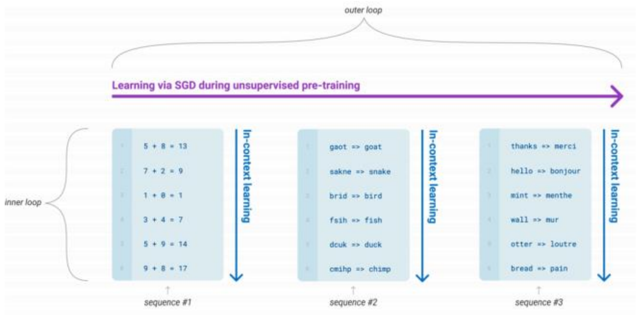
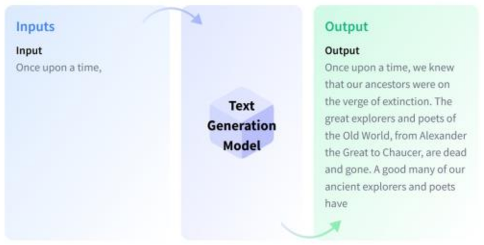
1. Target task chosen: Text generation

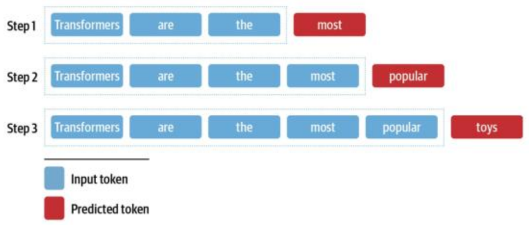
Text generation is a natural language processing task that involves creating coherent and contextually relevant textual content. It can be both creative and utilitarian, finding applications in various domains.

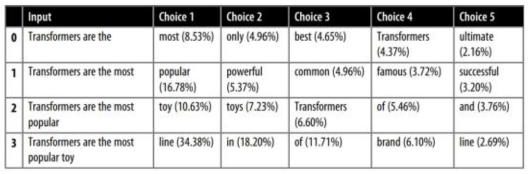




Once upon a time,

Once upon a time, we knew that our ancestors were on the verge of extinction. The great explorers and poets of the Old World, from Alexander the Great to Chaucer, are dead and gone. A good many of our ancient explorers and poets have





	Input	Choice 1	Choice 2	Choice 3	Choice 4	Choice 5
0	Transformers are the	most (8.53%)	only (4.96%)	best (4.65%)	Transformers (4.37%)	ultimate (2.16%)
1	Transformers are the most	popular (16.78%)	powerful (5.37%)	common (4.96%)	famous (3.72%)	successful (3.20%)
2	Transformers are the most popular	toy (10.63%)	toys (7.23%)	Transformers (6.60%)	of (5.46%)	and (3.76%)
3	Transformers are the most popular toy	line (34.38%)	in (18.20%)	of (11.71%)	brand (6.10%)	line (2.69%)

1.1 Challenges in Generating Coherent Text

Repetition: The model can repeat itself, generating the same text over and over again.

Limited Vocabulary: The model can use the same words and phrases over and over again.

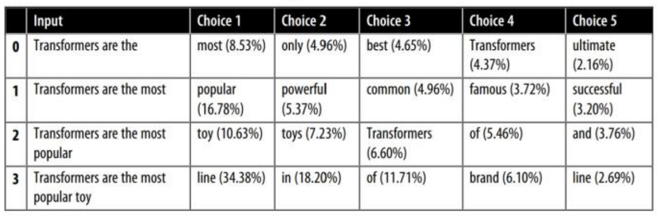
Lack of Context: The model can generate sentences that are grammatically correct, but lack context and meaning.

1.2 How to Generate Text

- **Greedy Search**: The model generates the word with the highest probability as the next word.
- **Beam Search**: The model generates the top k words and keeps track of the probability of each sequence. The sequence with the highest probability is used as the next sequence.
- **Top-K Sampling**: The model generates the top k words and samples from those words using their probabilities as weights.a

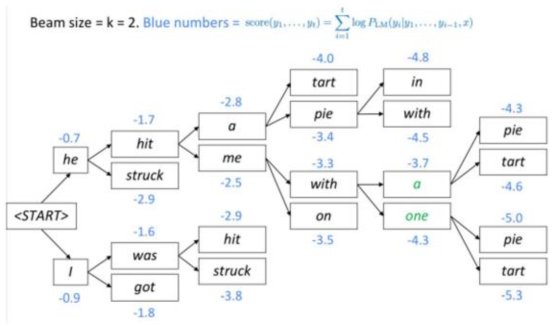
- **Top-p (nucleus) Sampling**: The model generates the smallest possible set of words whose cumulative probability exceeds the probability p. The model then samples from those words using their probabilities as weights.

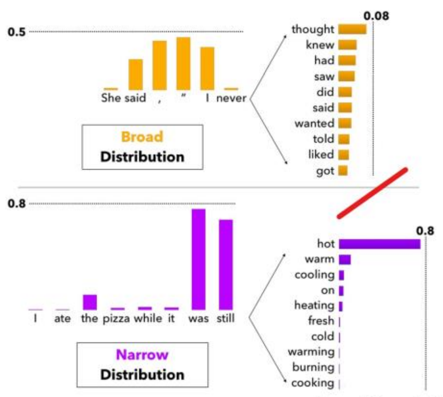
Greedy Search

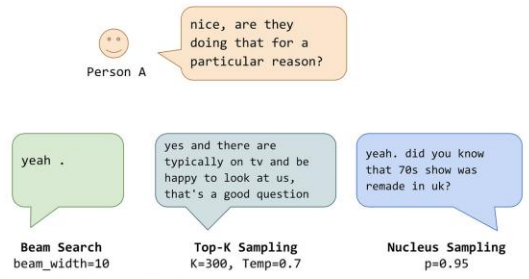


	Input	Choice 1	Choice 2	Choice 3	Choice 4	Choice 5
0	Transformers are the	most (8.53%)	only (4.96%)	best (4.65%)	Transformers (4.37%)	ultimate (2.16%)
1	Transformers are the most	popular (16.78%)	powerful (5.37%)	common (4.96%)	famous (3.72%)	successful (3.20%)
2	Transformers are the most popular	toy (10.63%)	toys (7.23%)	Transformers (6.60%)	of (5.46%)	and (3.76%)
3	Transformers are the most popular toy	line (34.38%)	in (18.20%)	of (11.71%)	brand (6.10%)	line (2.69%)

Beam Search Decoding







Person A: nice, are they doing that for a particular reason?

Beam Search beam_width=10: yeah .

Top-K Sampling K=300, Temp=0.7: yes and there are typically on tv and be happy to look at us, that's a good question

Nucleus Sampling p=0.95: yeah. did you know that 70s show was remade in uk?

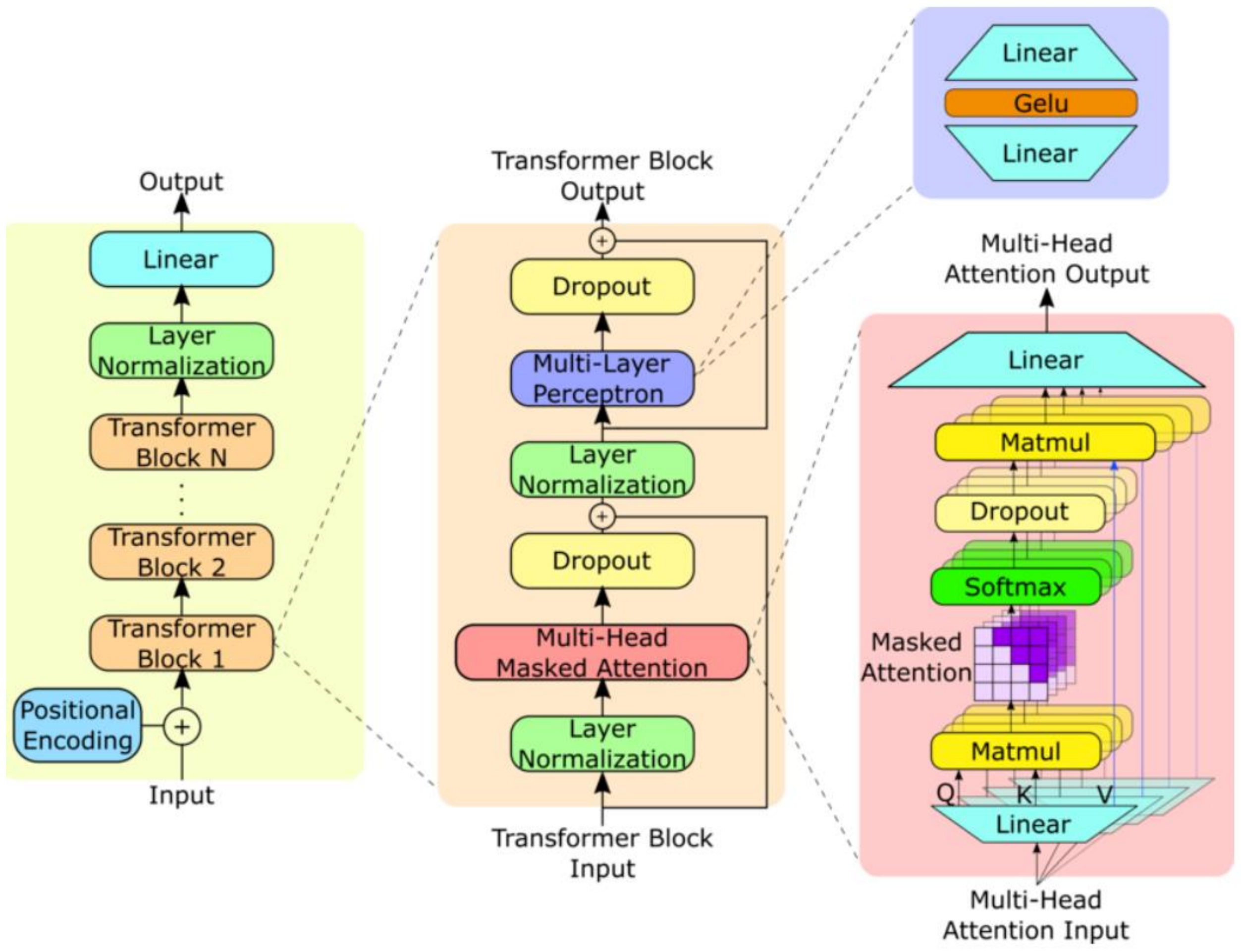
Top-K and Nucleus (Top-p) Search Decoding

2. Model selection: GPT-2

- https://huggingface.co/gpt2

2.1. Architecture:

- GPT-2 employs the transformer architecture, which utilizes self-attention mechanisms to capture contextual relationships in text.
- The GPT-2 model contains N Transformer decoder blocks, as shown in the left panel. Each decoder block (center panel) includes a multi-head masked attention layer, a multi-layer perceptron layer, normalization, and dropout layers. The residual connection (branching line to the addition operator) allows the block to learn from the previous block's input. The multi-head masked attention layer (right panel) calculates attention scores using Q, K, and V vectors to capture sequential relationships in the input sequence..



2.2. Pre-training:

- GPT-2 is initially pre-trained on a massive corpus of text data. During pre-training, the model learns to predict the next word in a sentence given the previous words. This process enables the model to learn contextual information and patterns from the input text.

2.3. Attention Mechanism:

- GPT-2's self-attention mechanism allows it to weigh the importance of each word/token in relation to others. This enables the model to understand the relationships between words regardless of their position in the sequence.

2.4. Fine-tuning and Adaptation:

- After pre-training, GPT-2 can be fine-tuned on specific tasks by providing task-specific training data. This process adapts the model to perform well on tasks like text completion, question answering, or text generation.

2.5. Text Generation:

- To generate text, GPT-2 takes an initial input (prompt) and generates new text by predicting the next word/token based on the context of the input and the words it has already generated. The model samples from its predicted word distribution to produce a sequence of words, resulting in coherent and contextually relevant text generation.

3. Quantization:

As their name suggests, Large Language Models (LLMs) are often too large to run on consumer hardware. These models may exceed billions of parameters and generally need GPUs with large amounts of VRAM to speed up inference.

As such, more and more research has been focused on making these models smaller through improved training, adapters, etc. One major technique in this field is called ***quantization***.

Post-Training Quantizatio:

One of the most popular quantization techniques is post-training quantization (PTQ). It involves quantizing a model's parameters (both weights and activations) **after** training the model.

Quantization of the *weights* is performed using either symmetric or asymmetric quantization.

Quantization of the *activations* , however, requires inference of the model to get their potential distribution since we do not know their range.

There are two forms of quantization of the activations:

- *Dynamic* Quantization
- *Static* Quantization

In our case we choose ***Dynamic Quantization.***

Here are some reasons why dynamic quantization is a suitable choice for the GPT-2 text generation task:

Reduced Model Size: Dynamic quantization reduces the size of the GPT-2 model by quantizing the weights from 32-bit floating point to 8-bit integers, making it more storage and memory-efficient without requiring extensive changes to the model architecture.

Improved Inference Speed: Quantized models, particularly when using dynamic quantization, can execute faster during inference as the computations are performed on lower-precision data types, which can be especially beneficial for deploying the model in environments with limited resources.

Minimal Impact on Performance: Dynamic quantization typically has a minimal impact on the accuracy or performance of the model, which is crucial for tasks like text generation where maintaining high-quality outputs is important.

Ease of Implementation: Dynamic quantization is relatively easy to apply and does not require retraining the model. This makes it a convenient choice when the primary goal is to optimize an already fine-tuned model for deployment.

4. Fine-Tuning the model

Fine-tune the GPT-2 model on a text generation task using a specific dataset.

4.1 Dataset Description

Dataset Used: Wikitext-2 (wikitext-2-raw-v1)

Source: The Wikitext-2 dataset is a popular dataset for language modeling tasks and is available via the Hugging Face datasets library.

4.2 Why This Dataset?

Relevance: Wikitext-2 contains over 100 million tokens extracted from Wikipedia articles, making it a rich source of diverse, high-quality text. The dataset's clean, well-structured nature aligns with the requirements for fine-tuning GPT-2, which benefits from diverse sentence structures and vocabulary.

Size: It is large enough to fine-tune a language model effectively without being too overwhelming in terms of computational resources. For me I am using Google Colab, the dataset was suitable for the GPU environment provided by Google Colab.

Benchmarking: Wikitext-2 is commonly used for benchmarking language models, allowing for easier comparison of the fine-tuned model's performance against other models in the literature.

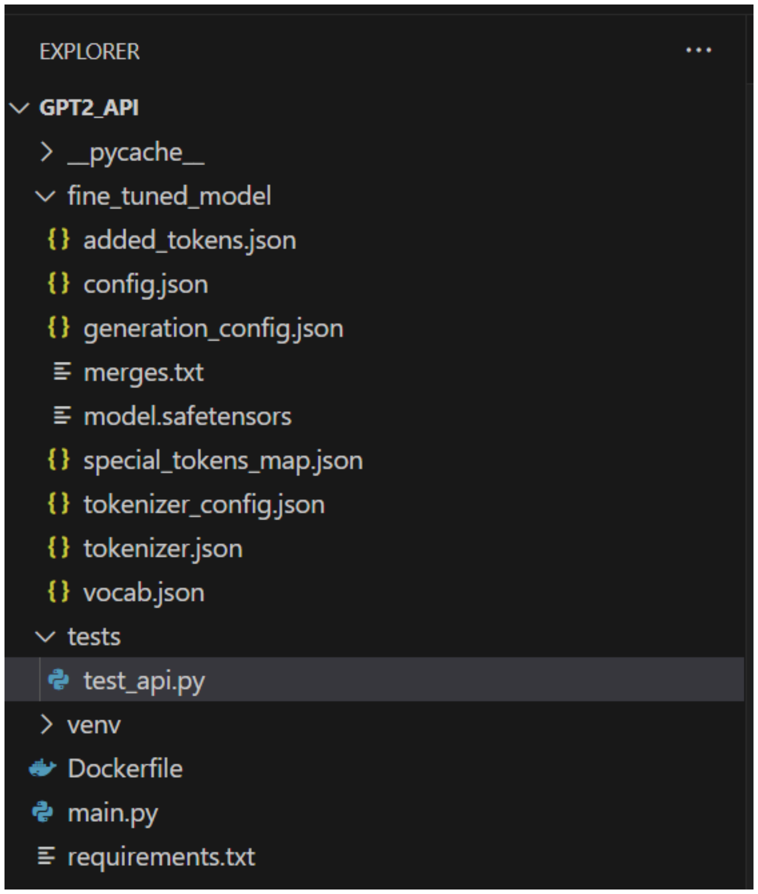
4.3 Preprocessing Steps:

Tokenization: The text was tokenized using the GPT-2 tokenizer, with special attention to padding, truncation, and ensuring that the labels matched the input IDs for language modeling.

Padding Token Addition: A padding token [PAD] was added to handle variable-length inputs during training.

Mapping: The dataset was processed to apply tokenization in batches, ensuring that each text segment was appropriately truncated or padded to a maximum length of 128 tokens.

5. API Creation



```
EXPLORER
GPT2_API
  __pycache__
  fine_tuned_model
    added_tokens.json
    config.json
    generation_config.json
    merges.txt
    model.safetensors
    special_tokens_map.json
    tokenizer_config.json
    tokenizer.json
    vocab.json
  tests
    test_api.py
  venv
  Dockerfile
  main.py
  requirements.txt
```

The file structure in the screenshot highlights the organization of our project, with the key files being:

fine_tuned_model/: Contains the model files and configurations needed to run the fine-tuned model.

tests/test_api.py: Contains the test cases for the API to ensure it works correctly.

Dockerfile: Defines the steps to create a Docker image, encapsulating the API and model.

main.py: contain the entry point for the API where the FastAPI server is started.

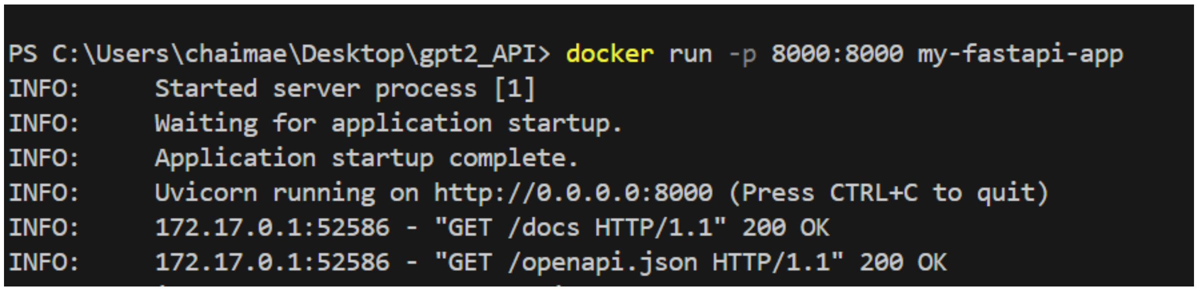
requirements.txt: Lists the Python dependencies needed for the project, which will be installed when building the Docker image.

6. Containerization

- Creating a Dockerfile that automates the setup of the environment, ensuring that all dependencies are installed correctly.
- Build and run the Docker image, verifying that the API functions as expected within the container.

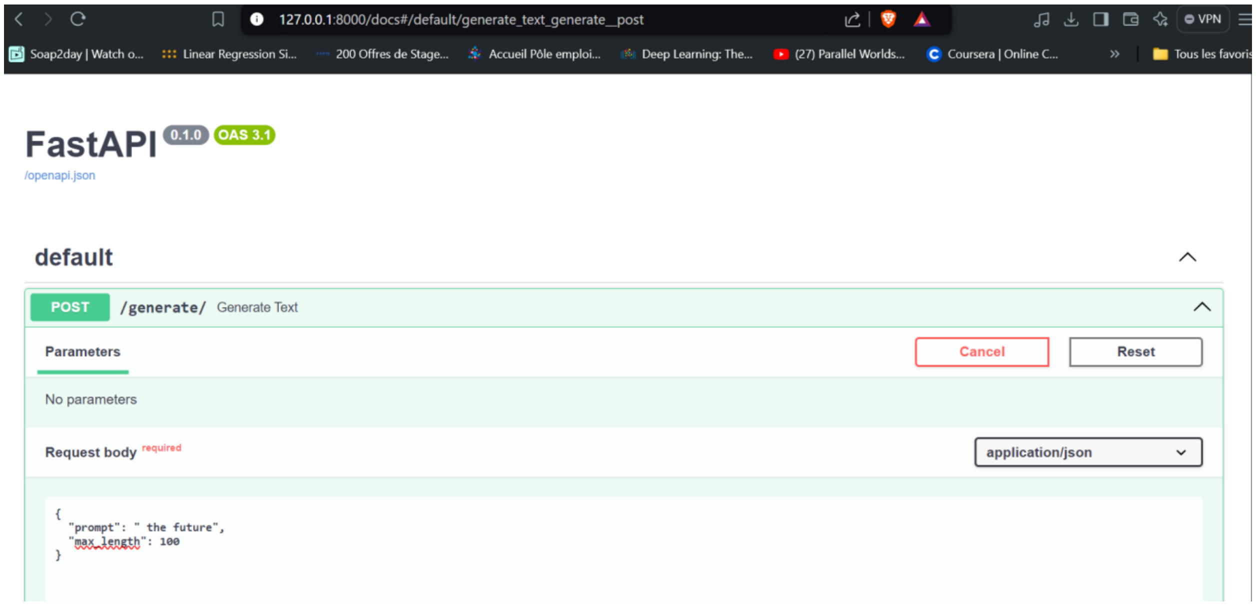
Let's Test ;)

1. First, we run our docker image



```
PS C:\Users\chaimae\Desktop\gpt2_API> docker run -p 8000:8000 my-fastapi-app
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     172.17.0.1:52586 - "GET /docs HTTP/1.1" 200 OK
INFO:     172.17.0.1:52586 - "GET /openapi.json HTTP/1.1" 200 OK
```

2. Time to test ;)



127.0.0.1:8000/docs#/default/generate_text_generate__post

FastAPI 0.1.0 OAS 3.1

/openapi.json

default

POST /generate/ Generate Text

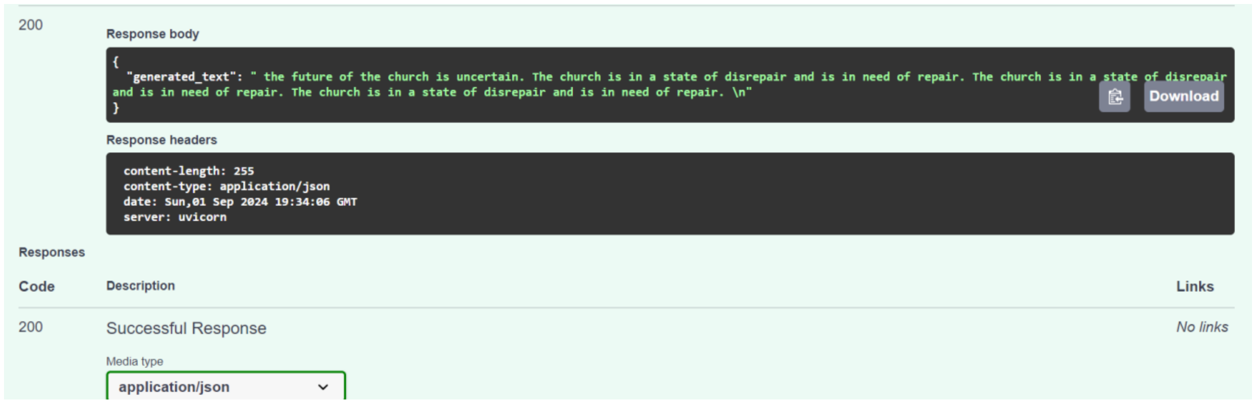
Parameters

No parameters

Request body required — application/json

```
{
  "prompt": " the future",
  "max_length": 100
}
```

3. Et voila



200

Response body

```
{
  "generated_text": " the future of the church is uncertain. The church is in a state of disrepair and is in need of repair. The church is in a state of disrepair and is in need of repair. The church is in a state of disrepair and is in need of repair. \n"
}
```

Response headers

```
 content-length: 255
 content-type: application/json
 date: Sun,01 Sep 2024 19:34:06 GMT
 server: uvicorn
```

Responses

Code	Description	Links
200	Successful Response	No links

Media type: application/json