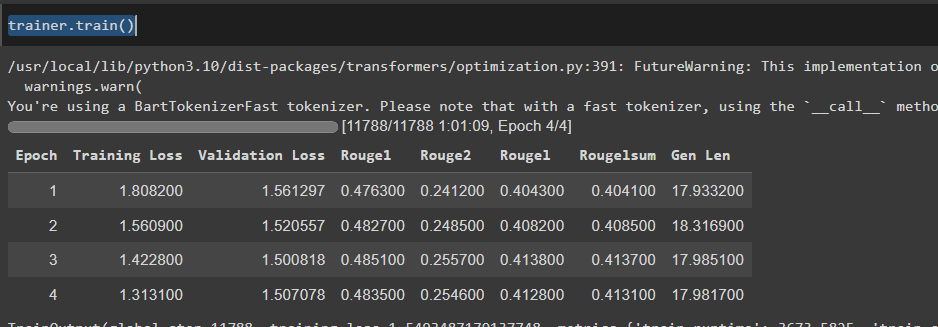
The `Seq2SeqTrainer` object is then instantiated, passing in several parameters:

* `model`: the model to train
* `args`: a `TrainingArguments` object containing all the training and optimization configurations
* `tokenizer`: the tokenizer used to tokenize the input and output texts
* `data\_collator`: the data collator used to preprocess the data before feeding it into the model
* `train\_dataset`: the dataset containing tokenized training samples
* `eval\_dataset`: the dataset containing tokenized validation samples
* `compute\_metrics`: a function for computing evaluation metrics during training

During training, the model will generate summaries for the input texts and adjust the model parameters to minimize a loss function. The `compute\_metrics` function will be used to compute the ROUGE metric during training to monitor the model's performance. At the end of each epoch, the validation dataset will be used to evaluate the model and compute the evaluation metrics.



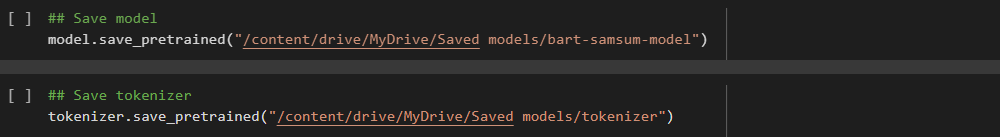
Finally, we train the model. During training, the model will generate summaries for the input texts and adjust the model parameters to minimize a loss function. The loss function will be backpropagated through the model, and the optimizer will update the model weights to improve the model's performance.

At the end of each epoch, the model's performance will be evaluated on the validation dataset and training dataset, and the evaluation metrics will be computed.

After training completed, we push the fine-tuned model to huggingface.

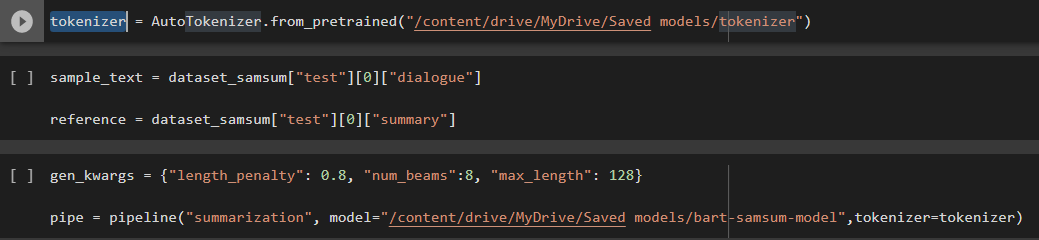


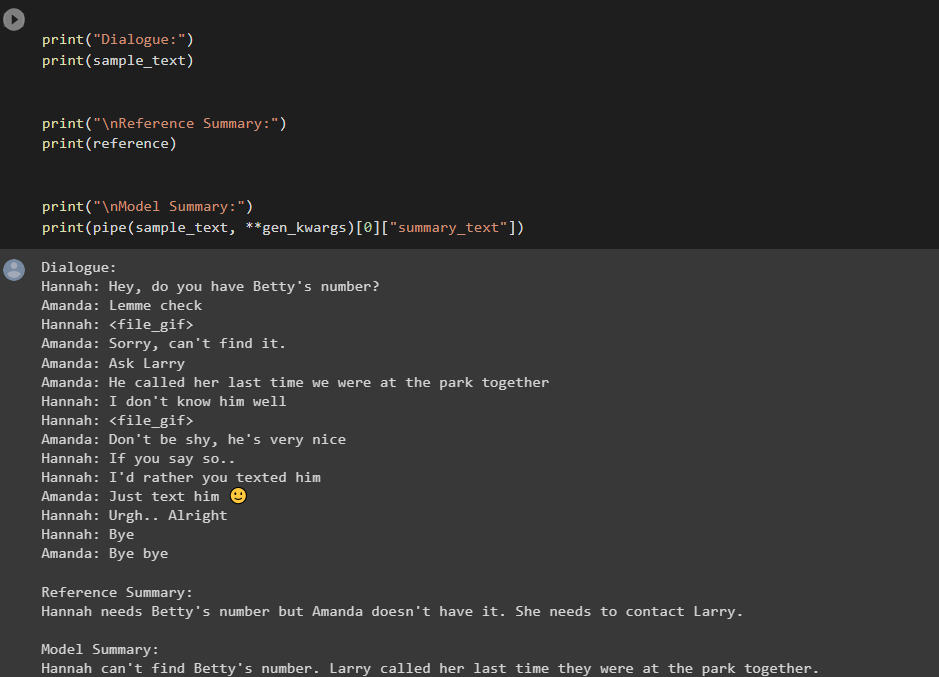
Lastly, we save the model to Google Drive for future uses.



**2.6. Test**

Last but not least, we test the model’s performance by comparing its produced summary with the exisiting summary.



Here is the result, you can see that it generate a summary that is quite closely to the existing one. and is quite accurate.

**Evaluation using rouge score**

