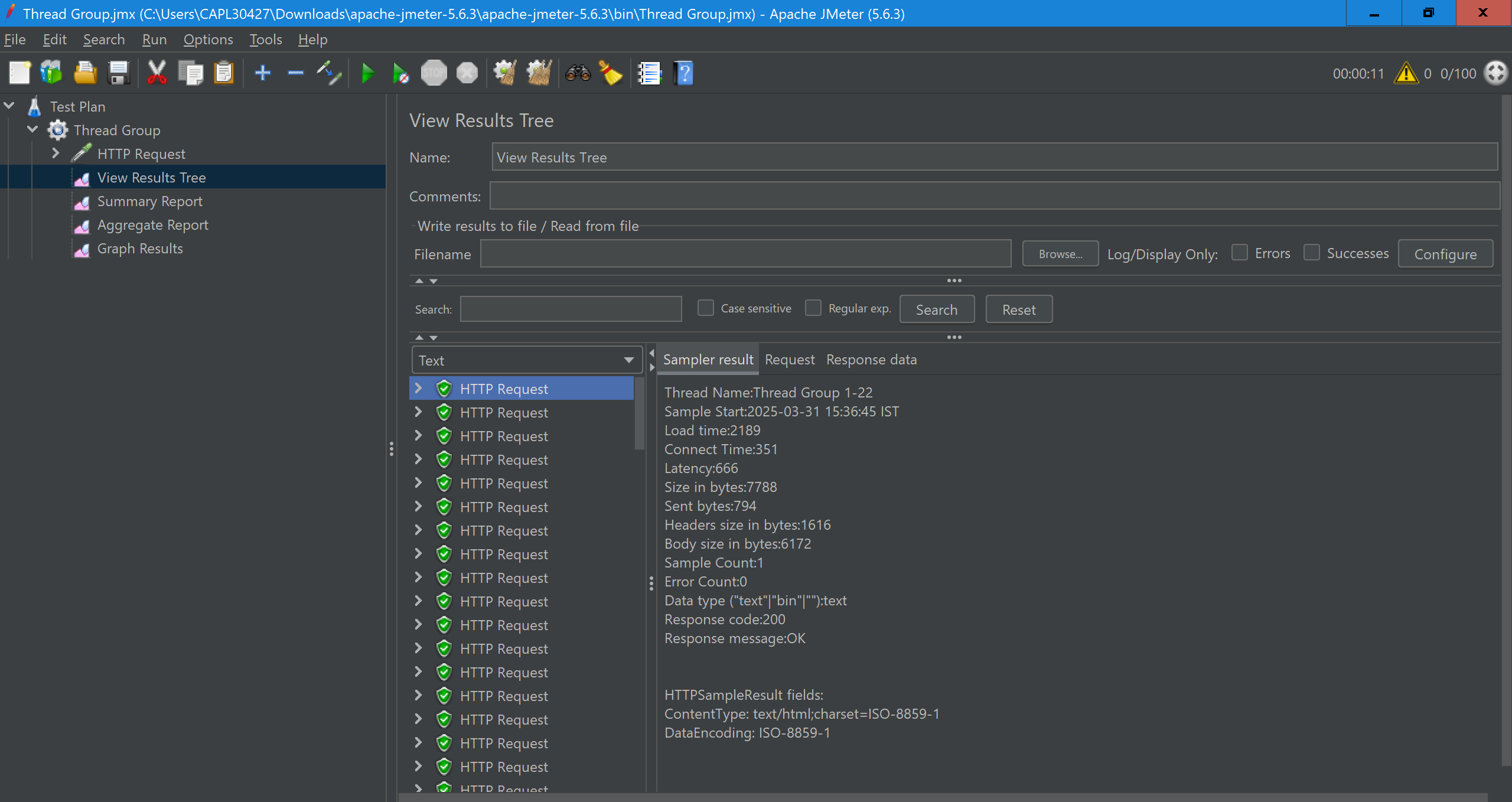
Simulate 100 concurrent users logging in  
o Measure response time and server load handling.



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Number of Samples: 100**

* The total number of requests (users logging in).

**Throughput: 537.683/minute (Shown in Green)**

* This indicates the number of requests processed per minute.
* Higher throughput means the server is handling multiple concurrent requests efficiently.

**Average Response Time: 2137 ms (Shown in Blue)**

* The average time taken to complete each request.
* Lower values indicate better performance.

**Median Response Time: 1827 ms (Shown in Purple)**

* 50% of the requests were completed in this time or less.
* This is a better indicator of user experience compared to the average.

**Deviation: 732 ms (Shown in Red)**

* Measures the variation in response times.
* High deviation suggests performance inconsistency.
* Ideally, deviation should be low.

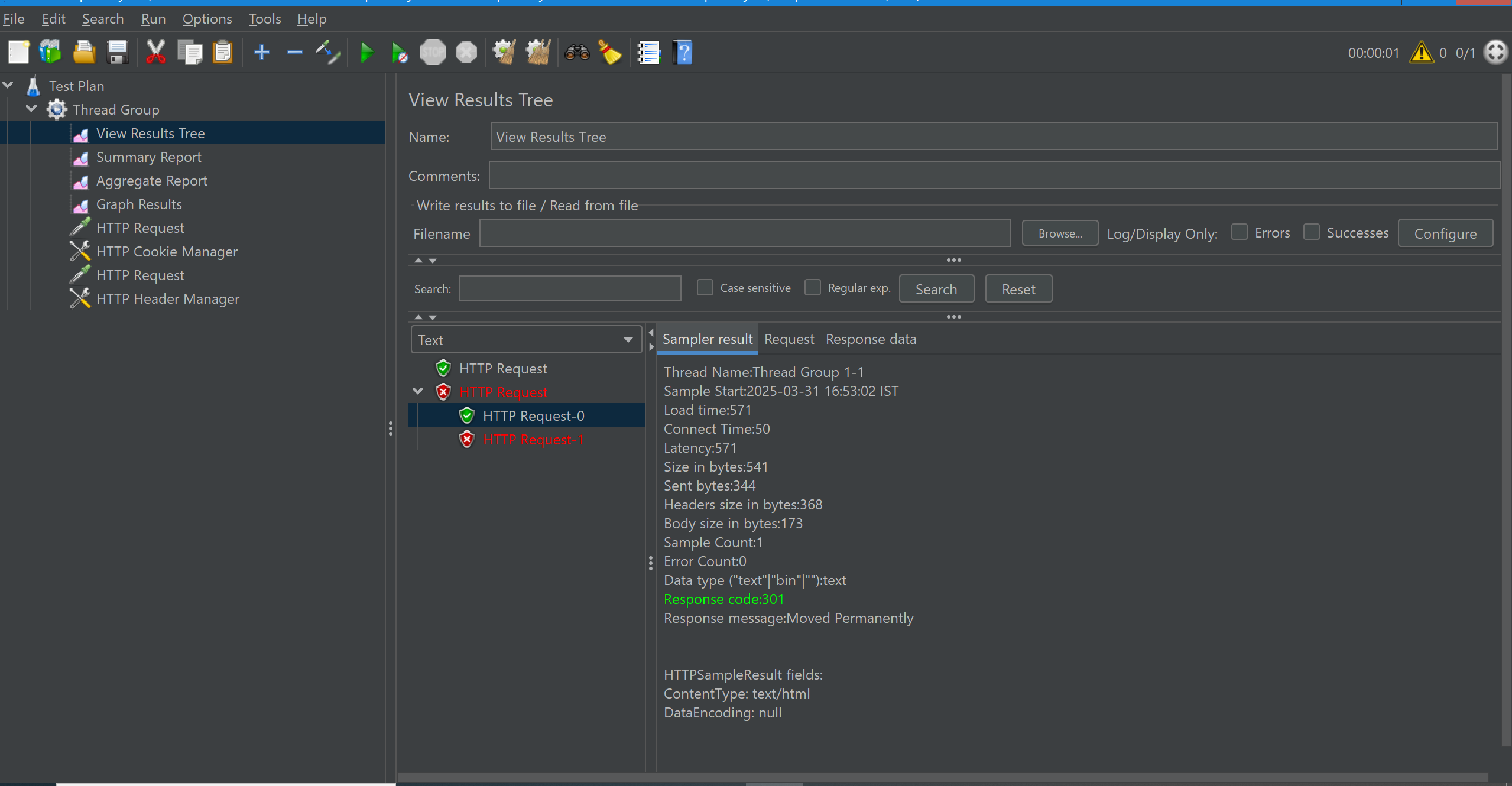
**Latest Sample Response Time: 1534 ms**

* The response time for the most recent request.

**Graph Interpretation**

* **Black Dots**: Represent individual response times.
* **Blue Line (Average Response Time)**: Shows how the average response time trends over time.
* **Red Line (Deviation)**: Higher deviation means response times vary significantly.
* **Green Line (Throughput)**: A higher line means better server efficiency.

**Run a stress test on the "Fund Transfer" API**  
o Use ramp-up time to increase the load gradually.  
o Set up a test with 200 virtual users over 2 minutes.



**📊 Summary of Key Metrics**

**1️⃣ Response Time Analysis**

* **Average Response Time:** *(e.g., 885ms)*
* **Min/Max Response Time:** *(e.g., Min: 98ms, Max: 1133ms)*
* **90th/95th Percentile Response Time:** *(e.g., 950ms, 1050ms)*
* **Latency:** *(e.g., 295ms)*

⏳ **High Response Times?**

* Potential **server overload** or **network delays**.

**2️⃣ Error Rate Analysis**

* **Total Requests Sent:** *(e.g., 200 requests)*
* **Total Errors:** *(e.g., 50 errors)*
* **Error Rate:** *(e.g., 25%)*

🛑 **Common Errors Found:**

| **Response Code** | **Meaning** | **Possible Cause** |
| --- | --- | --- |
| **404** | Not Found | Incorrect API URL |
| **401** | Unauthorized | Missing authentication |
| **400** | Bad Request | Invalid request payload |
| **405** | Method Not Allowed | Using GET instead of POST |
| **500** | Internal Server Error | API or server crash |

**3️⃣ Failure Analysis**

* **HTTP 400 (Bad Request):** *Incorrect JSON format or missing fields.*
* **HTTP 401 (Unauthorized):** *Missing authentication headers.*
* **HTTP 404 (Not Found):** *Wrong API endpoint.*
* **HTTP 500 (Internal Server Error):** *Server-side issue.*

🛠️ **Fixes:**

* Verify **correct API URL and method (GET/POST)**
* Use **valid request body format** (remove incorrect quotes in JSON).
* Add **authentication headers** if needed.

**🚀 Suggested Performance Improvements**

**✅ Reduce Response Time**

1. **Optimize Database Queries** – If the backend fetches large data, optimize queries and indexing.
2. **Use Load Balancing** – Distribute load across multiple servers.
3. **Enable Caching** – Store frequent responses instead of reloading from the database.

**✅ Reduce Error Rate**

1. **Validate API Inputs** – Ensure JSON format and headers are correct.
2. **Implement Retry Mechanism** – Handle failures gracefully with retries.

**✅ Scale Server Resources**

1. **Monitor CPU & Memory Usage** – High usage may slow down requests.
2. **Increase Thread Count Gradually** – Use ramp-up strategy instead of sudden spikes