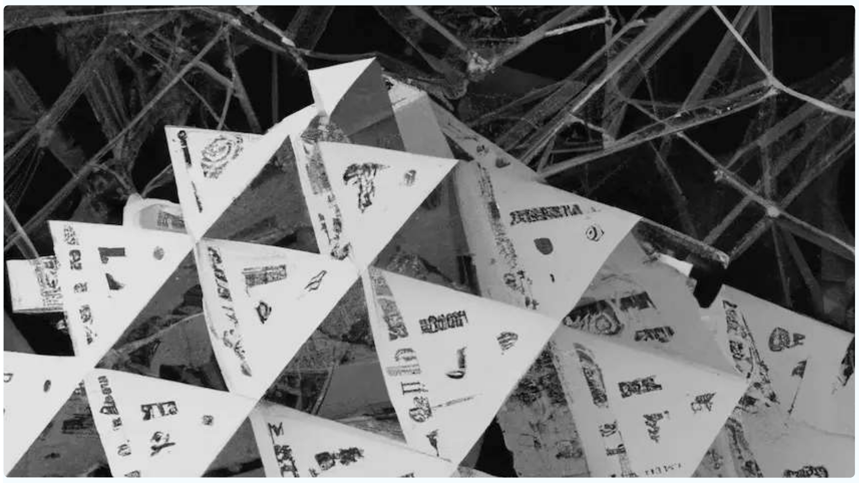
5 Approaches To Solve LLM Token Limits

Estimated Reading time: 3 minutes and 11 seconds

by Des Holmes



> ^ Thanks DALL·E for: *"An artistic interpretation of an AI LLM"*

TL;DR

The following approaches can be used to work around token limits in large language models (LLMs):

1. Truncation
2. Sampling
3. Chunking
4. Encoding and Decoding
5. Fine-tuning

Introduction

Language models have revolutionized natural language processing (NLP) tasks, such as text generation, summarisation, and translation. However, one of the challenges in working with language models is the limitation imposed by the maximum number of tokens they can handle.

> **What are tokens?** Tokens can be thought of as pieces of words. Tokens are not cut up exactly where the words start or end; tokens can include trailing spaces and even sub-words. Here are some helpful rules of thumb for understanding tokens in terms of lengths:
>
> - 1 token ~= 4 chars in English
> - 1 token ~= ¾ words
> - 100 tokens ~= 75 words
>
> Or
>
> - 1-2 sentence ~= 30 tokens
> - 1 paragraph ~= 100 tokens
> - 1,500 words ~= 2048 tokens
>
> Source: OpenAI: What are tokens and how to count them?

Models like GPT-3 have a maximum token limit, beyond which they cannot accept input or generate output.

In this blog post, we will explore different approaches to overcome token limits in language models, along with some TypeScript code examples.

Approaches for Solving Token Limits

1. Truncation

Truncation involves removing a portion of the input text to fit within the token limit. This can be done by either removing the beginning or the end of the text, or a combination of both. However, truncation may result in loss of important information and may impact the quality and coherence of the generated output.

Here's an example of truncation in TypeScript:

```
const inputText = "This is a long input text that exceeds the token limit.";
const maxTokens = 50;
const truncatedText = inputText.slice(0, maxTokens);
```

2. Sampling

Sampling is a technique where you randomly select a subset of tokens from the input text. This allows you to retain some diversity in the input and can help in generating varied outputs. However, this approach (similar to truncation) may result in loss of contextual information, and reduce the quality of the generated output.

Here's an example of sampling in TypeScript:

```
const inputText = "This is a long input text that exceeds the token limit.";
const maxTokens = 50;
const sampledText = inputText.split(" ").sort(() => 0.5 - Math.random()).slice(0, maxTokens).join(" ");
```

3. Chunking

Another approach is to split the input text into smaller chunks or segments that are within the token limit and process them sequentially. This way, each chunk can be processed independently, and the outputs can be concatenated to obtain the final result.

Here's an example in TypeScript:

```
const inputText = "This is a long text that exceeds the token limit of the language model.";
const maxTokens = 2048;

// Split the input text into smaller chunks
const chunks = inputText.match(/.{1, maxTokens}/g);

// Process each chunk separately
const outputs = [];
for (const chunk of chunks) {
  const output = await generateText(chunk); // Call to the language model API
  outputs.push(output);
}

// Concatenate the outputs to obtain the final result
const result = outputs.join("");
```

4. Encoding and Decoding

Encoding and decoding are common NLP techniques that convert text data into a numerical representation and vice versa. These techniques can be used to compress, decompress, truncate, or expand text to fit within the token limit of a language model. This approach requires additional pre-processing steps and may impact the readability of the generated output.

5 Fine-tuning

Fine-tuning is a transfer learning technique that allows adapting a pre-trained language model to a specific task or domain with a smaller amount of task-specific data. Fine-tuning can be leveraged to solve token limits in language models by training the model to predict the next token in a sequence of text that has been chunked or divided into smaller parts, each of which is within the token limit of the model.

High-level steps involved in fine-tuning a language model:

1. Prepare the training data
2. Create a fine-tuning model
3. Use the fine-tuning model to generate text

Check out the OpenAI fine-tuning guide for more details on this approach.

Popular related posts for Product Development, Technical Direction and SaaS:

> - Using The BlurHash Algorithm With A TypeScript Azure Function
> - ImageFly: On-The-Fly Image Transformations

Share:

---

tags: #openai #nlp #gpt-3 #typescript #llm #ai #technical-direction #web-development #product-development #saas #innovation