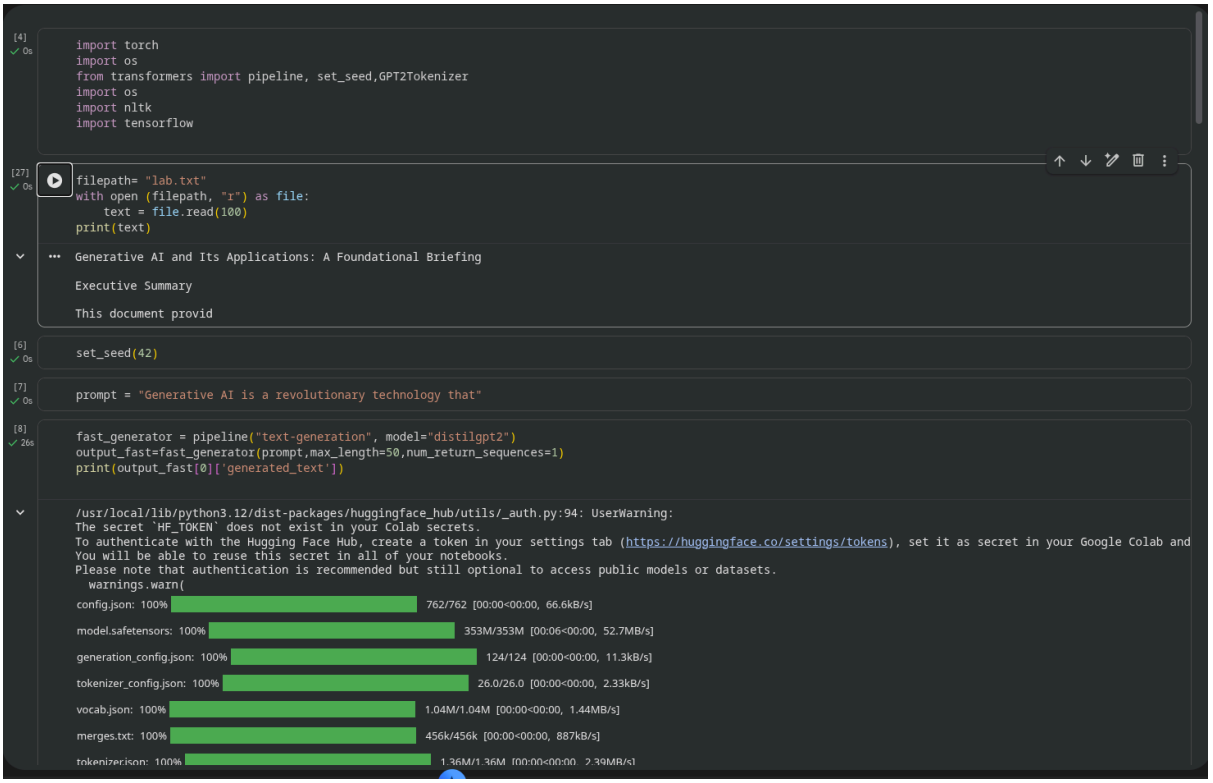
Gen AI Hands On- 1

Name: Suchitra Shankar
Class: 6J
SRN: PES2UG23CS608
Date: 27 Jan 2026
Email: suchitrashankar07@gmail.com
Phone number: 6366250274
Github link: https://github.com/SuchitraShankar07/GenAI2026

Submission 1: Hands on done in class



```
[4]
import torch
import os
from transformers import pipeline, set_seed,GPT2Tokenizer
import os
import nltk
import tensorflow

[27]
filepath= "lab.txt"
with open (filepath, "r") as file:
    text = file.read(100)
print(text)

Generative AI and Its Applications: A Foundational Briefing

Executive Summary

This document provid

[6]
set_seed(42)

[7]
prompt = "Generative AI is a revolutionary technology that"

[8]
fast_generator = pipeline("text-generation", model="distilgpt2")
output_fast=fast_generator(prompt,max_length=50,num_return_sequences=1)
print(output_fast[0]['generated_text'])

/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json: 100% 762/762 [00:00<00:00, 66.6kB/s]
model.safetensors: 100% 353M/353M [00:06<00:00, 52.7MB/s]
generation_config.json: 100% 124/124 [00:00<00:00, 11.3kB/s]
tokenizer_config.json: 100% 26.0/26.0 [00:00<00:00, 2.33kB/s]
vocab.json: 100% 1.04M/1.04M [00:00<00:00, 1.44MB/s]
merges.txt: 100% 456k/456k [00:00<00:00, 887kB/s]
tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 2.39MB/s]
```

```
merges.txt: 100% 456k/456k [00:00<00:00, 887kB/s]
tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 2.39MB/s]
Device set to use cpu
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Both `max_new_tokens` (=256) and `max_length`(=50) seem to have been set. `max_new_tokens` will take precedence. Please refer to the documentation for more inform
Generative AI is a revolutionary technology that is designed to work with existing AI systems. It has been developed by the University of California, Berkeley. It


The research team led by Professor Daniel Kranz, from the University of California, Berkeley, has developed a program to learn how to use the AI to improve the pe
The research team developed the program to learn how to use the AI to improve the performance of the software. It has been developed by the University of Californ
The research team developed the program to learn how to use the AI to improve the performance of the software. It has been developed by
```

[9] ✓ 1m

```
fast_generator=pipeline("text-generation",model="gpt2")
output_fast=fast_generator(prompt,max_length=100,num_return_sequences=2)
print(output_fast[0]['generated_text'])
```

```
config.json: 100% 665/665 [00:00<00:00, 53.3kB/s]
model.safetensors: 100% 548M/548M [00:06<00:00, 63.6MB/s]
generation_config.json: 100% 124/124 [00:00<00:00, 4.11kB/s]
tokenizer_config.json: 100% 26.0/26.0 [00:00<00:00, 1.09kB/s]
vocab.json: 100% 1.04M/1.04M [00:00<00:00, 1.45MB/s]
merges.txt: 100% 456k/456k [00:00<00:00, 878kB/s]
tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 1.69MB/s]
Device set to use cpu
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Both `max_new_tokens` (=256) and `max_length`(=100) seem to have been set. `max_new_tokens` will take precedence. Please refer to the documentation for more infor
Generative AI is a revolutionary technology that allows an AI to be used to solve any problem. For example, it is one of the most effective ways to train a human

The goal of the AI is to learn how to respond to an example. By knowing the basic mental models of human behavior, it can then apply those models to complex probl

The AI has its own set of problems. For example, it can solve problems that would take a human brain and a computer to solve.

This is how intelligent AI, or AI as a whole, can be used in today's tech world.

This technology will allow an AI to learn to respond to a problem, and to learn how to solve it.

This is a great way to build software that has the power to solve problems — to solve problems for a human, or for a machine.

This is a great way to build software that has the power to solve problems — to solve problems for a human, or for a machine.

I want to say that this is a great demonstration of the potential of AI to be used in the real world.
```

Tokenization

[10] ✓ 1s

```
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
```

[11] ✓ 0s

```
sample_sentence="The quick brown fox jumped over the lazy dog."
```

[12] ✓ 0s

```
tokens = tokenizer.tokenize(sample_sentence)
print(f"Tokens:{tokens}")
```

```
Tokens:['The', 'Ġquick', 'Ġbrown', 'Ġfox', 'Ġjumped', 'Ġover', 'Ġthe', 'Ġlazy', 'Ġdog', '.']
```

[13] ✓ 0s

```
token_ids = tokenizer.convert_tokens_to_ids(tokens)
print(f"Token IDs: {token_ids}")
```

```
Token IDs: [464, 2068, 7586, 21831, 11687, 625, 262, 16931, 3290, 13]
```

POS tagging: part of speech

[14] ✓ 0s

```
nltk.download('averaged_perceptron_tagger_eng', quiet = True)
```

```
True
```

[15] ✓ 0s

```
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

[16] ✓ 0s

```
pos_tags = nltk.pos_tag(nltk.word_tokenize(sample_sentence))
print(f"POS Tags: {pos_tags}")
#
```

```
POS Tags: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumped', 'VBD'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN'), ('.',
```

[17] ✓ 40s

```
ner_pipeline=pipeline("ner", model = "dbmdz/bert-large-cased-finetuned-conll03-english", aggregation_strategy="simple")
ner_pipeline("My name is Sylvain and I work at Hugging Face")
```

```
config.json: 100% 998/998 [00:00<00:00, 38.6kB/s]
model.safetensors: 100% 1.33G/1.33G [00:36<00:00, 32.4MB/s]
```

```
Some weights of the model checkpoint at dbmdz/bert-large-cased-finetuned-conll03-english were not used when initializing BertForTokenClassification: ['bert.pooler
- This IS expected if you are initializing BertForTokenClassification from the checkpoint of a model trained on another task or with another architecture (e.g. in
- This IS NOT expected if you are initializing BertForTokenClassification from the checkpoint of a model that you expect to be exactly identical (initializing a B
tokenizer_config.json: 100% 60.0/60.0 [00:00<00:00, 5.11kB/s]

vocab.txt: 213k/? [00:00<00:00, 9.96MB/s]
Device set to use cpu
[{'entity_group': 'PER',
  'score': np.float32(0.99846023),
  'word': 'Sylvain',
  'start': 11,
  'end': 18},
 {'entity_group': 'ORG',
  'score': np.float32(0.9917064),
  'word': 'Hugging Face',
  'start': 33,
  'end': 45}]
```

[18] ✓ 2s

```
snippet=text[:1000]
entities = ner_pipeline(snippet)
print(f"{'Entity':<25} | {'Type':<10} | {'Score':<10}")
print("-"*50)
for entity in entities:
  if entity['score']>0.90:
    print(f"{entity['word']:<25} | {entity['entity_group']:<10} | {entity['score']:.2f}")
```

```
Entity                    | Type       | Score
--------------------------------------------------
AI                        | MISC       | 0.98
PES University            | ORG        | 0.99
AI                        | MISC       | 0.98
Large Language Models     | MISC       | 0.91
LLMs                      | MISC       | 0.90
Transformer               | MISC       | 0.99
```

Comparative analysis

[19] ✓ 0s

```
transformer_section = '''

1. Pretraining: This is the initial, computationally expensive phase. The model is trained on a massive, diverse corpus of raw text (billions of tokens from book
2. Finetuning: In this stage, the pre-trained base model is further trained on a smaller, curated dataset specific to a particular task (e.g., medical question a

'''
```

[20] ✓ 1m

```
fast_sum = pipeline("summarization", model="sshleifer/distilbart-cnn-12-6")
res_fast = fast_sum(transformer_section, max_length=100, min_length=30, do_sample=False)
print(res_fast[0]['summary_text'])
```

```
config.json: 1.80k/? [00:00<00:00, 118kB/s]
pytorch_model.bin: 100% 1.22G/1.22G [00:49<00:00, 25.7MB/s]
model.safetensors: 100% 1.22G/1.22G [00:50<00:00, 4.40MB/s]
tokenizer_config.json: 100% 26.0/26.0 [00:00<00:00, 222B/s]
vocab.json: 899k/? [00:00<00:00, 8.29MB/s]
merges.txt: 456k/? [00:00<00:00, 3.91MB/s]
Device set to use cpu
 The model is trained on a massive, diverse corpus of raw text (billions of tokens from books, websites, etc.) The goal is to develop a broad, general understandi
```

[21] ✓ 1m

```
smart_sum = pipeline("summarization", model="facebook/bart-large-cnn")
res_smart = smart_sum(transformer_section, max_length=60, min_length=30, do_sample=False)
print(res_smart[0]['summary_text'])
```

```
config.json: 1.58k/? [00:00<00:00, 44.1kB/s]
model.safetensors: 100% 1.63G/1.63G [01:39<00:00, 23.5MB/s]
generation_config.json: 100% 363/363 [00:00<00:00, 25.0kB/s]
vocab.json: 899k/? [00:00<00:00, 30.9MB/s]
merges.txt: 456k/? [00:00<00:00, 17.4MB/s]
tokenizer.json: 1.36M/? [00:00<00:00, 35.3MB/s]
Device set to use cpu
The model is trained on a massive, diverse corpus of raw text (billions of tokens from books, websites, etc.) The goal is to develop a broad, general understandin
```

[22] ✓ 19s

```
qa_pipeline = pipeline("question-answering", model="distilbert-base-cased-distilled-squad")
```

```
config.json: 100% 473/473 [00:00<00:00, 34.4kB/s]
model.safetensors: 100% 261M/261M [00:10<00:00, 28.4MB/s]
tokenizer_config.json: 100% 49.0/49.0 [00:00<00:00, 3.05kB/s]
vocab.txt: 100% 213k/213k [00:00<00:00, 1.16MB/s]
tokenizer.json: 100% 436k/436k [00:00<00:00, 1.32MB/s]
Device set to use cpu
```

```
Device set to use cpu
```

```
[23] ✓ 6s
questions = [
    "What is the fundamental innovation of the Transformer?",
    "What are the risks of using Generative AI?"
]

for q in questions:
    res = qa_pipeline(question=q, context=text[:5000])
    print(f"\nQ: {q}")
    print(f"A: {res['answer']}")
```

```
Q: What is the fundamental innovation of the Transformer?
A: to identify hidden patterns, structures, and relationships within the data

Q: What are the risks of using Generative AI?
A: data privacy, intellectual property, and academic integrity
```

```
[24] ✓ 15s
mask_filler = pipeline("fill-mask", model="bert-base-uncased")
```

```
config.json: 100% 570/570 [00:00<00:00, 51.0kB/s]
model.safetensors: 100% 440M/440M [00:11<00:00, 28.2MB/s]
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertForMaskedLM: ['bert.pooler.dense.bias', 'bert.pooler.dense.weight',
- This IS expected if you are initializing BertForMaskedLM from the checkpoint of a model trained on another task or with another architecture (e.g. initializing
- This IS NOT expected if you are initializing BertForMaskedLM from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSeque
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 3.98kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 3.31MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 1.29MB/s]
Device set to use cpu
```

```
[25] ✓ 0s
masked_sentence = "The goal of Generative AI is to create new [MASK]."
preds = mask_filler(masked_sentence)

for p in preds:
    print(f"{p['token_str']}: {p['score']:.2f}")
```

```
applications: 0.06
ideas: 0.05
problems: 0.05
systems: 0.04
information: 0.03
```

Submission 2: Assignment: comparative analysis of BERT, BART, RoBERTa (filled in the ipynb file and copy pasted from there)

```
[9] ✓ 0s
from transformers import pipeline, set_seed
set_seed(42)
```

```
[10] ✓ 0s
models = {
    "BERT": "bert-base-uncased",
    "RoBERTa": "roberta-base",
    "BART": "facebook/bart-base"
}
```

Experiment 1: Text Generation

```
[11] ✓ 0s
prompt = "The future of Artificial Intelligence is"
```

```
[12] ✓ 10s
for name, model_id in models.items():
    print(f"\n=== {name} ===")
    try:
        generator = pipeline("text-generation", model=model_id)
        out = generator(prompt, max_new_tokens=30)
        print(out[0]["generated_text"])
    except Exception as e:
        print("ERROR:", e)
```

```
=== BERT ===
If you want to use `BertLMHeadModel` as a standalone, add `is_decoder=True.`
Device set to use cpu
The future of Artificial Intelligence is.............................

=== RoBERTa ===
If you want to use `RobertaLMHeadModel` as a standalone, add `is_decoder=True.`
Device set to use cpu
The future of Artificial Intelligence is

=== BART ===
Some weights of BartForCausalLM were not initialized from the model checkpoint at facebook/bart-base and are newly initialized: ['lm_head.weight', 'model.decoder.
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Device set to use cpu
The future of Artificial Intelligence is0303 apart03 humankind03 housing stimuli11110303 discriminatory discriminatory discriminatoryEStreamFrame discriminatory03
```

Masked Language modeling

```
masked_sentence = "The goal of Generative AI is to [MASK] new content."
print(f"\n === BERT ===")
try:
    masker = pipeline("fill-mask", model="bert-base-uncased")
    preds = masker(masked_sentence)
    for p in preds[:5]:
        print(f"{p['token_str']} ({p['score']:.3f})")
except Exception as e:
    print("ERROR:", e)
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertForMaskedLM: ['bert.pooler.dense.bias', 'bert.pooler.dense.weight',
- This IS expected if you are initializing BertForMaskedLM from the checkpoint of a model trained on another task or with another architecture (e.g. initializing
- This IS NOT expected if you are initializing BertForMaskedLM from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSeque

 === BERT ===
Device set to use cpu
create (0.540)
generate (0.156)
produce (0.054)
develop (0.045)
add (0.018)
```

```
masked_sentence = "The goal of Generative AI is to <mask> new content."
print(f"\n === RoBERTa ===")
try:
    masker = pipeline("fill-mask", model="roberta-base")
    preds = masker(masked_sentence)
    for p in preds[:5]:
        print(f"{p['token_str']} ({p['score']:.3f})")
except Exception as e:
    print("ERROR:", e)
```

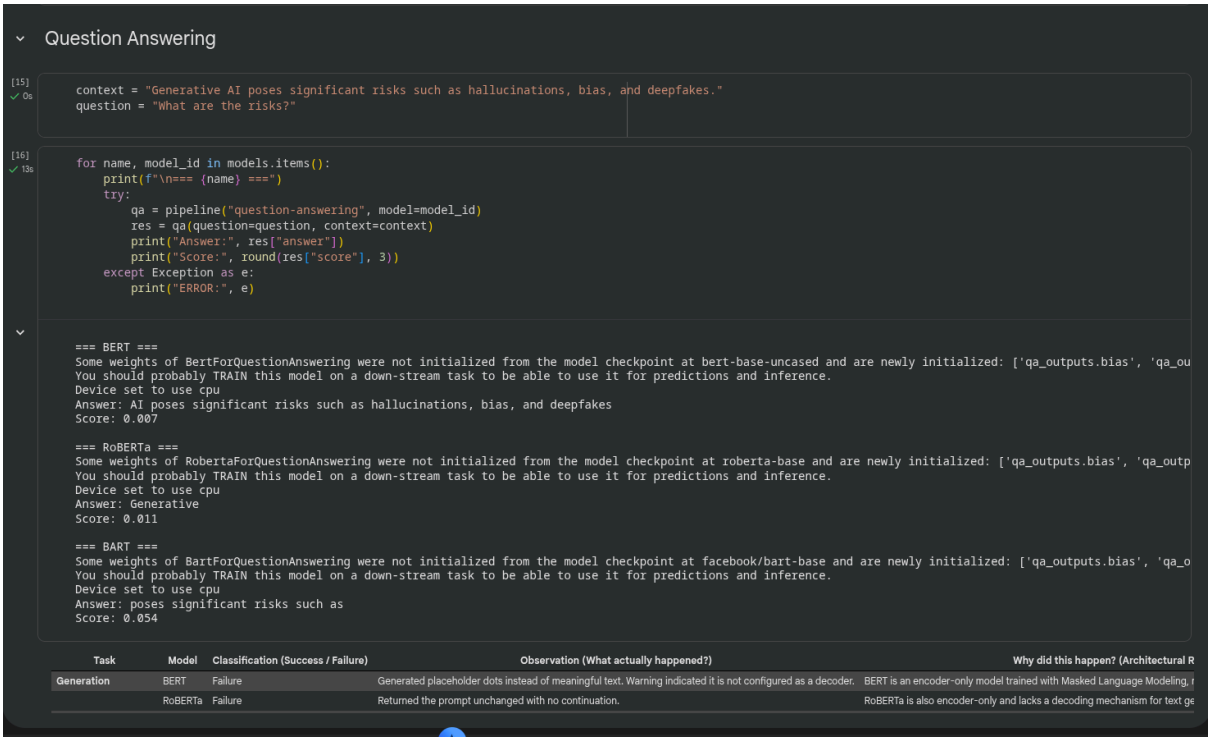
```
 === RoBERTa ===
Device set to use cpu
 generate (0.371)
 create (0.368)
 discover (0.084)
 find (0.021)
 provide (0.017)
```

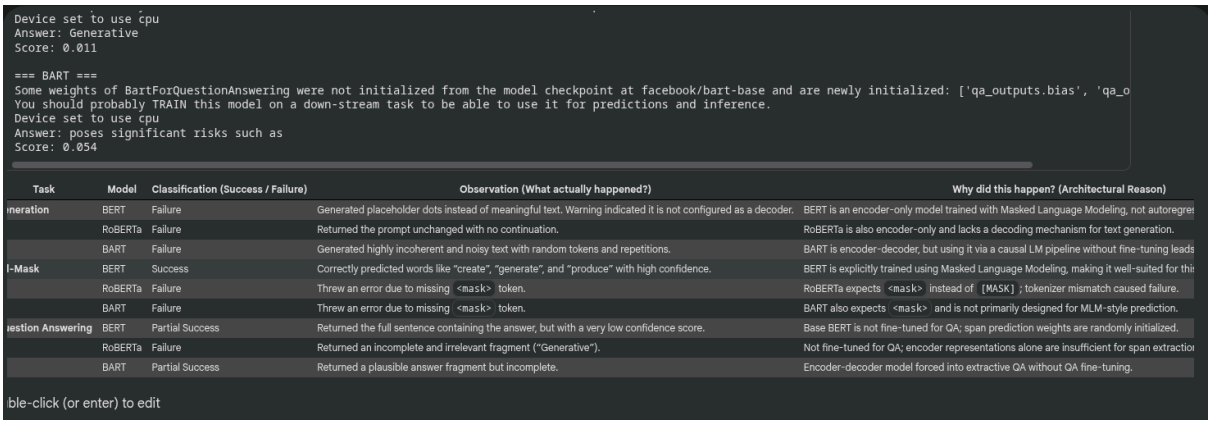
```
masked_sentence = "The goal of Generative AI is to <mask> new content."
print(f"\n === BART ===")
try:
    masker = pipeline("fill-mask", model="facebook/bart-base")
    preds = masker(masked_sentence)
    for p in preds[:5]:
        print(f"{p['token_str']} ({p['score']:.3f})")
except Exception as e:
    print("ERROR:", e)
```

```
 === BART ===
Device set to use cpu
 create (0.075)
 help (0.066)
 provide (0.061)
 enable (0.036)
 improve (0.033)
```

Question Answering

```
context = "Generative AI poses significant risks such as hallucinations, bias, and deepfakes."
question = "What are the risks?"
```



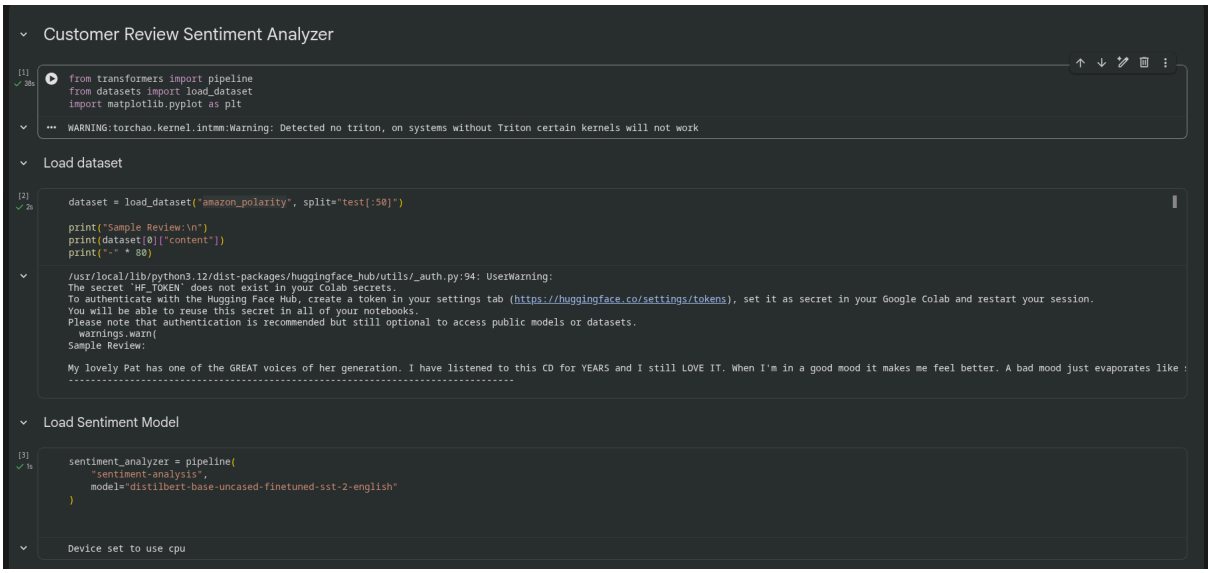


Task	Model	Classification (Success / Failure)	Observation (What actually happened?)	Why did this happen? (Architectural Reason)
Generation	BERT	Failure	Generated placeholder dots instead of meaningful text. Warning indicated it is not configured as a decoder.	BERT is an encoder-only model trained with Masked Language Modeling, not autoregressive next-token prediction.
	RoBERTa	Failure	Returned the prompt unchanged with no continuation.	RoBERTa is also encoder-only and lacks a decoding mechanism for text generation.
	BART	Failure	Generated highly incoherent and noisy text with random tokens and repetitions.	BART is encoder-decoder, but using it via a causal LM pipeline without fine-tuning leads to unstable generation.
Fill-Mask	BERT	Success	Predicted correct verbs like create and generate with high confidence.	BERT is trained using masked language modeling, so it is naturally good at predicting missing words.

	RoBERTa	Success	Gave meaningful predictions such as generate and create with similar scores.	RoBERTa is similar to BERT and is also trained with masked language modeling, but it requires the <mask> token.
	BART	Success	Predicted reasonable words like create and help, but with lower confidence compared to BERT and RoBERTa.	BART is mainly trained for sequence-to-sequence tasks, so fill-mask works but is not its main strength.
Question Answering	BERT	Partial Success	Returned the full sentence containing the answer, but with a very low confidence score.	Base BERT is not fine-tuned for QA; span prediction weights are randomly initialized.
	RoBERTa	Failure	Returned an incomplete and irrelevant fragment (“Generative”).	Not fine-tuned for QA; encoder representations alone are insufficient for span extraction.
	BART	Partial Success	Returned a plausible answer fragment but incomplete.	Encoder-decoder model forced into extractive QA without QA fine-tuning.

PROJECT

Customer Sentiment Analysis:



Run Sentiment Analysis

```
texts = [
    str(text)
    for text in dataset["content"]
    if isinstance(text, str) and text.strip() != ""
]
```

```
results = sentiment_analyzer(
    texts,
    truncation=True,
    batch_size=8
)
```

Predict!

```
positive = 0
negative = 0

print("\nSentiment Results:\n")

for i, result in enumerate(results):
    label = result["label"]
    score = result["score"]

    if label == "POSITIVE":
        positive += 1
    else:
        negative += 1

    print(f"Review {i+1}: {label} ({score:.2f})")
```

```
Sentiment Results:

Review 1: NEGATIVE (0.55)
Review 2: POSITIVE (1.00)
Review 3: NEGATIVE (1.00)
Review 4: NEGATIVE (0.98)
Review 5: POSITIVE (0.97)
Review 6: NEGATIVE (1.00)
Review 7: NEGATIVE (1.00)
Review 32: POSITIVE (0.86)
Review 33: POSITIVE (1.00)
Review 34: POSITIVE (1.00)
Review 35: POSITIVE (1.00)
Review 36: NEGATIVE (1.00)
Review 37: POSITIVE (1.00)
Review 38: NEGATIVE (1.00)
Review 39: POSITIVE (1.00)
Review 40: POSITIVE (0.99)
Review 41: POSITIVE (0.55)
Review 42: NEGATIVE (1.00)
Review 43: POSITIVE (0.98)
Review 44: POSITIVE (1.00)
Review 45: POSITIVE (1.00)
Review 46: POSITIVE (0.85)
Review 47: NEGATIVE (1.00)
Review 48: NEGATIVE (1.00)
Review 49: NEGATIVE (1.00)
Review 50: POSITIVE (1.00)
```

Summary Statistics

```
print("\n" + "=" * 50)
print("OVERALL SENTIMENT SUMMARY")
print("=" * 50)

print(f"Positive Reviews: {positive}")
print(f"Negative Reviews: {negative}")
```

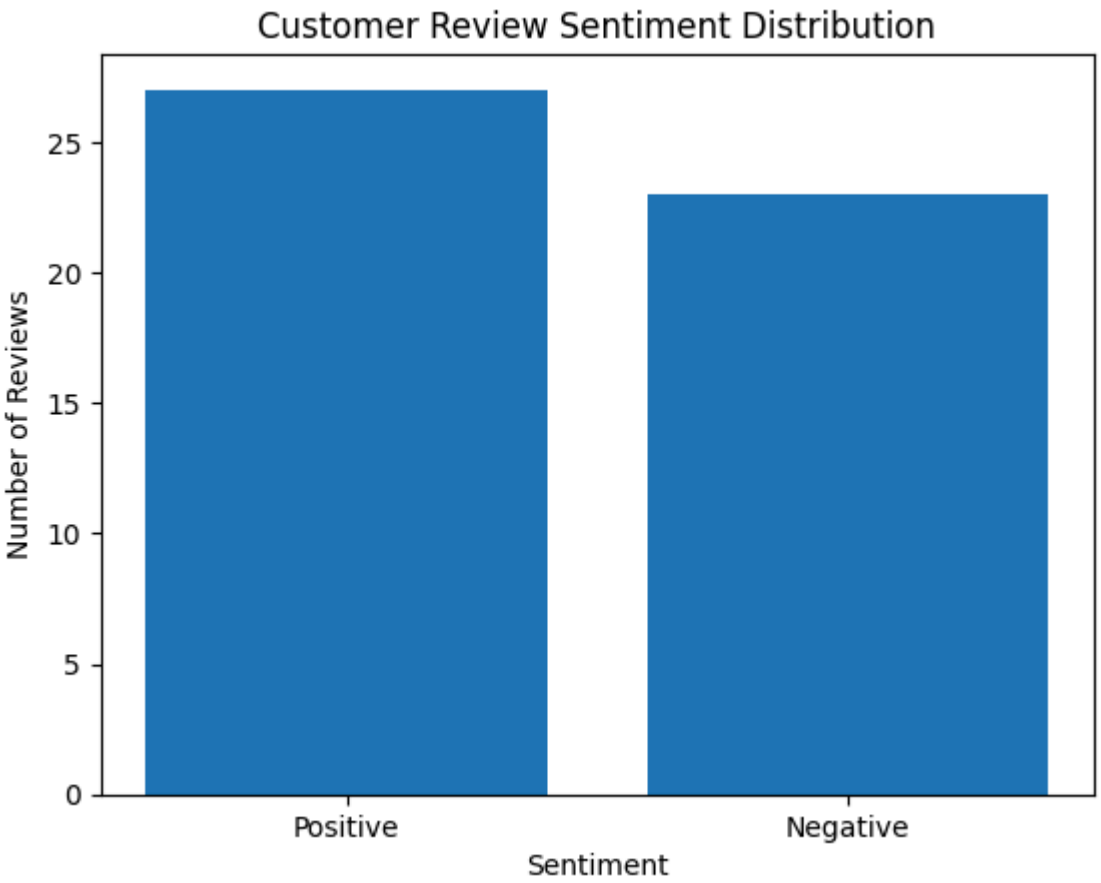
```
==================================================
OVERALL SENTIMENT SUMMARY
==================================================
Positive Reviews: 27
Negative Reviews: 23
```

Visualization

```
labels = ["Positive", "Negative"]
counts = [positive, negative]

plt.bar(labels, counts)
plt.title("Customer Review Sentiment Distribution")
plt.xlabel("Sentiment")
plt.ylabel("Number of Reviews")
plt.show()
```

Visualization:


Customer Review Sentiment Distribution
Number of Reviews
0
5
10
15
20
25
Positive
Negative
Sentiment