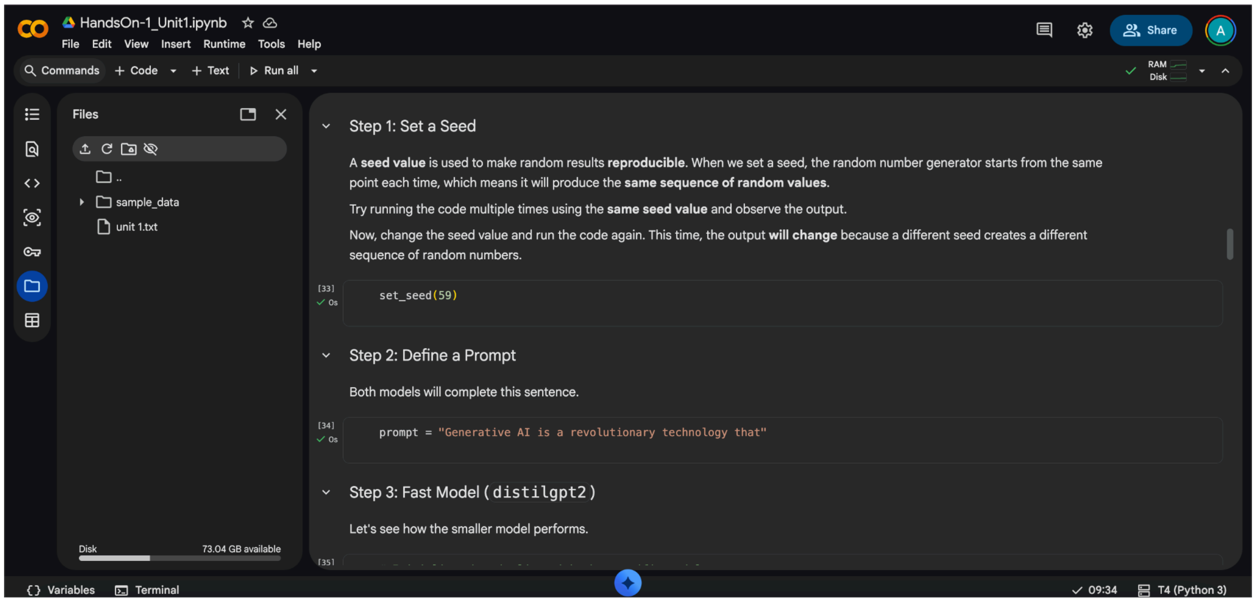
GEN AI HANDS ON OBSERVATIONS

NAME: Ananya Pandurang Prabhu
SECTION: A
SRN: PES2UG23CS063

- Hugging face is a library famous for carrying out nlp tasks like text generation, text to image, etc.
- The transformers library is used to load and make use of pre trained models like gpt 2 in this lab.
- The pipeline() function is used to abstract away the complex math and processing into three simple steps:
- Preprocessing: Converts your raw text into numbers that the model can understand.
- Model Inference: The model processes the numbers and outputs predictions.
- Post-processing: The raw predictions are converted back into human-readable text.



Step 1: Set a Seed

A **seed value** is used to make random results **reproducible**. When we set a seed, the random number generator starts from the same point each time, which means it will produce the **same sequence of random values**.

Try running the code multiple times using the **same seed value** and observe the output.

Now, change the seed value and run the code again. This time, the output **will change** because a different seed creates a different sequence of random numbers.

```
[33] set_seed(59)
```

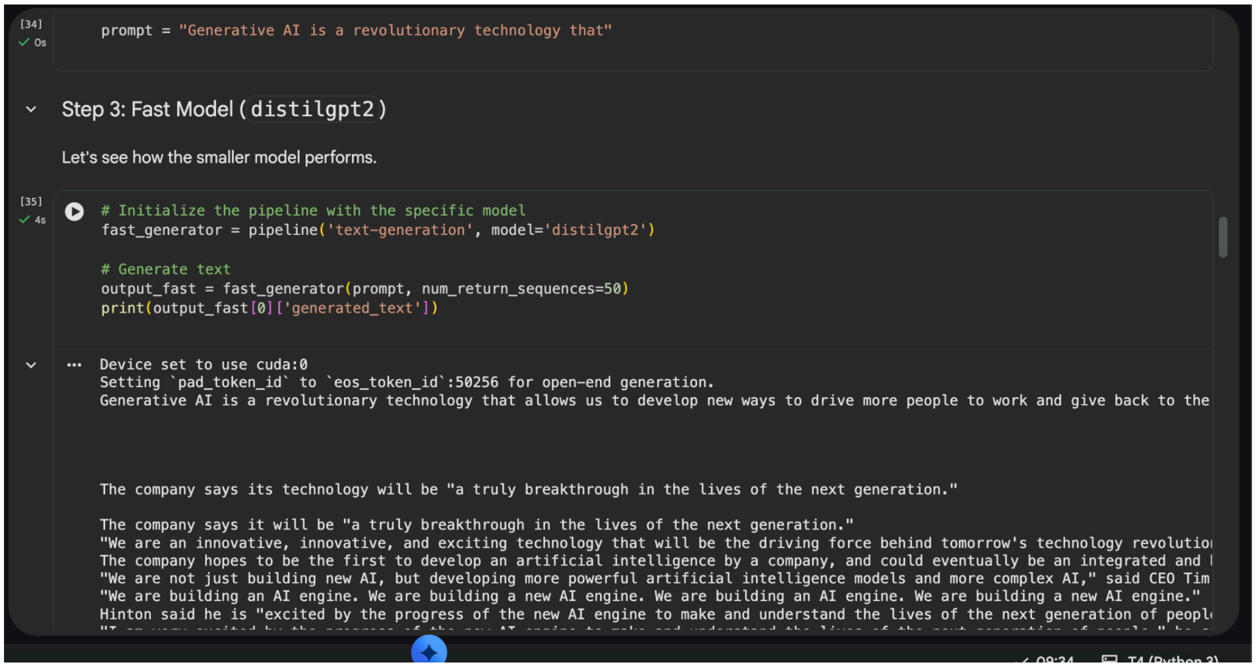
Step 2: Define a Prompt

Both models will complete this sentence.

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

Step 3: Fast Model (`distilgpt2`)

Let's see how the smaller model performs.



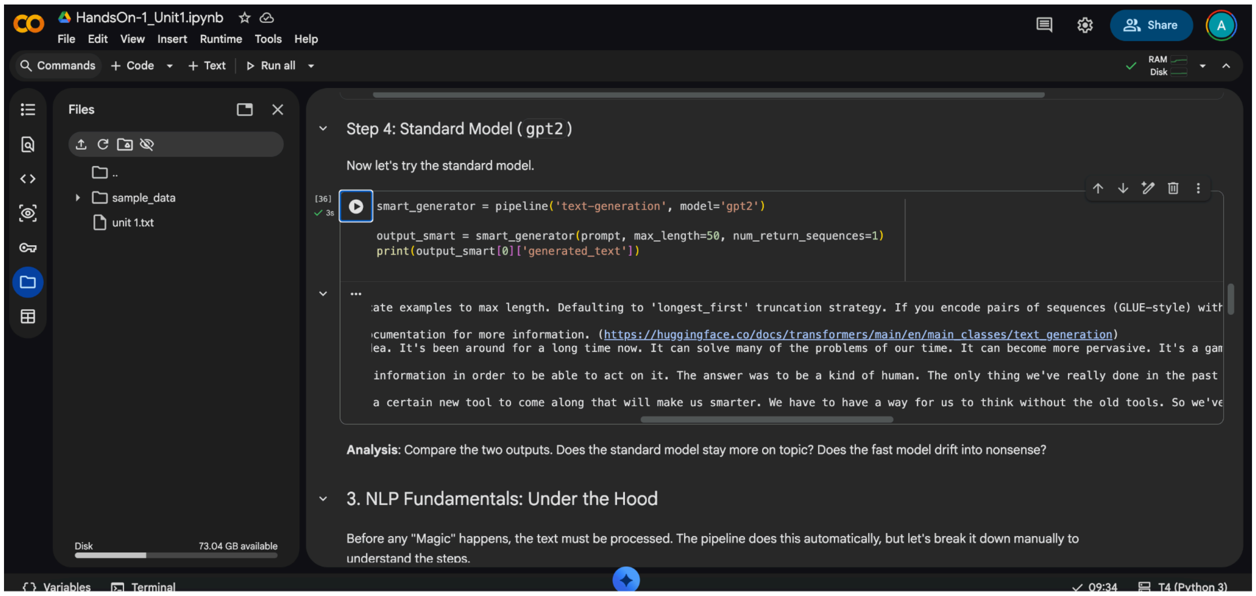
```
[34]
prompt = "Generative AI is a revolutionary technology that"
```

Step 3: Fast Model (`distilgpt2`)

Let's see how the smaller model performs.

```
[35]
# Initialize the pipeline with the specific model
fast_generator = pipeline('text-generation', model='distilgpt2')

# Generate text
output_fast = fast_generator(prompt, num_return_sequences=50)
print(output_fast[0]['generated_text'])
```

```
Device set to use cuda:0
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Generative AI is a revolutionary technology that allows us to develop new ways to drive more people to work and give back to the



The company says its technology will be "a truly breakthrough in the lives of the next generation."

The company says it will be "a truly breakthrough in the lives of the next generation."
"We are an innovative, innovative, and exciting technology that will be the driving force behind tomorrow's technology revolutio
The company hopes to be the first to develop an artificial intelligence by a company, and could eventually be an integrated and 
"We are not just building new AI, but developing more powerful artificial intelligence models and more complex AI," said CEO Tim
"We are building an AI engine. We are building a new AI engine. We are building an AI engine. We are building a new AI engine."
Hinton said he is "excited by the progress of the new AI engine to make and understand the lives of the next generation of peopl
```



Step 4: Standard Model (`gpt2`)

Now let's try the standard model.

```
[36]
smart_generator = pipeline('text-generation', model='gpt2')

output_smart = smart_generator(prompt, max_length=50, num_return_sequences=1)
print(output_smart[0]['generated_text'])
```

```
ate examples to max length. Defaulting to 'longest_first' truncation strategy. If you encode pairs of sequences (GLUE-style) with

cumentation for more information. (https://huggingface.co/docs/transformers/main/en/main_classes/text_generation)
lea. It's been around for a long time now. It can solve many of the problems of our time. It can become more pervasive. It's a gam

information in order to be able to act on it. The answer was to be a kind of human. The only thing we've really done in the past

a certain new tool to come along that will make us smarter. We have to have a way for us to think without the old tools. So we've
```

Analysis: Compare the two outputs. Does the standard model stay more on topic? Does the fast model drift into nonsense?

3. NLP Fundamentals: Under the Hood

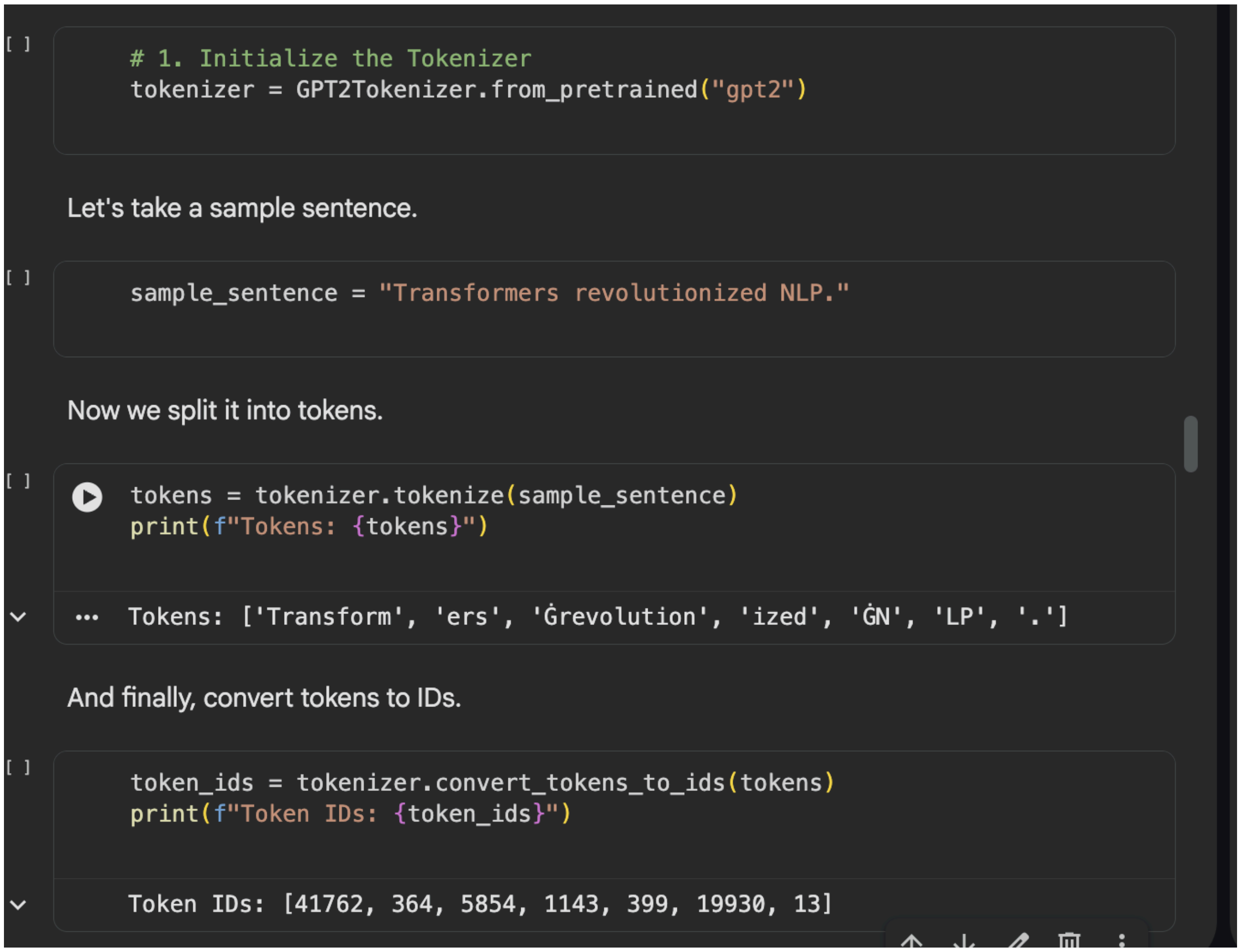
Before any "Magic" happens, the text must be processed. The pipeline does this automatically, but let's break it down manually to understand the steps.

- If we keep the same seed value, the results are reproducible and every run gives identical outputs.
- Changing the seed gives us different outputs.
- Temperature parameter can be set to control randomness and it decides how random the choice is whereas, seed decides which random choice is picked.
- Encoder knows to read and decoder knows to write ; Gpt 2 is a decoder only architecture, it cannot summarise text but only generate as it only writes being a decoder only architecture.
- Gpt2 tokenizer has its own logic and vectorization methods.

- I noticed that the distilgpt2 had repetitive sentences at seeds 42 and lesser and as i increased the seed value it started getting less repetitive. Upon looking for the reason behind this, my understanding was that changing the seed didn't reduce repetition in general it just made the model follow a different random path that happened to be less repetitive.
- Standard GPT-2 is a autoregressive transformer trained directly on large text corpora, whereas DistilGPT-2 is trained using knowledge distillation to mimic GPT-2's output distributions while using only 6 transformer layers.
- This makes DistilGPT-2 smaller and significantly faster, but it is more prone to issues like repetition and weaker long-range dependency modeling.
- GPT-2 is preferred when output quality matters, while DistilGPT-2 is better suited for low-resource or real-time applications.

The first step in processing text is of tokenization:

- It is the process of breaking text and sentences into tokens to which unique ids are then assigned.



```
# 1. Initialize the Tokenizer
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
```

Let's take a sample sentence.

```
sample_sentence = "Transformers revolutionized NLP."
```

Now we split it into tokens.

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

```
Tokens: ['Transform', 'ers', 'Ġrevolution', 'ized', 'ĠN', 'LP', '.']
```

And finally, convert tokens to IDs.

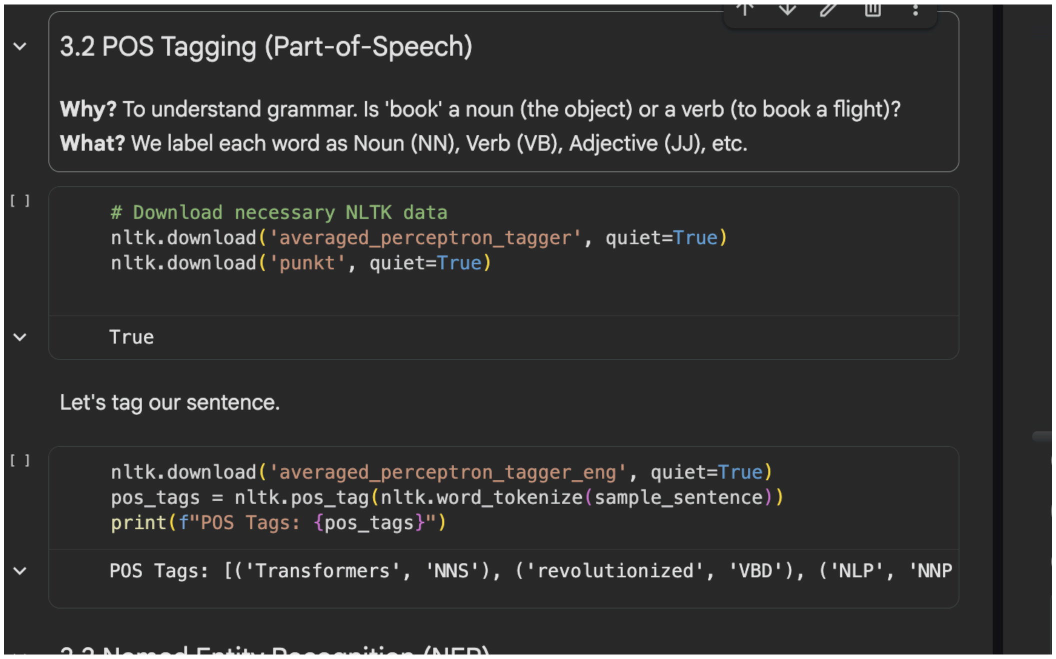
```
token_ids = tokenizer.convert_tokens_to_ids(tokens)
print(f"Token IDs: {token_ids}")
```

```
Token IDs: [41762, 364, 5854, 1143, 399, 19930, 13]
```

Next step is pos or parts of speech:

- POS helps us understand sentence structure like who is doing what, what is being described, etc.
- In NLP, POS is used for parsing, information extraction, grammar checking, and as features in traditional ML models.
- Main parts of speech are nouns, adjectives, verbs, adverbs, pronouns, prepositions, etc.
- NN → noun (dog)
- NNS → plural noun (dogs)
- NNP → proper noun (India)
- VB → verb base form (run)
- VBD → past tense (ran)
- VBG → gerund (running)
- VBZ → 3rd person present (runs)
- JJ → adjective (fast)
- RB → adverb (quickly)
- DT → determiner (the)
- IN → preposition (in)
- PRP → pronoun (she)

There was an error in this section of the file which turned out to be a LookupError, which simply means the program couldn't find a necessary piece of data it needed to perform a task. Specifically, the Natural Language Toolkit (NLTK) library, which we use for identifying parts of speech (like nouns, verbs, etc.), was missing a specific language model called averaged_perceptron_tagger_eng.
I made use of gemini in order to find its actual fix which involved downloading the correct missing model, averaged_perceptron_tagger_eng, using nltk.download('averaged_perceptron_tagger_eng').
Once that data was available, the program could successfully identify the parts of speech in your sample sentence.



3.2 POS Tagging (Part-of-Speech)

Why? To understand grammar. Is 'book' a noun (the object) or a verb (to book a flight)?
What? We label each word as Noun (NN), Verb (VB), Adjective (JJ), etc.

```
# Download necessary NLTK data
nltk.download('averaged_perceptron_tagger', quiet=True)
nltk.download('punkt', quiet=True)
```

```
True
```

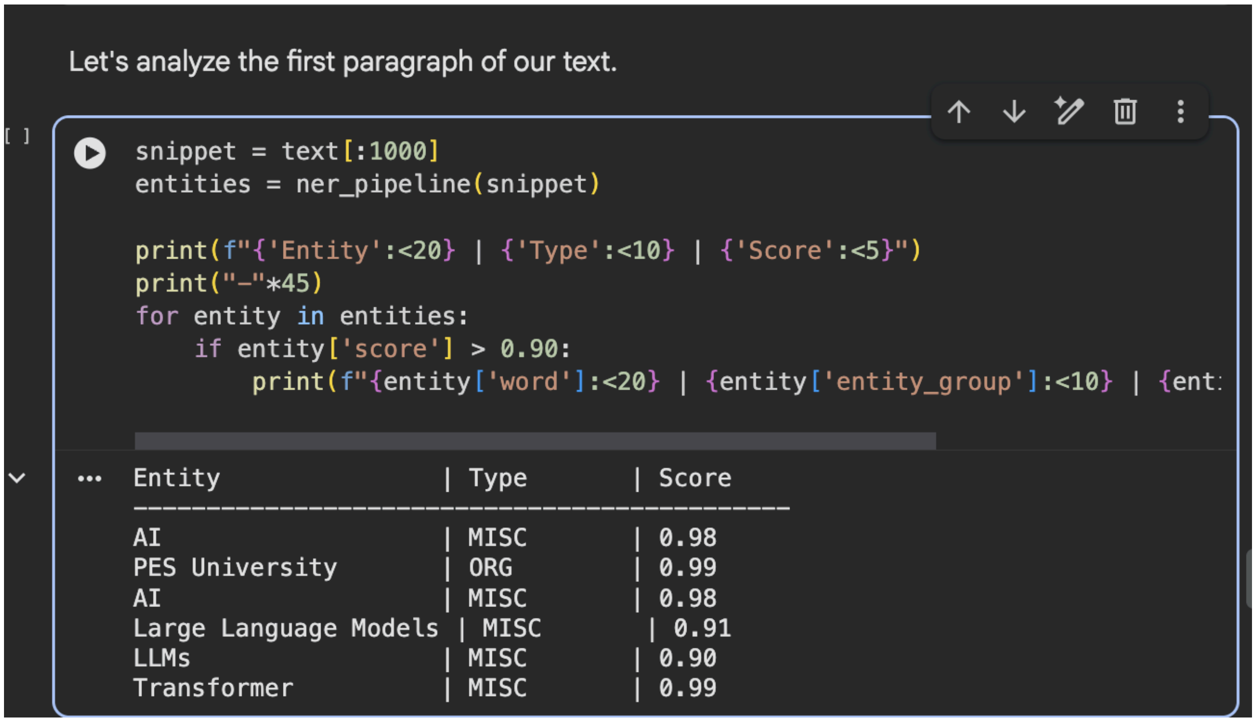
Let's tag our sentence.

```
nltk.download('averaged_perceptron_tagger_eng', quiet=True)
pos_tags = nltk.pos_tag(nltk.word_tokenize(sample_sentence))
print(f"POS Tags: {pos_tags}")
```

```
POS Tags: [('Transformers', 'NNS'), ('revolutionized', 'VBD'), ('NLP', 'NNP
```

Next came named entity recognition:

- This section focuses on identifying and classifying real-world entities in text.
- The code for this finds the important names and terms in the first part of input text and shows us what it found with a high degree of confidence (puts into categories of MISC or ORG).



Let's analyze the first paragraph of our text.

```
snippet = text[:1000]
entities = ner_pipeline(snippet)

print(f"{'Entity':<20} | {'Type':<10} | {'Score':<5}")
print("-"*45)
for entity in entities:
    if entity['score'] > 0.90:
        print(f"{entity['word']:<20} | {entity['entity_group']:<10} | {ent:
```

```
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
```

Lastly in section 4 we covered summarization and different models to do the same.

- Fast summarizers prioritize efficiency, while quality summarizers prioritize coherence and accuracy.
- Practical systems aim for a trade-off that delivers readable summaries with minimal latency.

For question answering:

- We call the pipeline("question-answering", ...),
- The specific model chosen, distilbert-base-cased-distilled-squad is a 'distilled' version of BERT, meaning it's smaller, faster, and more efficient than the full BERT model, making it suitable for practical applications without a significant loss in accuracy.
- Iit finds the answer within the provided context, rather than generating a new answer. This particular setup is highly efficient for quickly extracting factual information from large texts.
- For every question, the qa_pipeline is invoked, taking the current question and the first 5000 characters of the text (acting as the context) as input.
- The pipeline then processes these inputs to find the most relevant answer within the context, storing the result in the res variable.
- Finally, it clearly prints the question and then extracts and displays the model's identified answer by accessing res['answer'].

For fill in the blank:

- We initialize a Masked Language Modeling (MLM) pipeline, which is essentially setting up the model to play a "fill-in-the-blank" game.
- By using pipeline("fill-mask", ...), we're telling the system to prepare for a task where it will predict missing words in a sentence based on the surrounding context.
- The model selected, bert-base-uncased, is a version of BERT (Bidirectional Encoder Representations from Transformers). The 'uncased' part indicates that it treats words like 'The' and 'the' as identical, focusing purely on the lexical content without differentiating by capitalization. This setup makes it highly effective for tasks requiring an understanding of word relationships and context within sentences, as it excels at inferring the most plausible words for masked tokens.
-