ORACLE

OCI Generative AI Service

Timeout Error Handling

Generative AI Black Belt team
Oracle EMEA

Current concerns

How to effectively manage timeouts?

How to facilitate a seamless user experience?

1. Optimize Request Payloads

- **Concise Inputs**: Encourage user inputs to be succinct and relevant. Lengthy or complex prompts can increase processing time, leading to potential timeouts.

- **Limit Context Size**: When incorporating conversation history, include only essential context to reduce the payload size.

2. Model Selection and Routing

Implement a routing mechanism that selects the most appropriate LLM based on the complexity and nature of the user query. For instance, simpler queries can be handled by smaller, faster models, while more complex queries are directed to larger, more capable models. This approach optimizes resource utilization and response times. (https://arxiv.org/abs/2404.14618)

Use this routing mechanism as backup when timeout error raises.

3. Implement Exponential Backoff Strategy

We recommend using a back-off strategy to handle request rejections and timeouts. An exponential backoff strategy involves retrying failed requests after progressively longer intervals, which helps manage load and reduces the likelihood of repeated timeouts. (https://docs.oracle.com/en-us/iaas/tools/python/latest/sdk_behaviors/retries.html#exponential-backoff)

Third-party tools like LangChain also includes a retry mechanism that uses exponential backoff to handle API errors, including timeouts. (https://python.langchain.com/api_reference/core/runnables/langchain_core.runnables.retry.RunnableRetry.html)

Managed Service Recommendation: As a recommended step, especially for managed services, it's advisable to add a retry block with a maximum of three retries. This practice helps in handling transient errors effectively without overwhelming the system.

4. Utilize Retry Tokens

When making API calls, include an opc_retry_token to uniquely identify each request. This token allows the request to be retried in case of a timeout or server error without the risk of executing the same action multiple times. (https://docs.oracle.com/en-us/iaas/tools/python/latest/api/generative_ai/client/oci.generative_ai.GenerativeAiClient.html)

Additional Considerations

- **Circuit Breaker Pattern**: Implementing a circuit breaker pattern can prevent your system from making requests to an unresponsive service, allowing it to recover and preventing cascading failures.

- **Load Balancing**: Utilizing OCI Load Balancer to distribute incoming requests evenly across multiple instances can prevent any single instance from becoming overwhelmed, thereby reducing response times and preventing timeouts.

5. Set Appropriate Timeouts

Configure connection and read timeouts to balance responsiveness and reliability. OCI's Python SDK allows setting these timeouts to allow your application to handle delays appropriately. (https://docs.oracle.com/en-us/iaas/tools/python/latest/api/generative_ai_agent_runtime/client/oci.generative_ai_agent_runtime.GenerativeAiAgentRuntimeClient.html)

6. Monitor and Log Requests

Implement comprehensive logging of each API request and response, capturing timestamps, response times, and any error codes. This data can help identify patterns and frequency of timeouts, enabling proactive measures to enhance system reliability.