It’s Hard to preserve LLM streaming when using function calls. Don’t Fret, I Got Your Back!

\newline Team <us@fullstack.io>
To: <dsaad68@gmail.com>

Thu 31. Oct 2024 at 05:27

How do I have your back?

It’s with the help of Luise Sanna and his best work yet! **Responsive LLM Applications with Server-Sent Events with Louis Sanna.**

But let’s talk about that at the end of this email.

For now, let’s just focus on the matter at hand.

I’ve got three way for you to preserve LLM streaming when using function calls.

Yes, you heard me right. Three!!!

1. Use Asynchronous Functions

You need to ensure that all functions called during the streaming process are asynchronous. This will prevent blocking the main event loop and allow other tasks to execute at the same time.

Example:

```
python
Copy code
import asyncio
from fastapi import FastAPI, Request, Response
from fastapi.responses import StreamingResponse
app = FastAPI()
async def async_function(data):
await asyncio.sleep(1)
# Simulate a non-blocking I/O operation
    return f"Processed: {data}"
async def stream_generator(request: Request):
    data_chunks = ["chunk1", "chunk2", "chunk3"]
    for chunk in data_chunks:
        processed_chunk = await async_function(chunk)
        yield f"data: {processed_chunk}\n\n"
        await asyncio.sleep(0.1)  # Simulate delay between chunks
@app.get("/stream")
```

```
async def stream(request: Request):
return StreamingResponse(stream_generator(request),
       media_type="text/event-stream")
```

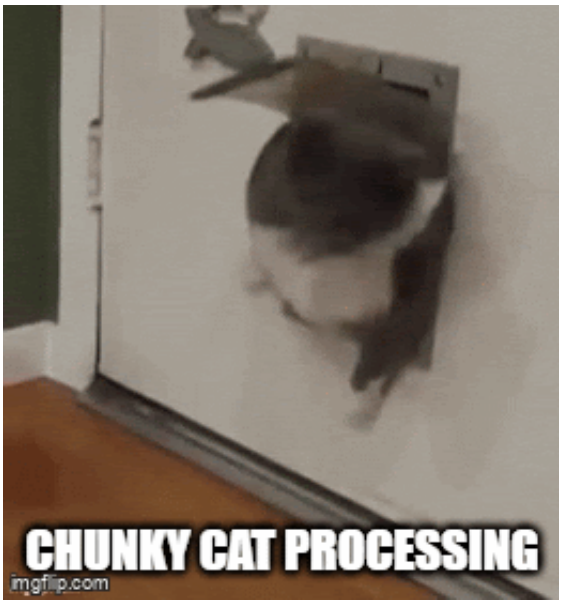
2. Leverage Background Tasks

If you have heavy computational tasks, you need to offload them to background tasks. This allows the main streaming function to continue yielding data without delays.

```
python
Copy code
import asyncio
from fastapi import FastAPI, BackgroundTasks
from fastapi.responses import StreamingResponse
app = FastAPI()
async def background_task(data, result_container):
    await asyncio.sleep(2)  # Simulate a long-running task
    result_container.append(f"Processed: {data}")
async def stream_generator(request, background_tasks):
    data_chunks = ["chunk1", "chunk2", "chunk3"]
    results = []
    for chunk in data_chunks:
        background_tasks.add_task(background_task, chunk, results)
        yield f"data: Processing {chunk}\n\n"
        await asyncio.sleep(0.1)  # Simulate delay between chunks
    while background_tasks.has_tasks:
        await asyncio.sleep(0.1)
    for result in results:
```

```
        yield f"data: {result}\n\n"
@app.get("/stream")
async def stream(request: Request, background_tasks: BackgroundTasks):
    return StreamingResponse(stream_generator(request, background_tasks), media
```

3. Chunked Data Processing



Instead of processing entire datasets at once, break down the data into smaller chunks. Process and stream each chunk sequentially to maintain responsiveness.

Example:

```
python
Copy code
import asyncio
from fastapi import FastAPI
from fastapi.responses import StreamingResponse
app = FastAPI()
async def process_chunk(chunk):
    await asyncio.sleep(1)  # Simulate processing time
    return f"Processed: {chunk}"
async def stream_generator(request):
    data_chunks = ["chunk1", "chunk2", "chunk3"]
    for chunk in data_chunks:
        processed_chunk = await process_chunk(chunk)
        yield f"data: {processed_chunk}\n\n"
        await asyncio.sleep(0.1)  # Simulate delay between chunks
@app.get("/stream")
async def stream(request: Request):
return StreamingResponse(stream_generator(request),
       media_type="text/event-stream")
```

These techniques help maintain the efficiency and responsiveness of your LLM streaming application, even when incorporating function calls.

And now, I introduce you to the Author of the course, the course which allowed me to write this email today.

Meet **Louis Sanna** not to be confused with **Louis Sama,** which according to Sheikh Google “Sama (様, さま) is **a more respectful version for individuals of a higher rank than oneself”**

Hah, based on the description I don’t think Louis will mind even if you do.

Louis is a Tech Lead with a decade of experience in startups and consulting. He’s built GenAI applications for international organizations (UNESCO, Singapore Government, etc.) and private companies (Renault, Private Equity Funds).

And now that you've stayed with me in my agony of writing this email, I have a **Gift,** especially for you:

You can check out the entire first 5 lessons of the course for free!

If you're hooked on this newfound enlightenment, don't forget to check out **Responsive LLM Applications with Server-Sent Events with Louis Sanna.** (Currently on sale)

Responsive LLM Applications with Server-Sent Events

1. **System Design for AI Applications**
2. **Building the Front-End**
3. **Building the Backend**
4. **Building a Chat**
5. **Retrieval Augmented Generation**
6. **[Advanced Package] Building an Agent**

End-to-end responsive application with server-sent events, this course has it all.

To top it all off, the entire first module "**System Design for AI Applications**" is free for everyone. And… a few sections of the "**Building the Front-End**" are also free.

Well, that was all from me for now.

More soon,

Zao

\newline Team
\newline
us@fullstack.io
www.newline.co/