Extensions

The pg_tiktoken extension

Efficiently tokenize data in your PostgreSQL database using OpenAI's `tiktoken` library

The `pg_tiktoken` extension enables fast and efficient tokenization of data in your PostgreSQL database using OpenAI's tiktoken library.

This topic provides guidance on installing the extension, utilizing its features for tokenization and token management, and integrating the extension with ChatGPT models.

What is a token?

Language models process text in units called tokens. A token can be as short as a single character or as long as a complete word, such as "a" or "apple." In some languages, tokens may comprise less than a single character or even extend beyond a single word.

For example, consider the sentence "Neon is serverless Postgres." It can be divided into seven tokens: ["Ne", "on", "is", "server", "less", "Post", "gres"].

pg_tiktoken functions

The `pg_tiktoken` offers two functions:

- `tiktoken_encode`: Accepts text inputs and returns tokenized output, allowing you to seamlessly tokenize your text data.
- `tiktoken_count`: Counts the number of tokens in a given text. This feature helps you adhere to text length limits, such as those set by OpenAI's language models.

Install the pg_tiktoken extension

You can install the `pg_tiktoken` extension by running the following `CREATE EXTENSION` statement in the Neon **SQL Editor** or from a client such as `psql` that is connected to Neon.

```
CREATE EXTENSION pg_tiktoken
```

For information about using the Neon **SQL Editor**, see Query with Neon's SQL Editor. For information about using the `psql` client with Neon, see Connect with psql.

Use the tiktoken_encode function

The `tiktoken_encode` function tokenizes text input and returns a tokenized output. The function accepts encoding names and OpenAI model names as the first argument and the text you want to tokenize as the second argument, as shown:

```
SELECT tiktoken_encode('text-davinci-003', 'The universe is a vast and captivating mystery,    t

 tiktoken_encode
--------------------------------------------------------------------------------
 {464,6881,318,257,5909,290,3144,39438,10715,11,4953,284,307,18782,290,7247,13}
(1 row)
```

The function tokenizes text using the Byte Pair Encoding (BPE) algorithm.

Use the tiktoken_count function

The `tiktoken_count` function counts the number of tokens in a text. The function accepts encoding names and OpenAI model names as the first argument and text as the second argument, as shown:

```
neondb=> SELECT tiktoken_count('text-davinci-003', 'The universe is a vast and captivating ı    e

 tiktoken_count
----------------
             17
(1 row)
```

Supported models

The `tiktoken_count` and `tiktoken_encode` functions accept both encoding and OpenAI model names as the first argument:

```
tiktoken_count(<encoding or model>,<text>)
```

The following models are supported:

Encoding name	OpenAI model
cl100k_base	ChatGPT models, text-embedding-ada-002
p50k_base	Code models, text-davinci-002, text-davinci-003
p50k_edit	Use for edit models like text-davinci-edit-001, code-davinci-edit-001
r50k_base (or gpt2)	GPT-3 models like davinci

Integrate pg_tiktoken with ChatGPT models

The `pg_tiktoken` extension allows you to store chat message history in a PostgreSQL database and retrieve messages that comply with OpenAI's model limitations.

For example, consider the `message` table below:

```
CREATE TABLE message (
  role VARCHAR(50) NOT NULL, -- equals to 'system', 'user' or 'assistant'
  content TEXT NOT NULL,
  created TIMESTAMP NOT NULL DEFAULT NOW(),
  n_tokens INTEGER -- number of content tokens
);
```

The gpt-3.5-turbo chat model requires specific parameters:

```
{
  "model": "gpt-3.5-turbo",
  "messages": [
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who won the world series in 2020?"},
        {"role": "assistant", "content": "The Los Angeles Dodgers won the World Series in 2020."
```

```
    ]
}
```

The `messages` parameter is an array of message objects, with each object containing two pieces of information: The `role` of the message sender (either `system`, `user`, or `assistant`) and the actual message `content`. Conversations can be brief, with just one message, or span multiple pages as long as the combined message tokens do not exceed the 4096-token limit.

To insert `role`, `content`, and the number of tokens into the database, use the following query:

```
INSERT INTO message (role, content, n_tokens)
VALUES ('user', 'Hello, how are you?', tiktoken_count('text-davinci-003','Hello, how are you?'))
```

Manage text tokens

When a conversation contains more tokens than a model can process (e.g., over 4096 tokens for `gpt-3.5-turbo`), you will need to truncate the text to fit within the model's limit.

Additionally, lengthy conversations may result in incomplete replies. For example, if a `gpt-3.5-turbo` conversation spans 4090 tokens, the response will be limited to just six tokens.

The following query retrieves messages up to your desired token limits:

```
WITH cte AS (
  SELECT role, content, created, n_tokens,
         SUM(tokens) OVER (ORDER BY created DESC) AS cumulative_sum
  FROM message
)

SELECT role, content, created, n_tokens, cumulative_sum
FROM cte
WHERE cumulative_sum <= <MAX_HISTORY_TOKENS>;
```

`<MAX_HISTORY_TOKENS>` represents the conversation history you want to keep for chat completion, following this formula:

```
MAX_HISTORY_TOKENS = MODEL_MAX_TOKENS - NUM_SYSTEM_TOKENS - NUM_COMPLETION_TOKENS
```

For example, assume the desired completion length is 100 tokens (`NUM_COMPLETION_TOKENS=90`).

```
MAX_HISTORY_TOKENS = 4096 - 6 - 90 = 4000
```

```
{
  "model": "gpt-3.5-turbo", // MODEL_MAX_TOKENS = 4096
  "messages": [
        {"role": "system", "content": "You are a helpful assistant."}, // NUM_SYSTEM_TOKENS = 6
        {"role": "user", "content": "Who won the world series in 2020?"},
        {"role": "assistant", "content": "The Los Angeles Dodgers won the World Series in 2020.
        {"role": ...}
        .
        .
        .
        {"role": "user", "content": "Great! Have a great day."}  // MAX_HISTORY_TOKENS = 4000
    ]
}
```

Conclusion

In conclusion, the `pg_tiktoken` extension is a valuable tool for tokenizing text data and managing tokens within PostgreSQL databases. By leveraging OpenAI's tiktoken library, it simplifies the process of tokenization and working with token limits, enabling you to integrate more easily with with OpenAI's language models.

As you explore the capabilities of the `pg_tiktoken extension` , we encourage you to provide feedback and suggest features you'd like to see added in future updates. We look forward to seeing the innovative natural language processing applications you create using `pg_tiktoken` .

Resources

- Open AI tiktoken source code on GitHub
- pg_tiktoken source code on GitHub

Need help?

Send a request to support@neon.tech, or join the Neon community forum.

NEON

All systems operational

Made in SF and the World