LLM_CHUNK_GPT2

- For chunking text to process with large language models (LLMs), you can create a simple function that splits long texts into smaller chunks.
- This is particularly useful when the input text exceeds the model's token limit. Below is an example using Python, leveraging the transformers library to illustrate how you can chunk text before feeding it into an LLM.

- The term "transformers" in the context of natural language processing (NLP) and machine learning can refer to various models and architectures built upon the original transformer architecture introduced in the paper "Attention is All You Need" by Vaswani et al. in 2017. Here are some key transformer models and variants that have been developed since then:

1. BERT (Bidirectional Encoder Representations from Transformers) Focuses on understanding context in both directions (left and right) using masked language modeling.
2. GPT (Generative Pre-trained Transformer) Developed by OpenAI, GPT models (like GPT-2 and GPT-3) are autoregressive models primarily used for text generation.
3. T5 (Text-to-Text Transfer Transformer) Treats every NLP task as a text-to-text problem, making it very flexible across various applications.
4. RoBERTa (A Robustly Optimized BERT Pretraining Approach) An improvement over BERT with more training data and different training strategies.
5. XLNet Combines the ideas of BERT and autoregressive models, allowing for better capturing of context and dependencies.
6. ALBERT (A Lite BERT) A smaller and more efficient version of BERT that reduces the number of parameters while maintaining performance.
7. DistilBERT A distilled version of BERT that is smaller and faster while retaining much of its performance.
8. ERNIE (Enhanced Representation through kNowledge Integration) Developed by Baidu, it incorporates external knowledge to improve language understanding.
9. ELECTRA Instead of masking tokens like BERT, ELECTRA predicts replaced tokens, leading to more efficient training.
10. DeBERTa (Decoding-enhanced BERT with Disentangled Attention) Uses a disentangled attention mechanism to improve performance on various NLP tasks.

11. Vision Transformers (ViT) Adapts the transformer architecture for image processing tasks, treating images as sequences of patches.
12. BART (Bidirectional and Auto-Regressive Transformers) Combines BERT's bidirectional encoding and GPT's autoregressive decoding for tasks like summarization and translation.
13. LayoutLM Designed for document understanding, incorporating layout information from scanned documents.
14. Swin Transformer A hierarchical vision transformer that can be used for both image classification and detection tasks.
15. Transformer-XL Introduces recurrence to the transformer architecture, allowing it to handle longer sequences more effectively. These are just some of the prominent transformer models and architectures. The field is rapidly evolving, with new variations and improvements continually being introduced, so the number and types of transformers are continually growing.

Double-click (or enter) to edit

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive
```

```
!nvidia-smi
```

```
Mon Sep 22 16:24:25 2025
+-----------------------------------------------------------------------------------------
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12
|-----------------------------------------+------------------------+----------------
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncor
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Comp
|                                         |                        |
|=========================================+========================+================
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |
| N/A   44C    P8              9W /   70W |       2MiB /  15360MiB |      0%      D
|                                         |                        |
+-----------------------------------------+------------------------+----------------

+-----------------------------------------------------------------------------------------
| Processes:
|  GPU   GI   CI        PID   Type   Process name                              GPU
|        ID   ID                                                               Usag
|=========================================================================================
|  No running processes found
+-----------------------------------------------------------------------------------------
```

```
pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in /usr/local/lib/pythc
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/pythor
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/di
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.1
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist
```

```
from transformers import AutoTokenizer

# Load the tokenizer for a specific model (e.g., GPT-2)
tokenizer = AutoTokenizer.from_pretrained("gpt2")

# Tokenize some input text
text = "Hello, how are you?"
tokens = tokenizer(text, return_tensors='pt')
print(tokens)
```

```
{'input_ids': tensor([[15496,    11,   703,   389,   345,    30]]), 'attention_mask
```

- input text "Hello, how are you?". Specifically, the output will be a dictionary containing the token IDs and attention masks in a format that PyTorch models can use.
- Token IDs: A tensor containing the integer representations of the tokens from the input text.
- Attention Mask: A tensor indicating which tokens should be attended to (1 for real tokens, 0 for padding).
- input_ids: This is a tensor containing the numerical IDs corresponding to the tokens. For the input text, it may look like [15496, 11, 703, 389, 345, 329, 30]. Each number corresponds to a specific token in the GPT-2 vocabulary.
- attention_mask: This tensor is used to indicate which tokens should be processed by the model. In this case, since there are no padding tokens, all values are 1.

```
from transformers import AutoModelForCausalLM

# Load the pre-trained GPT-2 model
model = AutoModelForCausalLM.from_pretrained("gpt2")

# Generate text
input_ids = tokenizer.encode("indian cricket", return_tensors='pt')
output = model.generate(input_ids, max_length=50)
generated_text = tokenizer.decode(output[0], skip_special_tokens=True)

print(generated_text)
```

```
The attention mask and the pad token id were not set. As a consequence, you may obs
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
indian cricket team, which has been in the country for over a decade.

The team's captain, Ravi Shankar, has been in the country for over a decade.

The team's captain, Ravi Shankar,
```

- When you run the provided code, it uses the GPT-2 model to generate text based on the prompt "Once upon a time." Here's a breakdown of what happens and what you can expect as output:
- Steps in the Code Load the Model: The AutoModelForCausalLM.from_pretrained("gpt2") line loads the pre-trained GPT-2 model.

Tokenization: The prompt "Once upon a time" is tokenized into input IDs that the model can understand.

- Text Generation: The model.generate() method generates text based on the input IDs. The max_length=50 argument specifies that the total length of the generated text (including the prompt) should not exceed 50 tokens.
- Decoding: The output is then decoded back into human-readable text using the tokenizer.

Output Characteristics

- Length: The length of the output will depend on the prompt and the max_length parameter. If the prompt is short and max_length is set to 50, the output will be roughly 30 to 40 tokens of generated text.

- Creativity: The continuation may include imaginative scenarios, characters, or events that align with the narrative style of fairy tales or stories.

```
from transformers import AutoModelForCausalLM

# Load the pre-trained GPT-2 model
model = AutoModelForCausalLM.from_pretrained("gpt2")

# Generate text
input_ids = tokenizer.encode("indian chess", return_tensors='pt')
output = model.generate(input_ids, max_length=50)
generated_text = tokenizer.decode(output[0], skip_special_tokens=True)

print(generated_text)
```

```
The attention mask and the pad token id were not set. As a consequence, you may obs
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
indian chess player, who was born in the United States, has been a member of the Ch

"I'm a big fan of the game and I'm very proud of the fact that I'm
```

```
from transformers import AutoTokenizer, AutoModelForCausalLM

# Load a pre-trained model and tokenizer
model_name = "gpt2"  # You can replace with any other LLM
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

def chunk_text(text, max_length=512):
    """Chunk text into smaller pieces."""
    tokens = tokenizer.encode(text, return_tensors='pt')[0]
    chunks = []

    for i in range(0, len(tokens), max_length):
        chunk = tokens[i:i + max_length]
        chunks.append(chunk)

    return chunks

def generate_responses(chunks):
    """Generate responses for each chunk using the LLM."""
    responses = []
    for chunk in chunks:
        input_ids = chunk.unsqueeze(0)  # Add batch dimension
        # Increase max_length to a value greater than or equal to the longest chun
        output = model.generate(input_ids, max_length=512)  # Generate response
        responses.append(tokenizer.decode(output[0], skip_special_tokens=True))

    return responses
```

```
# Example long text
long_text = "brief explain about generative ai " * 50  # Repeat to simulate long t

# Chunk the text
chunks = chunk_text(long_text)

# Generate responses for each chunk
responses = generate_responses(chunks)

# Print the responses
for i, response in enumerate(responses):
    print(f"Response for chunk {i+1}:\n{response}\n")
```

```
The attention mask and the pad token id were not set. As a consequence, you may obs
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Response for chunk 1:
brief explain about generative ai brief explain about generative ai brief explain a
```

- AutoTokenizer and AutoModelForCausalLM are part of the Hugging Face transformers library, which simplifies the process of working with various pre-trained transformer models.
- AutoTokenizer Purpose: AutoTokenizer is designed to automatically retrieve the appropriate tokenizer for a given model. Tokenizers convert raw text into tokens that the model can understand, and they also handle various tasks like adding special tokens, padding, and truncating.
- Usage: You can load a tokenizer by specifying the model name or path. The tokenizer will be automatically configured according to the model's requirements.
- Explanation of the Code
- Loading the Model and Tokenizer:
- The code loads a pre-trained GPT-2 model and its corresponding tokenizer. You can replace "gpt2" with any other compatible model.
- Chunking Function: The chunk_text function takes a string of text and splits it into chunks of a specified maximum length (in tokens). It encodes the text into tokens and then slices it into manageable pieces.
- Generating Responses: The generate_responses function iterates through each chunk, generates a response using the model, and decodes the output back into text.
- Putting It All Together: A long text is created (you can replace this with your actual text). The text is chunked, and responses are generated for each chunk.

- Output The output will show responses generated for each chunk of the input text, allowing you to process longer texts effectively without exceeding the token limit of the model.
- Note When processing multiple chunks, consider how to handle overlapping content, especially if the chunks are related, to maintain context. You might want to implement strategies like including the last few tokens of the previous chunk in the next one.