Given the endpoints described, I'll provide some hypothetical test results and then suggest potential improvements based on those results:

Hypothetical Test Results:

Endpoint1:

Load testing showed that this endpoint, being static, can handle a large number of requests but takes up unnecessary bandwidth for repeatedly sending the same static list.

Endpoint2:

Functionality testing identified that very large numbers cause the application to become sluggish or even crash due to extensive calculations.

Load testing demonstrated that this endpoint had consistent response times across a variety of inputs.

Endpoint3:

Error handling tests showed that the application crashes or hangs when a user inputs non-integer values.

Endurance testing suggested that the system, when constantly throwing exceptions for even numbers, increased CPU utilization and occasional memory leaks.

Potential Improvements:

Endpoint1:

Caching: Consider adding caching mechanisms to reduce the load on the server and speed up the response time. If the data remains static most of the time, serving it from a cache can significantly reduce the resource overhead.

Compression: Use data compression techniques or serve minimized versions of the data to reduce the bandwidth usage.

Endpoint2:

Input Validation: Before performing calculations, ensure the number isn't too large, which might cause performance issues. If a number is outside acceptable bounds, return an appropriate error message.

Optimization: If certain numbers are calculated frequently, consider caching their results to speed up subsequent requests with the same input.

Endpoint3:

Improved Error Handling: Instead of raising an exception for even numbers, return a well-structured error response. This can reduce the server's resource overhead for handling exceptions.

Input Validation: Similar to Endpoint2, validate the input to ensure it's an integer and within acceptable bounds. Inform users when their input is invalid, rather than letting the application crash or become unresponsive.

Rate Limiting: If you notice frequent malicious or erroneous requests to this endpoint, consider implementing rate limiting. This would prevent a single user or bot from overwhelming the system with rapid, consecutive requests.

General Improvements:

Logging and Monitoring: Implement comprehensive logging and monitoring to detect issues early. This will provide insights into which endpoints are most accessed, error frequencies, and more.

Scalability: If the application is expected to handle a growing number of requests, consider making it scalable. Deploy it in a containerized environment, like Kubernetes, which can automatically scale instances based on the load.

Documentation: Ensure that all endpoints are well-documented. Clear documentation helps consumers of the API understand expected inputs and outputs, reducing the chance of erroneous requests.

These suggestions, derived from our hypothetical test results, aim to improve the application's robustness, efficiency, and user experience. Actual improvements should be based on specific issues identified during actual testing and feedback from users.