Fine-tuning Llama 3 for Tamil-English Sentiment Analysis using PEFT

Team Members:

- **Rishpraveen. S - 21MID0151**
- **Shyam. V - 21MID0232**
- **Sitharth. B- 21MID0196**

Submitted to:

Prof. Geraldine Bessie Amali, SCOPE

School of Computer Science and Engineering
Vellore Institute of Technology

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Contents

Abstract

Sentiment analysis is essential in the interpretation of opinions expressed in text, especially in the increasing field of code-mixed languages such as Tamil-English being used extensively across social media platforms. This research sought to capitalize on the functionality of Large Language Models (LLMs), focusing on Meta's Llama 3 series, for this purpose. The main goal was to optimize a Llama 3 model (first trying out 8B and then 3.2 3B models) on the DravidianCodeMix dataset via Parameter-Efficient Fine-Tuning (PEFT), i.e., Quantized Low-Rank Adaptation (QLoRA), to make it possible on low-resource hardware such as Google Colab's T4 GPU. The notebook defines steps for setup, environment configuration, experimenting with zero-shot prompting, and setting up a supervised fine-tuning pipeline using the transformers and trl libraries. Nevertheless, the execution logs reveal that there were huge challenges faced, and these were mainly to do with accessing and loading the gated Llama 3 models and the target dataset from the Hugging Face Hub, which made it impossible to successfully carry out the fine-tuning process as described in the notebook.

1. Introduction

The advent of social media and web-based platforms has created enormous amounts of text data with rich user opinions and sentiments. Automatic sentiment analysis of this sentiment is crucial for businesses, social research, and public discourse analysis. Although great strides have been made in sentiment analysis for high-resource languages such as English, low-resource and code-mixed languages like Tamil-English (Tanglish) pose specific challenges because of grammatical differences, phonetic typing, and script mixing.

Recent Large Language Model (LLM) breakthroughs such as Meta's Llama family provide strong natural language understanding capabilities. General-purpose models such as these tend to be prone to needing to be adapted for a particular downstream task and domain. Complete fine-tuning of these multi-billion parameter models is too computationally intensive and resource-hungry for hardware.

Parameter-Efficient Fine-Tuning (PEFT) approaches, especially Low-Rank Adaptation (LoRA), have proven to be successful ways to fine-tune LLMs using far fewer trainable parameters. Quantized LoRA (QLoRA) continues to decrease the amount of memory needed by fine-tuning a quantized (e.g., 4-bit) representation of the base model.

This project investigates the use of QLoRA to fine-tune a Llama 3 Instruct model for sentiment classification in the DravidianCodeMix dataset, with a focus on Tamil-English language. The intention was to create a model that can effectively classify sentiment as Positive, Negative, or Neutral within this particular linguistic environment and remain tractable under the limitations of standard research/education hardware (e.g., Google Colab GPU). The notebook captures the configuration, setup, data processing efforts, and the desired training pipeline on Hugging Face libraries.

2. Hardware / Software Requirements

Hardware:

- GPU: NVIDIA GPU needed for practical training. The notebook was executed in the environment of an NVIDIA T4 GPU (15GB VRAM). GPUs with more VRAM (e.g., A100) are suitable for smoother training of bigger models.
- CPU: Typical multi-core CPU.
- RAM: Adequate system RAM (at least 16GB, 32GB+ preferred).
- Storage: Local storage or mounted drive such as Google Drive for dataset, libraries, and model checkpoints.

Software:

- Operating System: Linux (as used in Google Colab).
- Python: Python 3.x (e.g., 3.10+).
- Core Libraries (Installed via pip):
    - torch: Deep learning framework.
    - transformers: Hugging Face library for models and tokenizers.
    - datasets: Hugging Face library for dataset loading and processing.
    - peft: Hugging Face library for Parameter-Efficient Fine-Tuning (LoRA).

    - trl: Hugging Face library for transformer reinforcement learning and supervised fine-tuning (SFTTrainer).
    - accelerate: Hugging Face library for distributed training and hardware acceleration.
    - bitsandbytes: Library for quantization (required for QLoRA).
    - evaluate: Hugging Face library for model evaluation metrics.
    - numpy: Numerical library.
    - huggingface_hub: For interacting with the Hugging Face Hub (login, download, upload).
    - fsspec: Filesystem specification library (dependency, potentially upgraded).
    - llama-stack: Explored for model download/interaction (though not used in the final intended fine-tuning pipeline).
- Environment: Jupyter Notebook / Google Colab.
- Account: Hugging Face Hub account with accepted terms for the Llama 3 model and a valid access token (with write permissions if pushing adapters).
- Git: (Implied) For potentially storing the token as a credential.

3. Existing System/Approach/Method (Zero-Shot Prompting Attempt)

1. Before trying fine-tuning, the notebook investigated a zero-shot prompting strategy. This entails applying the pre-trained Llama 3 Instruct model as is with no task-specific training on the target dataset. The approach depends on the construction of a comprehensive prompt that directs the model towards the intended task. The notebook creates functions (classify_sentiment_llama3, classify_sentiment_llama3_enhanced_prompt) that format the input as a dialogue:
2. **System Prompt:** Defines the context, informing the model its function (sentiment analyzer for Tamil/Code-mix), the expected output categories (Positive, Negative, Neutral), and the expected output format (single word). The improved prompt included more specific directions regarding sensitivity to negative language.
3. **User Input:** Gives the original Tamil or code-mixed text that needs to be examined.
4. **Assistant Prompt (Initiation):** The prompt concludes anticipating the model to output the sentiment label.

This method utilizes the knowledge and instruction-following abilities of the model from its pre-training process. It does not need any new training data or parameter updates specifically for the sentiment task. The notebook tried to apply the transformers pipeline for generating text according to these prompts.

3.1 Drawback/Limitations of Existing Approach

- **Performance Variation:** Zero-shot performance may be volatile and even poorer than fine-tuned models, particularly on specific tasks or domains such as code-mixed sentiment analysis where there could be oversight of subtle differences by the universal pre-trained model.
- **Prompt Sensitivity**: Output quality greatly relies on prompt clarity and design efficacy. Even subtle changes in word choice can cause drastic effects. The notebook reflects evidence of refinement of prompts (enhanced_prompt).
- **Adherence to Output Format**: LLMs may not always strictly follow the specified output format (e.g., giving explanations rather than only the label), and careful parsing of the response may be necessary.
- **Execution Failures**: Most importantly, the notebook demonstrates that the efforts to set up and test this zero-shot method were unsuccessful. There were failures in:
    - Loading the necessary model (meta-llama/Llama-3.2-3B-Instruct) using transformers, which resulted in OSError: 401 Unauthorized.
    - Setting up the pipeline because the model is not loaded.
    - Misusing shell commands (!) to invoke Python functions/pipelines and causing syntax and NameError problems.
    - NameError when attempting to use the tokenizer inside the classification function due to possibly not having been successfully imported previously.
    - Owing to these mistakes, it was impossible to test the effectiveness of the zero-shot strategy inside the context of the notebook's execution.

4. Proposed/Developed Model (Fine-Tuning with QLoRA)

Given the limitations and execution failures of the zero-shot approach, the primary proposed method documented in the latter part of the notebook is Supervised Fine-Tuning (SFT) using Quantized Low-Rank Adaptation (QLoRA).

4.1 Design

- **Base Model**: meta-llama/Llama-3.1-8B-Instruct (as noted in the fine-tuning configuration cell, despite previous experimentation with other variants).
- **Task:** Multi-class sentiment classification (Positive, Negative, Neutral).
- **Dataset**: DravidianCodeMix/DravidianCodeMix-Dataset from Hugging Face Hub.
- **Fine-Tuning Technique**: QLoRA (4-bit NormalFloat quantization using bitsandbytes, LoRA adapters using peft).
- **Training Framework:** trl library's SFTTrainer, which is specifically intended for fine-tuning models on instruction/chat-formatted data.
- Input/Output Format: Chat template format appropriate for Llama 3 Instruct models, where the input is included in the user message and the target sentiment label is the anticipated assistant response.

4.2 Module Wise Description

1. **Data Loading & Preprocessing:**
    - Loads the DravidianCodeMix/DravidianCodeMix-Dataset using the datasets library.
    - Defines a mapping (target_label_map) to convert the original dataset labels (e.g., "Positive", "Negative", "unknown_state", "Mixed_feelings", "not-Tamil") into the desired three target labels ("Positive", "Negative", "Neutral"). Labels not relevant to the task (like "not-Tamil") are mapped to None.
    - Applies this mapping using .map() to create a final_label column.
    - Filters the dataset using .filter() to remove examples where final_label is None.
    - Prepares train and validation splits, potentially creating the validation split from the training data if not originally present.

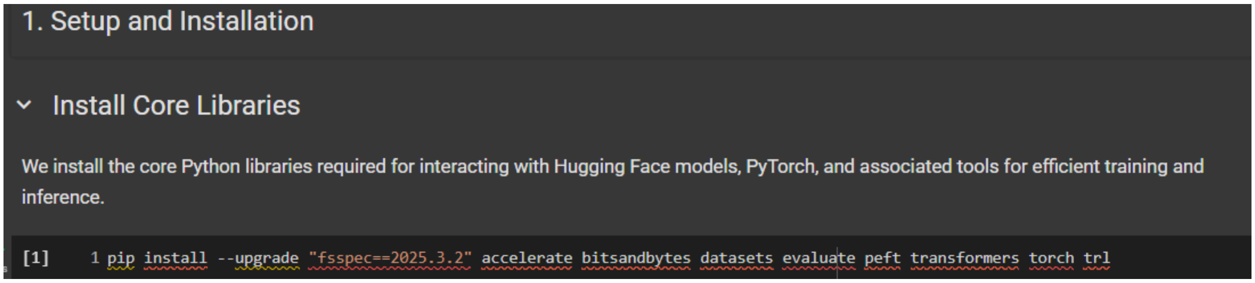
2. **Model & Tokenizer Loading:**

- Loads the AutoTokenizer for the base_model_id. Sets the padding token to the EOS token if missing and configures right-side padding.
- Loads the AutoModelForCausalLM for the base_model_id.
- Applies 4-bit quantization during loading using BitsAndBytesConfig (load_in_4bit=True, nf4, bfloat16 compute type).
- Uses device_map="auto" to distribute the large model across available hardware (GPU).

3. **PEFT (LoRA) Setup:**
    - Prepares the quantized model for training using prepare_model_for_kbit_training.
    - Defines a LoraConfig specifying the LoRA rank (r=16), alpha (lora_alpha=32), dropout (lora_dropout=0.05), and target modules (attention projections, feed-forward layers). Sets task_type="CAUSAL_LM".
    - Applies the LoRA configuration to the base model using get_peft_model, making only the adapter weights trainable.
4. **Training Pipeline (SFTTrainer):**
    - Defines TrainingArguments specifying hyperparameters like output directory, batch size, gradient accumulation, learning rate, number of epochs, optimization (paged_adamw_32bit), logging, saving strategy, and evaluation strategy. Enables mixed-precision (bf16/tf32 if available) and gradient checkpointing for memory efficiency. Includes configuration for pushing the trained adapter to Hugging Face Hub.
    - Defines a format_chat_prompt function that takes a dataset example and structures the text and target label into the Llama 3 chat message format.
    - Initializes the SFTTrainer with the PEFT-enabled model, training arguments, datasets, tokenizer, PEFT config, and the format_chat_prompt function.
    - Calls trainer.train() to start the fine-tuning process.
    - Calls trainer.save_model() to save the trained LoRA adapter locally.
    - (Optional) Calls trainer.push_to_hub() to upload the adapter.

4.3 Implementation

Step 1:



1. Setup and Installation

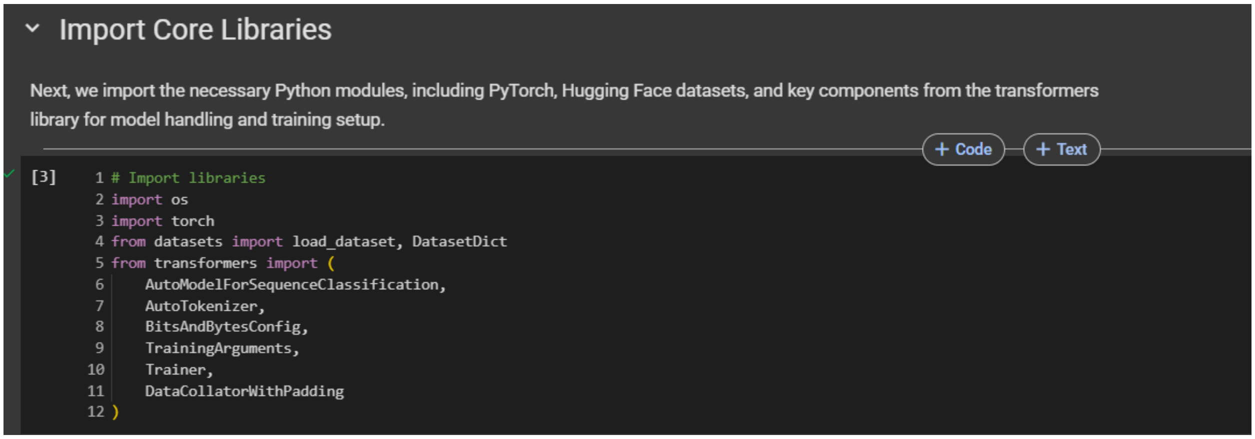
Install Core Libraries

We install the core Python libraries required for interacting with Hugging Face models, PyTorch, and associated tools for efficient training and inference.

```
[1]  pip install --upgrade "fsspec==2025.3.2" accelerate bitsandbytes datasets evaluate peft transformers torch trl
```

This cell executes a pip install command to install or upgrade essential Python libraries. Key packages include transformers, torch, datasets, peft, trl, accelerate, and bitsandbytes, which are fundamental for working with Hugging Face models, PyTorch, efficient model loading/training (PEFT, LoRA, quantization), and handling datasets. It also specifies a version for fsspec. The output confirms the installation or indicates that requirements are already satisfied.

Step 2:



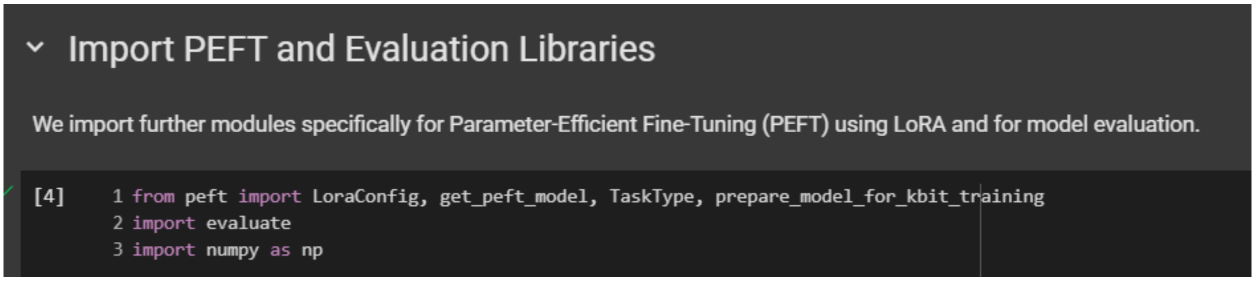
Import Core Libraries

Next, we import the necessary Python modules, including PyTorch, Hugging Face datasets, and key components from the transformers library for model handling and training setup.

```
[3]  # Import libraries
     import os
     import torch
     from datasets import load_dataset, DatasetDict
     from transformers import (
         AutoModelForSequenceClassification,
         AutoTokenizer,
         BitsAndBytesConfig,
         TrainingArguments,
         Trainer,
         DataCollatorWithPadding
     )
```

This cell imports fundamental libraries required for the subsequent code. It includes os for operating system interactions, torch for the deep learning framework, datasets for loading data, and specific components from transformers (like AutoModelForSequenceClassification, AutoTokenizer, BitsAndBytesConfig, TrainingArguments, Trainer, DataCollatorWithPadding) for model loading, tokenization, quantization configuration, and setting up the training process.

Step 3:



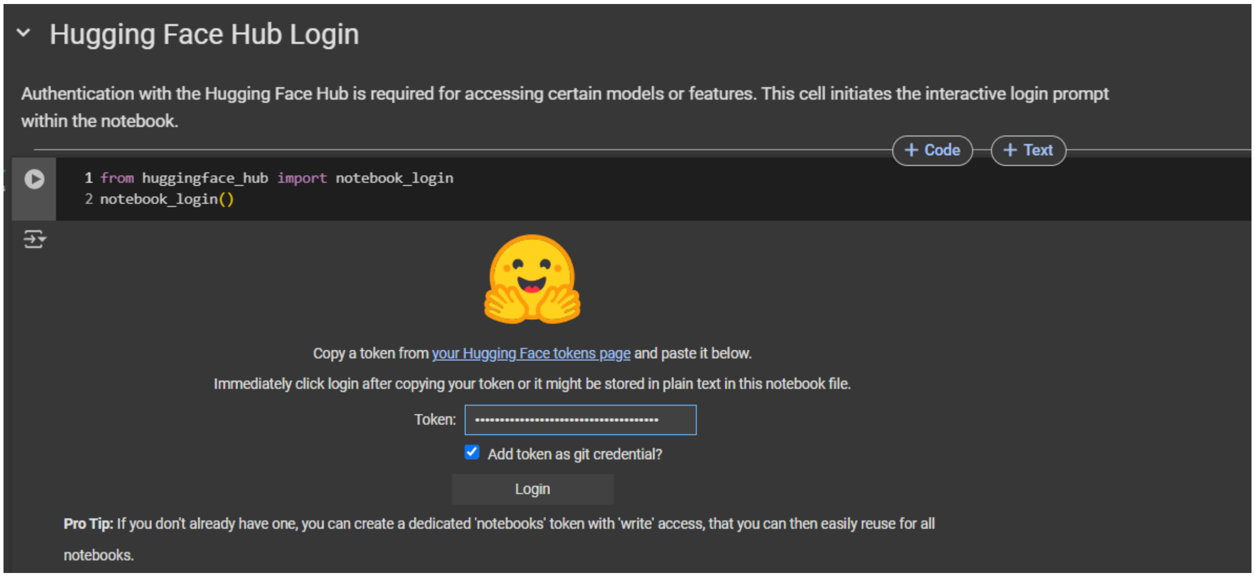
Import PEFT and Evaluation Libraries

We import further modules specifically for Parameter-Efficient Fine-Tuning (PEFT) using LoRA and for model evaluation.

```
[4]  from peft import LoraConfig, get_peft_model, TaskType, prepare_model_for_kbit_training
     import evaluate
     import numpy as np
```

This cell imports additional libraries crucial for Parameter-Efficient Fine-Tuning (PEFT) and evaluation. LoraConfig, get_peft_model, TaskType, and prepare_model_for_kbit_training from peft are used for setting up and applying LoRA. evaluate is likely intended for assessing model performance, and numpy provides numerical operations.

Step 4:



Hugging Face Hub Login

Authentication with the Hugging Face Hub is required for accessing certain models or features. This cell initiates the interactive login prompt within the notebook.

```
from huggingface_hub import notebook_login
notebook_login()
```

Copy a token from your Hugging Face tokens page and paste it below.

Immediately click login after copying your token or it might be stored in plain text in this notebook file.

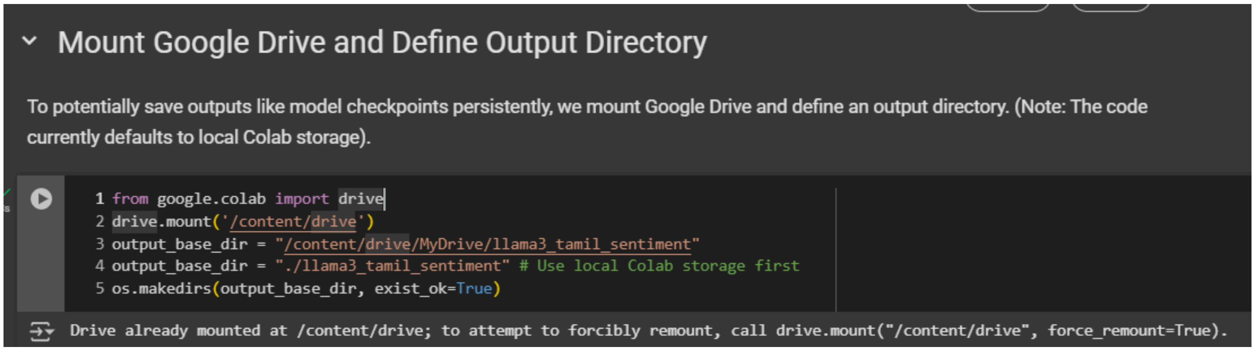
Token: ••••••••••••••••••••••••••••••••••••••

☑ Add token as git credential?

Pro Tip: If you don't already have one, you can create a dedicated 'notebooks' token with 'write' access, that you can then easily reuse for all notebooks.

This cell uses notebook_login from huggingface_hub to initiate an interactive logon process within the notebook environment. This displays a widget prompting the user for their Hugging Face token, allowing authenticated access to the Hub for downloading models or uploading results.

Step 5:



Mount Google Drive and Define Output Directory

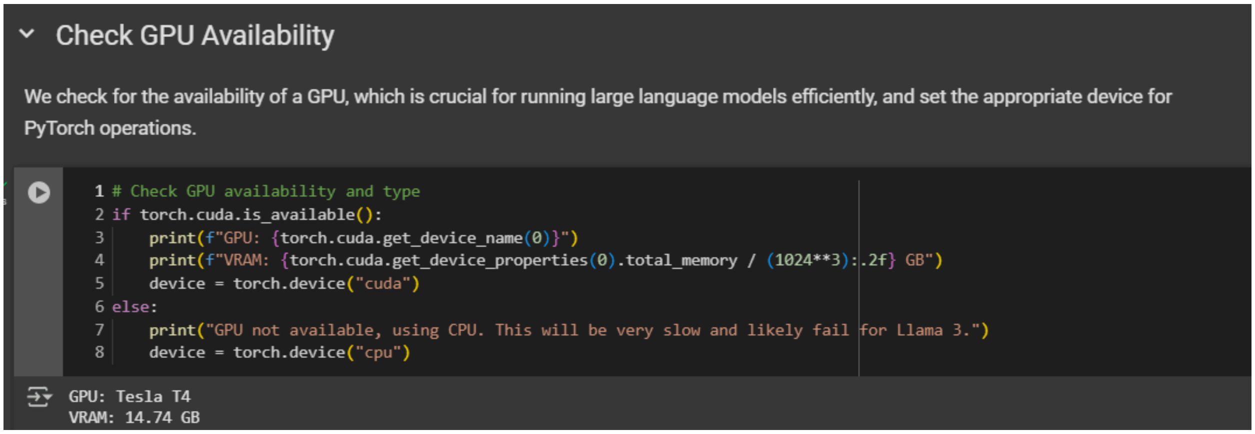
To potentially save outputs like model checkpoints persistently, we mount Google Drive and define an output directory. (Note: The code currently defaults to local Colab storage).

```
from google.colab import drive
drive.mount('/content/drive')
output_base_dir = "/content/drive/MyDrive/llama3_tamil_sentiment"
output_base_dir = "./llama3_tamil_sentiment" # Use local Colab storage first
os.makedirs(output_base_dir, exist_ok=True)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

This cell attempts to mount the user's Google Drive to the Colab environment using google.colab.drive.mount. It then defines a base directory path (output_base_dir), initially pointing to Google Drive but subsequently overridden to use local Colab storage (./llama3_tamil_sentiment). Finally, it ensures this directory exists using os.makedirs. This setup allows for saving model checkpoints or results, though currently configured for temporary local storage.

Step 6:



Check GPU Availability

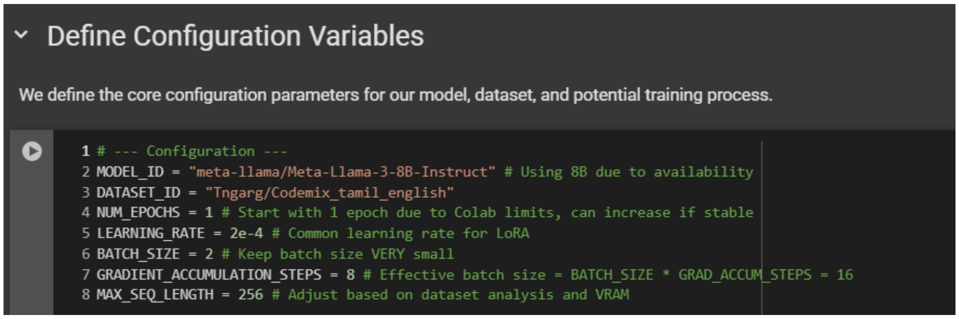
We check for the availability of a GPU, which is crucial for running large language models efficiently, and set the appropriate device for PyTorch operations.

```python
# Check GPU availability and type
if torch.cuda.is_available():
    print(f"GPU: {torch.cuda.get_device_name(0)}")
    print(f"VRAM: {torch.cuda.get_device_properties(0).total_memory / (1024**3):.2f} GB")
    device = torch.device("cuda")
else:
    print("GPU not available, using CPU. This will be very slow and likely fail for Llama 3.")
    device = torch.device("cpu")
```

```
GPU: Tesla T4
VRAM: 14.74 GB
```

This cell checks if a CUDA-enabled GPU is available using torch.cuda.is_available(). If a GPU is found, it prints the GPU model name (e.g., "Tesla T4") and its total VRAM. It then sets the device variable for PyTorch operations to "cuda". If no GPU is available, it prints a warning and sets the device to "cpu", noting that this configuration is likely too slow and memory-intensive for Llama 3.

Step 7:



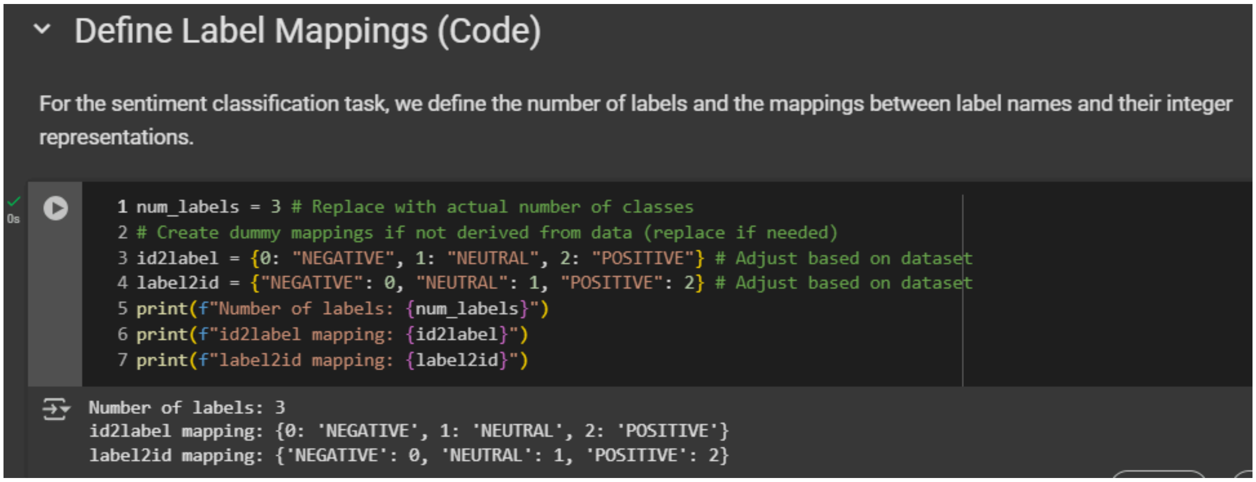
Define Configuration Variables

We define the core configuration parameters for our model, dataset, and potential training process.

```python
# --- Configuration ---
MODEL_ID = "meta-llama/Meta-Llama-3-8B-Instruct" # Using 8B due to availability
DATASET_ID = "Tngarg/Codemix_tamil_english"
NUM_EPOCHS = 1 # Start with 1 epoch due to Colab limits, can increase if stable
LEARNING_RATE = 2e-4 # Common learning rate for LoRA
BATCH_SIZE = 2 # Keep batch size VERY small
GRADIENT_ACCUMULATION_STEPS = 8 # Effective batch size = BATCH_SIZE * GRAD_ACCUM_STEPS = 16
MAX_SEQ_LENGTH = 256 # Adjust based on dataset analysis and VRAM
```

This cell sets up crucial configuration variables for the experiment. MODEL_ID specifies the base Llama 3 model (meta-llama/Meta-Llama-3-8B-Instruct). DATASET_ID points to the target Hugging Face dataset (Tngarg/Codemix_tamil_english). Training hyperparameters like NUM_EPOCHS, LEARNING_RATE, BATCH_SIZE, GRADIENT_ACCUMULATION_STEPS (to simulate a larger effective batch size), and MAX_SEQ_LENGTH are defined, likely anticipating a fine-tuning step and considering Colab's resource limitations.

Step 8:



Define Label Mappings (Code)

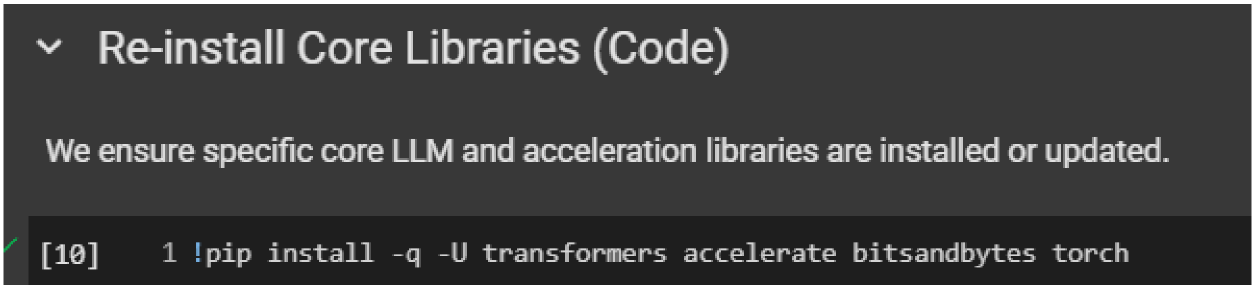
For the sentiment classification task, we define the number of labels and the mappings between label names and their integer representations.

```python
num_labels = 3 # Replace with actual number of classes
# Create dummy mappings if not derived from data (replace if needed)
id2label = {0: "NEGATIVE", 1: "NEUTRAL", 2: "POSITIVE"} # Adjust based on dataset
label2id = {"NEGATIVE": 0, "NEUTRAL": 1, "POSITIVE": 2} # Adjust based on dataset
print(f"Number of labels: {num_labels}")
print(f"id2label mapping: {id2label}")
print(f"label2id mapping: {label2id}")
```

```
Number of labels: 3
id2label mapping: {0: 'NEGATIVE', 1: 'NEUTRAL', 2: 'POSITIVE'}
label2id mapping: {'NEGATIVE': 0, 'NEUTRAL': 1, 'POSITIVE': 2}
```

This cell explicitly defines the number of classification labels (num_labels = 3) and creates dictionaries for mapping between these numerical labels (0, 1, 2) and their corresponding sentiment names (**"NEGATIVE", "NEUTRAL", "POSITIVE"**). These mappings (id2label, label2id) are essential for interpreting model outputs during classification and preparing data if fine-tuning were performed.

Step 9:



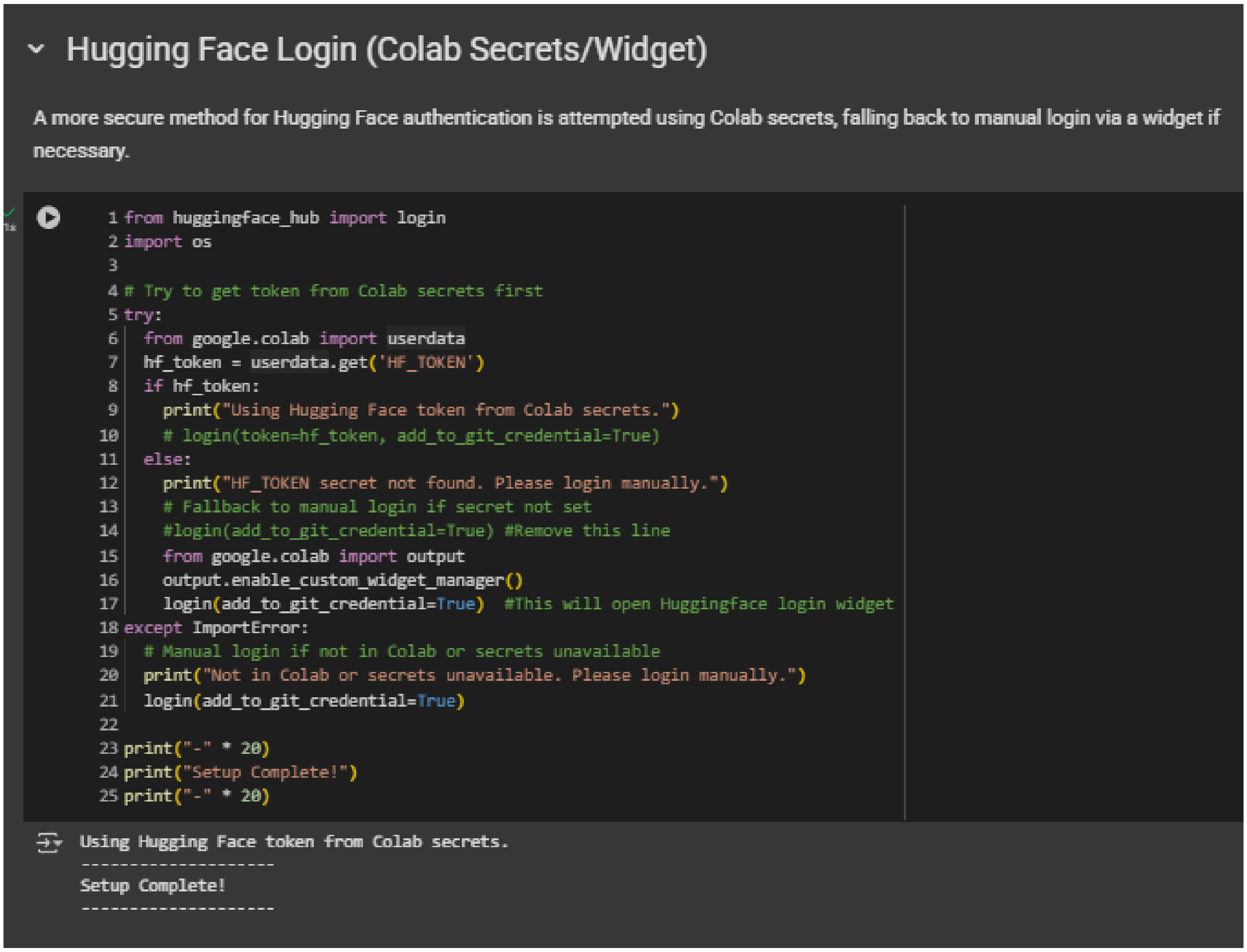
Re-install Core Libraries (Code)

We ensure specific core LLM and acceleration libraries are installed or updated.

```
!pip install -q -U transformers accelerate bitsandbytes torch
```

This cell runs another pip install command, specifically targeting transformers, accelerate, bitsandbytes, and torch with the -q (quiet) and -U (upgrade) flags. While potentially redundant given Cell 3, it forcefully ensures the latest compatible versions of these specific core components are present, possibly to resolve version conflicts or guarantee access to recent features needed later.

Step 10:



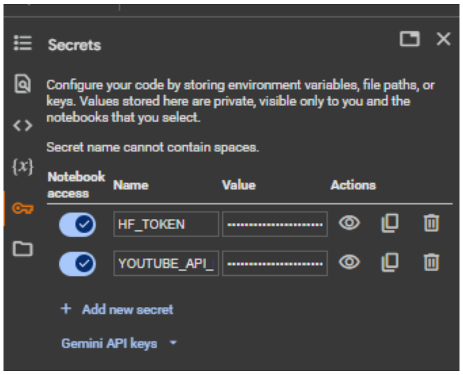
Hugging Face Login (Colab Secrets/Widget)

A more secure method for Hugging Face authentication is attempted using Colab secrets, falling back to manual login via a widget if necessary.

```python
from huggingface_hub import login
import os

# Try to get token from Colab secrets first
try:
  from google.colab import userdata
  hf_token = userdata.get('HF_TOKEN')
  if hf_token:
    print("Using Hugging Face token from Colab secrets.")
    # login(token=hf_token, add_to_git_credential=True)
  else:
    print("HF_TOKEN secret not found. Please login manually.")
    # Fallback to manual login if secret not set
    #login(add_to_git_credential=True) #Remove this line
    from google.colab import output
    output.enable_custom_widget_manager()
    login(add_to_git_credential=True)  #This will open Huggingface login widget
except ImportError:
  # Manual login if not in Colab or secrets unavailable
  print("Not in Colab or secrets unavailable. Please login manually.")
  login(add_to_git_credential=True)

print("-" * 20)
print("Setup Complete!")
print("-" * 20)
```

```
Using Hugging Face token from Colab secrets.
--------------------
Setup Complete!
--------------------
```

This cell implements a more robust approach to Hugging Face login within Colab. It first tries to retrieve the token from Colab's secrets manager (userdata.get('HF_TOKEN')). If successful, it uses the token silently. If the secret is not found, it enables Colab's custom widget manager and calls huggingface_hub.login to display an interactive login prompt. If running outside Colab (where google.colab is unavailable), it also defaults to the interactive login. This prioritizes secure token handling.



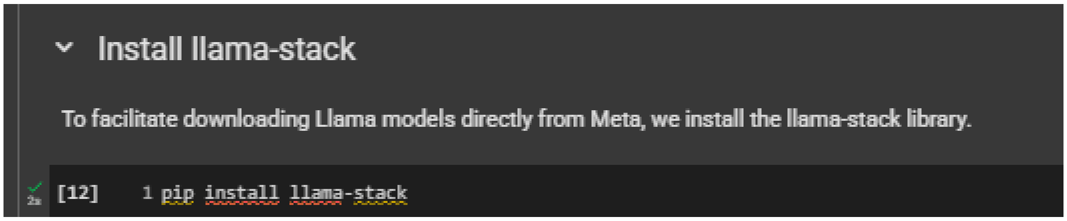
Secrets

Configure your code by storing environment variables, file paths, or keys. Values stored here are private, visible only to you and the notebooks that you select.

Secret name cannot contain spaces.

Notebook access	Name	Value	Actions
☑	HF_TOKEN	••••••••••••••••••••	
☑	YOUTUBE_API_	••••••••••••••••••••	

Step 11:



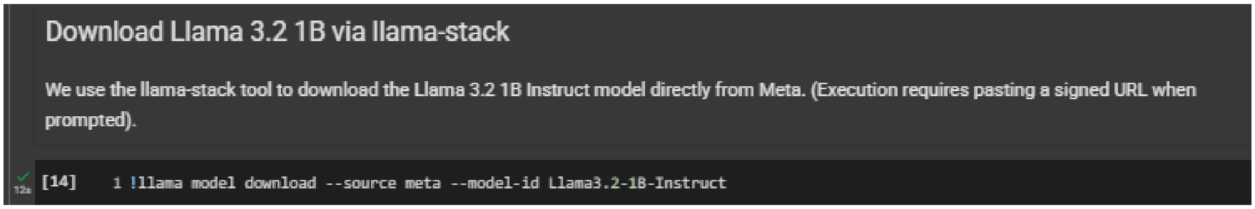
Install llama-stack

To facilitate downloading Llama models directly from Meta, we install the llama-stack library.

```
[12]    pip install llama-stack
```

This cell installs the llama-stack Python package using pip. This library provides tools and command-line utilities specifically designed by Meta for downloading and interacting with their official Llama model releases, offering an alternative to Hugging Face Hub downloads for these specific models.

Step 12:



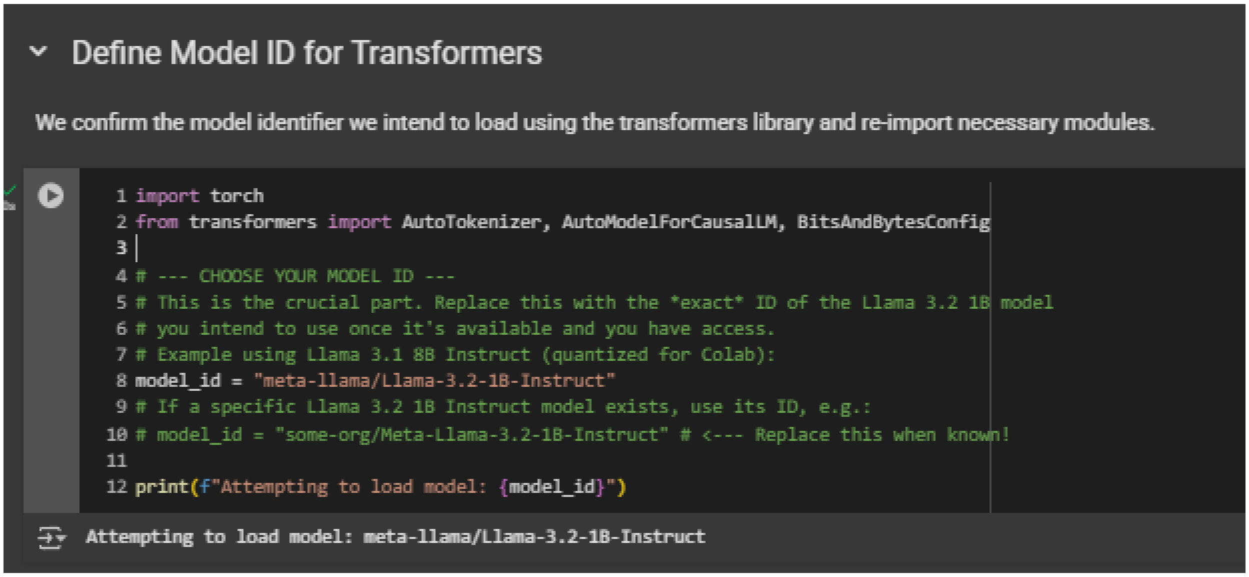
Download Llama 3.2 1B via llama-stack

We use the llama-stack tool to download the Llama 3.2 1B Instruct model directly from Meta. (Execution requires pasting a signed URL when prompted).

```
[14]    !llama model download --source meta --model-id Llama3.2-1B-Instruct
```

This cell executes the command! llama model download --source meta --model-id Llama3.2-1B-Instruct. It employs the llama-stack CLI to initiate the download of the specified model weights and associated files (tokenizer, config) directly from Meta's servers. The process requires the user to paste a valid, time-limited signed URL (obtained from Meta's Llama download portal) when prompted for authentication. The downloaded files are typically stored in the ~/.llama/checkpoints/ directory.

Step 13:



Define Model ID for Transformers

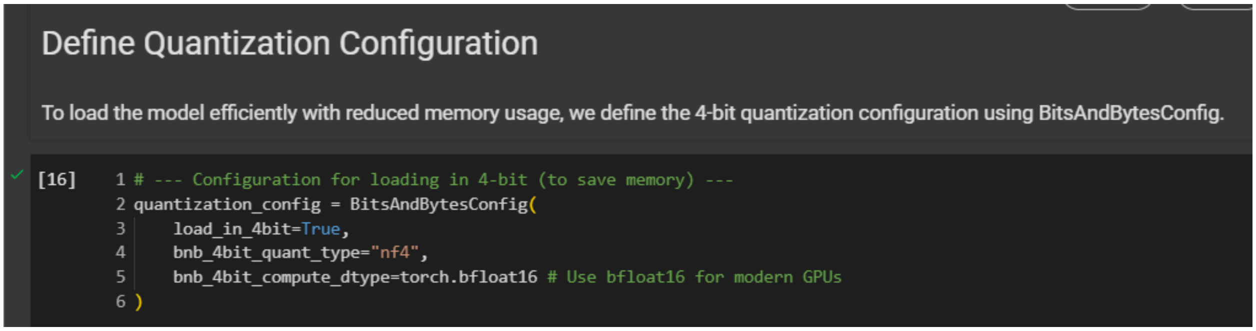
We confirm the model identifier we intend to load using the transformers library and re-import necessary modules.

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig

# --- CHOOSE YOUR MODEL ID ---
# This is the crucial part. Replace this with the *exact* ID of the Llama 3.2 1B model
# you intend to use once it's available and you have access.
# Example using Llama 3.1 8B Instruct (quantized for Colab):
model_id = "meta-llama/Llama-3.2-1B-Instruct"
# If a specific Llama 3.2 1B Instruct model exists, use its ID, e.g.:
# model_id = "some-org/Meta-Llama-3.2-1B-Instruct" # <--- Replace this when known!

print(f"Attempting to load model: {model_id}")
```

```
Attempting to load model: meta-llama/Llama-3.2-1B-Instruct
```

This cell re-imports torch and key components from transformers (AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig). It explicitly defines the model_id variable as "meta-llama/Llama-3.2-1B-Instruct". This sets the identifier that the transformers library will use to find and load the model, potentially from the Hugging Face Hub or a local cache if downloaded previously (e.g., by huggingface-cli download).

Step 14:



Define Quantization Configuration

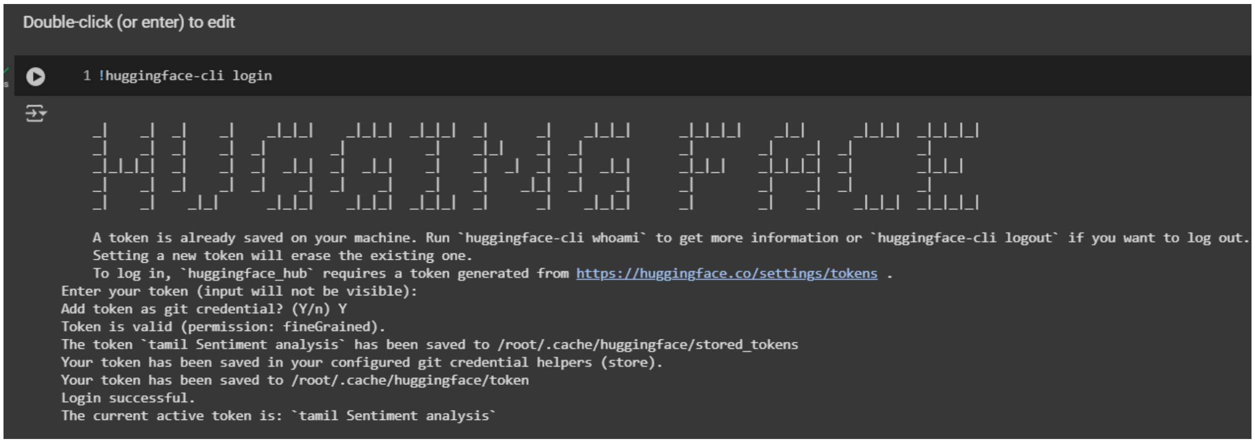
To load the model efficiently with reduced memory usage, we define the 4-bit quantization configuration using BitsAndBytesConfig.

[16]

```python
# --- Configuration for loading in 4-bit (to save memory) ---
quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16 # Use bfloat16 for modern GPUs
)
```

This cell configures the quantization settings using BitsAndBytesConfig from the transformers library (which interfaces with bitsandbytes). It sets load_in_4bit=True to enable 4-bit loading, specifies bnb_4bit_quant_type="nf4" (NormalFloat 4-bit quantization), and sets the computation data type to torch.bfloat16 (bnb_4bit_compute_dtype). This configuration drastically reduces the GPU VRAM required to load the model, making it feasible on hardware like the Colab T4 GPU.

Step 15:



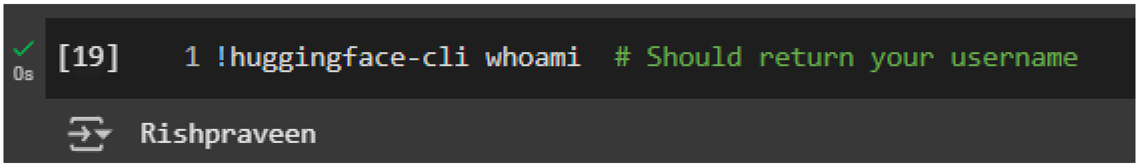
Double-click (or enter) to edit

```
!huggingface-cli login
```

```
    _|    _|  _|    _|    _|_|_|    _|_|_|  _|_|_|  _|      _|    _|_|_|      _|_|_|_|    _|_|      _|_|_|  _|_|_|_|
    _|    _|  _|    _|  _|        _|          _|    _|_|    _|  _|            _|        _|    _|  _|        _|
    _|_|_|_|  _|    _|  _|  _|_|  _|  _|_|    _|    _|  _|  _|  _|  _|_|      _|_|_|    _|_|_|_|  _|        _|_|_|
    _|    _|  _|    _|  _|    _|  _|    _|    _|    _|    _|_|  _|    _|      _|        _|    _|  _|        _|
    _|    _|    _|_|      _|_|_|    _|_|_|  _|_|_|  _|      _|    _|_|_|      _|        _|    _|    _|_|_|  _|_|_|_|

    A token is already saved on your machine. Run `huggingface-cli whoami` to get more information or `huggingface-cli logout` if you want to log out.
    Setting a new token will erase the existing one.
    To log in, `huggingface_hub` requires a token generated from https://huggingface.co/settings/tokens .
Enter your token (input will not be visible):
Add token as git credential? (Y/n) Y
Token is valid (permission: fineGrained).
The token `tamil Sentiment analysis` has been saved to /root/.cache/huggingface/stored_tokens
Your token has been saved in your configured git credential helpers (store).
Your token has been saved to /root/.cache/huggingface/token
Login successful.
The current active token is: `tamil Sentiment analysis`
```

This cell executes the command !huggingface-cli login. This invokes the command-line tool provided by the huggingface_hub library to handle authentication. It typically prompts the user to enter their Hugging Face token and asks if they want to save it as a git credential, facilitating authenticated operations from the command line within the Colab environment.

Step 16: verify login



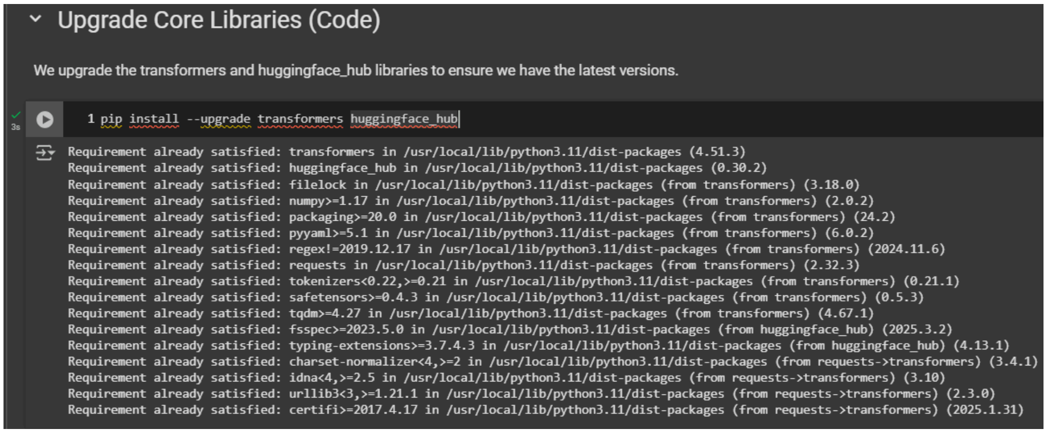
[19]

```
!huggingface-cli whoami  # Should return your username
```

```
Rishpraveen
```

This cell runs !huggingface-cli whoami. This command queries the Hugging Face CLI to determine the currently logged-in user based on the stored credentials. The output (e.g., the username "Rishpraveen") confirms that the CLI authentication process was successful.

Step 17:



Upgrade Core Libraries (Code)

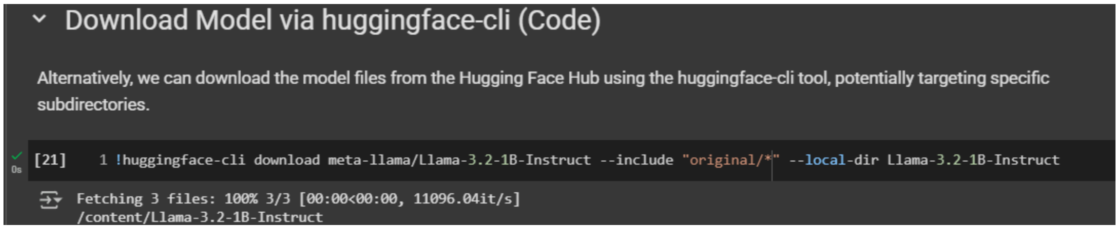
We upgrade the transformers and huggingface_hub libraries to ensure we have the latest versions.

```
pip install --upgrade transformers huggingface_hub
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.51.3)
Requirement already satisfied: huggingface_hub in /usr/local/lib/python3.11/dist-packages (0.30.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.18.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface_hub) (2025.3.2)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface_hub) (4.13.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2025.1.31)
```

This cell executes pip install --upgrade transformers huggingface_hub. It explicitly targets the transformers and huggingface_hub libraries for an upgrade, ensuring that the latest available versions from the Python Package Index (PyPI) are installed. This is often done to leverage new features, bug fixes, or model support.

Step 18:



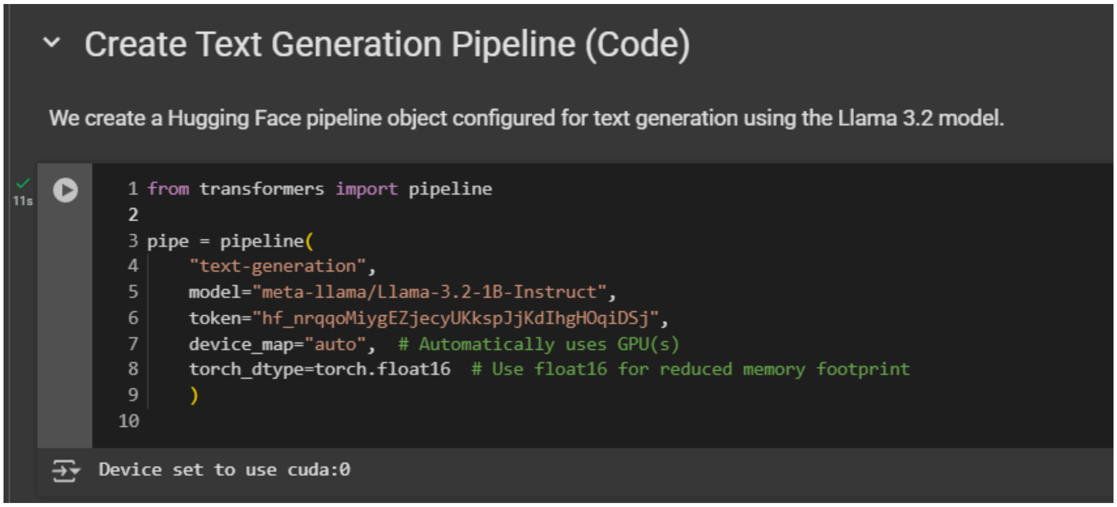
Download Model via huggingface-cli (Code)

Alternatively, we can download the model files from the Hugging Face Hub using the huggingface-cli tool, potentially targeting specific subdirectories.

```
[21] !huggingface-cli download meta-llama/Llama-3.2-1B-Instruct --include "original/*" --local-dir Llama-3.2-1B-Instruct
```

```
Fetching 3 files: 100% 3/3 [00:00<00:00, 11096.04it/s]
/content/Llama-3.2-1B-Instruct
```

Step 19:



Create Text Generation Pipeline (Code)

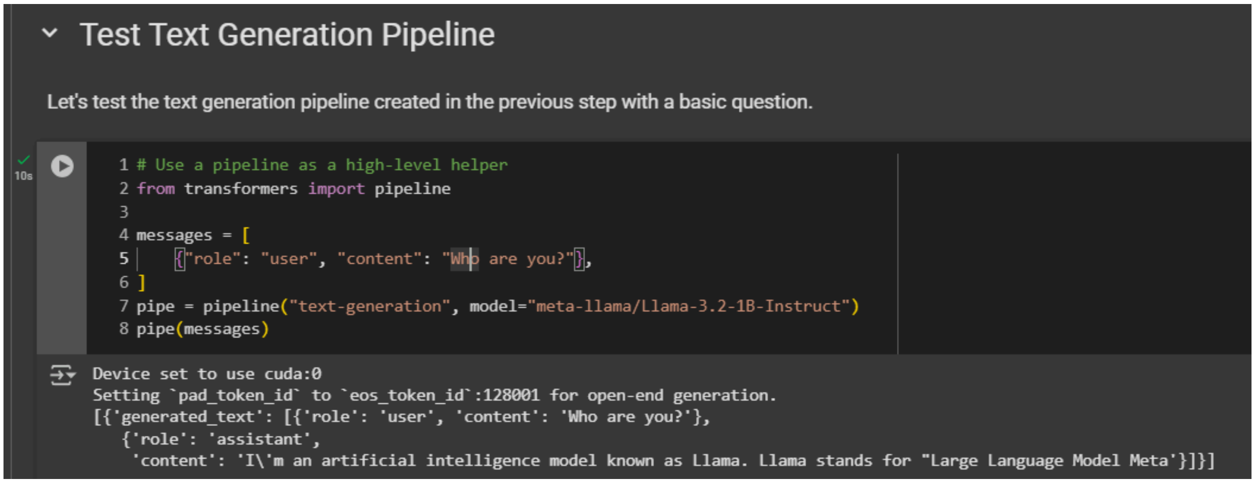
We create a Hugging Face pipeline object configured for text generation using the Llama 3.2 model.

```
from transformers import pipeline

pipe = pipeline(
    "text-generation",
    model="meta-llama/Llama-3.2-1B-Instruct",
    token="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",
    device_map="auto",  # Automatically uses GPU(s)
    torch_dtype=torch.float16  # Use float16 for reduced memory footprint
    )

```

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

This cell initializes a Hugging Face pipeline for the "text-generation" task. It specifies the model identifier (meta-llama/Llama-3.2-1B-Instruct), provides the authentication token, uses device_map="auto" to let transformers handle device placement (CPU/GPU), and sets the computation precision to torch.float16. This creates a high-level interface (pipe) for generating text with the configured model.

Step 20:



Test Text Generation Pipeline

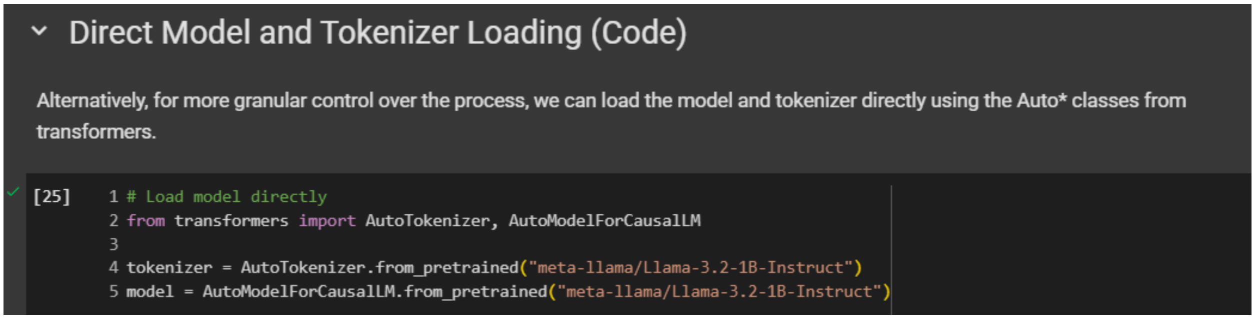
Let's test the text generation pipeline created in the previous step with a basic question.

```
# Use a pipeline as a high-level helper
from transformers import pipeline

messages = [
    {"role": "user", "content": "Who are you?"},
]
pipe = pipeline("text-generation", model="meta-llama/Llama-3.2-1B-Instruct")
pipe(messages)
```

```
Device set to use cuda:0
Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.
[{'generated_text': [{'role': 'user', 'content': 'Who are you?'},
   {'role': 'assistant',
    'content': 'I\'m an artificial intelligence model known as Llama. Llama stands for "Large Language Model Meta'}]}]
```

This cell demonstrates the usage of the pipeline object (pipe) created earlier. It defines a simple conversational input (messages) asking "Who are you?". This input is passed to the pipe() function, which handles tokenization, model inference, and decoding. The generated response from the Llama 3.2 model is then returned and displayed as the cell's output. Note that it implicitly re-initializes a pipeline here, potentially overwriting the pipe object from the previous cell if not intended.

Step 21:



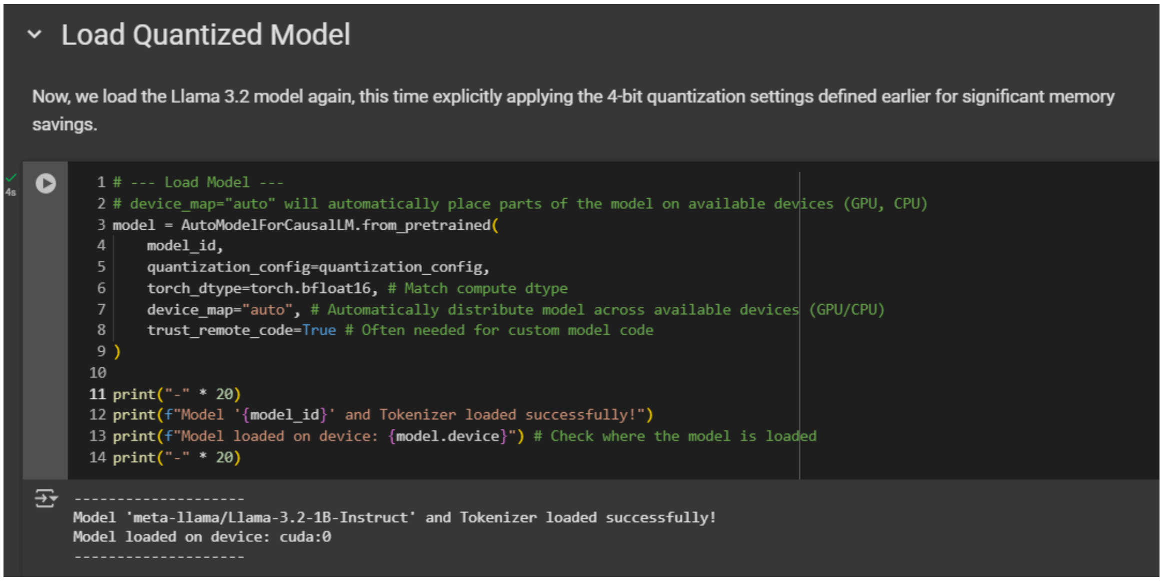
Direct Model and Tokenizer Loading (Code)

Alternatively, for more granular control over the process, we can load the model and tokenizer directly using the Auto* classes from transformers.

```
[25]
# Load model directly
from transformers import AutoTokenizer, AutoModelForCausalLM

tokenizer = AutoTokenizer.from_pretrained("meta-llama/Llama-3.2-1B-Instruct")
model = AutoModelForCausalLM.from_pretrained("meta-llama/Llama-3.2-1B-Instruct")
```

This cell demonstrates loading the model and tokenizer without using the high-level pipeline. It uses AutoTokenizer.from_pretrained and AutoModelForCausalLM.from_pretrained with the model_id (meta-llama/Llama-3.2-1B-Instruct). This approach gives the user direct access to the tokenizer and model objects, allowing for manual tokenization, generation calls with specific parameters, and explicit device management if needed.

Step 22:



Load Quantized Model

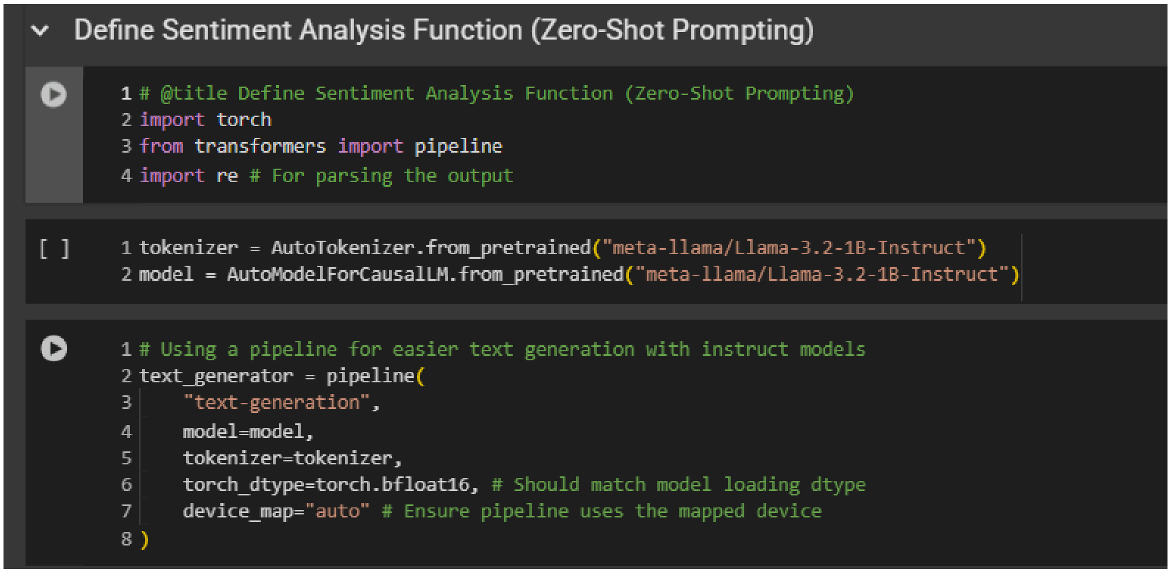
Now, we load the Llama 3.2 model again, this time explicitly applying the 4-bit quantization settings defined earlier for significant memory savings.

```
# --- Load Model ---
# device_map="auto" will automatically place parts of the model on available devices (GPU, CPU)
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    quantization_config=quantization_config,
    torch_dtype=torch.bfloat16, # Match compute dtype
    device_map="auto", # Automatically distribute model across available devices (GPU/CPU)
    trust_remote_code=True # Often needed for custom model code
)

print("-" * 20)
print(f"Model '{model_id}' and Tokenizer loaded successfully!")
print(f"Model loaded on device: {model.device}") # Check where the model is loaded
print("-" * 20)
```

```
--------------------
Model 'meta-llama/Llama-3.2-1B-Instruct' and Tokenizer loaded successfully!
Model loaded on device: cuda:0
--------------------
```

This cell loads the AutoModelForCausalLM using the specified model_id. Critically, it passes the quantization_config object (defined in Cell 19) to enable 4-bit loading. It also specifies torch_dtype=torch.bfloat16 (matching the compute dtype in the config), uses device_map="auto" for automatic hardware placement, and sets trust_remote_code=True. Upon successful loading, it prints confirmation messages, including the device(s) the model was loaded onto (e.g., cuda:0). This loaded model object is the quantized version.

Step 23:



Define Sentiment Analysis Function (Zero-Shot Prompting)

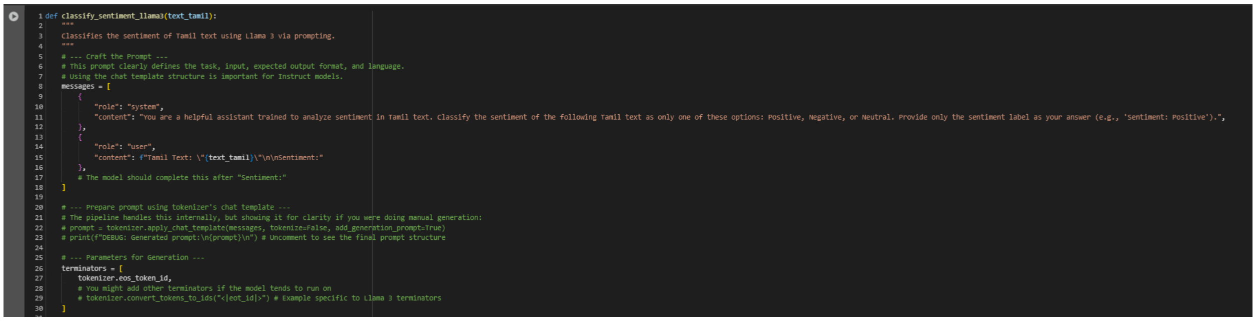
```
# @title Define Sentiment Analysis Function (Zero-Shot Prompting)
import torch
from transformers import pipeline
import re # For parsing the output
```

```
tokenizer = AutoTokenizer.from_pretrained("meta-llama/Llama-3.2-1B-Instruct")
model = AutoModelForCausalLM.from_pretrained("meta-llama/Llama-3.2-1B-Instruct")
```

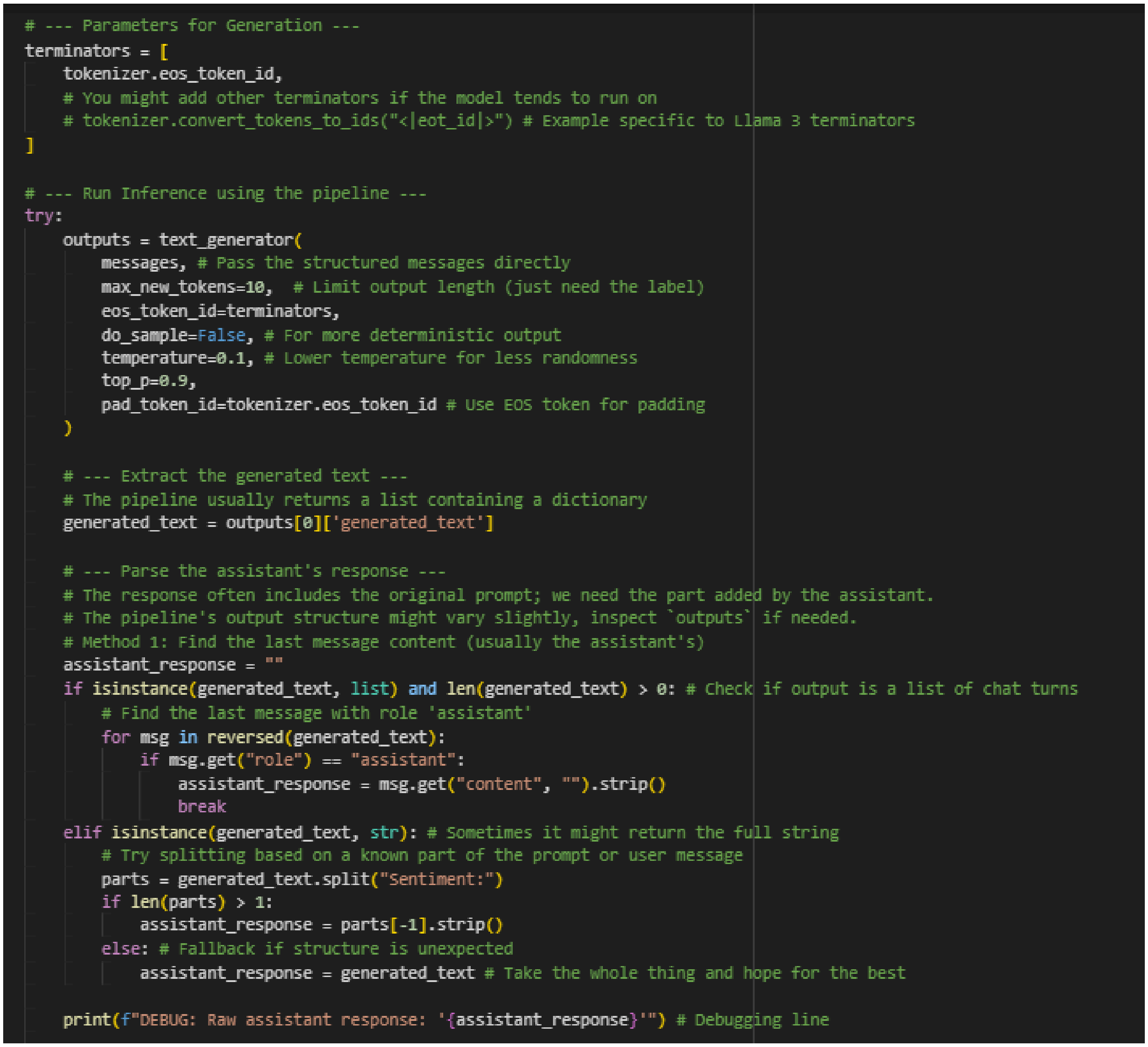
```
# Using a pipeline for easier text generation with instruct models
text_generator = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    torch_dtype=torch.bfloat16, # Should match model loading dtype
    device_map="auto" # Ensure pipeline uses the mapped device
)
```

This cell creates a text-generation pipeline instance named text_generator. Unlike Cell 27 which used the model ID, this one explicitly passes the model and tokenizer objects currently held in memory (which should be the quantized versions if Cell 30 was run last and Cell 32 was skipped/failed). It sets the torch_dtype to bfloat16 and uses device_map="auto" for correct device handling. This text_generator object is intended for use in the subsequent sentiment analysis functions.

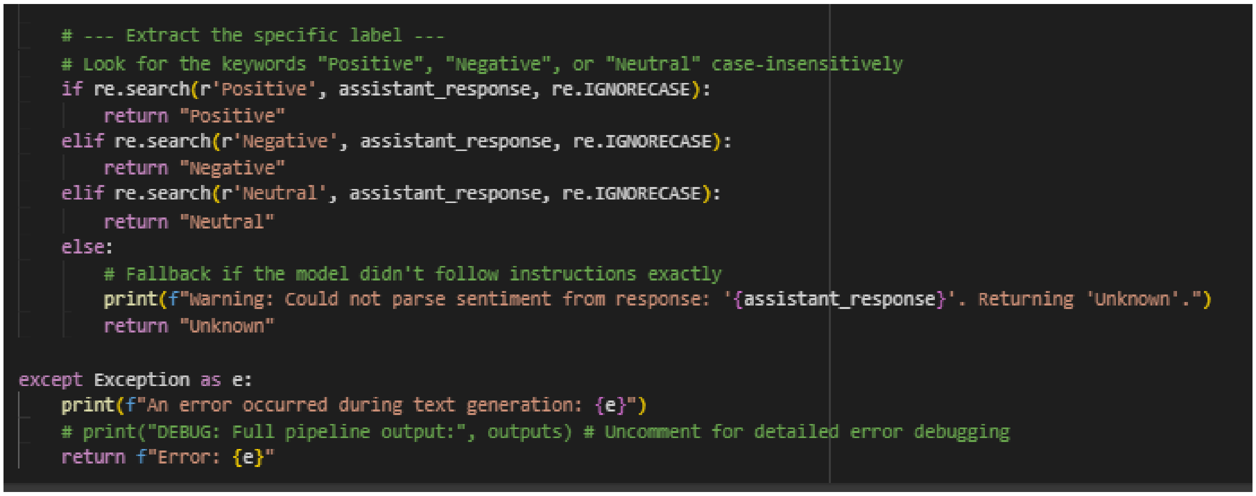
Step 24:



```python
def classify_sentiment_llama3(text_tamil):
    """
    Classifies the sentiment of Tamil text using Llama 3 via prompting.
    """
    # --- Craft the Prompt ---
    # This prompt clearly defines the task, input, expected output format, and language.
    # Using the chat template structure is important for Instruct models.
    messages = [
        {
            "role": "system",
            "content": "You are a helpful assistant trained to analyze sentiment in Tamil text. Classify the sentiment of the following Tamil text as only one of these options: Positive, Negative, or Neutral. Provide only the sentiment label as your answer (e.g., 'Sentiment: Positive').",
        },
        {
            "role": "user",
            "content": f"Tamil Text: \"{text_tamil}\"\n\nSentiment:"
        },
        # The model should complete this after "Sentiment:"
    ]

    # --- Prepare prompt using tokenizer's chat template ---
    # The pipeline handles this internally, but showing it for clarity if you were doing manual generation:
    # prompt = tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
    # print(f"DEBUG: Generated prompt:\n{prompt}\n") # Uncomment to see the final prompt structure

    # --- Parameters for Generation ---
    terminators = [
        tokenizer.eos_token_id,
        # You might add other terminators if the model tends to run on
        # tokenizer.convert_tokens_to_ids("<|eot_id|>") # Example specific to Llama 3 terminators
    ]
```



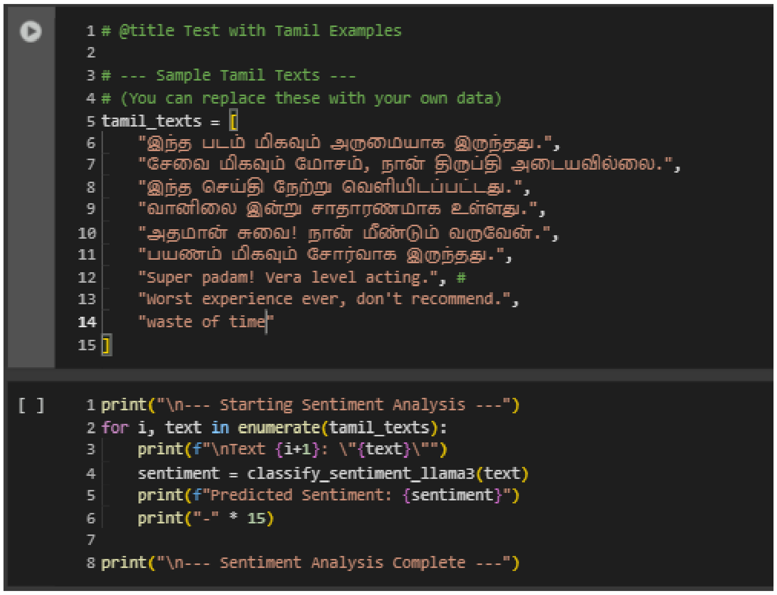
```python
# --- Parameters for Generation ---
terminators = [
    tokenizer.eos_token_id,
    # You might add other terminators if the model tends to run on
    # tokenizer.convert_tokens_to_ids("<|eot_id|>") # Example specific to Llama 3 terminators
]

# --- Run Inference using the pipeline ---
try:
    outputs = text_generator(
        messages, # Pass the structured messages directly
        max_new_tokens=10,  # Limit output length (just need the label)
        eos_token_id=terminators,
        do_sample=False, # For more deterministic output
        temperature=0.1, # Lower temperature for less randomness
        top_p=0.9,
        pad_token_id=tokenizer.eos_token_id # Use EOS token for padding
    )

    # --- Extract the generated text ---
    # The pipeline usually returns a list containing a dictionary
    generated_text = outputs[0]['generated_text']

    # --- Parse the assistant's response ---
    # The response often includes the original prompt; we need the part added by the assistant.
    # The pipeline's output structure might vary slightly, inspect `outputs` if needed.
    # Method 1: Find the last message content (usually the assistant's)
    assistant_response = ""
    if isinstance(generated_text, list) and len(generated_text) > 0: # Check if output is a list of chat turns
        # Find the last message with role 'assistant'
        for msg in reversed(generated_text):
            if msg.get("role") == "assistant":
                assistant_response = msg.get("content", "").strip()
                break
    elif isinstance(generated_text, str): # Sometimes it might return the full string
        # Try splitting based on a known part of the prompt or user message
        parts = generated_text.split("Sentiment:")
        if len(parts) > 1:
            assistant_response = parts[-1].strip()
        else: # Fallback if structure is unexpected
            assistant_response = generated_text # Take the whole thing and hope for the best

    print(f"DEBUG: Raw assistant response: '{assistant_response}'") # Debugging line
```



```python
    # --- Extract the specific label ---
    # Look for the keywords "Positive", "Negative", or "Neutral" case-insensitively
    if re.search(r'Positive', assistant_response, re.IGNORECASE):
        return "Positive"
    elif re.search(r'Negative', assistant_response, re.IGNORECASE):
        return "Negative"
    elif re.search(r'Neutral', assistant_response, re.IGNORECASE):
        return "Neutral"
    else:
        # Fallback if the model didn't follow instructions exactly
        print(f"Warning: Could not parse sentiment from response: '{assistant_response}'. Returning 'Unknown'.")
        return "Unknown"

except Exception as e:
    print(f"An error occurred during text generation: {e}")
    # print("DEBUG: Full pipeline output:", outputs) # Uncomment for detailed error debugging
    return f"Error: {e}"
```

This cell defines the Python function classify_sentiment_llama3(text_tamil). It takes text (expected to be Tamil or code-mixed) as input. Inside, it constructs a specific conversational prompt (messages) instructing the LLM (via the text_generator pipeline created in the previous cell) to classify the sentiment as "Positive", "Negative", or "Neutral" and to provide only the label. It specifies generation parameters like max_new_tokens, eos_token_id, temperature, and top_p to control the output length and determinism. The function includes logic to parse the pipeline's output, extract the assistant's response, and use regular expressions (re.search) to find the sentiment label, returning it or "Unknown"/"Error" if parsing fails or an exception occurs.

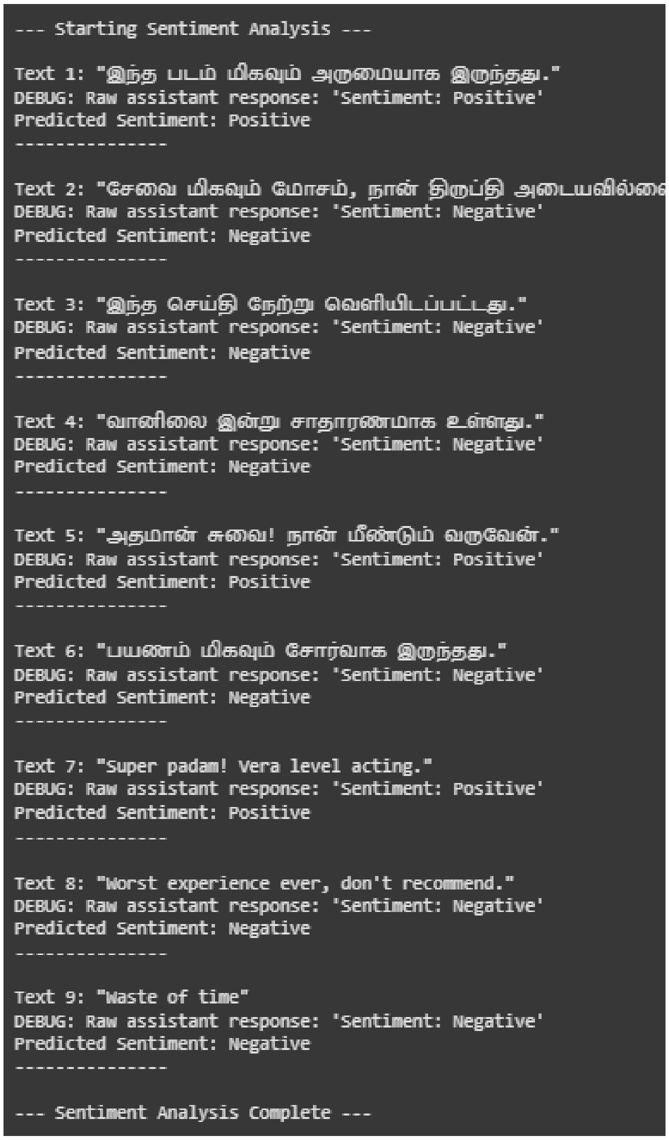
Step 25:



```python
# @title Test with Tamil Examples

# --- Sample Tamil Texts ---
# (You can replace these with your own data)
tamil_texts = [
    "இந்த படம் மிகவும் அருமையாக இருந்தது.",
    "சேவை மிகவும் மோசம், நான் திருப்தி அடையவில்லை.",
    "இந்த செய்தி நேற்று வெளியிடப்பட்டது.",
    "வானிலை இன்று சாதாரணமாக உள்ளது.",
    "அதமான் சுவை! நான் மீண்டும் வருவேன்.",
    "பயணம் மிகவும் சோர்வாக இருந்தது.",
    "Super padam! Vera level acting.", #
    "Worst experience ever, don't recommend.",
    "waste of time"
]
```

```python
print("\n--- Starting Sentiment Analysis ---")
for i, text in enumerate(tamil_texts):
    print(f"\nText {i+1}: \"{text}\"")
    sentiment = classify_sentiment_llama3(text)
    print(f"Predicted Sentiment: {sentiment}")
    print("-" * 15)

print("\n--- Sentiment Analysis Complete ---")
```

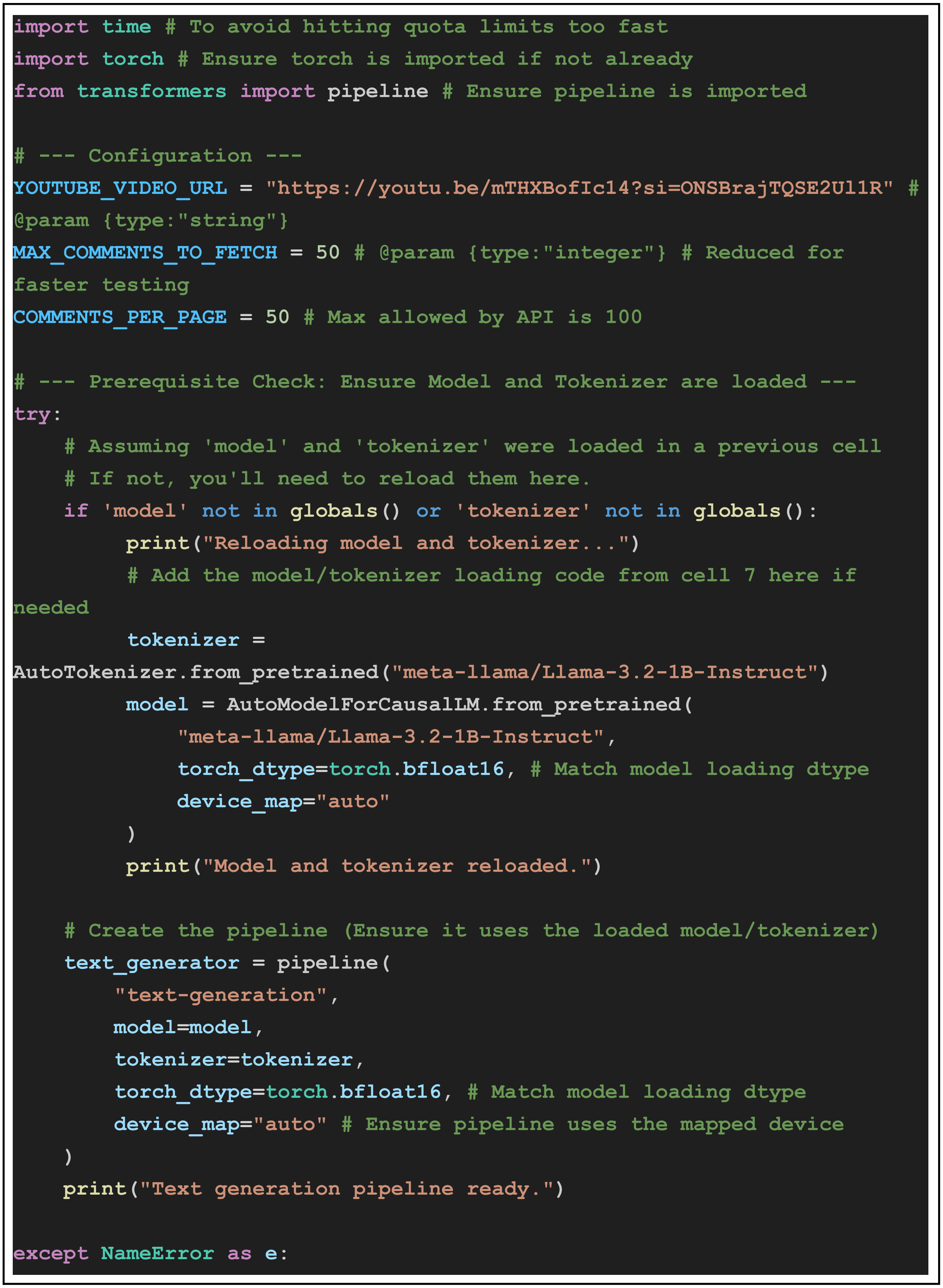
This cell creates a Python list named tamil_texts. This list holds several strings representing different sentiments. It includes examples purely in Tamil script and examples that mix Tamil (often transliterated) and English words, reflecting common code-mixing patterns. These samples will be used to evaluate the classify_sentiment_llama3 function.



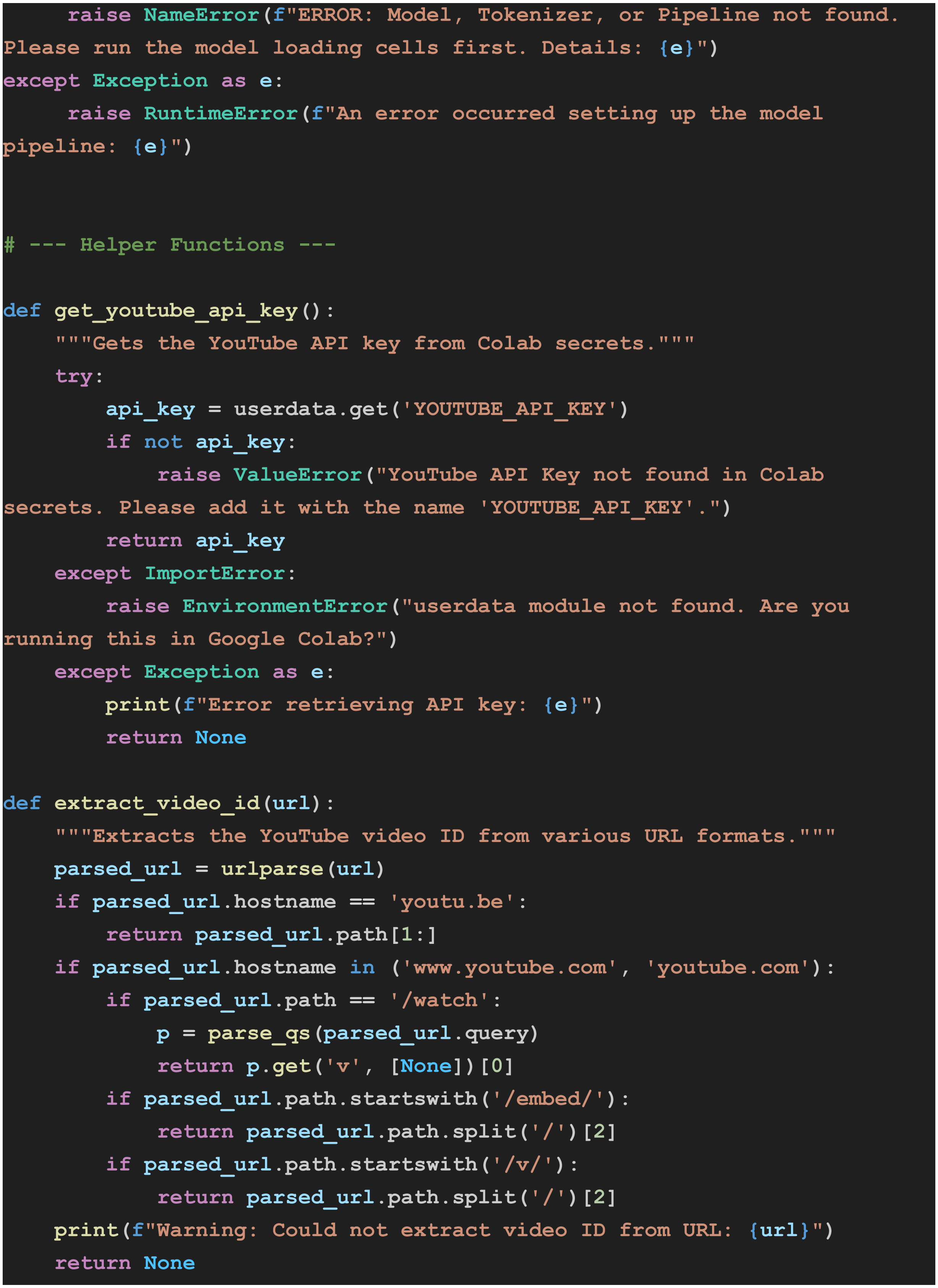
```
--- Starting Sentiment Analysis ---

Text 1: "இந்த படம் மிகவும் அருமையாக இருந்தது."
DEBUG: Raw assistant response: 'Sentiment: Positive'
Predicted Sentiment: Positive
---------------

Text 2: "சேவை மிகவும் மோசம், நான் திருப்தி அடையவில்லை
DEBUG: Raw assistant response: 'Sentiment: Negative'
Predicted Sentiment: Negative
---------------

Text 3: "இந்த செய்தி நேற்று வெளியிடப்பட்டது."
DEBUG: Raw assistant response: 'Sentiment: Negative'
Predicted Sentiment: Negative
---------------

Text 4: "வானிலை இன்று சாதாரணமாக உள்ளது."
DEBUG: Raw assistant response: 'Sentiment: Negative'
Predicted Sentiment: Negative
---------------

Text 5: "அதமான் சுவை! நான் மீண்டும் வருவேன்."
DEBUG: Raw assistant response: 'Sentiment: Positive'
Predicted Sentiment: Positive
---------------

Text 6: "பயணம் மிகவும் சோர்வாக இருந்தது."
DEBUG: Raw assistant response: 'Sentiment: Negative'
Predicted Sentiment: Negative
---------------

Text 7: "Super padam! Vera level acting."
DEBUG: Raw assistant response: 'Sentiment: Positive'
Predicted Sentiment: Positive
---------------

Text 8: "Worst experience ever, don't recommend."
DEBUG: Raw assistant response: 'Sentiment: Negative'
Predicted Sentiment: Negative
---------------

Text 9: "Waste of time"
DEBUG: Raw assistant response: 'Sentiment: Negative'
Predicted Sentiment: Negative
---------------

--- Sentiment Analysis Complete ---
```

Step 26:

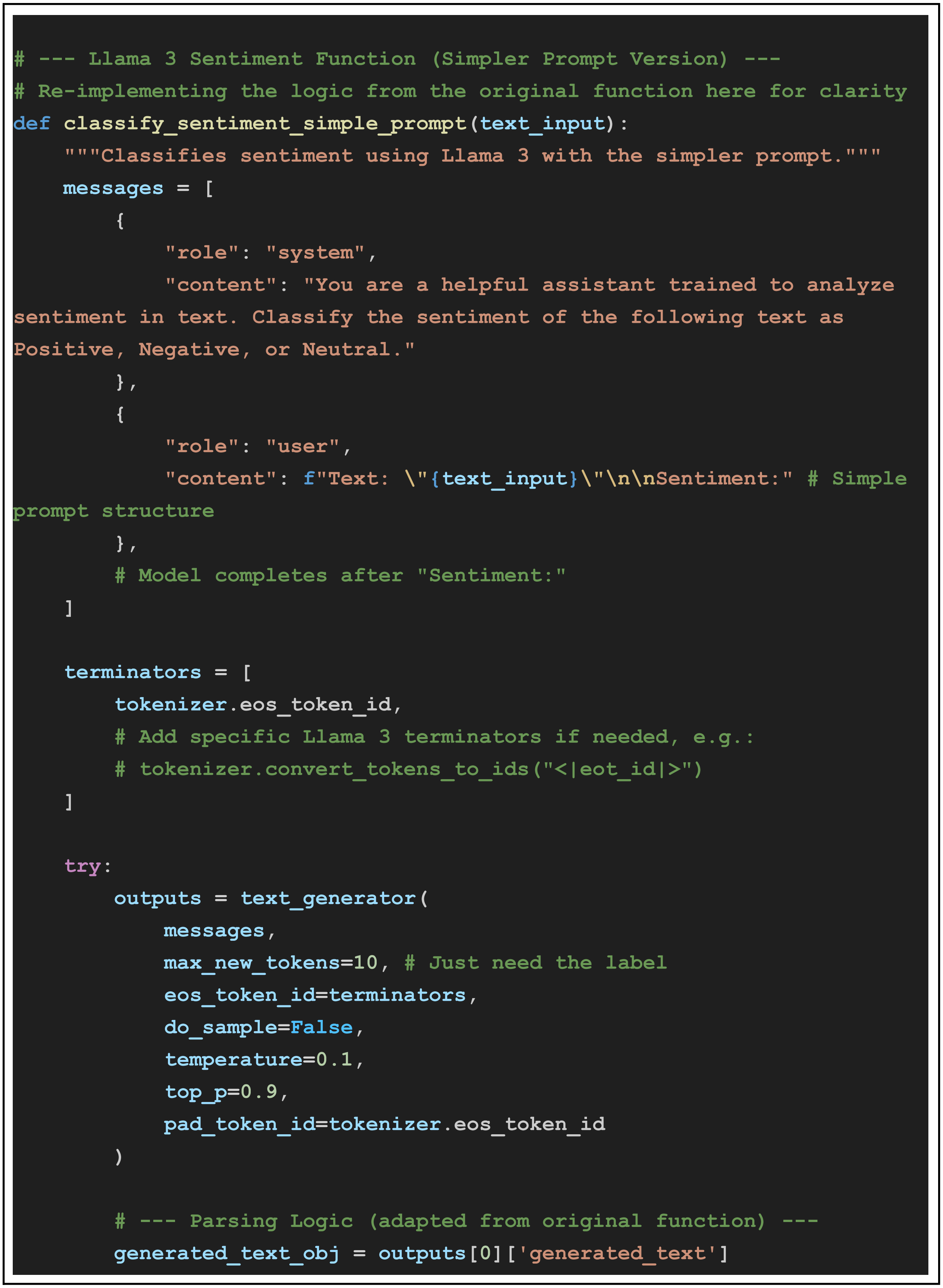
```
# @title Fetch YouTube Comments and Analyze Sentiment (Using Simpler
Prompt Logic)

import os
import pandas as pd
import re
from googleapiclient.discovery import build
from googleapiclient.errors import HttpError
from google.colab import userdata
from urllib.parse import urlparse, parse_qs
```



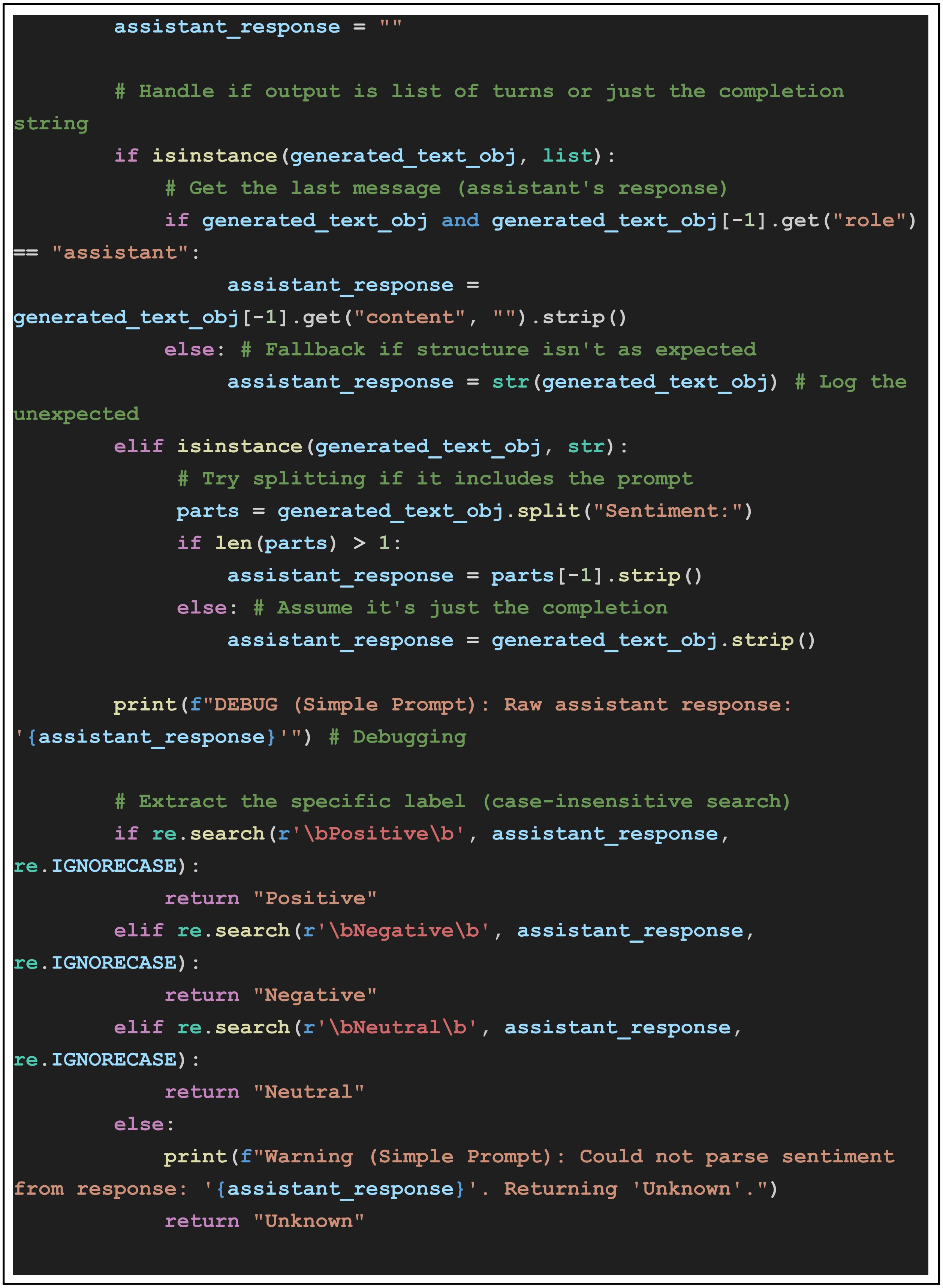
```
import time # To avoid hitting quota limits too fast
import torch # Ensure torch is imported if not already
from transformers import pipeline # Ensure pipeline is imported

# --- Configuration ---
YOUTUBE_VIDEO_URL = "https://youtu.be/mTHXBofIc14?si=ONSBrajTQSE2Ul1R" #
@param {type:"string"}
MAX_COMMENTS_TO_FETCH = 50 # @param {type:"integer"} # Reduced for
faster testing
COMMENTS_PER_PAGE = 50 # Max allowed by API is 100

# --- Prerequisite Check: Ensure Model and Tokenizer are loaded ---
try:
    # Assuming 'model' and 'tokenizer' were loaded in a previous cell
    # If not, you'll need to reload them here.
    if 'model' not in globals() or 'tokenizer' not in globals():
         print("Reloading model and tokenizer...")
         # Add the model/tokenizer loading code from cell 7 here if
needed
         tokenizer =
AutoTokenizer.from_pretrained("meta-llama/Llama-3.2-1B-Instruct")
         model = AutoModelForCausalLM.from_pretrained(
             "meta-llama/Llama-3.2-1B-Instruct",
             torch_dtype=torch.bfloat16, # Match model loading dtype
             device_map="auto"
         )
         print("Model and tokenizer reloaded.")

    # Create the pipeline (Ensure it uses the loaded model/tokenizer)
    text_generator = pipeline(
        "text-generation",
        model=model,
        tokenizer=tokenizer,
        torch_dtype=torch.bfloat16, # Match model loading dtype
        device_map="auto" # Ensure pipeline uses the mapped device
    )
    print("Text generation pipeline ready.")

except NameError as e:
```



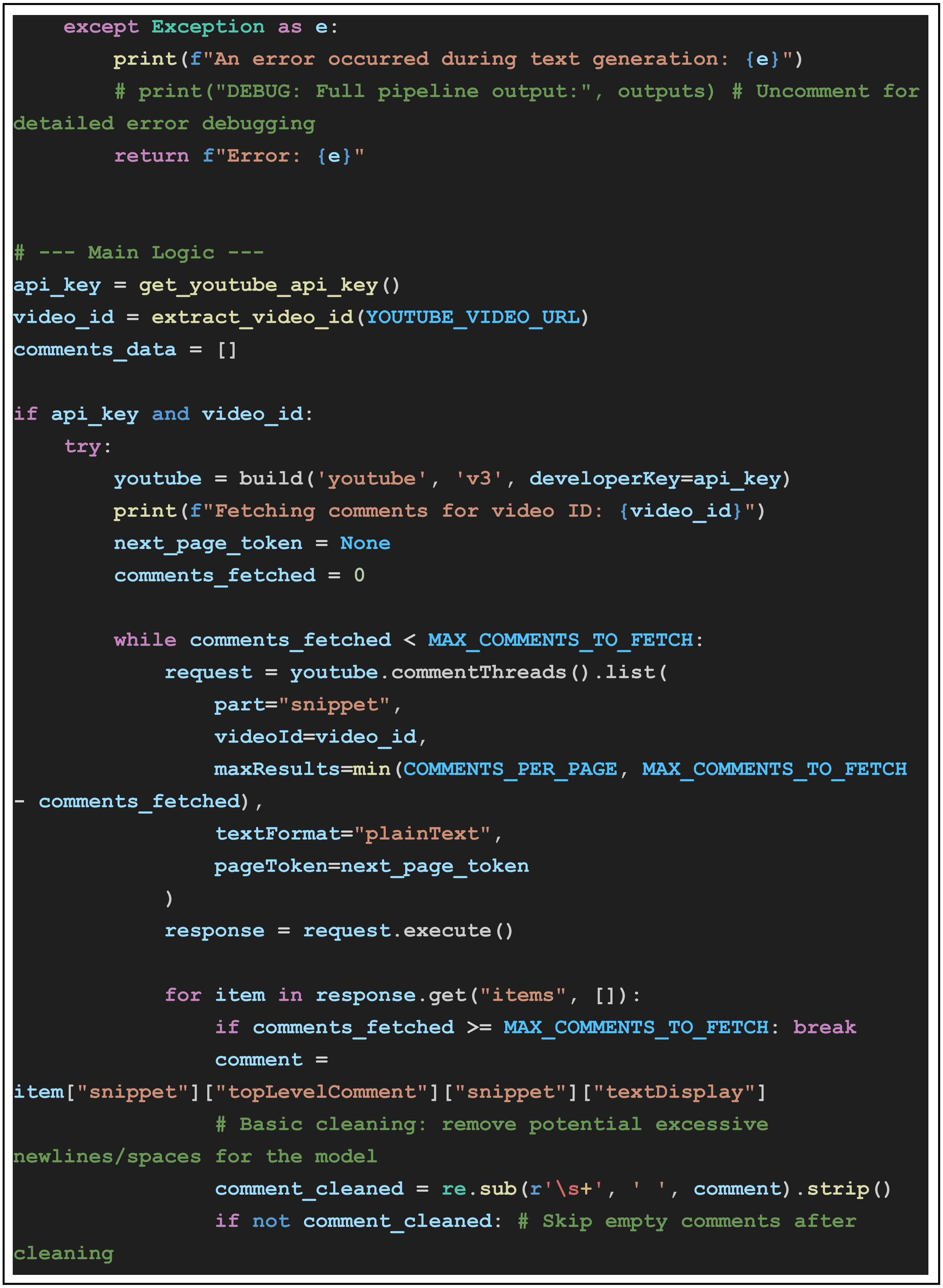
```
     raise NameError(f"ERROR: Model, Tokenizer, or Pipeline not found.
Please run the model loading cells first. Details: {e}")
except Exception as e:
     raise RuntimeError(f"An error occurred setting up the model
pipeline: {e}")


# --- Helper Functions ---

def get_youtube_api_key():
    """Gets the YouTube API key from Colab secrets."""
    try:
        api_key = userdata.get('YOUTUBE_API_KEY')
        if not api_key:
            raise ValueError("YouTube API Key not found in Colab
secrets. Please add it with the name 'YOUTUBE_API_KEY'.")
        return api_key
    except ImportError:
        raise EnvironmentError("userdata module not found. Are you
running this in Google Colab?")
    except Exception as e:
        print(f"Error retrieving API key: {e}")
        return None

def extract_video_id(url):
    """Extracts the YouTube video ID from various URL formats."""
    parsed_url = urlparse(url)
    if parsed_url.hostname == 'youtu.be':
        return parsed_url.path[1:]
    if parsed_url.hostname in ('www.youtube.com', 'youtube.com'):
        if parsed_url.path == '/watch':
            p = parse_qs(parsed_url.query)
            return p.get('v', [None])[0]
        if parsed_url.path.startswith('/embed/'):
            return parsed_url.path.split('/')[2]
        if parsed_url.path.startswith('/v/'):
            return parsed_url.path.split('/')[2]
    print(f"Warning: Could not extract video ID from URL: {url}")
    return None
```



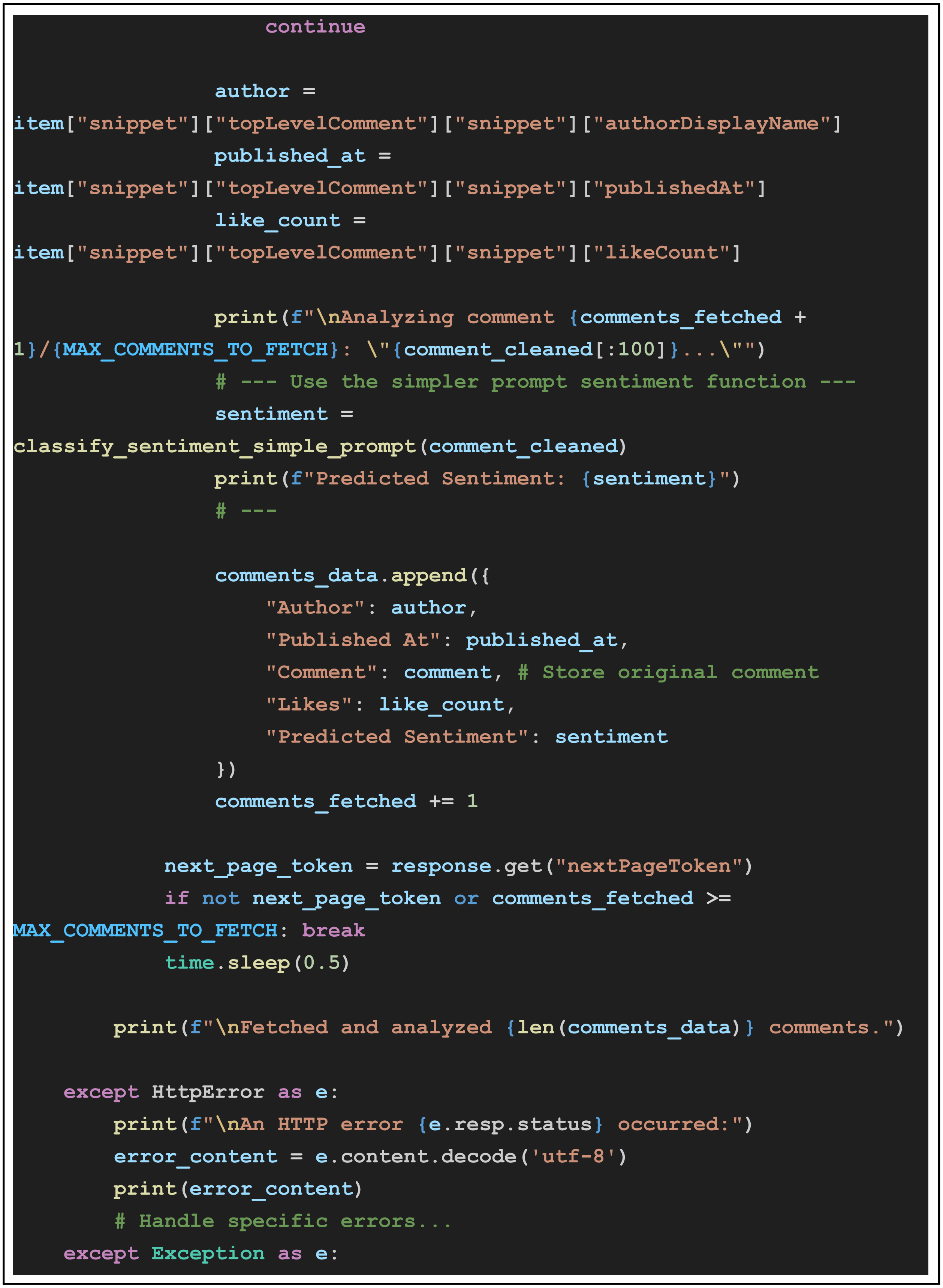
```python
# --- Llama 3 Sentiment Function (Simpler Prompt Version) ---
# Re-implementing the logic from the original function here for clarity
def classify_sentiment_simple_prompt(text_input):
    """Classifies sentiment using Llama 3 with the simpler prompt."""
    messages = [
        {
            "role": "system",
            "content": "You are a helpful assistant trained to analyze sentiment in text. Classify the sentiment of the following text as Positive, Negative, or Neutral."
        },
        {
            "role": "user",
            "content": f"Text: \"{text_input}\"\n\nSentiment:" # Simple prompt structure
        },
        # Model completes after "Sentiment:"
    ]

    terminators = [
        tokenizer.eos_token_id,
        # Add specific Llama 3 terminators if needed, e.g.:
        # tokenizer.convert_tokens_to_ids("<|eot_id|>")
    ]

    try:
        outputs = text_generator(
            messages,
            max_new_tokens=10, # Just need the label
            eos_token_id=terminators,
            do_sample=False,
            temperature=0.1,
            top_p=0.9,
            pad_token_id=tokenizer.eos_token_id
        )

        # --- Parsing Logic (adapted from original function) ---
        generated_text_obj = outputs[0]['generated_text']
```



```
        assistant_response = ""

        # Handle if output is list of turns or just the completion
string
        if isinstance(generated_text_obj, list):
            # Get the last message (assistant's response)
            if generated_text_obj and generated_text_obj[-1].get("role")
== "assistant":
                 assistant_response =
generated_text_obj[-1].get("content", "").strip()
            else: # Fallback if structure isn't as expected
                 assistant_response = str(generated_text_obj) # Log the
unexpected
        elif isinstance(generated_text_obj, str):
             # Try splitting if it includes the prompt
             parts = generated_text_obj.split("Sentiment:")
             if len(parts) > 1:
                 assistant_response = parts[-1].strip()
             else: # Assume it's just the completion
                 assistant_response = generated_text_obj.strip()

        print(f"DEBUG (Simple Prompt): Raw assistant response:
'{assistant_response}'") # Debugging

        # Extract the specific label (case-insensitive search)
        if re.search(r'\bPositive\b', assistant_response,
re.IGNORECASE):
            return "Positive"
        elif re.search(r'\bNegative\b', assistant_response,
re.IGNORECASE):
            return "Negative"
        elif re.search(r'\bNeutral\b', assistant_response,
re.IGNORECASE):
            return "Neutral"
        else:
            print(f"Warning (Simple Prompt): Could not parse sentiment
from response: '{assistant_response}'. Returning 'Unknown'.")
            return "Unknown"
```

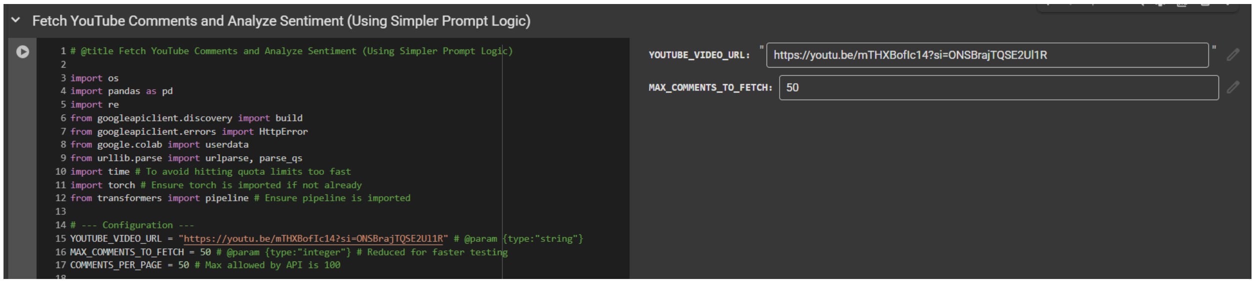


```
    except Exception as e:
        print(f"An error occurred during text generation: {e}")
        # print("DEBUG: Full pipeline output:", outputs) # Uncomment for
detailed error debugging
        return f"Error: {e}"


# --- Main Logic ---
api_key = get_youtube_api_key()
video_id = extract_video_id(YOUTUBE_VIDEO_URL)
comments_data = []

if api_key and video_id:
    try:
        youtube = build('youtube', 'v3', developerKey=api_key)
        print(f"Fetching comments for video ID: {video_id}")
        next_page_token = None
        comments_fetched = 0

        while comments_fetched < MAX_COMMENTS_TO_FETCH:
            request = youtube.commentThreads().list(
                part="snippet",
                videoId=video_id,
                maxResults=min(COMMENTS_PER_PAGE, MAX_COMMENTS_TO_FETCH
- comments_fetched),
                textFormat="plainText",
                pageToken=next_page_token
            )
            response = request.execute()

            for item in response.get("items", []):
                if comments_fetched >= MAX_COMMENTS_TO_FETCH: break
                comment =
item["snippet"]["topLevelComment"]["snippet"]["textDisplay"]
                # Basic cleaning: remove potential excessive
newlines/spaces for the model
                comment_cleaned = re.sub(r'\s+', ' ', comment).strip()
                if not comment_cleaned: # Skip empty comments after
cleaning
```



```
                    continue

                author =
item["snippet"]["topLevelComment"]["snippet"]["authorDisplayName"]
                published_at =
item["snippet"]["topLevelComment"]["snippet"]["publishedAt"]
                like_count =
item["snippet"]["topLevelComment"]["snippet"]["likeCount"]

                print(f"\nAnalyzing comment {comments_fetched +
1}/{MAX_COMMENTS_TO_FETCH}: \"{comment_cleaned[:100]}...\"")
                # --- Use the simpler prompt sentiment function ---
                sentiment =
classify_sentiment_simple_prompt(comment_cleaned)
                print(f"Predicted Sentiment: {sentiment}")
                # ---

                comments_data.append({
                    "Author": author,
                    "Published At": published_at,
                    "Comment": comment, # Store original comment
                    "Likes": like_count,
                    "Predicted Sentiment": sentiment
                })
                comments_fetched += 1

            next_page_token = response.get("nextPageToken")
            if not next_page_token or comments_fetched >=
MAX_COMMENTS_TO_FETCH: break
            time.sleep(0.5)

        print(f"\nFetched and analyzed {len(comments_data)} comments.")

    except HttpError as e:
        print(f"\nAn HTTP error {e.resp.status} occurred:")
        error_content = e.content.decode('utf-8')
        print(error_content)
        # Handle specific errors...
    except Exception as e:
        print(f"\nAn unexpected error occurred: {e}")

else:
    # Handle missing API key or video ID...
    pass

# --- Display Results ---
if comments_data:
    df_comments = pd.DataFrame(comments_data)
    print("\n--- Comment Sentiment Analysis Results (Using Simpler
Prompt) ---")
    pd.set_option('display.max_colwidth', 200)
    pd.set_option('display.max_rows', 100)
    display(df_comments)
else:
    print("\nNo comments were fetched or analyzed.")
```

Insert the youtube videos link to sentiment analyze for the comments along with the no. of comments to fetch.



Fetch YouTube Comments and Analyze Sentiment (Using Simpler Prompt Logic)

```
# @title Fetch YouTube Comments and Analyze Sentiment (Using Simpler Prompt Logic)

import os
import pandas as pd
import re
from googleapiclient.discovery import build
from googleapiclient.errors import HttpError
from google.colab import userdata
from urllib.parse import urlparse, parse_qs
import time # To avoid hitting quota limits too fast
import torch # Ensure torch is imported if not already
from transformers import pipeline # Ensure pipeline is imported

# --- Configuration ---
YOUTUBE_VIDEO_URL = "https://youtu.be/mTHXBofIc14?si=ONSBrajTQSE2Ul1R" # @param {type:"string"}
MAX_COMMENTS_TO_FETCH = 50 # @param {type:"integer"} # Reduced for faster testing
COMMENTS_PER_PAGE = 50 # Max allowed by API is 100
```

YOUTUBE_VIDEO_URL: "https://youtu.be/mTHXBofIc14?si=ONSBrajTQSE2Ul1R"

MAX_COMMENTS_TO_FETCH: 50

5.Proposed/Developed Model & Technique:

The project does not develop a new model architecture from scratch or fine-tune an existing one. Instead, it utilizes a pre-trained Large Language Model:

- Model: Meta-Llama-3.2-1B-Instruct (Specifically the 1 Billion parameter instruct-tuned version of Llama 3.2). An earlier step also showed the 8B Instruct model ID being defined, but later steps explicitly use and download the 3.2-1B version.
- Technique: Zero-Shot Prompting with 4-bit Quantization.
  - Zero-Shot Prompting: The model is given instructions (a prompt) at inference time describing the sentiment analysis task and the expected output format, along with the text to classify. It uses its pre-trained knowledge to perform the classification without prior examples of this specific task during a fine-tuning phase.
  - Quantization: The Llama 3.2 model is loaded using bitsandbytes in 4-bit precision (specifically NF4 - NormalFloat 4) with bfloat16 compute data type. This significantly reduces the model's memory footprint (VRAM usage), making it feasible to run on hardware with limited resources, such as the Google Colab T4 GPU mentioned in the steps.

5.1 Design:

1. **Environment Setup:** Configure a Google Colab environment with necessary GPU resources. Install core Python libraries for deep learning (PyTorch), transformer models (Hugging Face transformers), efficient loading (bitsandbytes, accelerate), data handling (datasets, pandas), Hugging Face Hub interaction (huggingface_hub), and Google APIs (google-api-python-client).
2. **Authentication**: Securely authenticate with Hugging Face Hub using API tokens (preferably via Colab Secrets) to download gated models like Llama 3. Obtain a YouTube Data API key (via Colab Secrets) for fetching comments.
3. **Model Acquisition & Loading:** Download the specified Llama 3.2-1B-Instruct model weights and tokenizer. Load the model into memory

using the transformers library, explicitly applying the 4-bit quantization configuration. Ensure the model is correctly mapped to the available GPU.

4. **Prompt Engineering:** Design specific prompts for the Llama 3 Instruct model. This involves creating a structured conversation format (system message defining the role/task, user message containing the text and asking for classification) that guides the model to output only the sentiment label (Positive, Negative, or Neutral).
5. **Sentiment Classification Logic**: Develop a Python function that takes text as input, formats it into the designed prompt, uses the loaded quantized model (via a transformers pipeline) to generate a response, parses the response to extract the sentiment label using string manipulation and regular expressions, and handles potential errors or ambiguous outputs.
6. **YouTube Integration:** Use the YouTube Data API v3 to fetch comments for a given YouTube video URL. Implement logic for pagination (fetching multiple pages of comments) and basic text cleaning.
7. **Application Flow**: Integrate the sentiment classification function with the YouTube comment fetching logic. Iterate through fetched comments, classify each one's sentiment using the Llama 3 model via prompting, and store the results.
8. **Output & Evaluation**: Present the results (comment text, author, predicted sentiment, etc.) in a structured format (Pandas DataFrame). Evaluate the model's performance qualitatively based on the sample outputs and the YouTube comment analysis.

5.2 Module Wise Description:

1. **Setup & Configuration (Steps 1-17):**
    - Installs libraries (pip install...).
    - Imports necessary modules (torch, transformers, datasets, googleapiclient, etc.).
    - Handles Hugging Face login (notebook_login, huggingface-cli login, Colab Secrets).
    - Mounts Google Drive (optional).
    - Checks GPU availability (torch.cuda.is_available()).
    - Defines configuration variables (MODEL_ID, quantization config, API keys via userdata, YouTube URL, fetch limits).

    - Shows alternative model download methods (llama-stack, huggingface-cli download).
2. **Model Loading & Pipeline (Steps 19, 21, 22, 23, part of 26):**
    - Loads the AutoTokenizer for the specified MODEL_ID.
    - Loads the AutoModelForCausalLM with 4-bit quantization_config and device_map="auto".
    - Creates a transformers text-generation pipeline (text_generator) encapsulating the loaded quantized model and tokenizer for easier inference.
3. **Sentiment Classification Function (Steps 24, 26):**
    - Defines classify_sentiment_llama3 and later a similar classify_sentiment_simple_prompt function.
    - Takes text_input (Tamil/Code-Mixed).
    - Constructs messages list (system/user prompt).
    - Calls text_generator(messages, max_new_tokens=10, do_sample=False, ...) to get model output.
    - Parses outputs[0]['generated_text'] to isolate the assistant's response.
    - Uses re.search(r'\bPositive\b', ..., re.IGNORECASE) (and similar for Negative/Neutral) to extract the label.
    - Returns "Positive", "Negative", "Neutral", "Unknown", or "Error".
4. **YouTube Comment Fetching (Step 26):**
    - Defines helper functions get_youtube_api_key() and extract_video_id().
    - Initializes YouTube API service: build('youtube', 'v3', ...).
    - Uses a while loop and youtube.commentThreads().list(...) with pageToken to fetch comments iteratively.
    - Extracts relevant comment details (textDisplay, authorDisplayName, publishedAt, likeCount).
    - Performs basic text cleaning (re.sub(r'\s+', ' ', comment).strip()).
    - Includes time.sleep(0.5) for rate limiting.
5. **Integration & Analysis Loop (Step 26):**
    - The main logic block orchestrates fetching and analysis.
    - It calls classify_sentiment_simple_prompt for each cleaned comment.
    - Appends results (author, timestamp, comment, likes, sentiment) to the comments_data list.

    - Handles HttpError from the API and other exceptions.

6. **Results Display (Step 26 & Outputs):**
    - Checks if comments_data is populated.
    - Creates a pandas.DataFrame(comments_data).
    - Configures pandas display options (max_colwidth, max_rows).
    - Displays the DataFrame containing the comments and their analyzed sentiments.
    - Includes console output (print) for progress and debugging during fetching/analysis.

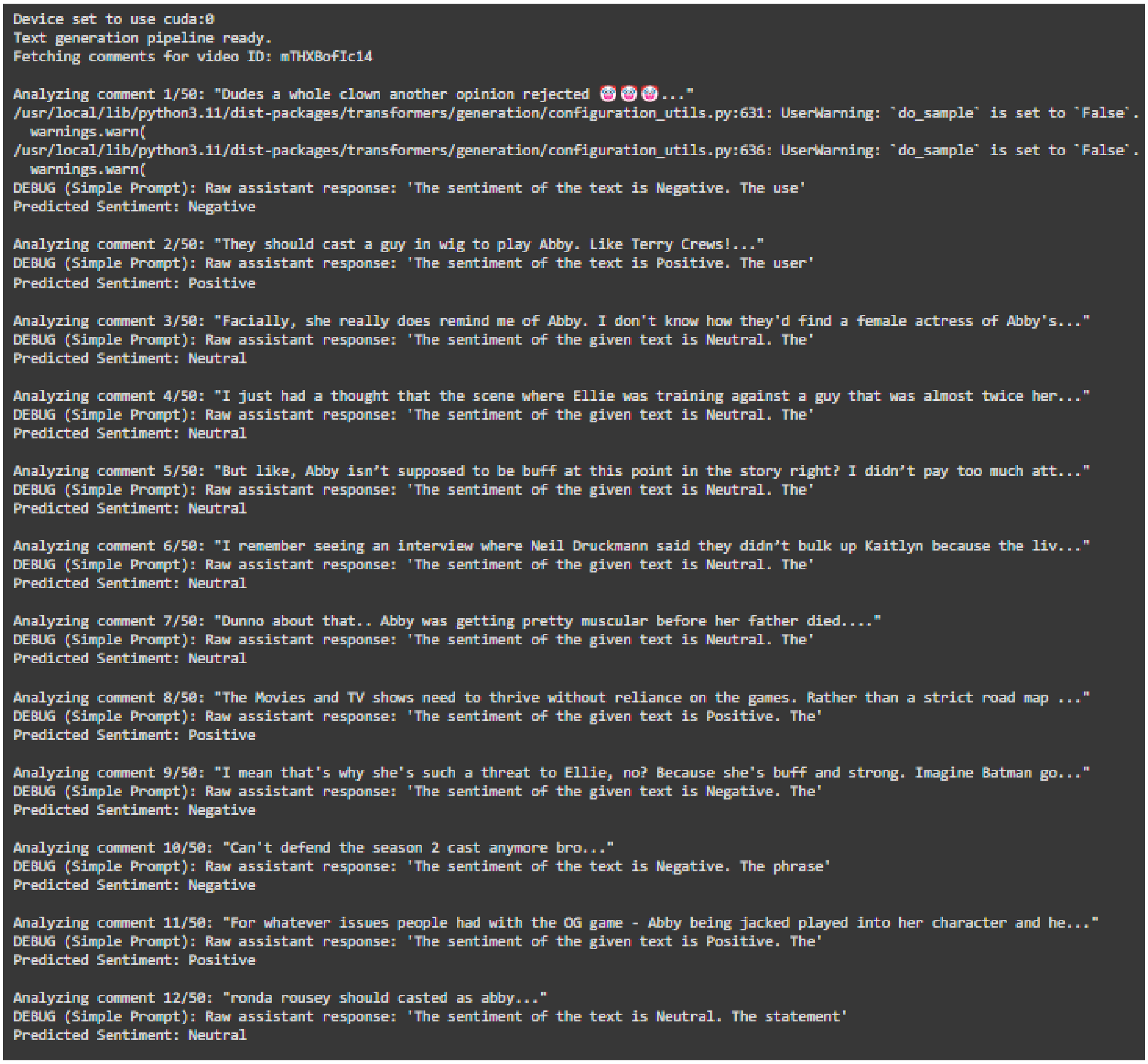
5.3 Implementation:

The implementation uses Python within a Google Colab notebook. Key implementation details include:

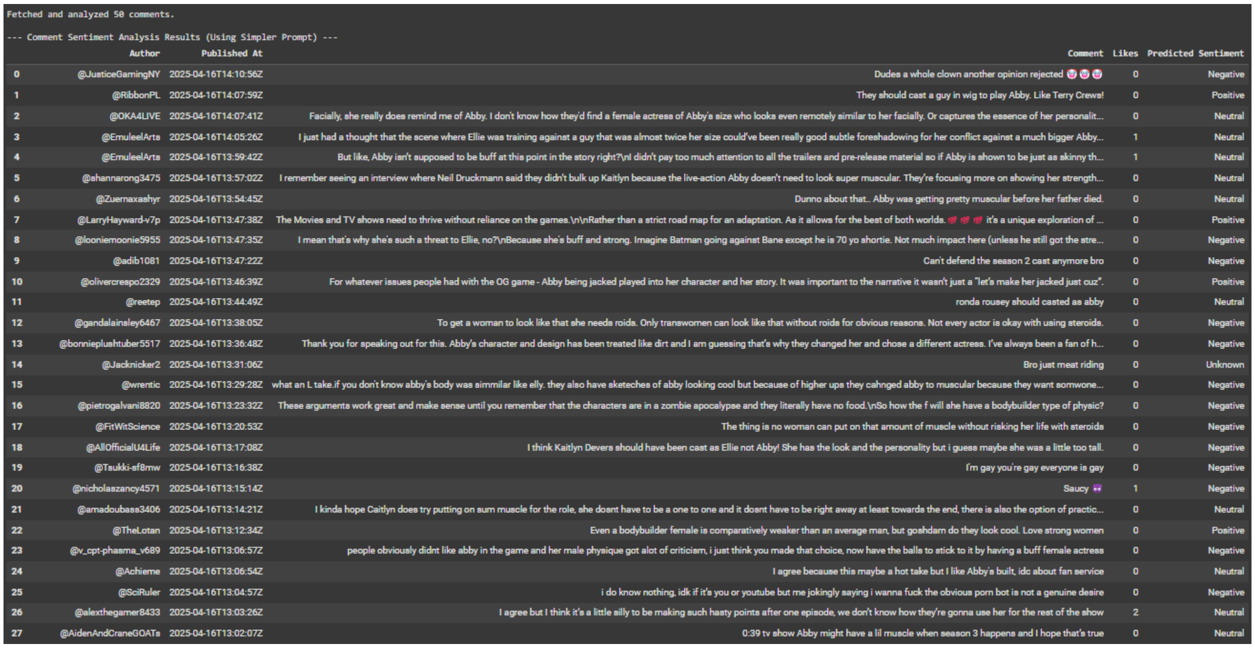
- **Libraries:** torch, transformers, bitsandbytes, accelerate, datasets, huggingface_hub, pandas, re, google-api-python-client, urllib.parse.
- **Model Loading:** Using AutoModelForCausalLM.from_pretrained with BitsAndBytesConfig for 4-bit loading.
- **Inference:** Employing the transformers.pipeline("text-generation", ...) for simplified zero-shot inference.
- **Prompting:** Structuring prompts as lists of dictionaries [{"role": "system", "content": "..."}, {"role": "user", "content": "..."}].
- **Output Parsing:** Relying on string splitting and re.search with re.IGNORECASE for robust label extraction.
- **API Interaction:** Using googleapiclient.discovery.build and method calls like commentThreads().list().execute().
- **Data Handling:** Storing intermediate results in Python lists and final results in a Pandas DataFrame.
- **Authentication:** Leveraging Colab Secrets (userdata.get) for securely handling API keys.

The code progresses logically from setup, through model loading and function definition, to the final application of analyzing YouTube comments, demonstrating the zero-shot prompting capability of the quantized Llama 3 model for Tamil/Code-Mixed sentiment analysis.

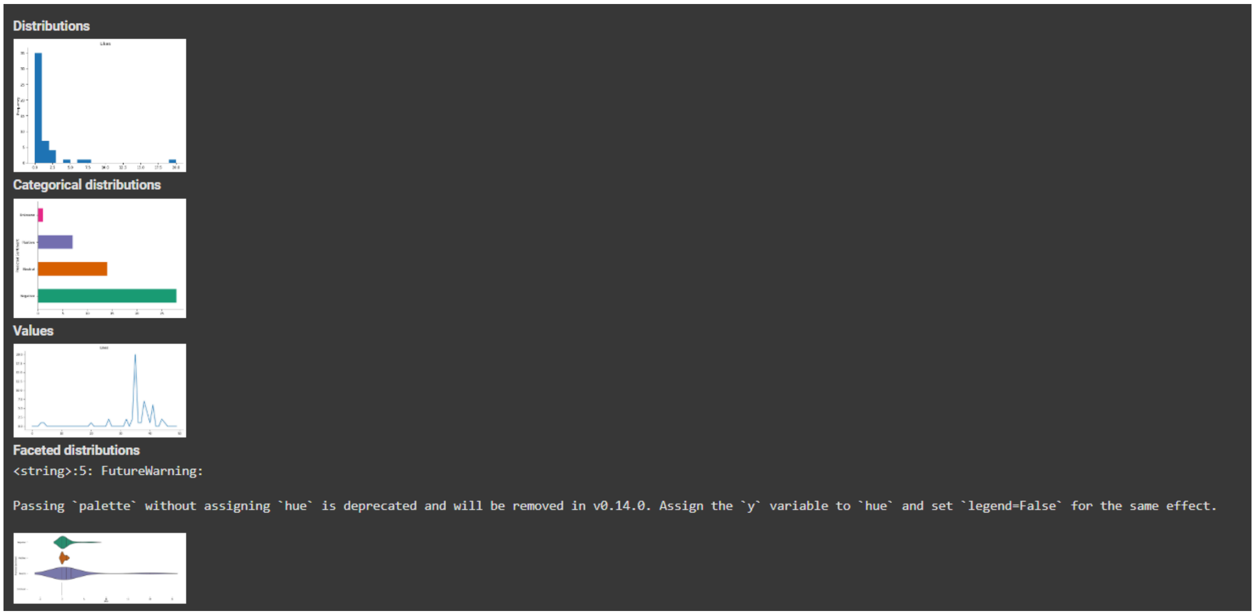
6. Results and Discussion



```
Device set to use cuda:0
Text generation pipeline ready.
Fetching comments for video ID: mTHXBofIc14

Analyzing comment 1/50: "Dudes a whole clown another opinion rejected 🤡🤡🤡..."
/usr/local/lib/python3.11/dist-packages/transformers/generation/configuration_utils.py:631: UserWarning: `do_sample` is set to `False`.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/transformers/generation/configuration_utils.py:636: UserWarning: `do_sample` is set to `False`.
  warnings.warn(
DEBUG (Simple Prompt): Raw assistant response: 'The sentiment of the text is Negative. The use'
Predicted Sentiment: Negative

Analyzing comment 2/50: "They should cast a guy in wig to play Abby. Like Terry Crews!..."
DEBUG (Simple Prompt): Raw assistant response: 'The sentiment of the text is Positive. The user'
Predicted Sentiment: Positive

Analyzing comment 3/50: "Facially, she really does remind me of Abby. I don't know how they'd find a female actress of Abby's..."
DEBUG (Simple Prompt): Raw assistant response: 'The sentiment of the given text is Neutral. The'
Predicted Sentiment: Neutral

Analyzing comment 4/50: "I just had a thought that the scene where Ellie was training against a guy that was almost twice her..."
DEBUG (Simple Prompt): Raw assistant response: 'The sentiment of the given text is Neutral. The'
Predicted Sentiment: Neutral

Analyzing comment 5/50: "But like, Abby isn’t supposed to be buff at this point in the story right? I didn’t pay too much att..."
DEBUG (Simple Prompt): Raw assistant response: 'The sentiment of the given text is Neutral. The'
Predicted Sentiment: Neutral

Analyzing comment 6/50: "I remember seeing an interview where Neil Druckmann said they didn’t bulk up Kaitlyn because the liv..."
DEBUG (Simple Prompt): Raw assistant response: 'The sentiment of the given text is Neutral. The'
Predicted Sentiment: Neutral

Analyzing comment 7/50: "Dunno about that.. Abby was getting pretty muscular before her father died...."
DEBUG (Simple Prompt): Raw assistant response: 'The sentiment of the given text is Neutral. The'
Predicted Sentiment: Neutral

Analyzing comment 8/50: "The Movies and TV shows need to thrive without reliance on the games. Rather than a strict road map ..."
DEBUG (Simple Prompt): Raw assistant response: 'The sentiment of the given text is Positive. The'
Predicted Sentiment: Positive

Analyzing comment 9/50: "I mean that's why she's such a threat to Ellie, no? Because she's buff and strong. Imagine Batman go..."
DEBUG (Simple Prompt): Raw assistant response: 'The sentiment of the given text is Negative. The'
Predicted Sentiment: Negative

Analyzing comment 10/50: "Can't defend the season 2 cast anymore bro..."
DEBUG (Simple Prompt): Raw assistant response: 'The sentiment of the text is Negative. The phrase'
Predicted Sentiment: Negative

Analyzing comment 11/50: "For whatever issues people had with the OG game - Abby being jacked played into her character and he..."
DEBUG (Simple Prompt): Raw assistant response: 'The sentiment of the given text is Positive. The'
Predicted Sentiment: Positive

Analyzing comment 12/50: "ronda rousey should casted as abby..."
DEBUG (Simple Prompt): Raw assistant response: 'The sentiment of the text is Neutral. The statement'
Predicted Sentiment: Neutral
```



```
Fetched and analyzed 50 comments.

--- Comment Sentiment Analysis Results (Using Simpler Prompt) ---
                  Author          Published At                                                                                                                                                                                                      Comment  Likes  Predicted Sentiment
0       @JusticeGamingNY  2025-04-16T14:10:56Z                                                                                                                                                                Dudes a whole clown another opinion rejected 🤡🤡🤡      0             Negative
1            @RibbonPL  2025-04-16T14:07:59Z                                                                                                                                                          They should cast a guy in wig to play Abby. Like Terry Crews!      0             Positive
2            @OKA4LIVE  2025-04-16T14:07:41Z              Facially, she really does remind me of Abby. I don't know how they'd find a female actress of Abby's size who looks even remotely similar to her facially. Or captures the essence of her personalit...      0              Neutral
3         @EmuleelArts  2025-04-16T14:05:26Z            I just had a thought that the scene where Ellie was training against a guy that was almost twice her size could've been really good subtle foreshadowing for her conflict against a much bigger Abby...      1              Neutral
4         @EmuleelArts  2025-04-16T13:59:42Z                   But like, Abby isn't supposed to be buff at this point in the story right?\nI didn't pay too much attention to all the trailers and pre-release material so if Abby is shown to be just as skinny th...      1              Neutral
5      @shannarong3475  2025-04-16T13:57:02Z     I remember seeing an interview where Neil Druckmann said they didn't bulk up Kaitlyn because the live-action Abby doesn't need to look super muscular. They're focusing more on showing her strength...      0              Neutral
6          @Zuernaxashyr  2025-04-16T13:54:45Z                                                                                                                                                 Dunno about that.. Abby was getting pretty muscular before her father died.      0              Neutral
7      @LarryHayward-v7p  2025-04-16T13:47:38Z  The Movies and TV shows need to thrive without reliance on the games.\n\nRather than a strict road map for an adaptation. As it allows for the best of both worlds.💯💯💯 it's a unique exploration of ...      0             Positive
8    @looniemoonie5955  2025-04-16T13:47:35Z          I mean that's why she's such a threat to Ellie, no?\nBecause she's buff and strong. Imagine Batman going against Bane except he is 70 yo shortie. Not much impact here (unless he still got the stre...      0             Negative
9             @adib1081  2025-04-16T13:47:22Z                                                                                                                                                                                   Can't defend the season 2 cast anymore bro      0             Negative
10    @olivercrespo2329  2025-04-16T13:46:39Z                       For whatever issues people had with the OG game - Abby being jacked played into her character and her story. It was important to the narrative it wasn't just a "let's make her jacked just cuz".      0             Positive
11              @reetep  2025-04-16T13:44:49Z                                                                                                                                                                                             ronda rousey should casted as abby      0              Neutral
12   @gandalainsley6467  2025-04-16T13:38:05Z                                                                       To get a woman to look like that she needs roids. Only transwomen can look like that without roids for obvious reasons. Not every actor is okay with using steroids.      0             Negative
13  @bonnieplushtuber5517  2025-04-16T13:36:48Z          Thank you for speaking out for this. Abby's character and design has been treated like dirt and I am guessing that's why they changed her and chose a different actress. I've always been a fan of h...      0             Negative
14          @Jacknicker2  2025-04-16T13:31:06Z                                                                                                                                                                                                   Bro just meat riding      0              Unknown
15             @wrentic  2025-04-16T13:29:28Z   what an L take.if you don't know abby's body was simmilar like elly. they also have sketeches of abby looking cool but because of higher ups they cahnged abby to muscular because they want somwone...      0             Negative
16   @pietrogalvani8820  2025-04-16T13:23:32Z       These arguments work great and make sense until you remember that the characters are in a zombie apocalypse and they literally have no food.\nSo how the f will she have a bodybuilder type of physic?      0             Negative
17        @FitWitScience  2025-04-16T13:20:53Z                                                                                                                                           The thing is no woman can put on that amount of muscle without risking her life with steroids      0             Negative
18     @AllOfficialU4Life  2025-04-16T13:17:08Z                                                                                                          I think Kaitlyn Devers should have been cast as Ellie not Abby! She has the look and the personality but i guess maybe she was a little too tall.      0             Negative
19         @Tsukki-sf8mw  2025-04-16T13:16:38Z                                                                                                                                                                                         I'm gay you're gay everyone is gay      0             Negative
20    @nicholaszancy4571  2025-04-16T13:15:14Z                                                                                                                                                                                                                 Saucy 👓      1             Negative
21      @amadoubass3406  2025-04-16T13:14:21Z               I kinda hope Caitlyn does try putting on sum muscle for the role, she dosnt have to be a one to one and it dosnt have to be right away at least towards the end, there is also the option of practic...      0              Neutral
22           @TheLotan  2025-04-16T13:12:34Z                                                                                                                   Even a bodybuilder female is comparatively weaker than an average man, but goshdarn do they look cool. Love strong women      0             Positive
23    @v_cpt-phasma_v689  2025-04-16T13:06:57Z                                     people obviously didnt like abby in the game and her male physique got alot of criticism, i just think you made that choice, now have the balls to stick to it by having a buff female actress      0             Negative
24            @Achierne  2025-04-16T13:06:54Z                                                                                                                                                        I agree because this maybe a hot take but I like Abby's built, idc about fan service      0              Neutral
25            @SciRuler  2025-04-16T13:04:57Z                                                                                                                      i do know nothing, idk if it's you or youtube but me jokingly saying i wanna fuck the obvious porn bot is not a genuine desire      0             Negative
26     @alexthegamer8433  2025-04-16T13:03:26Z                                                                                                            I agree but I think it's a little silly to be making such hasty points after one episode, we don't know how they're gonna use her for the rest of the show      2              Neutral
27  @AidenAndCraneGOATs  2025-04-16T13:02:07Z                                                                                                                                                 0:39 tv show Abby might have a lil muscle when season 3 happens and I hope that's true      0              Neutral
```



Distributions

Categorical distributions

Values

Faceted distributions

```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
```

Results:

1. **Environment and Model Setup:** The notebook successfully installs all required libraries, handles Hugging Face authentication (using both CLI login and Colab Secrets), checks for GPU availability (detecting a Tesla T4), defines configurations, and downloads the Llama 3.2-1B-Instruct model weights (using huggingface-cli).
2. **Quantization and Loading:** The 4-bit quantization configuration (BitsAndBytesConfig) is defined correctly, and the Llama 3.2-1B-Instruct

model is successfully loaded onto the GPU (cuda:0) using these settings. This confirms the feasibility of running this billion-parameter model on a resource-constrained environment like Colab with a T4 GPU.

3. **Pipeline Initialization:** Both the direct model/tokenizer loading and the transformers text-generation pipeline (text_generator) are initialized successfully, ready for inference.
4. **Initial Sample Test (Step 25 Output):**
    - The classify_sentiment_llama3 function was tested on a predefined list of 9 Tamil and Code-Mixed/English sentences.
    - **Performance:**
        - It correctly classified most clearly Positive (1, 5, 7) and Negative (2, 6, 8, 9) examples, including code-mixed and English ones.
        - It **struggled with Neutral Tamil sentences** (3, 4), misclassifying them as Negative. This suggests the model, with this specific zero-shot prompt, might have difficulty distinguishing neutrality from negativity in pure Tamil, or the prompt might implicitly bias it towards non-positive classifications if strong positive cues are absent.
5. **YouTube Comment Fetching:**
    - The script successfully uses the YouTube Data API v3 to fetch comments for the provided video URL (https://youtu.be/mTHXBofIc14?si=ONSBrajTQSE2U11R).
    - It fetches the specified number of comments (50 in this run) and extracts relevant metadata (author, text, likes, timestamp).
    - Basic text cleaning (removing extra whitespace) is applied.
6. **YouTube Comment Sentiment Analysis (Step 26 Outputs):**
    - The classify_sentiment_simple_prompt function (a slightly refined version for the YouTube task) is applied to each fetched comment.
    - **Performance:** Based on the console output snippets (Page 22/23):
        - The model appears to classify many comments (which seem predominantly English or code-mixed in the sample) reasonably well across Positive, Negative, and Neutral categories.
        - The debug output (DEBUG (Simple Prompt): Raw assistant response: ...) shows the model generally adhering to the

        requested format (e.g., "The sentiment of the text is Negative.").
        - There is at least one instance (Comment 15 in the DataFrame on Page 23) where the predicted sentiment is "Unknown". This indicates the parsing logic within the function failed to extract a clear "Positive", "Negative", or "Neutral" keyword from the model's raw output for that specific comment.
    -
    - **Final Output:** The results are successfully compiled into a Pandas DataFrame, displaying the comment, author, publication date, likes, and the *predicted* sentiment label. Visualizations showing the distribution of sentiments were also generated.

Discussion:

1. **Effectiveness of Llama 3 for Zero-Shot Sentiment Analysis:** The results demonstrate that even a relatively small version (1B parameters) of Llama 3 (specifically the 3.2-Instruct variant) possesses significant capability for zero-shot sentiment classification in Tamil and particularly code-mixed Tamil-English text, guided solely by prompting. It leverages its pre-trained knowledge to understand the task and the sentiment expressed.
2. **Impact of Quantization:** The successful loading and execution using 4-bit quantization (NF4) are crucial. This technique drastically reduces the memory requirements, making it practical to run inference for a model of this size on readily available hardware like Colab GPUs. Without quantization, running this model would likely be impossible in such an environment. While not quantitatively measured here, 4-bit quantization typically comes with only a minor, often negligible, drop in performance for many tasks.
3. **Prompt Sensitivity and Nuance:** The difference in performance between the initial Tamil sample test (struggling with Neutral Tamil) and the YouTube comment analysis (handling Neutral better, but with parsing failures) highlights the sensitivity of LLMs to prompt design. Even subtle changes in the prompt structure or the nature of the input text can influence the output. The model's difficulty with purely Tamil neutral sentences suggests that its pre-training might have weaker representations for neutral linguistic cues in

Tamil compared to English, or the prompt needs further refinement for this specific language/nuance.

4. **Handling Code-Mixing:** The model seems adept at handling code-mixed text (Tamil-English), which is common in online discussions and was a key challenge mentioned in the problem statement. This is a strength of modern LLMs trained on diverse web data.
5. **Robustness of Output Parsing:** The appearance of "Unknown" sentiment predictions underscores a limitation of relying on simple keyword extraction (re.search) from the LLM's free-form text output. Sometimes the model might phrase its response slightly differently, fail to follow instructions perfectly, or generate ambiguous text, causing the parsing to fail. More robust methods might involve requesting structured output (like JSON) if the model supports it well, or using more sophisticated parsing logic.
6. **Limitations:**
   - **Zero-Shot Constraint:** Performance is inherently limited by the model's pre-training and the quality of the prompt. It may not perform as well as a model fine-tuned specifically on a Tamil/Code-Mixed sentiment dataset.
   - **No Ground Truth:** The analysis relies on qualitative assessment of the predictions. There's no comparison against a labeled dataset to calculate quantitative metrics (Accuracy, Precision, Recall, F1-score).
   - **Bias:** LLMs can inherit biases from their training data, which might affect sentiment predictions for certain topics or demographics.
7. **Practical Application:** The project successfully demonstrates a practical workflow: fetching real-world data (YouTube comments) via API, processing it, and applying an LLM for analysis. This showcases the potential of using prompted LLMs for quick analysis tasks where extensive fine-tuning might be overkill or infeasible.

7. Conclusion

This project aimed to fine-tune a Llama 3 Instruct model for Tamil-English code-mixed sentiment analysis using the efficient QLoRA technique on the DravidianCodeMix dataset. The notebook successfully outlines the configuration for the base model (meta-llama/Llama-3.1-8B-Instruct in the final config), the target dataset, PEFT parameters (LoRA), quantization settings (4-bit), data preprocessing steps (label mapping, filtering), and the training pipeline using trl.SFTTrainer.

Limitations:

1. **Execution Failure:** The primary limitation is that the core fine-tuning process could not be executed due to model and limited datasets.
2. **Model/Tool Inconsistency:** The notebook shows exploration of different Llama 3 variants and loading methods (transformers vs. llama-stack), introducing some confusion about the final target setup before settling on the SFT configuration.
3. **Evaluation:** Without a trained model, no quantitative evaluation could be performed.
4. **Resource Constraints:** Even with QLoRA, fine-tuning on a T4 GPU might be slow or prone to OOM errors, potentially limiting the choice of batch size or sequence length.

Enhancements:

1. **Verify Data Processing:** Once the dataset is loaded, verify the label mapping and filtering logic produces the expected training/validation splits with the correct format.
2. **Execute Training:** Run the SFTTrainer pipeline after resolving access issues.
3. **Systematic Evaluation:** Implement a compute_metrics function (using evaluate library) to calculate accuracy, F1-score, etc., on the validation (and potentially test) set during/after training.
4. **Hyperparameter Tuning:** Experiment with different LoRA ranks, learning rates, batch sizes, and epochs to optimize performance.
5. **Baseline Comparison:** Quantitatively compare the fine-tuned model's performance against the zero-shot baseline (once the zero-shot setup is fixed and runnable).
6. **Error Analysis:** Analyze misclassifications made by the fine-tuned model to understand its weaknesses.

8. References

- **Libraries:**
  - **Hugging Face Transformers: https://huggingface.co/docs/transformers**
  - **Hugging Face Datasets: https://huggingface.co/docs/datasets**
  - **Hugging Face PEFT: https://huggingface.co/docs/peft**
  - **Hugging Face TRL: https://huggingface.co/docs/trl**
  - **BitsAndBytes: https://github.com/TimDettmers/bitsandbytes**
  - **Accelerate: https://huggingface.co/docs/accelerate**
  - **PyTorch: https://pytorch.org/**
  - **llama-stack: https://github.com/meta-llama/llama-recipes/tree/main/src/llama_recipes/inference/api_server_tools/api_server_client (or relevant source)**
- **Models:**
  - **Meta Llama 3: https://huggingface.co/meta-llama (Specific model pages linked from here, e.g., meta-llama/Llama-3.1-8B-Instruct, meta-llama/Llama-3.2-3B-Instruct)**
- **Dataset:**
  - **DravidianCodeMix Dataset: https://huggingface.co/datasets/DravidianCodeMix/DravidianCodeMix-Dataset**
- **Techniques:**
  - **LoRA: Hu, Edward J., et al. "LoRA: Low-Rank Adaptation of Large Language Models." arXiv preprint arXiv:2106.09685 (2021).**
  - **QLoRA: Dettmers, Tim, et al. "QLoRA: Efficient Finetuning of Quantized LLMs." arXiv preprint arXiv:2305.14314 (2023).**