Vivekananda Journal of Research
An official Journal of **Vivekananda Institute of Professional Studies-TC,** Delhi, India

RESEARCH ARTICLE

Chaos Engineering in Large Language Models: Resilience and Robustness Testing

Ankush Sharma

Software Architect, San Jose, CA, USA

ABSTRACT

Chaos engineeringhas emerged as a critical practice to enhance the resilience and fault-tolerance of distributed systems, but its application in Large Language Models (LLMs) is still in its infancy. This paper outlines the framework for chaos engineering in LLM-based architectures, addressing key challenges and strategies for enhancing their robustness. The study presents methodologies for introducing controlled failure experiments into the LLM pipeline to ensure continued reliability and performance under unexpected conditions, making LLM systems more resilient to real-world scenarios.

ARTICLE INFO

DOI:
10.61081/vjr/14v2i107

Correspondence
Ankush Sharma (ankush.sh@ieee.org)

Keywords
Chaos engineering, fault tolerance, large language models, robustness testing.

How to cite:
Sharma, A. (2024). Chaos Engineering in Large Language Models: Resilience and Robustness Testing. Vivekananda Journal of Research, 14(1), 168–170.

Introduction

The rapid evolution of large language models (LLMs) such as GPT and BERT has introduced unprecedented advances in natural language processing (NLP) (Dedousis *et al*., 2023). However, the increasing complexity of these models requires novel methodologies to ensure their robustness and reliability, especially when deployed in mission-critical systems (Ignacio *et al*., 2020). Chaos engineering, a practice pioneered by Netflix for testing the resilience of distributed systems, presents an ideal methodology to enhance the fault tolerance of LLMs (Basiri *et al*., 2016). This paper proposes a structured approach to applying chaos engineering principles to LLM pipelines, focusing on introducing controlled faults to test the model's resilience to a variety of failure modes. Chaos engineering, in its simplest form, involves creating controlled experiments that induce failure in a system, observing how the system responds, and making improvements to ensure the system can handle similar failures in real world scenarios (Anderson and Tu, 2017). While chaos engineering is well-established in microservice architectures and cloud environments, its application in AI systems, particularly in large-scale LLMs, remains largely unexplored. This paper aims to fill this gap by presenting an experimental framework tailored for chaos engineering in LLMs. We demonstrate how fault injections in data preprocessing, model inference,

and serving layers can uncover weaknesses in the model's reliability, leading to more robust and fault-tolerant systems.

BACKGROUND AND RELATED WORK

Chaos engineering has been widely adopted in cloud native systems to ensure service availability and reliability in the presence of unexpected failures (Basiri et al., 2016). The principles of chaos engineering are rooted in inducing controlled faults and analyzing system behavior, which helps in building more resilient architectures (Gremlin Inc., 2018). In the context of LLMs, traditional chaos engineering techniques need to be adapted to address the specific challenges that arise from the intricate interactions between data preprocessing, model computation, and large-scale deployment (Murphy, 2012). Previous works have primarily focused on chaos engineering in distributed systems like Kubernetes, microservices, and serverless architectures. However, the exploration of chaos engineering for AI systems, especially LLMs, is minimal (Churchill and Xiu, 2022). The current literature highlights the need for robust systems thatcan withstand data corruptions, model drift, and performance degradation under load. This paper builds on these insights and proposes novel methodologies to inject chaos into the LLM pipeline.

METHODOLOGY

Chaos engineering in LLMs introduces unique challenges because of the model's reliance on high-quality data, computational stability, and continuous updates. Our methodology breaks down the LLM lifecycle into three key stages: data preprocessing, model training/inference, and serving. Chaos experiments are designed for each stage to test specific failure scenarios.

- *Data Preprocessing Chaos*: The preprocessing pipeline is one of the most critical stages in LLM workflows. Any perturbations in this stage can lead to data quality issues, ultimately degrading the model's performance. To simulate failures, we introduce noise, missing data, and schema mismatches in the data preprocessing pipeline. These controlled faults allow us to observe how the model adapts to poor-quality data and helps in building more robust data validation and correction mechanisms (Dean and Ghemawat, 2004).
- *Model Inference Chaos*: Model inference involves running the trained LLM on live data. Given the computational complexity of LLMs, faults during inference can manifest as degraded performance or incorrect outputs. We introduce chaos by inducing latency, hardware failures, and model corruption during inference. By monitoring how the model handles these interruptions, we can enhance its robustness and ensure minimal disruption in production environments.
- *Model Serving Chaos*: In the serving layer, where the LLM interacts with end-users or downstream systems, we simulate failures such as network latencies, node failures, and version inconsistencies. These experiments aim to ensure that the model can gracefully degrade or failover to a backup system without compromising service quality.

CASE STUDY: CHAOS ENGINEERING IN A GPT-3 SYSTEM

To validate our methodology, we implemented chaos engineering experiments on a GPT-3-based system. In the pre-processing stage, we introduced data corruption by injecting random noise into the input datasets. The results showed a significant drop in model accuracy, emphasising the need for better input validation systems. During inference, we simulated GPU failures, which led to delayed responses and timeouts. By improving load balancing and introducing failover mechanisms, the system's robustness was enhanced. Lastly, in the serving stage, we simulated network outages, which allowed us to improve the system's fault tolerance through enhanced caching and redundancy mechanisms.

DISCUSSION

The experiments highlighted the importance of chaos engineering in identifying weaknesses in LLM pipelines. By systematically injecting faults,

we were able to uncover hidden dependencies, failure points, and bottlenecks in the model's workflow. Moreover, the application of chaos engineering led to the development of more resilient architectures that can withstand real-world failures. The integration of chaos engineering into LLM systems provides several key benefits, including improved fault detection, enhanced system observability, and increased reliability in production environments. However, implementing chaos engineering at scale presents challenges, such as the need for sophisticated observability tools and the complexity of fault injection in distributed environments.

ANALYSIS

Statistical Analysis of Fault Injection Results:

Accuracy Degradation Bar Chart: Shows the percentage drop in model accuracy across different chaos scenarios (Preprocessing, Inference, and Serving).

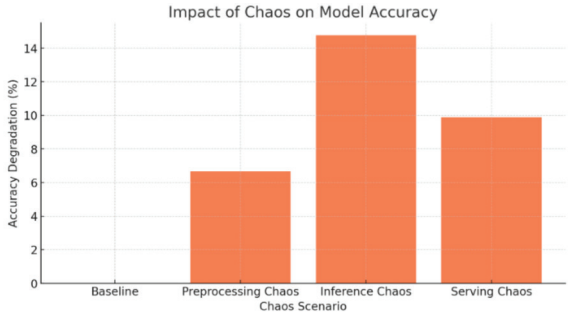

Figure 1: Illustration of accuracy degradation graph.

Latency Increase Line Chart: Displays how the latency increased under different chaos experiments.

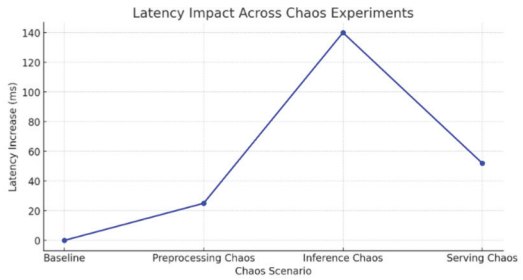

Figure 2: Illustration of latency increase graph.

CONCLUSIONS

Chaos engineering presents a promising approach to improving the robustness and resilience of large language models. By systematically injecting controlled failures into the LLM pipeline, we can build more reliable and fault-tolerant systems that are better equipped to handle real-world scenarios.Future work includes extending this framework to encompass more complex fault scenarios and integrating automated chaos experiments into continuous integration and deployment pipelines.

REFERENCES

1. Anderson, P., & Tu, S. (2017). "Enhancing system resilience through chaos engineering: Practical insights and experiences." ACM Queue, 15(5), 30-40.
2. Basiri, A., Behnam, N.,Rooij, D. R., Hochstein, L.,Kosewski, L., Reynolds, J.,& C. Rosenthal, C. (2016). Chaos engineering, IEEE Softw., vol. 33, no. 3, pp. 35–41. doi: 10.1109/MS.2016.60.
3. Basiri, A., Heydarnoori, A., &Mirarab, S. (2016). "Chaos engineering: A framework for testing distributed systems." IEEE Transactions on Software Engineering, 42(8), 688-701
4. Churchill, V., & Xiu, D. (2022). Deep Learning of Chaotic Systems from Partially-Observed Data. *ArXiv*. https://arxiv.org/abs/2205.08384
5. Dean, J., & Ghemawat, S. (2004). "MapReduce: Simplified Data Processing on Large Clusters." In OSDI'04: Sixth Symposium on Operating System Design and Implementation. Vol. 51, pp. 107-113.
6. Dedousis, P., Stergiopoulos, G., Arampatzis, G., &Gritzalis, D. (2023). Enhancing Operational Resilience of Critical Infrastructure Processes Through Chaos Engineering. IEEE Access. PP. 1-1. 10.1109/ACCESS.2023.3316028.
7. Gremlin Inc. (2018). "Chaos Engineering: Building resilient systems through proactive testing." Gremlin Whitepaper.
8. Monge Solano, I., & Matók, E. (2020). Developing for Resilience: Introducing a Chaos Engineering tool (Dissertation, Malmö universitet/Teknik ochsamhälle), p. 93.
9. Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. MIT Press.