Resource usage in production environment

In the case of LLM Verbatim, the production scenario of model serving is driven by a REST API call to LLMSuite library. LLMSuite library then leverages GPT4o-mini model for inference. In such case, the returned payload consist of results sought indicated by the system prompt. Also, such payload returned by GPT4o-mini does not contain information regarding resource utilization on the server side. This means we can only measure the resource utilization in the client side.

Implementing MLFlow to track or monitor metrics may be easily done using a context wrap:

```
with mlflow.start_run():
```

Below are examples showing how to use MLFlow's logging capability for resource utilization metrics.

Resource monitoring

As an example, below is a function that measures CPU and memory usage of an application running on client side:

```
import mlflow
import psutil

def log_system_usage():
    # Get CPU and memory stats
    cpu_percent = psutil.cpu_percent(interval=1)
    memory_info = psutil.virtual_memory()

    # Log metrics to MLflow
    mlflow.log_metric("cpu_usage_percent", cpu_percent)
    mlflow.log_metric("memory_usage_percent", memory_info.percent)
    mlflow.log_metric("memory_available_mb", memory_info.available / (1024
** 2))

    # Print for reference (optional)
    print(f"CPU Usage: {cpu_percent}%")
    print(f"Memory Usage: {memory_info.percent}%")
    print(f"Memory Available: {memory_info.available / (1024 ** 2)} MB")

# Use this function within an MLflow run
with mlflow.start_run():
    log_system_usage()
```

The example above shows that as long as a metric may be defined and captured, it may be passed into `mlflow.log_metric` function to register the value anjd be displayed in the MLFlow UI.

Specifically, the example above shows:

- Logging CPU and Memory Metrics: Each time log_system_usage() runs, it logs the CPU and memory usage to MLflow, allowing you to monitor system load alongside model metrics.
- MLflow UI: You'll be able to see system performance in the MLflow Metrics tab, providing insight into resource usage during model serving.

This also provides flexibility for users to define and calaulate their own metrics, and pass these into `mlflow.log_metric` funcrtion.

Continuous resource monitoring

For ongoing logging, a way to extend the above example is to call `log_system_usage()` at set intervals or integrate it into a model prediction loop to capture system metrics during every prediction request. Here's a sample usage in a loop:

```
import time

# Start an MLflow run
with mlflow.start_run():
    for i in range(100):  # Simulate ongoing monitoring, e.g., during
production
        log_system_usage()
        time.sleep(5)  # Adjust the interval as needed (e.g., every 5
seconds)
```

This setup provides a continuous view of system resource usage over time, which is especially helpful for monitoring production environments.