**FLAN – T5 fine-tuning on Stanford Alpaca Dataset**

By

Aishwarya and Deepshika

Advised by Prof. Ausif Mahmood

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIRMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN ARTIFICIAL INTELLIGENCE

THE SCHOOL OF ENGINEERING

UNIVERSITY OF BRIDGEPORT CONNECTICUT

APRIL 2024

**Contents**

1. **Acknowledgments**
2. **Abstract**
3. **List of Acronyms.**
4. **Introduction**

4.1 Background on T5 (Text-to-Text Transfer Transformer)

4.2 Evaluating Flan-T5

4.3 Flan-T5-Large Performance

4.4 Use of Hugging Face Transformers

4.5 Use of Hugging face

**5. Background**

5.1 About Rule-based QA systems

5.2 Machine Learning based QA systems

5.3 Description of Alpaca and use of Dataset

5.4 A Picture illustrating pipeline to create the Alpaca dataset and fine-tuning below

5.5 Prerequisites

**Chapter 1**

**6. Methodology**

6.1 Installing Python packages Torch and Dependencies

1. Python Environment Setup
2. Installing Package Manager
3. Installing Dependencies

6.2 Preparing the Data

6.3 Training the Flan T5 Alpaca Model

6.4 Troubleshooting and Optimizations

6.5 About Transformers in Machine Learning

6.6 Conclusion

6.7 Resources

6.8 Q&A

**Chapter 2**

**7. Model Architecture with Code explanation**

**7.1** **Python Environment**

7.1.1 Python packages Installed

7.1.2 Setting up kernel and required dependencies

**7.2 Datasets**

7.2.1 Create dataset in hugging face from original huge d dataset

7.2.2 Uploaded modified dataset back to our personal hugging face hub

7.2.3 Loading Dataset and LLM

7.2.4 Load the pre-trained FLAN-T5 model and its tokenizer directly from HuggingFace.

**7.3 Testing the model with zero shot Inferencing**

7.3.1 Zero-Shot Inference Overview

7.3.2 Test Data Selection

7.3.3 Evaluation Metrics

7.3.4 Baseline Comparison

**Chapter3**

**7.4 Full Fine-tuning**

7.4.1 What is tokenizing

7.4.2 How to Fine-Tune the Model with the Preprocessed Dataset

**7.5 Model Evaluation**

7.5.1 What is Qualitative Evaluation (Human Evaluation)

7.5.2 What is ROUGE metric

7.5.3 Evaluating the Model Quantitatively (with ROUGE Metric)

**Chapter 4**

**7.6 Parameter Efficient Fine-Tuning (PEFT)**

7.6.1 What is PEFT

7.6.2 What is LORA model

7.6.3 Setting up PEFT/LORA model for fine tuning

7.6.4 How to train PEFT adaptor

**Acknowledgments**

* Huggingface.com

* <https://github.com/tatsu-lab/stanford_alpaca?tab=readme-ov-fil>
* <https://www.traceloop.com/blog/evaluating-model-performance-with-the-rouge-metric-a-comprehensive-guide>
* <https://vegavid.com/blog/parameter-efficient-fine-tuning-guide/>#
* <https://www.datacamp.com/tutorial/flan-t5-tutorial>
* <https://medium.com/version-1/stanford-alpaca-a-small-yet-mighty-language-model-for-instruction-following-tasks-af9e92e87d9a>
* <https://stanford-alpaca.gitbook.io/everything-you-need-to-know/stanford-alpaca-guides/understanding-project-stanford-alpaca>

**ABSTRACT**

**Fine-Tuning FLAN-T5 LLM on Stanford Alpaca Dataset**

Flan-T5 is an open-source LLM that’s available for commercial usage. FLAN-T5 was released in the paper Scaling Instruction-Finetuned Language Models - it is an enhanced version of T5 that has been finetuned in a mixture of tasks. Published by Google researchers, Flan-T5 is an encoder-decoder model pre-trained on a variety of language tasks. The model has been trained on supervised and unsupervised datasets to learn mappings between sequences of text, i.e., text-to-text. In our view, what sets Flan-T5 apart from other models is that its training is based on prompting. In other words, the model knows about performing specific tasks such as summarization, classification, Question and Answering, and translation to name a few. If you download the model from Hugging Face, you can start using it right away for general-purpose applications. One can directly use FLAN-T5 weights without finetuning the model.

In this paper, our objective is to refine the FLAN-T5 Language Model (LLM) through prompt-based fine-tuning on the Stanford Alpaca Dataset for Question and Answering. Additionally, we seek to create a chatbot for this communication, enabling the model to respond to user queries accurately.

**List of Acronyms**

FLAN Federated Learning for Adaptive Networks

T5Text-To-Text Transfer Transformer

LLM Large Language Model

NLP Natural Language Processing

BERT Bidirectional Encoder Representations from Transformers

ROUGE Recall-Oriented Understudy for Gisting Evaluation

RFC Request for Comments

ADAM Adaptive Moment Estimation

DNN Deep Neural Network

GPU Graphic Processor Unit

LSTM Long Short-Term Memory

MSE Mean Squared Error

Seq2seq Sequence to Sequence

SGD Stochastic Gradient Descent

ALPACA Adversarial Language Pair Challeng ALPACA dataset

**Introduction**

T5 (Text-to-Text Transfer Transformer) is a state-of-the-art natural language processing (NLP) model developed by Google Research. It is based on the transformer architecture, which was introduced in the 2017 paper “[Attention is All You Need](https://arxiv.org/abs/1706.03762)” and has since become the foundation for many other NLP models, including BERT, GPT-2, and GPT-3.

T5 is a text-to-text transfer model, which means that it can be fine-tuned to perform a wide range of natural language understanding tasks, such as text classification, language translation, and question-answering. It’s trained on a massive amount of text data, which allows it to understand and generate a wide range of natural language.

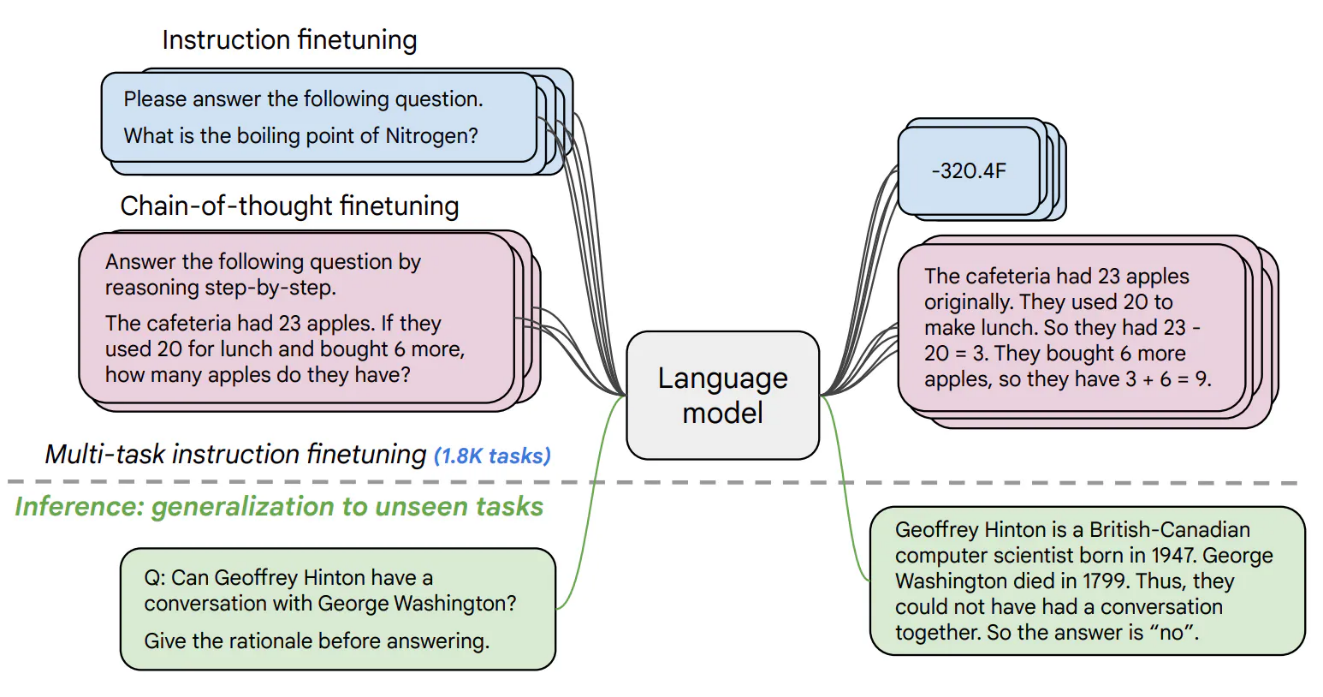
One of the key innovations of T5 is its “prefix” approach to transfer learning, where the model is fine-tuned for a specific task by training it with a prefix added to the input text. For example, to fine-tune T5 for a text classification task, the input text would be prefixed with the task name and a separator, such as “classify: This is the input text.”.

T5 has been shown to achieve state-of-the-art results on a wide range of NLP tasks, and it’s considered a highly sophisticated and powerful NLP model, showing a high level of versatility, fine-tuning capability, and an efficient way to transfer knowledge.

Flan-T5 is an open-source LLM that’s available for commercial usage. Published by Google researchers, Flan-T5 is an encoder-decoder model pre-trained on a variety of language tasks. The model has been trained on supervised and unsupervised datasets with the goal of learning mappings between sequences of text, i.e., text-to-text.

In our view, what sets Flan-T5 apart from other models is that its training is based on prompting. In other words, the model has knowledge of performing specific tasks such as summarization, classification and translation to name a few. For instance, if you were to feed some data into Flan-T5 and tell it to “summarize this article”, it would know that it needs to generate a shorter version of this article. If you download the model from Hugging Face, you can start using it right away for general purpose applications.

**Below is a picture which illustrates Model Card for FLAN-T5 base**



# **Evaluating Flan-T5**

However, to build verticalized solutions on data with this model, we first have to consider: **Performance**, **Time to train**, **Costs** and **Inference**.

# **Flan-T5-Large Performance**

Let’s move on to the first pillar of our evaluation framework:

**Performance**.

We compare the performance of fine-tuning Flan-T5-Base on different tasks given in Alpaca dataset: **Classification and Summarization and Question and answering etc**. Flan-T5 comes in various sizes; for our experiments, we chose Flan-T5-Base, which has 780M parameters. Furthermore, all experiments were conducted on Google Collab using A100GPU which is a free version available.

Fine-tuning the full Flan-T5 model comes with significant hardware and overfitting challenges. Imagine fitting such a huge model in 1 GPU! A nightmare, right? To circumvent this, we will be performing Parameter Efficient fine tuning (PEFT) via [LoRA](https://huggingface.co/docs/peft/conceptual_guides/lora%22%20/t%20%22_blank). In simple words, LoRA is a technique to fine-tune massive models (LLMs) without actually fine-tuning the base model. It works by adding some extra layers throughout the base model and **tuning only those layers**. In the case of Flan-T5-Large, LoRA adds layers that contain only 4.7M parameters. These layers are experts that are focused on learning patterns present in the given dataset while leveraging the generalist knowledge already available in Flan-T5-Base’s 780M parameters.

**USE OF HUGGING FACE TRANSFORMERS :**

Hugging Face Transformers provides APIs to easily download and train state-of-the-art pre-trained models. With Transformers you can train Text, Image, Audio, and Multi-Modal models.

The Hugging Face Transformers library provides pre-trained models for a variety of tasks with deep interoperability between TensorFlow 2.0 and PyTorch. This provides a:

* Low barrier to entry for educators and practitioners
* State-of-the-art NLP for everyone:
* Lower compute costs, smaller carbon footprint
* The encoder receives inputs and iteratively processes the inputs to generate information about which parts of inputs are relevant to each other. The model will be optimized to get the best understanding of the input.
* The decoder generates a target sequence using representation from the encoder and uses the contextual information to generate outputs.
* The key to this structure for transformers is the addition of the "attention" and "self-attention" mechanisms.

**Background**

In today’s fast-paced world, we’re bombarded by information from all directions, making it difficult to sort through all that information to find the answers we need. This is where question-answering systems come in. The systems are designed to understand natural language questions and respond with accurate and relevant answers. Applications range from customer service chatbots to virtual assistants to help desks. They’ve revolutionized the way we access information.

Question-answering (QA) systems are computer programs that are designed to understand natural language questions and provide accurate and relevant answers. They are a type of natural language processing (NLP) system that combines techniques from **information retrieval**, **natural language understanding**, and **knowledge representation** to understand a user’s question and provide a response.

There are several different types of QA systems, but they can generally be grouped into two categories: rule-based systems and machine learning-based systems.

* **Rule-based QA systems** use a set of predefined rules and heuristics to understand the user’s question and generate a response. These systems are typically based on a knowledge base that contains a collection of facts and rules.
* **Machine learning-based QA systems**, on the other hand, use machine learning algorithms to learn from a large dataset of questions and answers. These systems are trained to understand natural language questions and generate relevant responses. They typically rely on deep learning algorithms such as recurrent neural networks (RNNs) and transformer architectures.

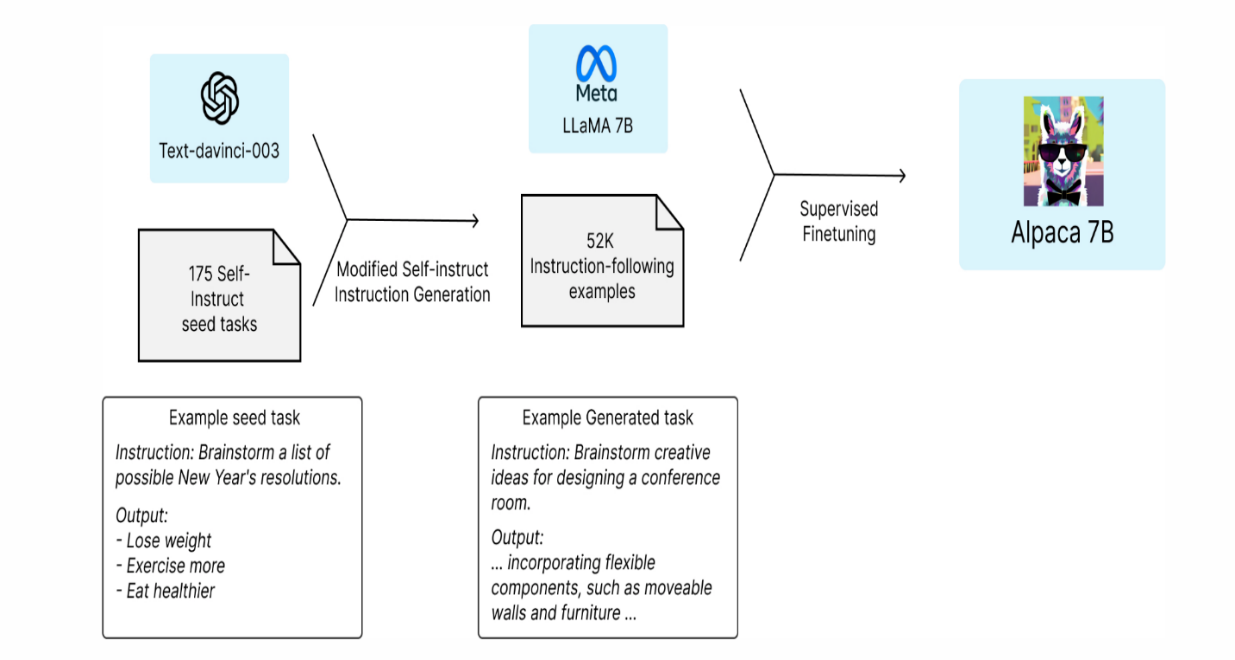
QA systems are used in a wide range of applications, such as customer service chatbots, virtual assistants, help desks, etc. They’re also used in knowledge-based systems, where they help users access information more intuitively and efficiently. Some QA systems can answer simple questions, while others can provide answers to more complex questions that require reasoning and inference. Overall, QA systems are aimed to improve human-computer interaction and make it easier for people to find the information they need more efficiently.

**Alpaca and use of the dataset:**

The Alpaca dataset serves as a valuable resource for training and testing our FLAN t5 model. It includes a wide range of tasks such as summarization, translation and question and answer which further enriches the learning process of our model.

By harnessing the combined power of FLAN t5 and the Alpaca dataset, we aim to develop a cutting-edge response for the system. This project represents a significant step towards leveraging advanced AI technologies for various tasks and addressing real-world challenges in industries.

**The Below Picture illustrates pipeline to create the Alpaca dataset and fine-tuning below:**



**Chapter 1**

**Prerequisites**

Now that we have a better understanding of FLAN-T5, let’s see how to fine-tune it for a question-answering use case. To begin, the following libraries and tools are required:

**Hugging Face**: A platform that provides access to the FLAN-T5 model, facilitating its download and usage for fine-tuning

**Transformers**: This is used to simplify the process of loading the pre-trained FLAN-T5 model and provides useful functions for fine-tuning

**Datasets**: a collection of ready-to-use datasets, crucial for sourcing relevant data for fine-tuning**.**

**Tokenizers**: a tokenization library for converting text into a suitable format for the use case

**Evaluate**: This library provides a wide range of metrics for model evaluation, ensuring that the fine-tuned model meets the desired performance standard

**Rouge score**: specific metric used to evaluate the quality of text generated by large language models.

**NLTK**: useful for data preprocessing steps such as tokenization and stemming.

Throughout this project, we will explore the possibility of using the Alpaca idea to train a Flan T5 Large model. We will discuss the installation process, data preparation, and training steps, and address any potential issues along the way. By the end, we aim to give a clear understanding of how to leverage the Alpaca concept to fine-tune the Flan T5 Large model successfully.

**Methodology**

**1. Installing Torch and Dependencies**

Before we dive into the training process, we need to ensure that we have all the necessary tools and libraries installed. To get started, we will install Torch 1.13.1, as it provides a faster compile function. Additionally, we'll need to install other dependencies such as datasets, transformers, evaluate, rouge, fire, PyTorch, and lightning. We'll also make sure to use the latest version of the transformer library (v4.28).

Python Package Installation

Before starting a FLAN T5 LLM (Federated Learning for Adaptive Networks with T5 Large Language Model) project, you may need to install several Python packages to support your development and experimentation. Here's a list of commonly used packages

Below are the dependencies we need to install

**Transformers** : Required for using T5 Large Language Model and other transformer-based models. We can install it using below command

Pip install transformers

**Pip install transformers**

**Datasets :** Necessary for handling datasets efficiently,Including downloading and preprocessing . We can install it using below command

Pip install datasets

**Torch**: PyTorch is commonly used for deep learning tasks. We can install it using below command

pip install torch

**Torchdata**: Useful for creating custom datasets and data loading pipelines with PyTorch. Install it using

pip install torchdata

**Scikit-learn**: Provides tools for machine learning tasks such as data preprocessing, model evaluation, and more. Install it via pip

pip install scikit-learn

**Numpy**: Essential for numerical operations and array manipulation in Python. Install it using

pip install numpy

**Pandas**: Useful for data manipulation and analysis, especially when working with structured data. Install it via

pip install pandas

**2. Preparing the Data**

To train the Flan T5 model, we need a dataset. In this case, we will use the Stanford Alpaca dataset provided by a Hugging face hub called "tural/stanfordalpaca". If a clean version of the data is available, we will use that for our training. We will create a new dataset in the hub to store our data with reduced size of records “Aishwarya30998/updated\_alpaca\_dataset\_3k” and proceed to download and load the dataset using the available Python scripts.

**3. Training the Flan T5 Alpaca Model**

Now that we have the data ready, we can move on to the fine-tuning process. We will specifically focus on training the Flan T5 Large model using our chosen dataset. It is essential to check if our GPU has enough memory to handle this model. We'll also set the necessary parameters for the training, including learning rate, source length, target length, and various other configurations. We'll train the model using PyTorch and monitor the progress throughout the training process.

4. Troubleshooting and Optimizations

During the training process, we may encounter some issues or challenges. In this section, we will address common troubleshooting scenarios and provide optimization techniques to improve the training performance. We will explore potential solutions for CUDA device compatibility issues, RAM limitations, and any other problems that may arise during the training process.

5. Conclusion

In conclusion, we have learned how to utilize the Alpaca idea to train a Flan T5 Large model. We covered the installation process, data preparation, training steps, and comparing error metrics. Have successfully trained the Flan T5 Alpaca model and achieved excellent results for your natural language processing tasks. The possibilities with Alpaca and Flan models are vast.

**Highlights:**

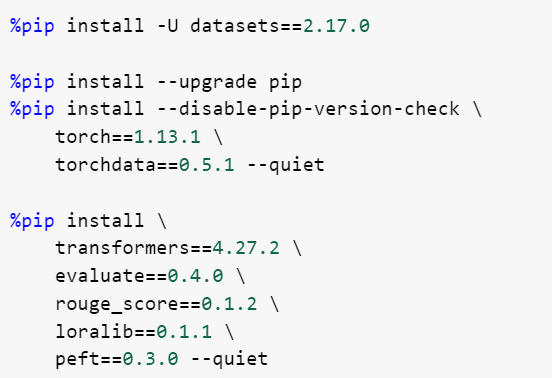
1. Explore the possibility of training a Flan T5 Large model using the Alpaca idea.
2. Install Torch and all necessary dependencies.
3. Prepare the dataset for training.
4. Fine-tune the Flan T5 Alpaca model.
5. Compare the metrics with different finetuned models.

Going In depth on how Model and Code evaluation on how to Fine-Tune a Generative AI Model for sequence-to-sequence generation.

Chapter 2

**Setting up kernel and required dependencies**

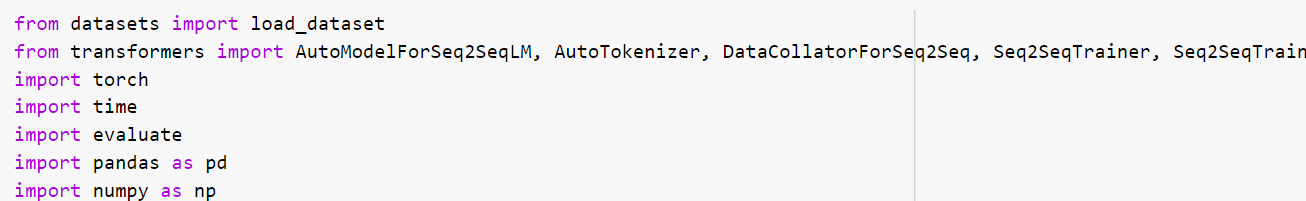
**Code :**



**Explanation for the code above** :

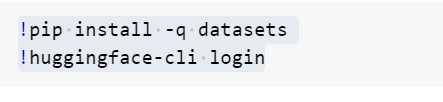
* The provided code is a series of commands written in the Python programming language. These commands are executed in Google colab for this specific project or a Python script to install specific Python packages and their dependencies. Let's break down each line of code and explain its purpose:
* %pip install -U datasets==2.17.0
* This command uses the %pip magic function, and allows the package installations within the environment.
* The -U flag stands for "upgrade," which instructs pip to upgrade the specified package to the latest version if it's already installed.
* datasets==2.17.0 specifies the exact version of the datasets package to be installed. This package is commonly used for handling datasets in machine learning projects.
* %pip install --upgrade pip
* This command upgrades the pip package manager itself to the latest version available.
* The --upgrade flag ensures that if pip is already installed, it will be upgraded to the latest version.
* %pip install --disable-pip-version-check \
* This command is like the previous one, but it includes the --disable-pip-version-check flag.
* This flag disables the check for the latest version of pip when installing packages. It can be useful to speed up the installation process, especially in automated environments.
* torch==1.13.1 \
* This line installs a specific version of the PyTorch library, which is a popular deep learning framework.
* torchdata==0.5.1 installs a specific version of the torchdata package, which is often used for data loading and preprocessing in PyTorch projects.
* %pip install \ transformers==4.27.2 \
* These lines install several additional Python packages used in natural language processing (NLP) tasks.
* transformers==4.27.2 installs a specific version of the Hugging Face Transformers library, which provides easy-to-use interfaces to pre-trained NLP models like BERT, GPT, and T5.
* evaluate==0.4.0 installs a package for evaluating NLP models, though it's less commonly used compared to other packages like rouge\_score.
* rouge\_score==0.1.2 installs a package for calculating the ROUGE metric, which is commonly used in evaluating text summarization models.
* loralib==0.1.1 and peft==0.3.0 are additional packages, possibly custom libraries or tools required for specific tasks in the research project.
* Overall, these commands are used to set up the Python environment for a machine learning or NLP project, ensuring that specific versions of required packages are installed

**Importing necessary dependencies**



As shown above this code snippet sets up the necessary libraries and tools for working with Seq2Seq models, loading datasets, training models, evaluating performance, and manipulating data. It's commonly used in NLP research projects or experiments involving sequence generation tasks.

**Create dataset in hugging face from original huge dataset**

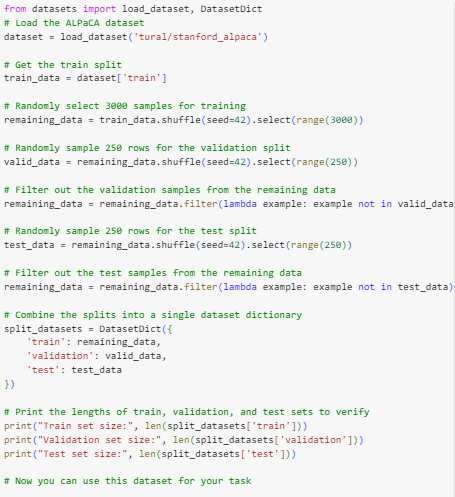
**pip install -q datasets**

* This line uses the !pip syntax to execute a pip command within the Jupyter Notebook environment.
* pip is the package installer for Python, and the install command is used to install Python packages.
* -q flag stands for "quiet" mode, which suppresses output except for errors and warnings. It's commonly used to make the installation process less verbose.
* datasets is the name of the Python package being installed. In this case, it's likely the Hugging Face datasets library, which provides a convenient interface for accessing and working with various datasets commonly used in natural language processing (NLP) tasks.

**Huggingface-cli login**

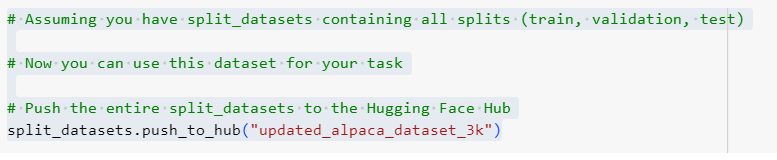
The provided code snippet installs the Hugging Face datasets library using pip and authenticates the user with the Hugging Face platform using the command-line interface. These steps are often the initial setup required for accessing and working with Hugging Face datasets and models in a Python environment.

**Collecting dataset from tural/stanford\_alpaca which has 52K rows and modifying to our needs to 3k rows and splitting it to train, validation and test datasets below is the code**



In the above code we demonstrating how to split a dataset into training, validation, and test sets while ensuring that the splits are disjoint and the sizes are as specified. It's a common preprocessing step in machine learning projects to prepare data for model training and evaluation.

**Uploaded modified dataset back to our personal hugging face hub**

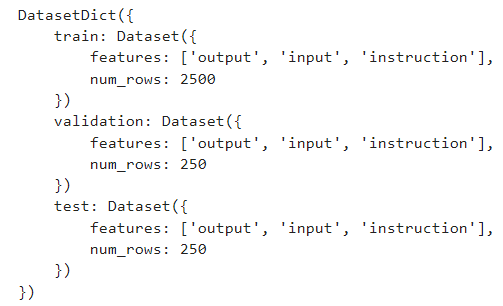


**Loading Dataset and LLM**

Here we are experimenting with the Stanford Alpaca Dataset Hugging Face dataset. It contains 3000+ instructions with the corresponding manually labeled inputs and outputs.

Stanford Alpaca Dataset has 52k Rows of data from which we have taken only 3K and further divided them into Training, Testing and Validation sets

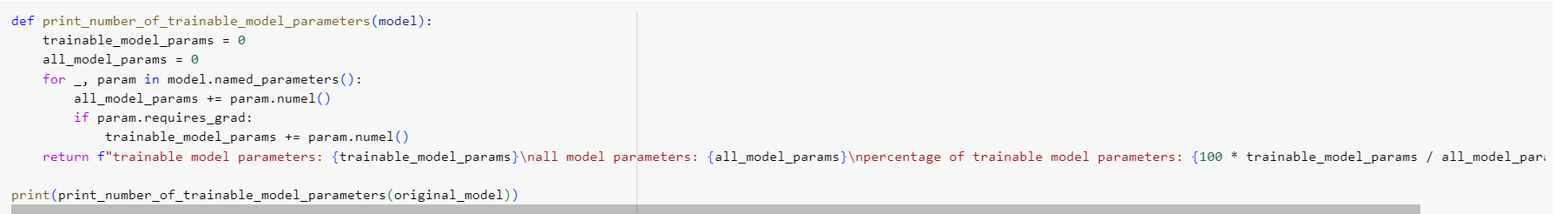
Below is the code how you divide the data inro three different sets



**Load the pre-trained FLAN-T5 model and its tokenizer directly from HuggingFace**

Here we are using the small version of FLAN-T5. Setting torch\_dtype=torch.bfloat16 specifies the memory type to be used by this model.

Below is the code to determine how many of the model parameters are and find out how many of them are trainable. The following function can be used to do that, at this stage, you do not need to go into details of it.



In PyTorch, the .numel() method is used to compute the number of elements in a tensor. The name "numel" stands for "number of elements". It returns the total number of elements in the tensor, which is equal to the product of the sizes of all dimensions of the tensor.

### **Test the Model with Zero Shot Inferencing**

**Zero shot Inferencing**

Zero-shot learning in NLP allows a pre-trained LLM to generate responses to tasks that it hasn’t been specifically trained for. In this technique, the model is provided with an input text and a prompt that describes the expected output from the model in natural language. The pre-trained models can use its knowledge to generate coherent and relevant responses even for prompts it hasn’t specifically been trained on. Zero-shot learning can reduce the time and data required while improving efficiency and accuracy of NLP tasks. Zero-shot learning is used in a variety of NLP tasks, such as question answering, summarization, and text generation.

Few-shot learning involves training a model to perform new tasks by providing only a few examples. This is useful where limited labeled data is available for training. Although this post primarily focuses on zero-shot learning, the referenced models are also capable of generating responses to few-shot learning prompts.

**Types of T5:**

This includes Flan-T5 Small, Flan-T5 Base, Flan-T5 Large, Flan-T5 XL, and Flan-T5 XXL. Furthermore, JumpStart provides three versions of Flan-T5 XXL at different levels of quantization:

* Flan-T5 XXL – The full model, loaded in single-precision floating-point format (FP32).
* Flan-T5 XXL FP16 – A half-precision floating-point format (FP16) version of the full model. This implementation consumes less GPU memory and performs faster inference than the FP32 version.
* Flan-T5 XXL BNB INT8 – An 8-bit quantized version of the full model, loaded onto the GPU context using the accelerate and bitsandbytes libraries. This implementation provides accessibility to this LLM on instances with less compute, such as a single-GPU ml.g5.xlarge instance.

**Chapter 3**

## **Perform Full Fine-Tuning**

We will Tokenize the dataset values using tokenize function.

NLP and Tokenization :

Tokenization is a fundamental part of natural language processing (NLP)

What is natural language processing?

Natural language processing (NLP) is a type of artificial intelligence focused on developing computer algorithms that allow computers to read, understand, and generate language. This includes written or spoken language, with the goal of computers being able to comprehend and communicate language in the same way that humans do. To do this, NLP models use a combination of pre-defined rules and machine learning algorithms to analyze the content and intent of messages and interpret the underlying meaning.

Several key components work in tandem to allow NLP algorithms to process and manipulate human language. Some of the common ones include:

Tokenization: Tokenization involves segmenting text into smaller units that are analyzed individually.

Embeddings: Embeddings are words or phrases represented as numerical vectors to make it easier for computers to read and process the input.

Architecture type: NLP systems can use various architectures, including recurrent neural networks (RNNs) and transformers, to perform specific language tasks effectively. The architecture type will play a role in how the algorithm performs.

Types of tokenization

Depending on your algorithm, you can choose to define your tokens at various levels of granularity. This will often depend on your use case, and different languages may lend themselves better to different types of tokenization. While words are a common choice for token type, you can also choose to reduce your token size to characters or morphemes, or you can expand to words or phrases. Some types of tokens you can define through tokenization include:

Character tokenization: You can tokenize words into individual characters at the most basic level. Doing so can be useful as it limits the number of defined entities.

Word tokenization: Word tokenization involves splitting text into individual words. For instance, the sentence “The grass is green” is tokenized into four tokens with this method: [“The”, “grass”, “is”, “green”].

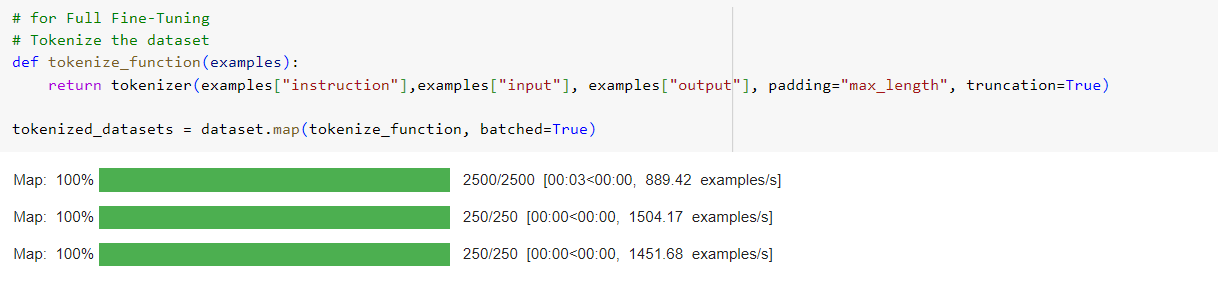
Phrase tokenization: You can also tokenize text into phrases or chunks that convey a specific meaning. For instance, the phrase “Los Angeles” might be a single token instead of two separate words.

Sentence tokenization: This type of tokenization segments text into sentences. It separates paragraphs or long blocks of text into distinct sentences for analysis.

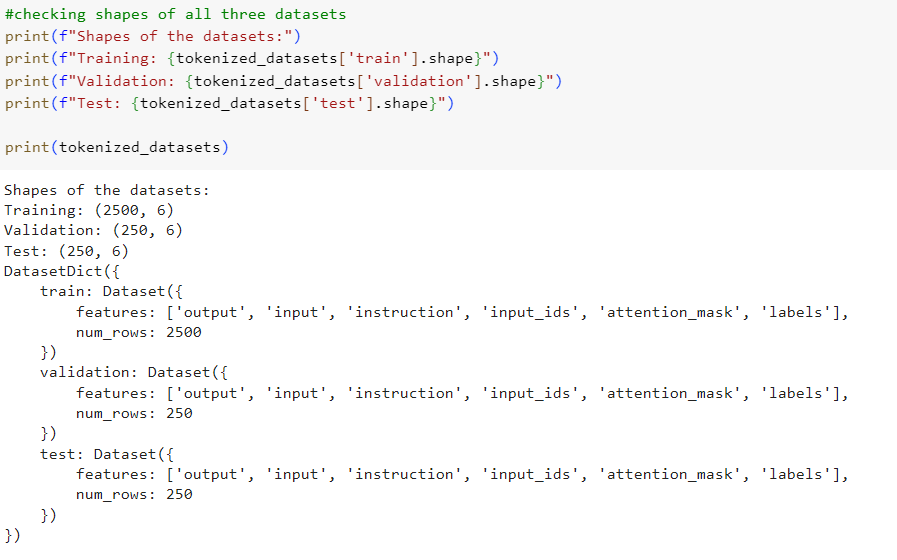
Subword tokenization: Subword tokenization dissects words into their constituent morphemes, which are the smallest units of meaning in a language. For instance, “unusual” becomes [“un”, “usual”].

Number tokenization: Number tokenization uses digits as the primary component of the token and segments numbers from the rest of the body of text. For example, if the phrase was, “She had 15 cats,” then “15” would be the number token.

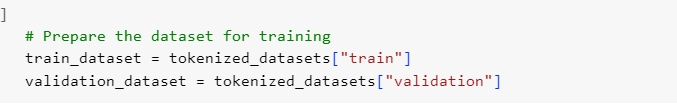
Below is how we full fine tune the data model and tokenize function



After tokenizing, few more ids such as input\_ids, attention\_mask and labels are added to the dataset



Splitting the dataset into train and validation dataset for training



**3.1 Fine-Tune the Model with the Preprocessed Dataset**

Now utilizing the built-in Hugging Face Trainer class here. Passing the preprocessed dataset with reference to the original model.



**EPOC**

**Understanding the Concept of Epochs in Machine Learning**

In the field of machine learning, one important concept that plays a key role in the training of models is that of epochs. Epochs refer to the number of times an entire dataset is passed through a machine learning algorithm during the training phase. Understanding epochs is crucial for effectively training models and improving their performance.

**Content**

Epochs in machine learning refer to the number of times the entire dataset is passed through the model during training

Why are Epochs Important?

Determining the Number of Epochs

**In Conclusion**

Increasing the number of epochs can improve the model's performance and accuracy.

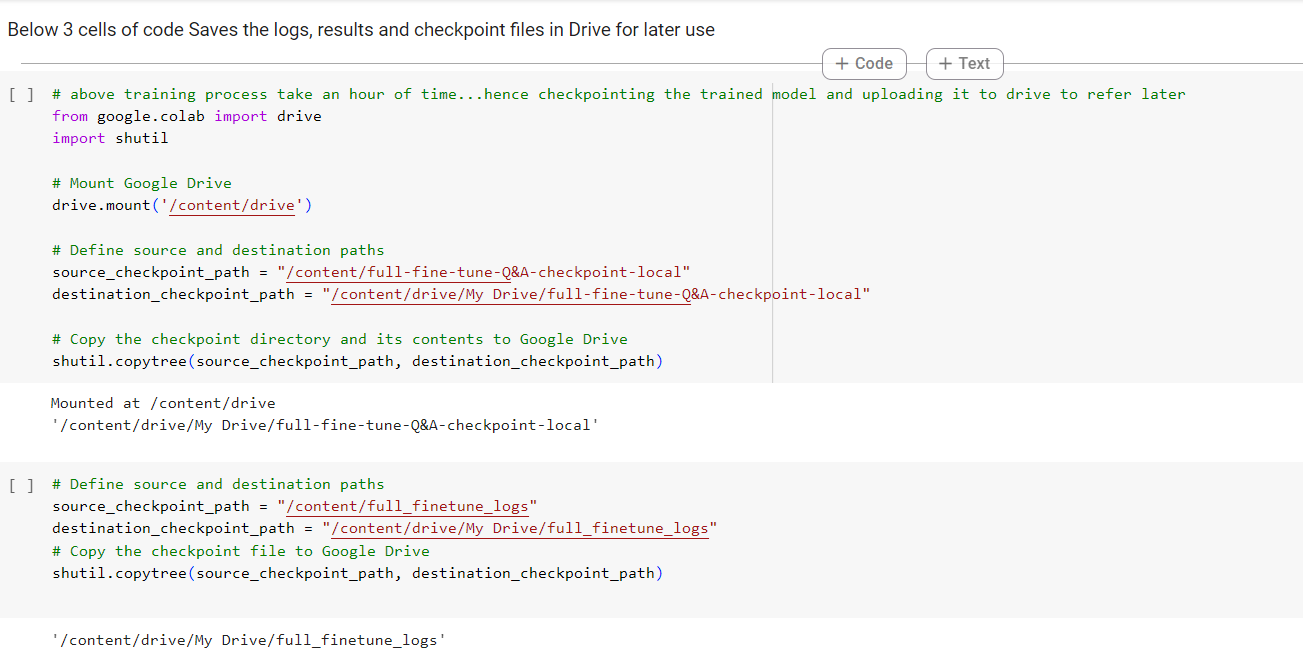
However, too many epochs can also lead to overfitting, where the model becomes too specialized to the training data and performs poorly on new data. Choosing the right number of epochs requires experimentation and monitoring the model's performance on a validation set. Techniques such as early stopping can be used to automatically determine the optimal number of epochs.

In deep learning, where models have many layers, training for a large number of epochs is often necessary to achieve good results

It is important to balance the number of epochs with other factors such as computational resources and time constraints

Factors to Consider when Choosing the Number of Epochs:

Regularization techniques, such as dropout or L1/L2 regularization, can help mitigate overfitting and improve performance even with a smaller number of epochs



**Now we will Perform qualitative evaluation (Human Evaluation) for the given dataset: For original model**

To perform a qualitative evaluation (human evaluation) for the original model on a given dataset, you'll need to follow a structured process to gather feedback from human evaluators. In our Model, the evaluator is the Model itself.

Here's a step-by-step guide:

Step 1: Define Evaluation Metrics

Define the criteria or metrics based on which you'll evaluate the original model's performance. Metrics such as fluency, coherence, relevance, and overall quality can be used. Human judges assess the generated text's quality compared to reference texts.

Step 2: Select Human Evaluators

Select a diverse group of human evaluators who are proficient in the language and context of the dataset. Ensure they are unbiased and representative of your target audience.

Step 3: Provide Instructions

Clearly explain the evaluation criteria and provide instructions on how evaluators should assess the model's outputs. It's crucial to ensure that evaluators understand the task and what is expected from them.

Step 4: Sample Selection

Randomly select a subset of data from the dataset for evaluation. Ensure that the subset covers various aspects of the dataset to get a comprehensive evaluation.

Step 5: Evaluation Process

Individual Evaluation: Provide each evaluator with a set of model outputs along with the corresponding inputs. Ask them to evaluate each output independently based on the defined criteria.

Scoring: Use a scoring system (e.g., Rouge or BLUE) to quantify the evaluation. For example, you can use a scale from 0 to 1, where 0 indicates poor performance and 1 indicates excellent performance.

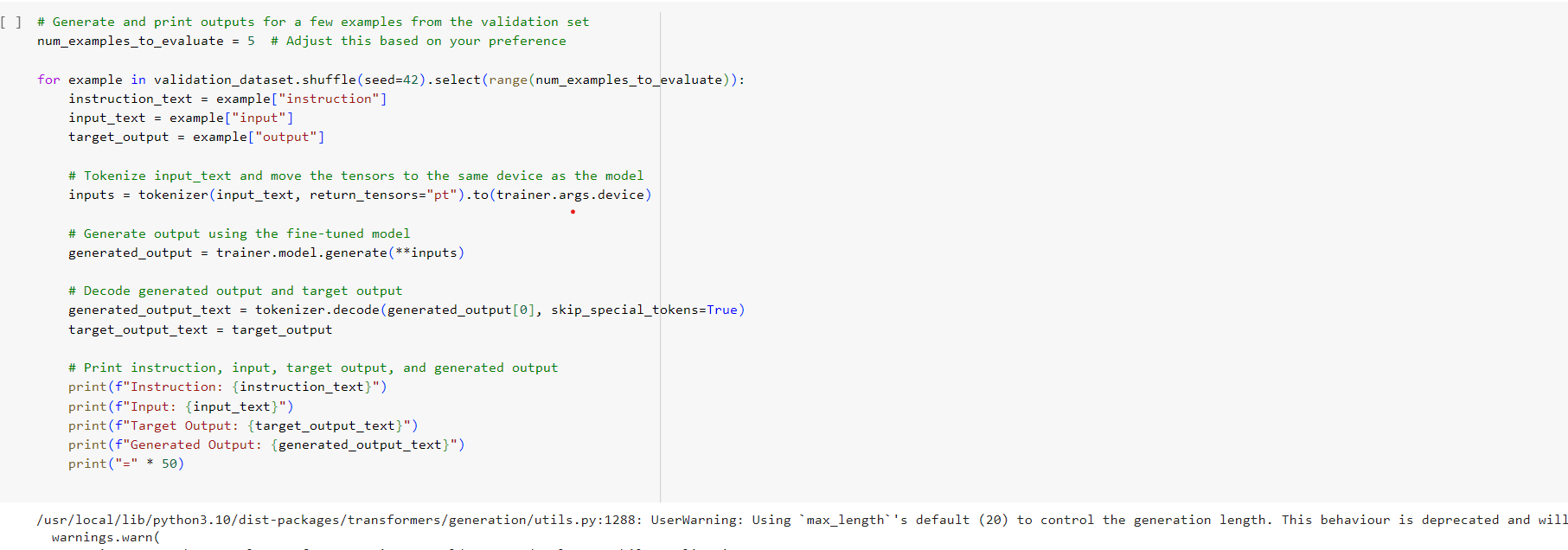
Step 7: Interpretation and Reporting

Interpret the evaluation results and report findings. Highlight areas where the model performs well and areas that need improvement. Provide actionable insights for model refinement based on the feedback received.

Step 8: Iteration

Use the insights gained from the evaluation to iterate on the model if necessary. Implement improvements and repeat the evaluation process to gauge the impact of changes.

To evaluate the model qualitatively, you can generate summaries for a few examples from the validation set and examine them manually to assess the quality. You can print out the generated summaries along with the corresponding input and target summaries.



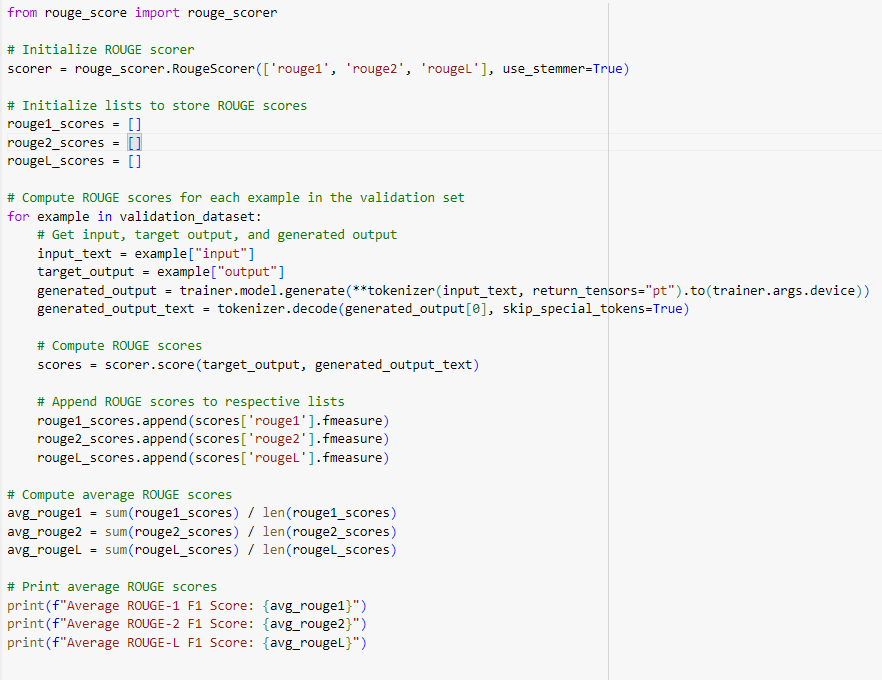
**Quantitative Evaluation (ROUGE Metric): for initial original model**

You can use the ROUGE metric to quantitatively evaluate the model's performance. The datasets library provides an easy way to compute ROUGE scores for generated summaries compared to the target summaries.

# **An intro to ROUGE, and how to use it to evaluate summaries**

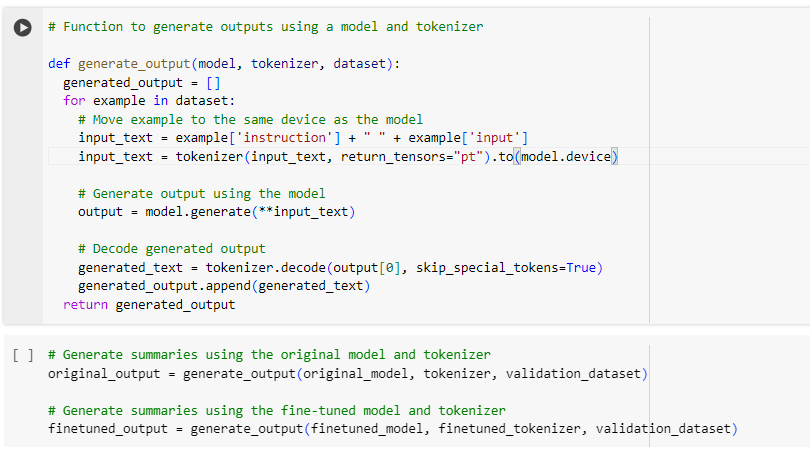
ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. It is essentially a set of metrics for evaluating automatic summarization of texts as well as machine translations.

It works by comparing an automatically produced summary or translation against a set of reference summaries (typically human-produced).



To evaluate the model and compute ROUGE metrics for both the original flan-t5-base model and the fine-tuned model, you can follow these steps:

Now using finetuned model to compare with original model Load the original flan-t5-base model and tokenizer that we have Also Load the finetuned model and full fine tuned tokenizer trained before. Tokenize the evaluation dataset. Generate outputs using both models. Compute ROUGE metrics for the generated outputs compared to the reference outputs in the dataset.

Evaluate the Finetuned Model Quantitatively (with ROUGE Metric)

Evaluating text quality can be extremely tricky, yet it’s foundational to any use of generative AI models. But although assessing the quality of text might be easy for humans, it’s usually much harder for machines. One reason is that the criteria that determine the quality of text are not well defined. Furthermore, those criteria might change, depending on the text’s purpose and context.

**Understanding the ROUGE Metric**

ROUGE is a set of metrics originally designed for evaluating text-summarization algorithms. The metrics compare an automatically produced summary against a human-produced reference summary.

For illustration, consider the following human-produced reference (denoted with R) and a model-generated text (denoted with C):

R: Dan loves chocolate cakes

C: Dan loves chocolate chip cookies

ROUGE-N

The ROUGE-n score measures the overlap of n-grams (sometimes referred to as gram length) in the reference and the generated text. This means the number of n consecutive words that appear both in the evaluated text and the measured text. Formally, we define:

ROUGE[n]-recall = # n-grams that appear in both R and C / number of n-grams in R

ROUGE[n]-precision= # n-grams that appear in both R and C / number of n-grams in C

The ROUGE[n]-F1 score is then defined as:

2\* ROUGE[n]-recall\* ROUGE[n]-precision/( ROUGE[n]-recall + ROUGE[n]-precision)

Back to our example:

ROUGE-1:

ROUGE[1]-recall = 3/4 = 0.75

ROUGE[1]-precision = 3/5 = 0.6

ROUGE[n]-F1 = 2\*(0.75\*0.6)/(0.75+0.6) = 0.66

ROUGE-2:

ROUGE[2]-recall = 2/3 = 0.66

ROUGE[2]-precision = 2/4 = 0.5

ROUGE[2]-F1 = 2\*(0.66\*0.5)/(0.66+0.5) = 0.56

ROUGE-L

While ROUGE-N is based on the overlap of n-consecutive words in both the reference and the automatically produced text, ROUGE-L considers the longest common subsequence of words (LCS) — even if they aren’t consecutive, but still in order. For instance, in our example, the LCS is “Dan loves chocolate” — and has a length of 3. Therefore:

ROUGE-L-recall = ¾ = 0.75

ROUGE-L-precision = 3/5 = 0.6

ROUGE-L-F1 = 0.66

ROUGE-S (n)

ROUGE-S is a skip-gram concurrence metric: it considers n-grams that appear in the reference text and allows the words to be separated by one or more words in the model output (but they must still appear in order).

For example, consider the following sentences:

R: Dan loves chocolate cakes

C: Dan loves chocolate chip cookies and cakes

**Chapter 4**

## **Perform Parameter Efficient Fine-Tuning (PEFT)**

Parameter-efficient first rate-tuning (PEFT) refers to a hard and fast of techniques that propose to fine-tune large pre-trained models with the usage of a small subset of parameters, even as preserving most of the authentic pre-trained weights fixed. This facilitates coping with issues like high computational and storage expenses concerned with complete exceptional tuning of massive fashions. PEFT techniques alter select parameters like embedding layers or extra assignment-precise parameters delivered on top of pre-trained models.

During best-tuning, these project-precise parameters are up to date through backpropagation, at the same time as the authentic weights continue to be unchanged. This permits pre-trained models to be correctly tailored to new tasks and datasets without incurring the useful resource-intensive manner or dangers of full excellent tuning. PEFT leverages the powerful capabilities of huge language fashions through their pre-trained representations, whilst enabling challenge-unique customization via lightweight parameter updates.

**Benefits of PEFT**

**Here are the benefits of PEFT in individual paragraphs:**

**Decreased computational and storage costs**

One of the main advantages of parameter-green fine-tuning is lower computational and storage requirements compared to great-tuning big pre-skilled fashions. These models can include hundreds of tens of millions or even billions of parameters, making full high-quality tuning quite high-priced in terms of computing electricity and reminiscence needed. Fine-tuning the whole model often requires high-end GPUs and can take days or weeks to finish.

**Overcoming catastrophic forgetting**

Another key benefit of parameter-efficient fine-tuning is its ability to triumph over catastrophic forgetting. When huge pre-educated fashions are fine-tuned on a brand-new assignment, it could frequently purpose the version to overlook what it had found out in pre-schooling. The expertise from the earlier assignment is overwritten using the updates for the new assignment.

**Portability**

One of the advantages of parameter-efficient fine-tuning is the high portability. It provides across different domains and tasks. Because PEFT only introduces a small set of task-specific parameters while keeping the general-purpose parameters of the pre-trained model intact, it is very easy to transfer a PEFT-adapted model to new unlabeled datasets. This allows the same model to be readily applied to related tasks with little to no additional training.

**Performance comparable to full fine-tuning**

While aiming to overcome the limitations of traditional full fine-tuning, a key requirement for PEFT techniques is that they must not significantly degrade performance on the downstream task.

**Parameter-Efficient Fine-Tuning Techniques**

Here are some key parameter-efficient fine-tuning techniques that could be discussed:

**Adapter**

One of the most popular and effective PEFT methods is using adapter modules. Adapters involve adding a simple “bottleneck” architecture, such as a two-layer feedforward network, between existing layers or blocks of the pre-trained model. Only the parameters in the adapter modules are fine-tuned for the new task, while the original model weights remain untouched. At inference, the adapters can be combined with the base model by merging the outputs.

**Lora**

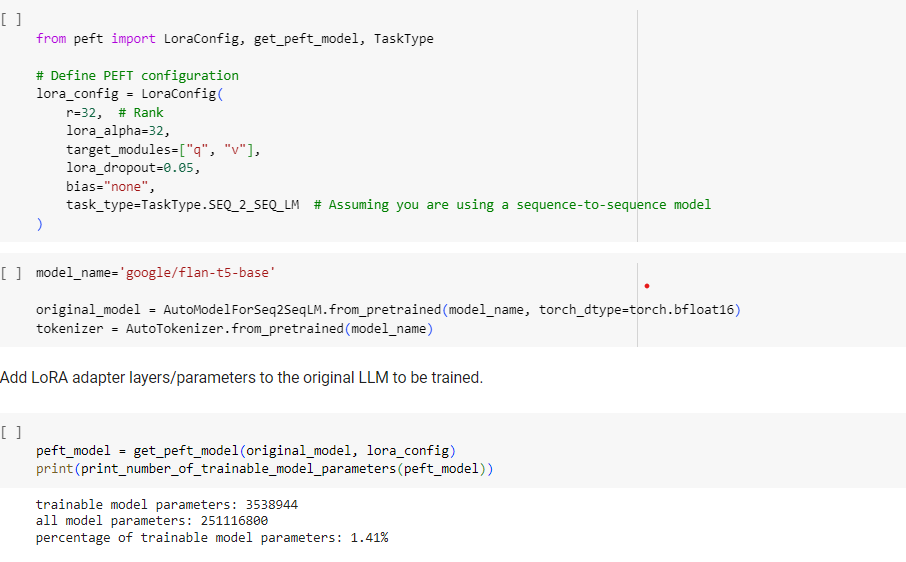
Lora (Learnable Mixing of Representations) is another approach for parameter-efficient fine-tuning. Lora augments the pre-trained model with a learnable ‘mixer’ module that mixes the task-specific representations with the original contextualized representations from the pre-trained model. The mixer contains task-specific learnable parameters optimized during fine-tuning while keeping the pre-trained parameters fixed. This allows the model to learn new capabilities for the task without forgetting the original pre-trained knowledge. Lora has been shown to outperform adapter-based methods in highly multitask learning scenarios. The mixing operation allows flexible sharing of information between tasks. However, Lora requires more parameters than adapters since it uses a full-rank matrix for mixing.

**Conclusion**

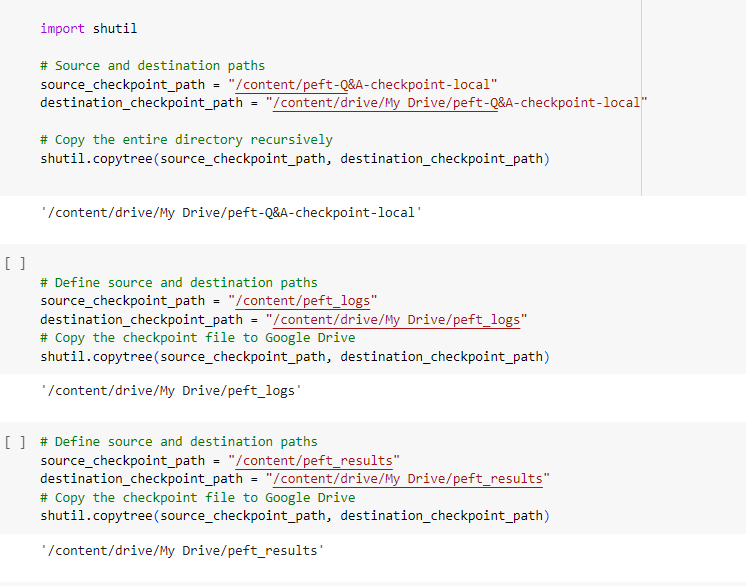
PEFT presents an exciting avenue for persevering to enhance NLP by way of building upon effective pre-skilled fashions. The techniques discussed here provide very good stability in the capacity to specialize models for new tasks at the same time as keeping their widespread capabilities. PEFT approaches make it feasible to deploy effective models even in statistics- or resource-restrained eventualities. Moving ahead, in addition to growing those methods and exploring hybrid combinations holds promise to maximize transfer mastering and fuel persistent development in natural language understanding.

**Setup the PEFT/Lora model for Fine-Tuning**

We have set up the PEFT/Lora model for fine-tuning with a new layer/parameter adapter. Using PEFT/Lora, we are freezing the underlying LLM and only training the adapter. Have a look at the Lora configuration below. Note the rank (r) hyper-parameter, which defines the rank/dimension of the adapter to be trained.

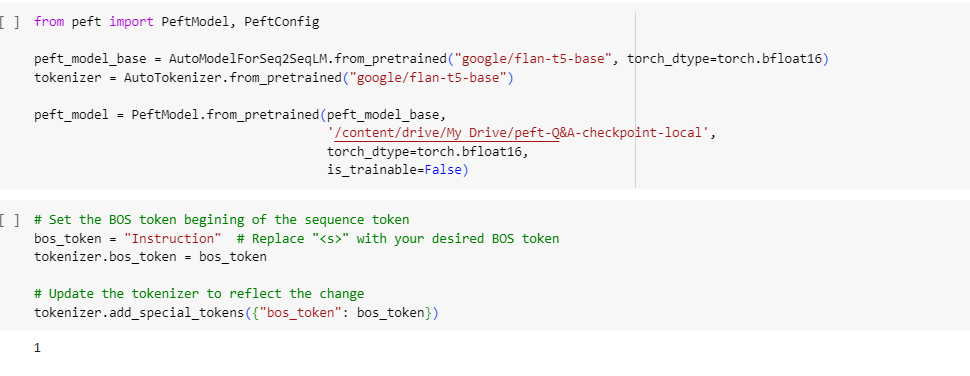


### Train PEFT Adapter



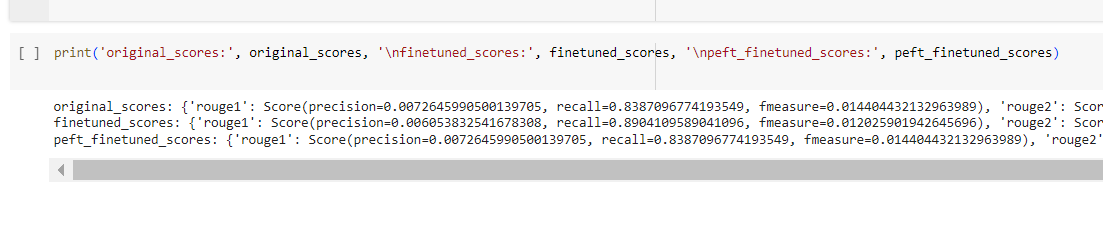
The above code auto created files with logs in Drive for further reference.

Prepare this model by adding an adapter to the original FLAN-T5 model. You are setting is\_trainable=False because the plan is only to perform inference with this PEFT model. If you were preparing the model for further training, you would set is\_trainable=True.



### Finally, we evaluate the model with ROUGE metric.

Below is a picture of final output how the rouge metric compares and gives accuracy of original model, Full fine-tuned model and PEFT fine-tuned model.



**Conclusion**: In this Entire project we have finetuned the original FLAN T5 model on a small subset of the Alpaca dataset. But the maximum efficiency of this can be seen if it's trained on the entire dataset and also here, we used only human evaluation and Rouge metric but can use BLUE metric and so on to get a clear idea of the models performance.

**Resources:**

* Alpaca Dataset: <https://huggingface.co/datasets/Aishwarya30998/updated_alpaca_dataset_3k>
* Transformer Library: <https://github.com/huggingface/transformers>
* Stanford Alpaca: <https://github.com/tatsu-lab/stanford_alpaca>
* Flan T5 Base: google/flan-t5-base.
* <https://stanford-alpaca.gitbook.io/everything-you-need-to-know/stanford-alpaca-guides/understanding-project-stanford-alpaca>
* <https://medium.com/version-1/stanford-alpaca-a-small-yet-mighty-language-model-for-instruction-following-tasks-af9e92e87d9a>