GPU Check

```
!nvidia-smi
```

```
Sat Jan 31 06:34:39 2026
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 580.95.05              Driver Version: 580.95.05      CUDA Version: 13.0     |
+-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce GTX 1650        Off |   00000000:01:00.0  On |                  N/A |
| N/A   30C    P8              3W /   50W |      45MiB /   4096MiB |     24%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                        GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|    0   N/A  N/A            3783      G   /usr/lib/xorg/Xorg                       39MiB |
+-----------------------------------------------------------------------------------------+
```

Imports

```
import os
import zipfile
import evaluate
import torch
import pandas as pd
import urllib.request
from datasets import Dataset
from transformers import pipeline
from transformers import DataCollatorForSeq2Seq
from transformers import TrainingArguments, Trainer
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer

import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /home/siddhu/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

Set Device

```
device = (
    "cuda" if torch.cuda.is_available() else "cpu"
)  # Choose GPU if available, else CPU
print("Device : ", device)  # Print the selected device
```

```
Device :  cuda
```

Load Model & Tokenizer

```
MODEL_NAME = "google/flan-t5-small"  # Specify the pre-trained model name

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)  # Load tokenizer for the model
model = AutoModelForSeq2SeqLM.from_pretrained(MODEL_NAME).to(
    device
)  # Load model and move to device
print(f"Model loaded successfully on {device}")  # Confirm model is loaded
```

```
Model loaded successfully on cuda
```

Test Model on Sample Input

```
article = """
The stock market saw a significant dip on Monday as investors reacted to new inflation data.
Tech stocks were the hardest hit, with major indices dropping by over 2%. Analysts suggest
that while the volatility is concerning, long-term projections remain stable if interest
rates hold steady.
"""  # Sample article to summarize

input_text = "summarize: " + article  # Add prefix to indicate summarization task
input_ids = tokenizer(input_text, return_tensors="pt").input_ids.to(
```

```
    device
)  # Tokenize and move to device

outputs = model.generate(
    input_ids,
    max_length=50,  # Maximum length of generated summary
    min_length=15,  # Minimum length of generated summary
    num_beams=4,  # Beam search for better quality
    no_repeat_ngram_size=3,  # Prevent repeating trigrams
    repetition_penalty=2.5,  # Penalize repetition
    length_penalty=1.0,  # Control length preference
    early_stopping=True,  # Stop when all beams finish
)

print(
    tokenizer.decode(outputs[0], skip_special_tokens=True)
)  # Decode and print summary
```

```
Stock markets saw a significant dip on Monday as investors reacted to new inflation data.
```

Download & Extract Dataset (Data Ingestion)

```
def download_and_extract(url: str, zip_name: str = "summarizer-data.zip", extract_dir: str = "summarizer-data"):

    # Download
    if not os.path.exists(zip_name):
        print("Downloading dataset...")
        urllib.request.urlretrieve(url, zip_name)
    else:
        print("Zip file already exists. Skipping download.")

    # Extract
    if not os.path.exists(extract_dir):
        os.makedirs(extract_dir)

    with zipfile.ZipFile(zip_name, "r") as zip_ref:
        # Get top-level folder(s) in zip
        top_level_dirs = set(f.split("/")[0] for f in zip_ref.namelist() if f.strip())
        # Check if top-level folder(s) exist already
        already_extracted = all(os.path.exists(os.path.join(extract_dir, d)) for d in top_level_dirs)
        if already_extracted:
            print("Dataset already extracted. Skipping unzip.")
        else:
            print("Extracting dataset...")
            # Flatten: remove the top-level folder from the zip paths
            for f in zip_ref.namelist():
                if f.strip():  # skip empty entries
                    path_parts = f.split("/")
                    # Skip the first folder in path
                    target_path = os.path.join(extract_dir, *path_parts[1:])
                    if f.endswith("/"):
                        os.makedirs(target_path, exist_ok=True)
                    else:
                        os.makedirs(os.path.dirname(target_path), exist_ok=True)
                        with open(target_path, "wb") as out_file:
                            out_file.write(zip_ref.read(f))
            print("Extraction complete.")
```

```
download_and_extract(
    url="https://github.com/SiddhuShkya/Text-Summarizer-With-HF/raw/main/data/summarizer-data.zip"
)  # Download and unzip dataset from GitHub
```

```
Zip file already exists. Skipping download.
Extracting dataset...
Extraction complete.
```

Load Data as Dataframe

```
train_df = pd.read_csv("./summarizer-data/train.csv")  # Load training dataset
test_df = pd.read_csv("./summarizer-data/test.csv")  # Load test dataset
val_df = pd.read_csv("./summarizer-data/validation.csv")  # Load validation dataset

print(
    "Features in the dataset: ", train_df.columns.tolist()
)  # Print dataset column names
print("=" * 70)
print("Number of samples in each dataset:")
print("Train data samples: ", len(train_df))  # Print number of training samples
print("Test data samples: ", len(test_df))  # Print number of test samples
print("Validation data samples: ", len(val_df))  # Print number of validation samples
```

```
Features in the dataset:  ['id', 'dialogue', 'summary']
======================================================================
Number of samples in each dataset:
Train data samples:  14731
Test data samples:  819
Validation data samples:  818
```

```
print(train_df["dialogue"][0])  # Print first dialogue/sample from training data
print(
    "\nSummary: ", train_df["summary"][0]
)  # Print corresponding summary of the first sample
```

```
Amanda: I baked  cookies. Do you want some?
Jerry: Sure!
Amanda: I'll bring you tomorrow :-)

Summary:  Amanda baked cookies and will bring Jerry some tomorrow.
```

Tokenize & Prepare Features (Data Transformation)

```
def convert_examples_to_features(example_batch):
    model_inputs = tokenizer(
        example_batch["dialogue"],  # Tokenize input dialogue
        text_target=example_batch["summary"],  # Tokenize target summary for seq2seq
        max_length=1024,  # Max length for input tokens
        truncation=True,  # Truncate if longer than max_length
    )
    labels = tokenizer(
        text_target=example_batch["summary"],
        max_length=128,
        truncation=True,  # Tokenize summary as labels
    )
    model_inputs["labels"] = labels["input_ids"]  # Add tokenized labels to model inputs
    return model_inputs
```

```
# Convert dataframes → Dataset
train_dataset = Dataset.from_pandas(train_df)
test_dataset = Dataset.from_pandas(test_df)
val_dataset = Dataset.from_pandas(val_df)
# Apply the function with .map()
train_dataset = train_dataset.map(convert_examples_to_features, batched=True)
test_dataset = test_dataset.map(convert_examples_to_features, batched=True)
val_dataset = val_dataset.map(convert_examples_to_features, batched=True)
```

```
Map:   0%|          | 0/14731 [00:00<?, ? examples/s]
Map:   0%|          | 0/819 [00:00<?, ? examples/s]
Map:   0%|          | 0/818 [00:00<?, ? examples/s]
```

```
print("Train Dataset:\n", train_dataset)  # Display tokenized training dataset
print("Test Dataset:\n", test_dataset)  # Display tokenized test dataset
print("Val Dataset:\n", val_dataset)  # Display tokenized validation dataset
```

```
Train Dataset:
 Dataset({
    features: ['id', 'dialogue', 'summary', 'input_ids', 'attention_mask', 'labels'],
    num_rows: 14731
})
Test Dataset:
 Dataset({
    features: ['id', 'dialogue', 'summary', 'input_ids', 'attention_mask', 'labels'],
    num_rows: 819
})
Val Dataset:
 Dataset({
    features: ['id', 'dialogue', 'summary', 'input_ids', 'attention_mask', 'labels'],
```

```
    num_rows: 818
})
```

Set Up Training & Trainer

```
data_collator = DataCollatorForSeq2Seq(
    tokenizer=tokenizer,  # Tokenizer to dynamically pad inputs
    model=model,  # Model to handle special tokens for seq2seq
)
```

```
training_args = TrainingArguments(
    output_dir="flan-t5-small-finetuned",  # Directory to save model checkpoints
    num_train_epochs=1,  # Number of training epochs
    warmup_steps=500,  # Steps for learning rate warmup
    per_device_train_batch_size=1,  # Batch size per device during training
    per_device_eval_batch_size=1,  # Batch size per device during evaluation
    weight_decay=0.01,  # L2 regularization
    logging_steps=10,  # Log training metrics every 10 steps
    eval_strategy="steps",  # Evaluate model at specified steps
    eval_steps=500,  # Evaluation interval in steps
    save_steps=100000,  # Save checkpoint every 100k steps
    gradient_accumulation_steps=16,  # Accumulate gradients over multiple steps
)
```

```
trainer = Trainer(
    model=model,  # Model to train
    args=training_args,  # Training arguments
    train_dataset=test_dataset,  # Dataset used for training (here using test dataset)
    eval_dataset=val_dataset,  # Dataset used for evaluation
    data_collator=data_collator,  # Collate batches dynamically
    processing_class=tokenizer,  # Tokenizer for preprocessing inputs
)
```

Finetuning the model

```
trainer.train()  # Start the training process
```


Step	Training Loss	Validation Loss

```
TrainOutput(global_step=52, training_loss=2.004348736542922, metrics={'train_runtime': 52.7321, 'train_samples_per_second':
15.531, 'train steps per second': 0.986, 'total flos': 45322428678144.0, 'train loss': 2.004348736542922, 'epoch': 1.0})
```

Test the fine-tuned Model (Generate Summary)

```
# Move model to evaluation mode
model.eval()               # Set model to evaluation mode (disable dropout, etc.)
model.to(device)           # Move model to the selected device (CPU/GPU)
```

```
T5ForConditionalGeneration(
  (shared): Embedding(32128, 512)
  (encoder): T5Stack(
    (embed_tokens): Embedding(32128, 512)
    (block): ModuleList(
      (0): T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=384, bias=False)
              (k): Linear(in_features=512, out_features=384, bias=False)
              (v): Linear(in_features=512, out_features=384, bias=False)
              (o): Linear(in_features=384, out_features=512, bias=False)
              (relative_attention_bias): Embedding(32, 6)
            )
```

```
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerFF(
            (DenseReluDense): T5DenseGatedActDense(
              (wi_0): Linear(in_features=512, out_features=1024, bias=False)
              (wi_1): Linear(in_features=512, out_features=1024, bias=False)
              (wo): Linear(in_features=1024, out_features=512, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): NewGELUActivation()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
      (1-7): 7 x T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=384, bias=False)
              (k): Linear(in_features=512, out_features=384, bias=False)
              (v): Linear(in_features=512, out_features=384, bias=False)
              (o): Linear(in_features=384, out_features=512, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerFF(
            (DenseReluDense): T5DenseGatedActDense(
              (wi_0): Linear(in_features=512, out_features=1024, bias=False)
              (wi_1): Linear(in_features=512, out_features=1024, bias=False)
              (wo): Linear(in_features=1024, out_features=512, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): NewGELUActivation()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (final_layer_norm): T5LayerNorm()
```

```
# Test text
test_text = """summarize: The quick brown fox jumps over the lazy dog.
            The dog was so lazy that it didn't even bark at the fox.
            This event was caught on camera by a local hiker."""

inputs = tokenizer(
    test_text,
    return_tensors="pt",  # Return PyTorch tensors
    truncation=True,  # Truncate if text exceeds max_length
    padding="max_length",  # Pad to max_length
    max_length=512,  # Maximum input length
).to(device)

# Generate Summary
with torch.no_grad():  # Disable gradient calculation for inference
    outputs = model.generate(
        inputs["input_ids"],  # Input token IDs
        max_length=50,  # Maximum summary length
        num_beams=4,  # Beam search for better quality
        early_stopping=True,  # Stop generation when beam search is complete
    )

print("Summary:", tokenizer.decode(outputs[0], skip_special_tokens=True))
```

```
Summary: The dog was so lazy that it didn't even bark at the fox.
```

Model Evaluation

```
def generate_batch_sized_chunks(list_of_elements, batch_size):
    """Split a list into smaller batches of given size"""
    for i in range(0, len(list_of_elements), batch_size):
        yield list_of_elements[i : i + batch_size]  # Yield each batch
```

```
def calculate_metric_on_test_ds(
    dataset,
    metric,
    model,
    tokenizer,
    batch_size=1,
    device=device,
    column_text="article",
    column_summary="highlights",
```

```
):
    model.eval()  # Set model to evaluation mode

    # Split dataset into batches
    article_batches = list(
        generate_batch_sized_chunks(dataset[column_text], batch_size)
    )
    target_batches = list(
        generate_batch_sized_chunks(dataset[column_summary], batch_size)
    )

    with torch.no_grad():  # Disable gradient calculation
        for article_batch, target_batch in zip(article_batches, target_batches):
            inputs = tokenizer(
                article_batch,
                max_length=256,  # Max input length
                truncation=True,  # Truncate if too long
                padding="max_length",  # Pad to max length
                return_tensors="pt",  # Return PyTorch tensors
            )

            summaries = model.generate(
                input_ids=inputs["input_ids"].to(device),  # Move inputs to device
                attention_mask=inputs["attention_mask"].to(device),
                max_new_tokens=128,  # Max tokens for output
                num_beams=1,  # Beam search width
                do_sample=False,  # Deterministic generation
                use_cache=True,  # Use past key values for speed
            )

            decoded_summaries = tokenizer.batch_decode(
                summaries,
                skip_special_tokens=True,  # Decode outputs to text
            )

            metric.add_batch(
                predictions=decoded_summaries,  # Add generated summaries
                references=target_batch,  # Add reference summaries
            )

    return metric.compute()  # Compute final metric (e.g., ROUGE, BLEU)
```

```
rouge_metric = evaluate.load("rouge")  # Load ROUGE evaluation metric
rouge_names = [
    "rouge1",
    "rouge2",
    "rougeL",
    "rougeLsum",
]  # Specify ROUGE variants to compute
```

```
score = calculate_metric_on_test_ds(
    dataset=test_dataset[0:50],  # Evaluate on first 50 samples of test dataset
    metric=rouge_metric,  # Use the loaded ROUGE metric
    model=trainer.model,  # Model to generate summaries
    tokenizer=tokenizer,  # Tokenizer for preprocessing
    column_text="dialogue",  # Column containing input text
    column_summary="summary",  # Column containing reference summaries
)

# Extract only the specified ROUGE scores
rouge_dict = {name: score[name] for name in rouge_names}

# Convert to DataFrame for display
pd.DataFrame(rouge_dict, index=["flan-t5-small-finetuned"])
```

	rouge1	rouge2	rougeL	rougeLsum
flan-t5-small-finetuned	0.364824	0.116339	0.286061	0.286229

Save the model

```
model.save_pretrained("t5-model")  # Save the fine-tuned model to disk
tokenizer.save_pretrained("t5-tokenizer")  # Save the tokenizer to disk
```

```
('t5-tokenizer/tokenizer_config.json',
 't5-tokenizer/special_tokens_map.json',
 't5-tokenizer/spiece.model',
```

```
 't5-tokenizer/added_tokens.json',
 't5-tokenizer/tokenizer.json')
```

Testing the saved model & tokenizer

```
model_path = "./t5-model"  # Path to the saved fine-tuned model
tokenizer_path = "./t5-tokenizer"  # Path to the saved tokenizer

tokenizer = AutoTokenizer.from_pretrained(tokenizer_path)  # Load the saved tokenizer
model = AutoModelForSeq2SeqLM.from_pretrained(model_path)  # Load the saved model
```

```
gen_kwargs = {
    "max_length": 20,  # Maximum length of generated text
    "min_length": 5,  # Minimum length of generated text
    "length_penalty": 2.0,  # Penalize shorter sequences
    "num_beams": 4,  # Beam search width for better quality
}
```

```
sample_text = train_dataset[0]["dialogue"]  # Take first dialogue from training dataset
reference = train_dataset[0]["summary"]  # Take corresponding reference summary

pipe = pipeline(
    "summarization",  # Use pipeline for text summarization
    model=model,  # Use the loaded/fine-tuned model
    tokenizer=tokenizer,  # Use the corresponding tokenizer
)
```

```
Device set to use cuda:0
```

```
print("Dialogue : \n", sample_text)  # Print the input dialogue
print("\nReference Summary : \n", reference)  # Print the reference summary
print(
    "\nModel Summary : \n", pipe(sample_text, **gen_kwargs)[0]["summary_text"]
)  # Generate and print model summary
```

```
Both `max_new_tokens` (=256) and `max_length`(=20) seem to have been set. `max_new_tokens` will take precedence. Please ref
Dialogue :
 Amanda: I baked  cookies. Do you want some?
Jerry: Sure!
Amanda: I'll bring you tomorrow :-)

Reference Summary :
 Amanda baked cookies and will bring Jerry some tomorrow.

Model Summary :
 Amanda baked cookies. Jerry will bring Amanda tomorrow.
```

```
# 1. Grab your text from the dataset
sample_text = train_dataset[0]["dialogue"]  # Input dialogue from dataset
reference = train_dataset[0]["summary"]  # Reference summary for comparison

# 2. Tokenize the input dialogue
# truncation=True ensures it fits within the model's 1024 token limit
inputs = tokenizer(
    sample_text, truncation=True, padding="longest", return_tensors="pt"
).to(device)

# 3. Generate the summary
# Model outputs token IDs for the summary
summary_ids = model.generate(
    inputs["input_ids"],
    max_length=128,  # Maximum tokens for generated summary
    num_beams=4,  # Beam search width for better quality
    length_penalty=2.0,  # Favor longer sequences
    early_stopping=True,  # Stop generation when beams finish
)

# 4. Decode the IDs back into a string
decoded_summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)

# 5. Compare the results
print("--- DIALOGUE ---")
print(sample_text)  # Print original dialogue
print("\n--- REFERENCE SUMMARY (Ground Truth) ---")
print(reference)  # Print reference summary
print("\n--- MODEL GENERATED SUMMARY ---")
print(decoded_summary)  # Print model-generated summary
```

```
--- DIALOGUE ---
Amanda: I baked  cookies. Do you want some?
Jerry: Sure!
```

```
Amanda: I'll bring you tomorrow :-)

--- REFERENCE SUMMARY (Ground Truth) ---
Amanda baked cookies and will bring Jerry some tomorrow.

--- MODEL GENERATED SUMMARY ---
Amanda baked cookies. Jerry will bring Amanda tomorrow.
```