Unit 1 HandsOn – Observations and Learnings

HandsOn-1_Unit1.ipynb

NAME- ANANYAA SREE SP

SRN- PES2UG23CS066

SECTION: A

Aim

The aim of this practical session was to understand the basic NLP pipeline and to study how different Large Language Models behave when used for various NLP tasks using the Hugging Face library.

Hugging Face Setup and NLP Pipeline

In this practical, the Hugging Face transformers library was used along with the pipeline() function. The pipeline function simplifies the entire NLP workflow by handling tokenization, model loading, and inference internally. This made it easier to focus on understanding model behavior rather than implementation details.

Through this setup, multiple NLP tasks such as text generation, summarization, named entity recognition, masked language modeling, and question answering were performed.

Tokenization

Tokenization is the process of breaking input text into smaller units called tokens so that the model can process them. Since language models cannot directly understand raw text, tokenization converts text into a form suitable for numerical processing.

Tokenization plays a very important role because the quality of tokens directly affects how well the model understands the input sentence. Sub-word tokenization also helps in handling unknown or rare words.

Part-of-Speech (POS) Tagging

POS tagging is used to identify the grammatical role of each word in a sentence, such as noun, verb, or adjective.

Example: [('Transformers', 'NNS'), ('revolutionized', 'VBD'), ('NLP', 'NNP'), ('.', '.')]

In this output:

- **NNS** represents a plural noun
- **VBD** represents a verb in past tense
- **NNP** represents a proper noun

POS tagging helps in understanding sentence structure and is useful in tasks like syntactic analysis, information extraction, and question answering.

Named Entity Recognition (NER)

Named Entity Recognition is used to identify real-world entities such as names of people, organizations, locations, and dates in a sentence. Instead of only understanding grammar, NER helps the model understand what the sentence is actually referring to.

This technique is widely used in applications such as search engines, chatbots, and document analysis systems.

Seed Value and Its Effect

A seed value is used to control randomness during text generation. When the same seed and prompt are used, the model produces the same output every time.

When the seed value is changed or increased, the model still follows the same logic but generates slightly different word choices or sentence structures. This increases variation in the output but does not improve the model's intelligence or accuracy.

Using a fixed seed is helpful for reproducibility and fair comparison between different models.

Comparison of Distilled and Standard Models

In this practical, **distilgpt2** and **gpt2** models were compared.

The distilled model, distilgpt2, is smaller and faster, making it suitable for applications where speed and low resource usage are important. However, its output is sometimes less detailed and less coherent.

The standard gpt2 model is larger and takes more time to generate responses, but it produces better structured and more meaningful text.

This comparison shows that model selection depends on whether performance speed or output quality is the priority.

Summarization: Fast vs Qualitative

The fast summarizer generates short and direct summaries by focusing only on the main points. It is computationally efficient and suitable when quick results are required.

The qualitative summarizer produces more detailed summaries that preserve context and important information. Although slower, it provides better readability and understanding.

Masked Language Modeling

Masked language modeling involves predicting missing words in a sentence based on surrounding context. This task demonstrates how models understand relationships between words rather than processing them independently.

Question Answering (Q&A)

In the question answering task, the model answered questions based strictly on the provided context. Accurate answers were generated when the information was clearly mentioned in the text, but the model struggled when the answer had to be inferred indirectly.

This shows that Q&A models work best with explicit and well-defined context.

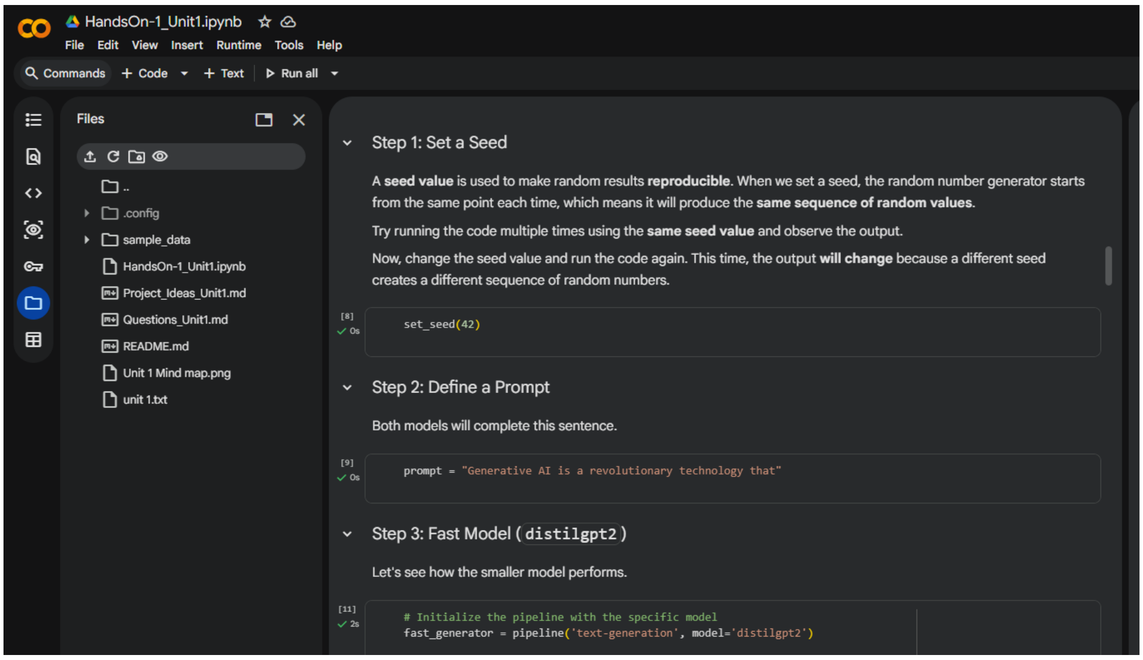
Practical Debugging Skills

By reading and understanding error messages, appropriate fixes were applied:

- Adding
  nltk.download('punkt_tab', quiet=True)
  resolved the tokenization error.

The errors clearly showed that NLTK does not always automatically download all dependent resources. Explicitly identifying missing resources from error messages was necessary to fix the issues.

Seed 42



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.

```
[8]
set_seed(42)
```

Step 2: Define a Prompt

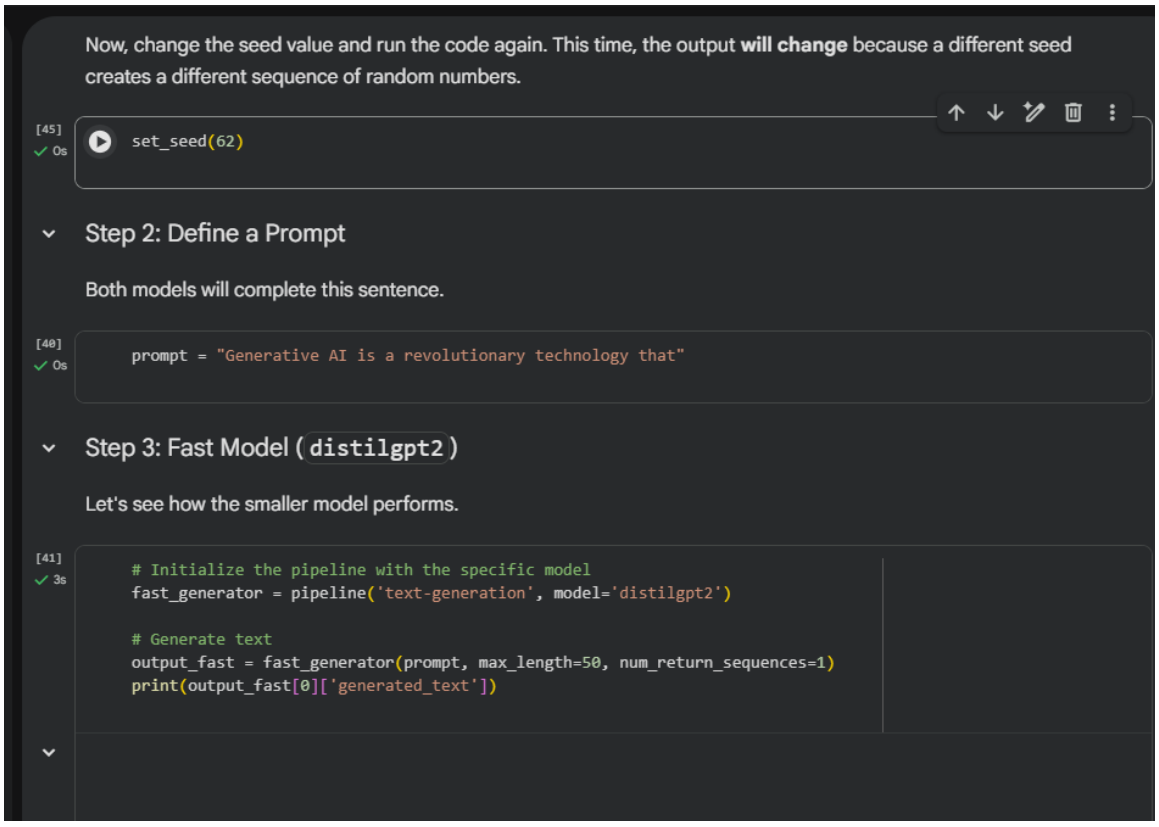
Both models will complete this sentence.

```
[9]
prompt = "Generative AI is a revolutionary technology that"
```

Step 3: Fast Model (`distilgpt2`)

Let's see how the smaller model performs.

```
[11]
# Initialize the pipeline with the specific model
fast_generator = pipeline('text-generation', model='distilgpt2')
```



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.

```
[45]
set_seed(62)
```

Step 2: Define a Prompt

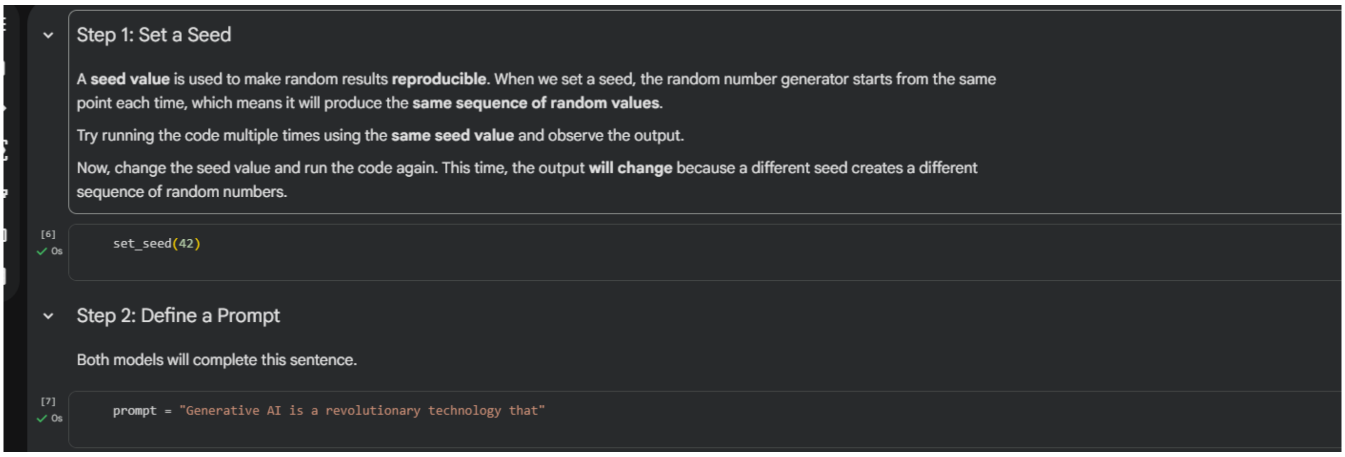
Both models will complete this sentence.

```
[40]
prompt = "Generative AI is a revolutionary technology that"
```

Step 3: Fast Model (`distilgpt2`)

Let's see how the smaller model performs.

```
[41]
# Initialize the pipeline with the specific model
fast_generator = pipeline('text-generation', model='distilgpt2')

# Generate text
output_fast = fast_generator(prompt, max_length=50, num_return_sequences=1)
print(output_fast[0]['generated_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.

[6]
✓ 0s

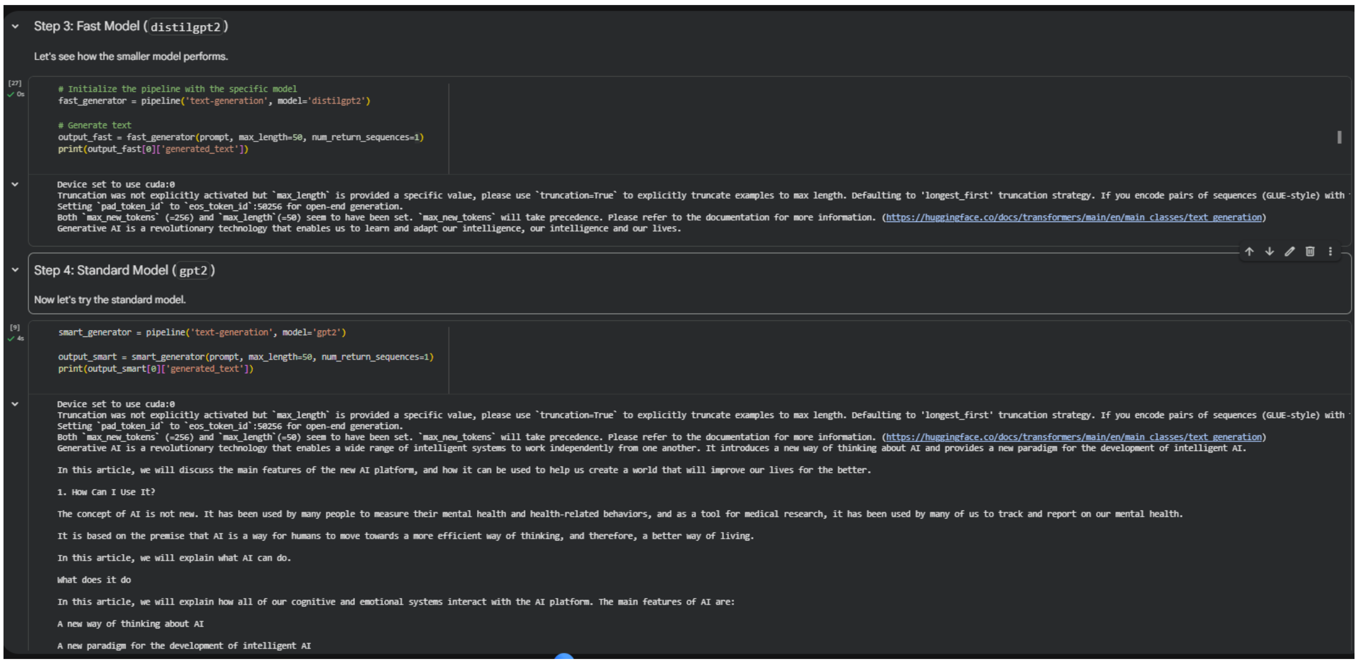
```python
set_seed(42)
```

Step 2: Define a Prompt

Both models will complete this sentence.

[7]
✓ 0s

```python
prompt = "Generative AI is a revolutionary technology that"
```



Step 3: Fast Model (`distilgpt2`)

Let's see how the smaller model performs.

[27]
✓ 0s

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

# Generate text
output_fast = fast_generator(prompt, max_length=50, num_return_sequences=1)
print(output_fast[0]['generated_text'])
```

```
Device set to use cuda:0
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length. Defaulting to 'longest_first' truncation strategy. If you encode pairs of sequences (GLUE-style) with
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 information. (https://huggingface.co/docs/transformers/main/en/main_classes/text_generation)
Generative AI is a revolutionary technology that enables us to learn and adapt our intelligence, our intelligence and our lives.
```

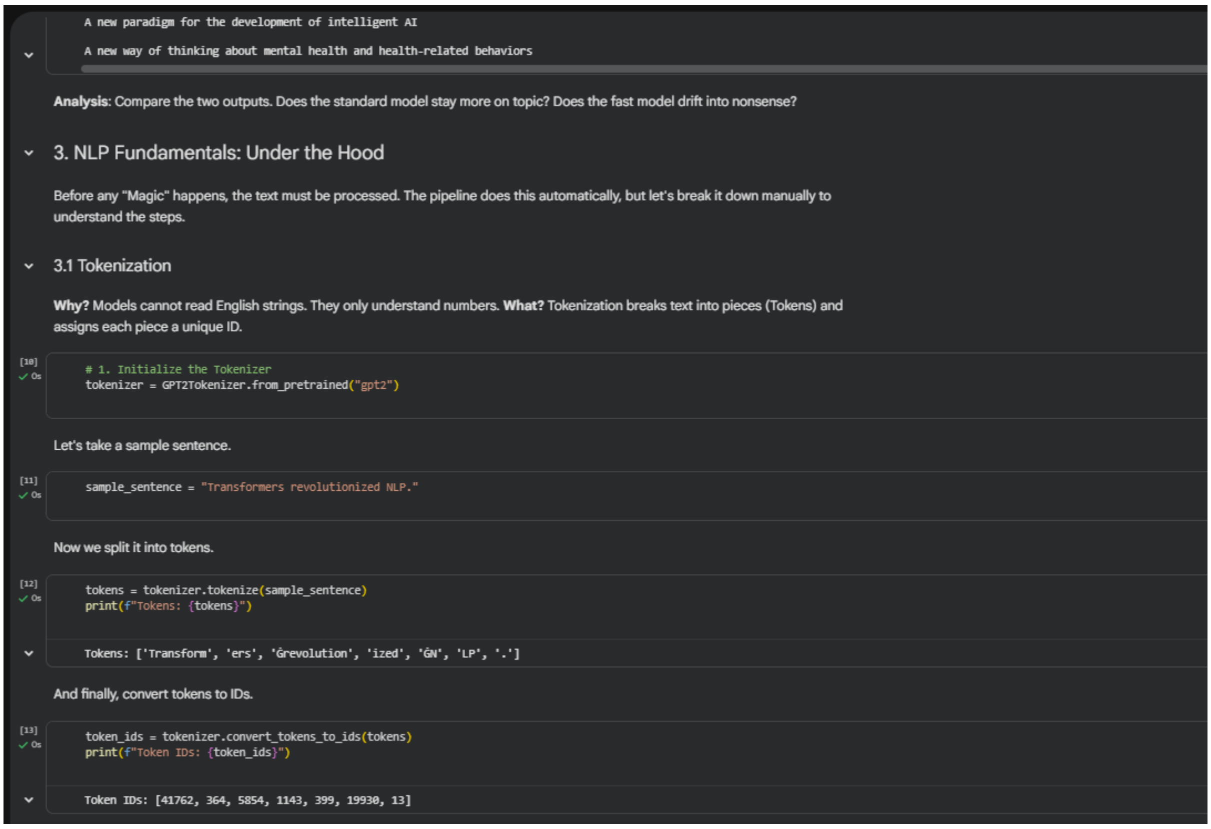
Step 4: Standard Model (`gpt2`)

Now let's try the standard model.

[9]
✓ 4s

```python
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'])
```

```
Device set to use cuda:0
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length. Defaulting to 'longest_first' truncation strategy. If you encode pairs of sequences (GLUE-style) with
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 information. (https://huggingface.co/docs/transformers/main/en/main_classes/text_generation)
Generative AI is a revolutionary technology that enables a wide range of intelligent systems to work independently from one another. It introduces a new way of thinking about AI and provides a new paradigm for the development of intelligent AI.

In this article, we will discuss the main features of the new AI platform, and how it can be used to help us create a world that will improve our lives for the better.

1. How Can I Use It?

The concept of AI is not new. It has been used by many people to measure their mental health and health-related behaviors, and as a tool for medical research, it has been used by many of us to track and report on our mental health.

It is based on the premise that AI is a way for humans to move towards a more efficient way of thinking, and therefore, a better way of living.

In this article, we will explain what AI can do.

What does it do

In this article, we will explain how all of our cognitive and emotional systems interact with the AI platform. The main features of AI are:

A new way of thinking about AI

A new paradigm for the development of intelligent AI
```



```
A new paradigm for the development of intelligent AI

A new way of thinking about mental health and health-related behaviors
```

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.

3.1 Tokenization

Why? Models cannot read English strings. They only understand numbers. **What?** Tokenization breaks text into pieces (Tokens) and assigns each piece a unique ID.

[10] ✓ 0s
```python
# 1. Initialize the Tokenizer
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
```

Let's take a sample sentence.

[11] ✓ 0s
```python
sample_sentence = "Transformers revolutionized NLP."
```

Now we split it into tokens.

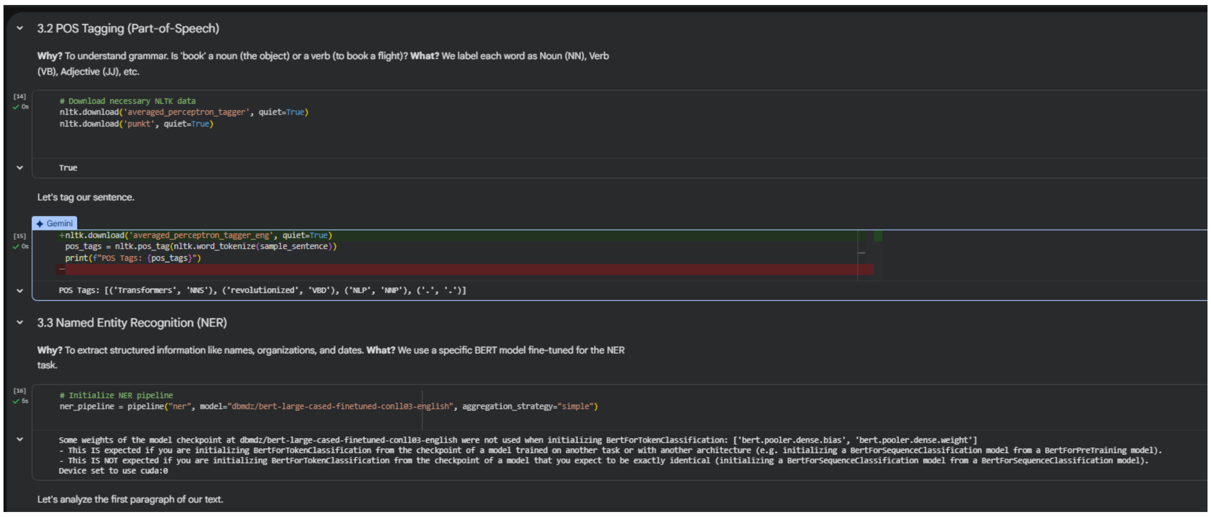
[12] ✓ 0s
```python
tokens = tokenizer.tokenize(sample_sentence)
print(f"Tokens: {tokens}")
```

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

And finally, convert tokens to IDs.

[13] ✓ 0s
```python
token_ids = tokenizer.convert_tokens_to_ids(tokens)
print(f"Token IDs: {token_ids}")
```

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



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.

[14] ✓ 0s
```python
# Download necessary NLTK data
nltk.download('averaged_perceptron_tagger', quiet=True)
nltk.download('punkt', quiet=True)
```

```
True
```

Let's tag our sentence.

[15] ✓ 0s
```python
+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'), ('.', '.')]
```

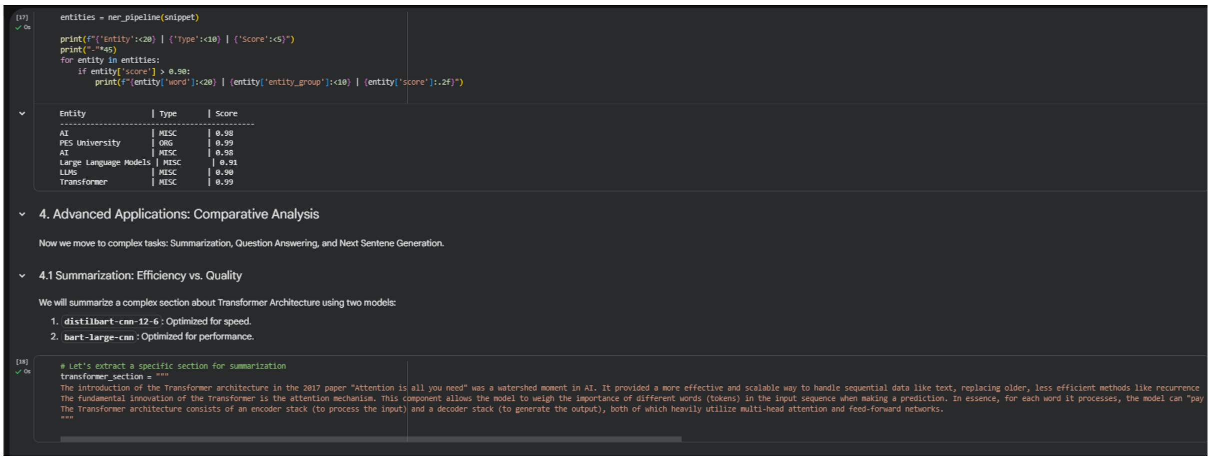
3.3 Named Entity Recognition (NER)

Why? To extract structured information like names, organizations, and dates. **What?** We use a specific BERT model fine-tuned for the NER task.

[16] ✓ 5s
```python
# Initialize NER pipeline
ner_pipeline = pipeline("ner", model="dbmdz/bert-large-cased-finetuned-conll03-english", aggregation_strategy="simple")
```

```
Some weights of the model checkpoint at dbmdz/bert-large-cased-finetuned-conll03-english were not used when initializing BertForTokenClassification: ['bert.pooler.dense.bias', 'bert.pooler.dense.weight']
- This IS expected if you are initializing BertForTokenClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForTokenClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
Device set to use cuda:0
```

Let's analyze the first paragraph of our text.



[17] ✓ 0s
```python
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} | {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
```

4. Advanced Applications: Comparative Analysis

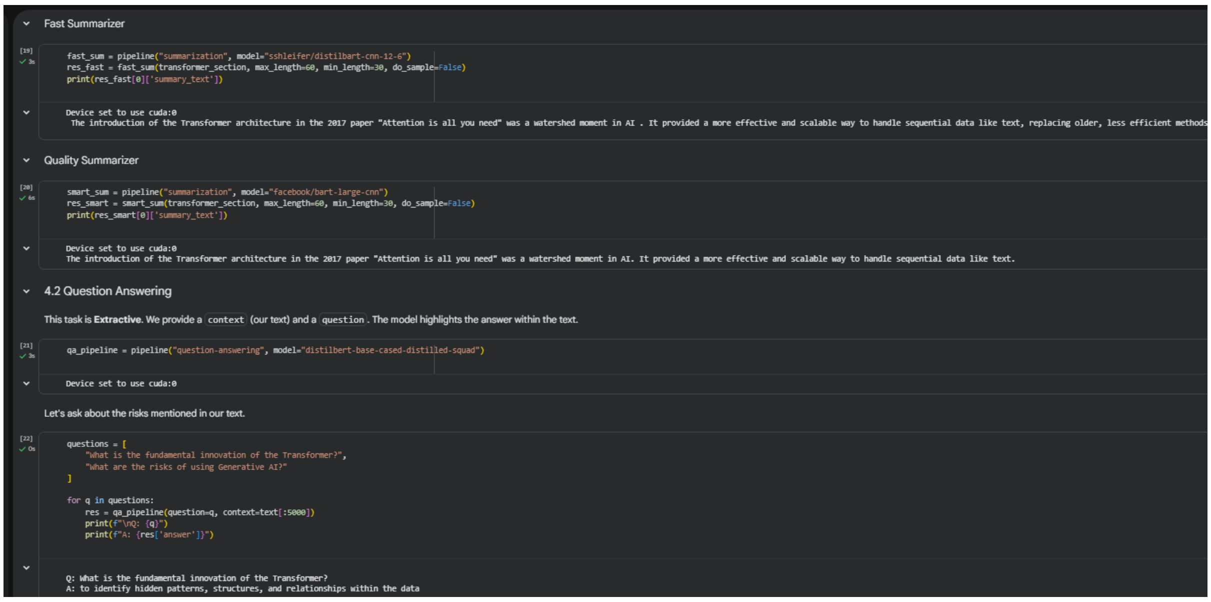
Now we move to complex tasks: Summarization, Question Answering, and Next Sentene Generation.

4.1 Summarization: Efficiency vs. Quality

We will summarize a complex section about Transformer Architecture using two models:

1. `distilbart-cnn-12-6` : Optimized for speed.
2. `bart-large-cnn` : Optimized for performance.

[18] ✓ 0s
```python
# Let's extract a specific section for summarization
transformer_section = """
The introduction of the Transformer architecture in the 2017 paper "Attention is all you need" was a watershed moment in AI. It provided a more effective and scalable way to handle sequential data like text, replacing older, less efficient methods like recurrence
The fundamental innovation of the Transformer is the attention mechanism. This component allows the model to weigh the importance of different words (tokens) in the input sequence when making a prediction. In essence, for each word it processes, the model can "pay
The Transformer architecture consists of an encoder stack (to process the input) and a decoder stack (to generate the output), both of which heavily utilize multi-head attention and feed-forward networks.
"""
```



Fast Summarizer

```
fast_sum = pipeline("summarization", model="sshleifer/distilbart-cnn-12-6")
res_fast = fast_sum(transformer_section, max_length=60, min_length=30, do_sample=False)
print(res_fast[0]['summary_text'])
```

```
Device set to use cuda:0
 The introduction of the Transformer architecture in the 2017 paper "Attention is all you need" was a watershed moment in AI . It provided a more effective and scalable way to handle sequential data like text, replacing older, less efficient methods
```

Quality Summarizer

```
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'])
```

```
Device set to use cuda:0
The introduction of the Transformer architecture in the 2017 paper "Attention is all you need" was a watershed moment in AI. It provided a more effective and scalable way to handle sequential data like text.
```

4.2 Question Answering

This task is **Extractive**. We provide a `context` (our text) and a `question`. The model highlights the answer within the text.

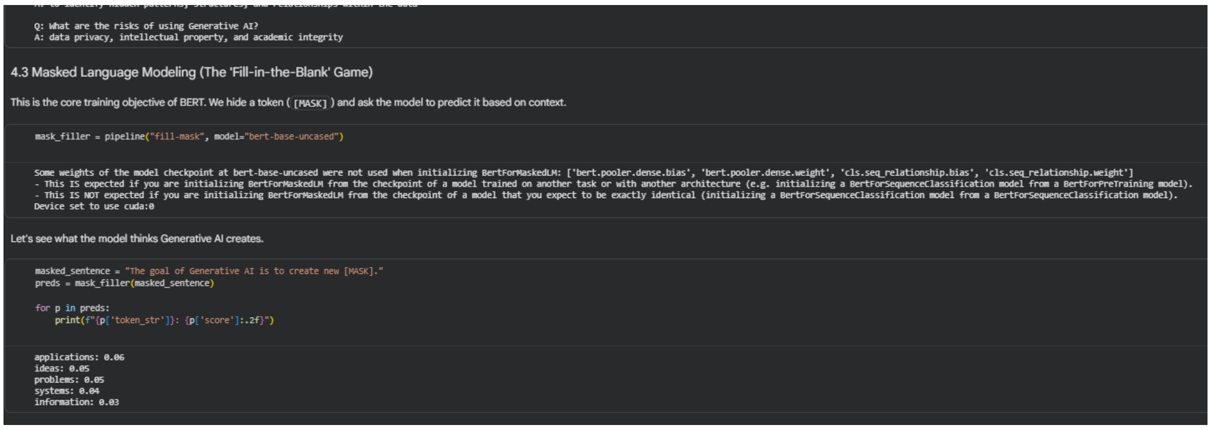
```
qa_pipeline = pipeline("question-answering", model="distilbert-base-cased-distilled-squad")
```

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

Let's ask about the risks mentioned in our text.

```
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
```

4.3 Masked Language Modeling (The 'Fill-in-the-Blank' Game)

This is the core training objective of BERT. We hide a token (`[MASK]`) and ask the model to predict it based on context.

```
mask_filler = pipeline("fill-mask", model="bert-base-uncased")
```

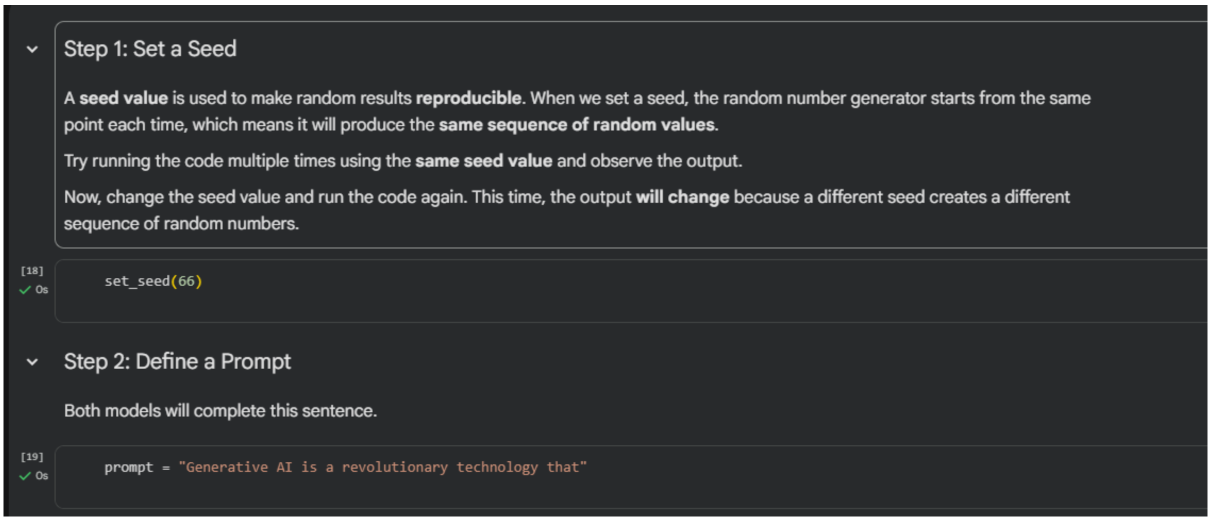
```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertForMaskedLM: ['bert.pooler.dense.bias', 'bert.pooler.dense.weight', 'cls.seq_relationship.bias', 'cls.seq_relationship.weight']
- This IS expected if you are initializing BertForMaskedLM from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForMaskedLM from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
Device set to use cuda:0
```

Let's see what the model thinks Generative AI creates.

```
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
```

Seed 66



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.

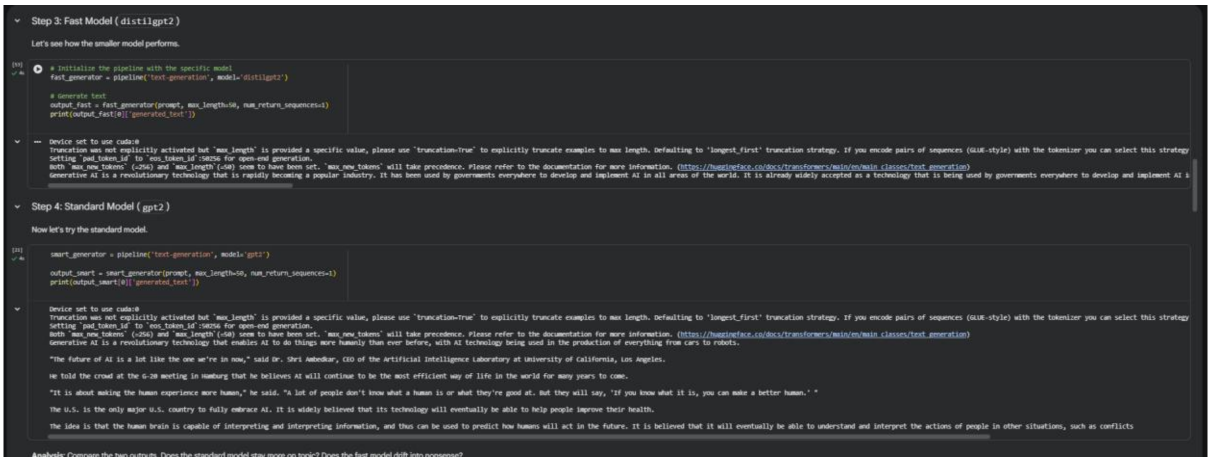
Now, change the seed value and run the code again. This time, the output **will change** because a different seed creates a different sequence of random numbers.

```
set_seed(66)
```

Step 2: Define a Prompt

Both models will complete this sentence.

```
prompt = "Generative AI is a revolutionary technology that"
```



Step 3: Fast Model (`distilgpt2`)

Let's see how the smaller model performs.

[53]

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

# Generate text
output_fast = fast_generator(prompt, max_length=50, num_return_sequences=1)
print(output_fast[0]['generated_text'])
```

```
Device set to use cuda:0
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length. Defaulting to 'longest_first' truncation strategy. If you encode pairs of sequences (GLUE-style) with the tokenizer you can select this strategy
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 information. (https://huggingface.co/docs/transformers/main/en/main_classes/text_generation)
Generative AI is a revolutionary technology that is rapidly becoming a popular industry. It has been used by governments everywhere to develop and implement AI in all areas of the world. It is already widely accepted as a technology that is being used by governments everywhere to develop and implement AI i
```

Step 4: Standard Model (`gpt2`)

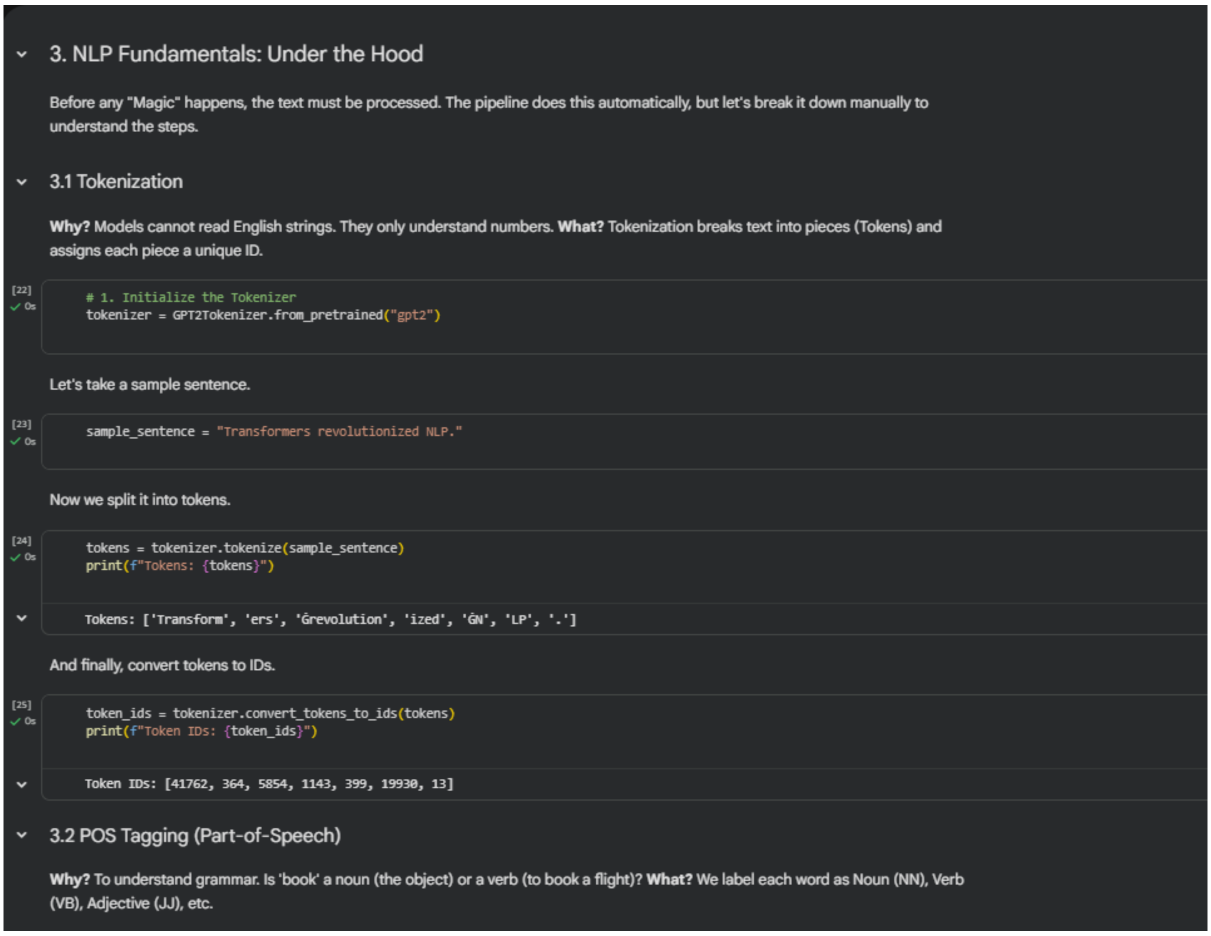
Now let's try the standard model.

[21]

```python
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'])
```

```
Device set to use cuda:0
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length. Defaulting to 'longest_first' truncation strategy. If you encode pairs of sequences (GLUE-style) with the tokenizer you can select this strategy
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 information. (https://huggingface.co/docs/transformers/main/en/main_classes/text_generation)
Generative AI is a revolutionary technology that enables AI to do things more humanly than ever before, with AI technology being used in the production of everything from cars to robots.

"The future of AI is a lot like the one we're in now," said Dr. Shri Ambedkar, CEO of the Artificial Intelligence Laboratory at University of California, Los Angeles.

He told the crowd at the G-20 meeting in Hamburg that he believes AI will continue to be the most efficient way of life in the world for many years to come.

"It is about making the human experience more human," he said. "A lot of people don't know what a human is or what they're good at. But they will say, 'If you know what it is, you can make a better human.' "

The U.S. is the only major U.S. country to fully embrace AI. It is widely believed that its technology will eventually be able to help people improve their health.

The idea is that the human brain is capable of interpreting and interpreting information, and thus can be used to predict how humans will act in the future. It is believed that it will eventually be able to understand and interpret the actions of people in other situations, such as conflicts
```

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.

3.1 Tokenization

Why? Models cannot read English strings. They only understand numbers. **What?** Tokenization breaks text into pieces (Tokens) and assigns each piece a unique ID.

[22]

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

Let's take a sample sentence.

[23]

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

Now we split it into tokens.

[24]

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

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

And finally, convert tokens to IDs.

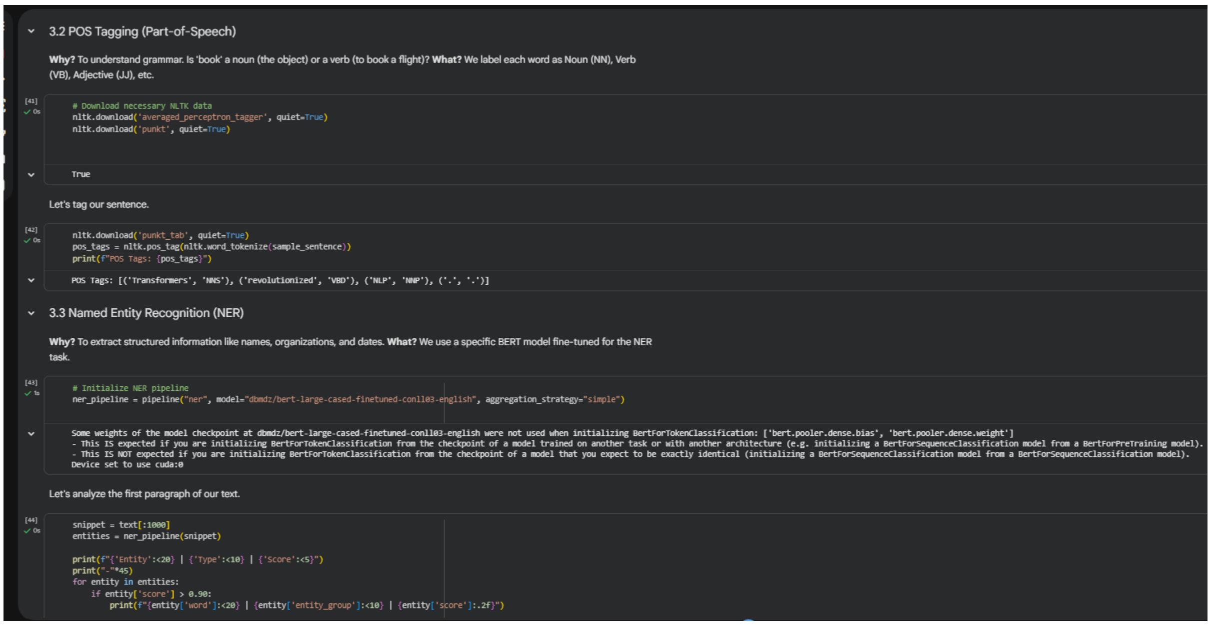
[25]

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

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

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.



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('punkt_tab', 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'), ('.', '.')]
```

3.3 Named Entity Recognition (NER)

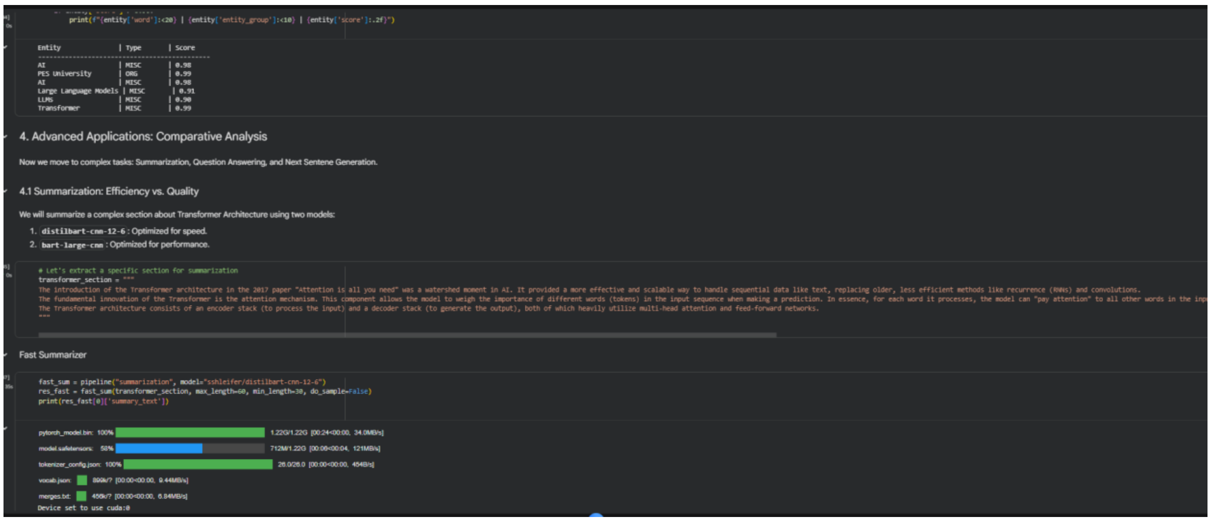
Why? To extract structured information like names, organizations, and dates. **What?** We use a specific BERT model fine-tuned for the NER task.

```
# Initialize NER pipeline
ner_pipeline = pipeline("ner", model="dbmdz/bert-large-cased-finetuned-conll03-english", aggregation_strategy="simple")
```

```
Some weights of the model checkpoint at dbmdz/bert-large-cased-finetuned-conll03-english were not used when initializing BertForTokenClassification: ['bert.pooler.dense.bias', 'bert.pooler.dense.weight']
- This IS expected if you are initializing BertForTokenClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForTokenClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
Device set to use cuda:0
```

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} | {entity['score']:.2f}")
```



```
        print(f"{entity['word']:<20} | {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
```

4. Advanced Applications: Comparative Analysis

Now we move to complex tasks: Summarization, Question Answering, and Next Sentene Generation.

4.1 Summarization: Efficiency vs. Quality

We will summarize a complex section about Transformer Architecture using two models:

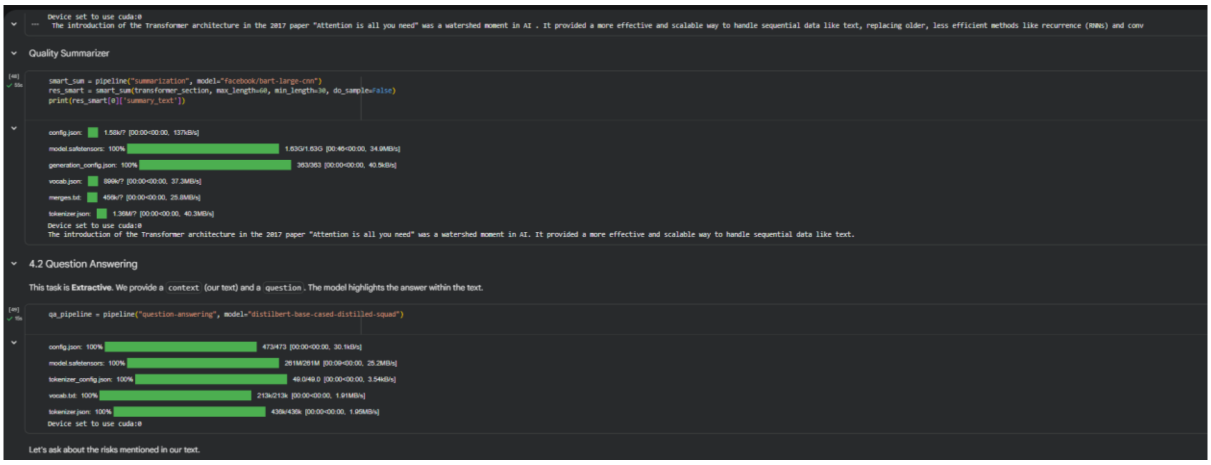
1. `distilbart-cnn-12-6` : Optimized for speed.
2. `bart-large-cnn` : Optimized for performance.

```
# Let's extract a specific section for summarization
transformer_section = """
The introduction of the Transformer architecture in the 2017 paper "Attention is all you need" was a watershed moment in AI. It provided a more effective and scalable way to handle sequential data like text, replacing older, less efficient methods like recurrence (RNNs) and convolutions.
The fundamental innovation of the Transformer is the attention mechanism. This component allows the model to weigh the importance of different words (tokens) in the input sequence when making a prediction. In essence, for each word it processes, the model can "pay attention" to all other words in the inp
The Transformer architecture consists of an encoder stack (to process the input) and a decoder stack (to generate the output), both of which heavily utilize multi-head attention and feed-forward networks.
"""
```

Fast Summarizer

```
fast_sum = pipeline("summarization", model="sshleifer/distilbart-cnn-12-6")
res_fast = fast_sum(transformer_section, max_length=60, min_length=30, do_sample=False)
print(res_fast[0]['summary_text'])
```

```
pytorch_model.bin: 100%  1.22G/1.22G [00:24<00:00, 34.0MB/s]
model.safetensors:  58%  712M/1.22G [00:06<00:04, 121MB/s]
tokenizer_config.json: 100%  26.0/26.0 [00:00<00:00, 454B/s]
vocab.json:  899k/? [00:00<00:00, 9.44MB/s]
merges.txt:  456k/? [00:00<00:00, 6.84MB/s]
Device set to use cuda:0
```



```
Device set to use cuda:0
 The introduction of the Transformer architecture in the 2017 paper "Attention is all you need" was a watershed moment in AI . It provided a more effective and scalable way to handle sequential data like text, replacing older, less efficient methods like recurrence (RNNs) and conv
```

Quality Summarizer

```
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, 137kB/s]
model.safetensors: 100%  1.63G/1.63G [00:46<00:00, 34.9MB/s]
generation_config.json: 100%  363/363 [00:00<00:00, 40.5kB/s]
vocab.json:  899k/? [00:00<00:00, 37.3MB/s]
merges.txt:  456k/? [00:00<00:00, 25.8MB/s]
tokenizer.json:  1.36M/? [00:00<00:00, 40.3MB/s]
Device set to use cuda:0
The introduction of the Transformer architecture in the 2017 paper "Attention is all you need" was a watershed moment in AI. It provided a more effective and scalable way to handle sequential data like text.
```

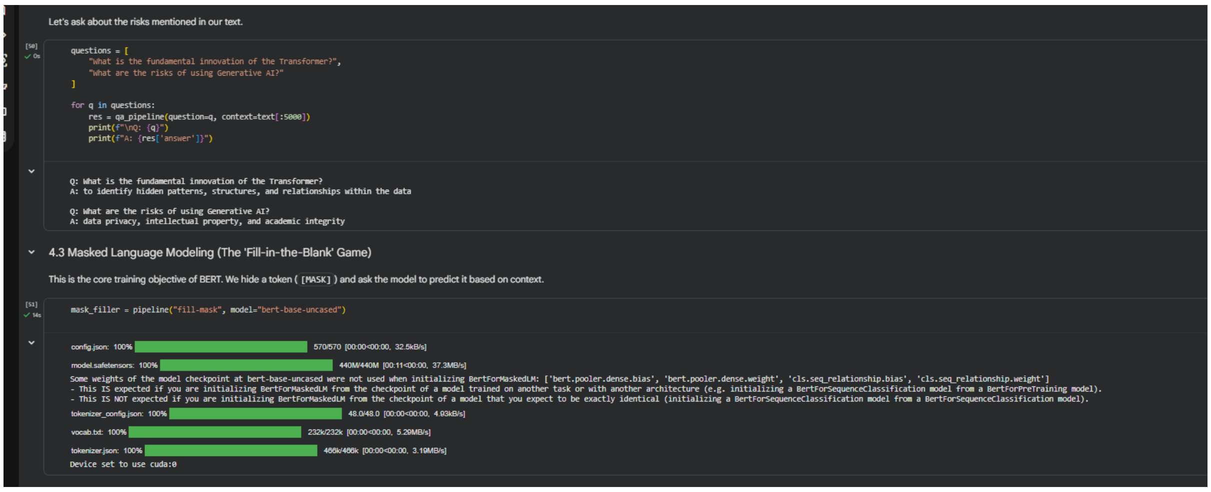
4.2 Question Answering

This task is **Extractive.** We provide a `context` (our text) and a `question`. The model highlights the answer within the text.

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

```
config.json: 100%  473/473 [00:00<00:00, 30.1kB/s]
model.safetensors: 100%  261M/261M [00:09<00:00, 25.2MB/s]
tokenizer_config.json: 100%  49.0/49.0 [00:00<00:00, 3.54kB/s]
vocab.txt: 100%  213k/213k [00:00<00:00, 1.91MB/s]
tokenizer.json: 100%  436k/436k [00:00<00:00, 1.95MB/s]
Device set to use cuda:0
```

Let's ask about the risks mentioned in our text.



Let's ask about the risks mentioned in our text.

[50]

```python
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
```

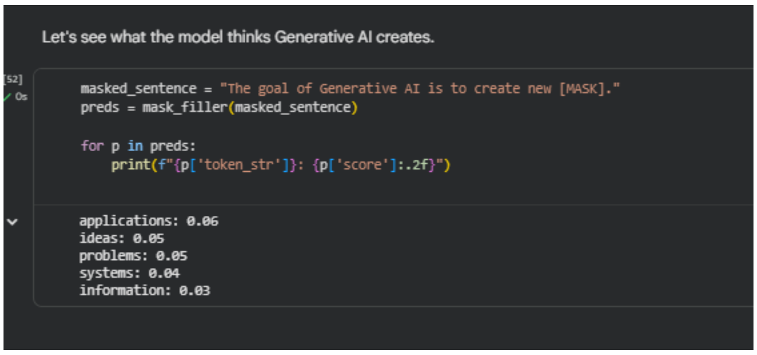
4.3 Masked Language Modeling (The 'Fill-in-the-Blank' Game)

This is the core training objective of BERT. We hide a token (`[MASK]`) and ask the model to predict it based on context.

[51]

```python
mask_filler = pipeline("fill-mask", model="bert-base-uncased")
```

```
config.json: 100% 570/570 [00:00<00:00, 32.5kB/s]
model.safetensors: 100% 440M/440M [00:11<00:00, 37.3MB/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', 'cls.seq_relationship.bias', 'cls.seq_relationship.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 a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForMaskedLM from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 4.93kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 5.29MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 3.19MB/s]
Device set to use cuda:0
```



Let's see what the model thinks Generative AI creates.

[52]

```python
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
```