To create an architecture diagram for generative AI log analyzer using the LangChain framework and the OpenAI API, the system can be broken down into several key components. Here's an outline of the architecture:

# Architecture Diagram Overview

## Log Source (Input Layer):

* A folder (`C:\Log`) containing multiple `.log` files.
* Each log file contains raw log data that needs to be processed and analyzed.

## Log Loader:

* load\_logs\_from\_folder: This function reads all `.log` files from the specified folder and stores them in memory for further processing.

### 3. Text Chunking Layer:

* CharacterTextSplitter: Splits log data into smaller chunks to ensure each part can be analyzed efficiently. It breaks the log data into chunks of 500 characters with a 100-character overlap to maintain context between chunks.

### 4. LLM (Large Language Model) Chain (Processing Layer):

* LLM (OpenAI API): Interacts with OpenAI's language model to process and analyze each chunk of log data.
* Prompt Template: A predefined prompt is passed to the LLM for each chunk of log data, asking it to analyze and detect patterns, anomalies, or errors.
* LLM Chain: A chain combining the prompt and LLM to handle interactions and execute the analysis.

### 5. Error Handling & Retry Logic:

* Exponential Backoff\*\*: In case of rate limits or API errors, the system retries the API call with exponential backoff to avoid overwhelming the service.
* Mock API (Testing Mode)\*\*: A testing environment using a mock API to simulate OpenAI responses, ensuring the system can be tested without calling the real API.

### 6. Aggregation Layer (Output):

* The analyzed chunks are aggregated into a final report.
* The report is either printed to the console or saved as a text file (`log\_analysis\_report.txt`).

### 7. Testing with Mocking:

* The system can be run in testing mode with mocked API responses to simulate real behavior without using the actual API.

### Key Components in the Diagram:

* **Input**: Log folder path (C:\Log) with multiple `.log` files.
* **Processing Pipeline**: Load logs → Split logs → Analyze chunks via OpenAI API (or mock for testing).
* **Error Handling**: Exponential backoff for handling rate limits or errors.
* **Output**: Aggregated report saved as `log\_analysis\_report.txt` and printed.

### Diagram Representation:

Log Folder  
(Multiple .log files from C:\Log)

Log Loader Component

(load\_logs\_from\_folder function)

Text Splitter (CharacterTextSplitter) Splitting logs into smaller chunks (500 chars)

LLM Chain

(LLM OpenAI API + Prompt Template)

Chunk analysis

Exponential backoff (Rate limit retries) Mock API for testing

Aggregation & Output Layer

Final analysis report generated and saved

log\_analysis\_report.txt