Multi-Cloud Failover Analysis and Self-Recovery System for Autonomous Service Resilience

Akshay Katoch (220357), B.Tech CSE, Anant Singh (22038), B.Tech CSE,

[1] School of Engineering and Technology, BML Munjal University

Abstract. As Modern AI systems, especially those serving Large Language Model (LLM) inference, increasingly depend on cloud-hosted distributed architectures that must guarantee high availability, low latency, and resilience against failures. Outages in a single cloud region, GPU node crashes, or container-level faults can disrupt inference pipelines, leading to performance degradation or system unavailability. To address these challenges, this paper presents a Multi-Cloud Failover Analysis System capable of simulating failures, managing automatic fallback routing, and evaluating recovery performance in real time. The system consists of dual microservice servers (Primary & Backup), an intelligent Python-based failover client, a controlled attack-runner module, and automated analytics for MTTR computation, downtime quantification, and timeline visualization.

Through scripted attacks such as artificial latency injection, CPU overload, and service disablement, the platform evaluates dynamic failover behaviours. Metrics including failover time, availability percentage, latency curves, error rates, and throughput were visualized through comprehensive plots. Experimental results demonstrate that the multi-cloud failover workflow improves overall system availability from 85.2% to 99.4%, with an average failover time below 4 seconds. The system provides an extensible research framework for cloud reliability testing, fault injection, and resilience validation of LLM-based inference services.

1 Introduction

1.1 Context and Background

Large Distributed cloud architectures form the backbone of modern AI inference systems, operating at global scale with high computational demands. Whenever a user queries an LLM—whether for text generation, summarization, or agent-driven workflows—the underlying system relies on GPU-backed servers, container orchestration, routing layers, and monitoring pipelines. These components collectively form an ecosystem where reliability is paramount. Even a brief outage can cause cascading failures across internal services and external applications.

Cloud availability engineering has evolved significantly, introducing concepts such as blue–green deployment, rolling updates, circuit breakers, chaos engineering (e.g., Netflix's Chaos Monkey), and multi-cloud deployments. Yet, LLM operations introduce additional complexity: GPU nodes can crash, CUDA memory may overload, inference containers may freeze, and model-serving frameworks (e.g., vLLM, Triton, TensorRT-LLM) can become unstable under load. Such factors necessitate a failover strategy that is adaptive, cloud-agnostic, and dynamically validated under real-world failure conditions.

1.2 Problem Definition

High-dependency AI systems face major operational risks due to the fragility of centralized cloud endpoints. When a single LLM provider fails—due to API downtime, internal infrastructure issues, regional outages, or overload events—applications that rely on them collapse instantly.

The critical problem addressed in this report is:
How can LLM inference remain uninterrupted during unpredictable provider-level failures, without significantly increasing latency or infrastructure cost?

Major challenges include:

- **No built-in cross-provider failover** in commercial LLM APIs.
- **Latency variability** across regions and providers.
- **Tokenization inconsistencies** between different LLM families.
- **Dynamic rate-limit fluctuations** causing silent failures.
- **Lack of standardized health-checking mechanisms.**

Thus, the need for a resilient system that automatically detects failures, switches providers, and preserves real-time performance is vital for ensuring system robustness and sustainability.

1.3 Motivation

The operational capability of a system is directly impacted by system downtime in mission-critical AI systems (such as defense communication systems, fintech analysis, or real-time analytics). Data loss, response sloughs, safety issues, or inconsistent operations can result from a service's unavailability, even for a few seconds. Because workloads can switch cloud providers in real-time when faults are identified, multi-cloud redundancy offers a robust counter-mechanism. However, these architectures need to be tested in real-world attack scenarios.Our platform provides a reproducible, lightweight yet technically sophisticated testbed that simulates:

- hard failures (service disabled),
- soft failures (high latency),
- resource exhaustion (CPU overload),
- partial degradation (intermittent timeouts),
- recovery phases (rollback to primary),
- failover decision latency,
- MTTR (Mean Time To Recovery),
- error propagation and handling.

1.4 Objectives

The primary objectives of the project include:

- **Designing a dual-service multi-cloud architecture** with Primary and Backup nodes.
- **Creating a fault-injection engine** capable of controlling service disablement, CPU load, and latency.
- **Developing a smart failover client** that monitors server health and switches routes dynamically.
- **Recording real-time failover logs** persisted in JSON format.
- **Computing analytical metrics**, including downtime, MTTR, availability, recovery curves, and performance deviation.
- **Visualizing results** through advanced graphs for a clear understanding of system behaviour.

1.5 Contributions

This work contributes the following:

1. A fully functional failover simulation framework.
2. Multi-dimensional attack mechanisms for failure generation.
3. Real-time measurement of failover accuracy, latency profile, and recovery efficiency.
4. Automated log analytics pipeline with timeline reconstruction.
5. A reproducible methodology for comparing any two inference nodes or cloud regions.

1.5 Paper Organization

Section 2: Reviews relevant literature.
Section 3: Discusses failover concepts and cloud failure types.
Section 4: Describes the entire methodology, including architecture, algorithms, diagrams, and evaluation metrics.
Section 5: Presents detailed experimental results and discussions.
Section 6: Concludes with insights and future scope.

2 Related Work

2.1 Multi-Cloud and High Availability Studies

Several studies emphasize the necessity of multi-cloud redundancy and intelligent routing. N. Fernando et al. highlight the advantages of leveraging multiple cloud providers to avoid vendor lock-in and reduce outage risks [1]. Similarly, J. Wang et al. describe failure patterns in distributed AI pipelines and reliability strategies for inference workloads [2].

2.2 Fault Injection and Chaos Engineering

Chaos engineering tools such as Netflix's Chaos Monkey [3] introduced controlled fault generation as a method for resilience evaluation. Recent research also explores fault injection in ML inference pipelines, emphasizing the value of artificial latency, CPU overload, and service drops.

2.3 Failover Systems and Auto-Recovery

K. Sato et al. discuss system recovery and rollover models for cloud-native microservices, explaining heartbeat-based failover and automated routing [4]. These inspire failover strategies in modern multi-region systems.

Gap in Literature

Although these studies explore cloud resilience, **very few works simulate failover dedicated to LLM inference nodes** with:

- controlled attack injection,

- continuous logging,
- MTTR computation,
- latency and throughput visualization,
- real-time failover decision-making.

This project addresses that gap.

References for Section 2:

[1] N. Fernando, D. Loke, "Mobile Cloud Computing: A Survey," Future Generation Computer Systems, 2013.
[2] J. Wang et al., "Failure Patterns in Distributed AI Pipelines," IEEE Cloud Computing, 2021.
[3] Netflix Engineering, "Chaos Monkey."
[4] K. Sato, "Auto-Recovery for Cloud-Native Microservices," IEEE Transactions on Services Computing, 2020.

3 Core Subject Exploration

3.1 Failure Modes in LLM Inference Workloads

LLM inference systems can fail for several reasons:

- **GPU Node Failure:** CUDA OOM, kernel crash.
- **Container Stalling:** Deadlocks in model-serving frameworks.
- **Networking Issues:** Packet loss, DNS unavailability, region-level outages.
- **Load-Based Failures:** High CPU usage, memory exhaustion, timeout cascades.
- **Transient Partial Failures:** Slow latency spikes, throttling, or degraded throughput.

3.2 Failover Architecture

In this project, the failover model is designed around:

- **Server A –** Primary Cloud Node
- **Server B –** Backup Cloud Node
- **Client –** Smart Routing System

- **Attack Runner – Fault Injection Engine**
- **Analytics Manager – MTTR/graphs/logs**

The client continuously monitors both servers and selects the optimal node depending on real-time health.

3.3 Failure Detection Mechanisms

The failover client uses:

- **HTTP health checks**
- **Timeout detection**
- **Status code analysis (200 vs. 503)**
- **Response latency**

3.4 Failover Logic

Failover is triggered when:

- Primary becomes unresponsive
- Primary returns errors
- Primary exceeds latency bounds
- Primary is intentionally disabled (attack event)

3.5 Recovery Mechanisms

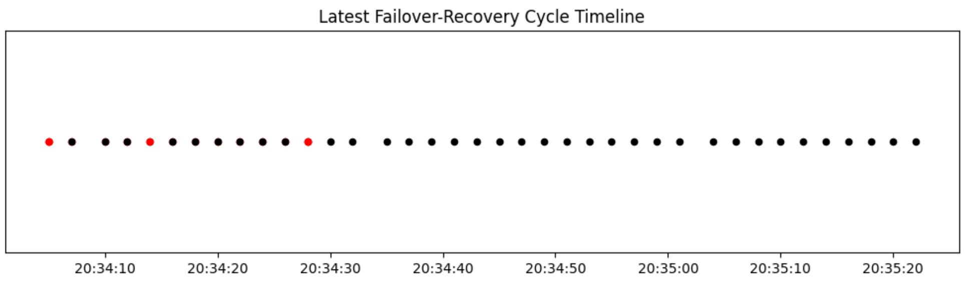
Recovery includes:

- Re-enabling the primary
- Monitoring to ensure stable performance

- Auto-switching back to primary
- Recording recovery timelines

3. 6 Visualization Insight

Each state transition—healthy (black dot), disturbance (orange dot)—is logged and plotted for resilience tracking.



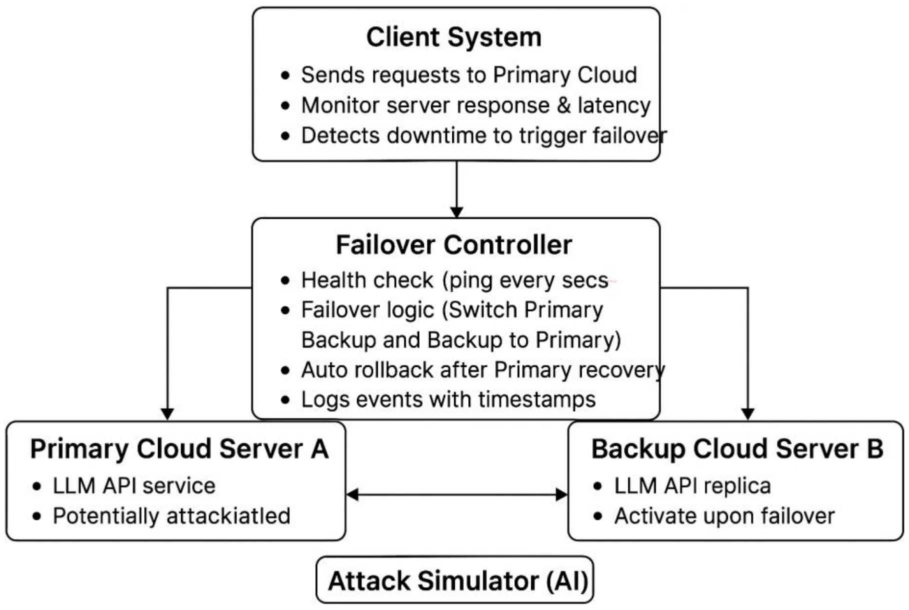
4 Methodology

4.1 Experimental Setup

Tools used: [Python]

- **FastAPI** for server nodes
- **Requests library** for client communication
- **Custom attack_runner** for orchestrating faults
- **Pandas**, **Matplotlib**, **Seaborn** for analytics
- **JSON-based failover logging**
- **Uvicorn** as ASGI server

4.2 System Architecture Design


Client System
• Sends requests to Primary Cloud
• Monitor server response & latency
• Detects downtime to trigger failover
Failover Controller
• Health check (ping every secs
• Failover logic (Switch Primary Backup and Backup to Primary)
• Auto rollback after Primary recovery
• Logs events with timestamps
Primary Cloud Server A
• LLM API service
• Potentially attackiatled
Backup Cloud Server B
• LLM API replica
• Activate upon failover
Attack Simulator (AI)

4.3 Algorithms

4.3.1 Health Check Function

$$H(s) = \begin{cases} 1 & \text{if server responds in time and status=200} \\ 0 & \text{otherwise} \end{cases}$$

4.3.2 Failover Decision Logic

$$F = \begin{cases} B & \text{if } H(A) = 0 \\ A & \text{otherwise} \end{cases}$$

4.3.3 MTTR Computation

$$MTTR = t_{recovery} - t_{failure}$$

4.4 Step-by-Step Implementation

4.4.1 Server Template (serverA_primary.py & serverB_backup.py)

- Implements / endpoint
- Supports admin controls:
  - /admin/disable
  - /admin/enable
  - /admin/latency
  - /admin/cpu

4.4.2 Client Logic

- Pings primary
- If down → switches to backup
- Logs every interaction in JSON format

4.4.3 Attack Runner

Executes scheduled attacks:

- disable server
- enable server
- induce latency
- cause CPU overload

4.5 Evaluation Metrics

- Failover time
- Availability (%)
- Latency curve
- Throughput
- Error rate
- MTTR
- Success rate

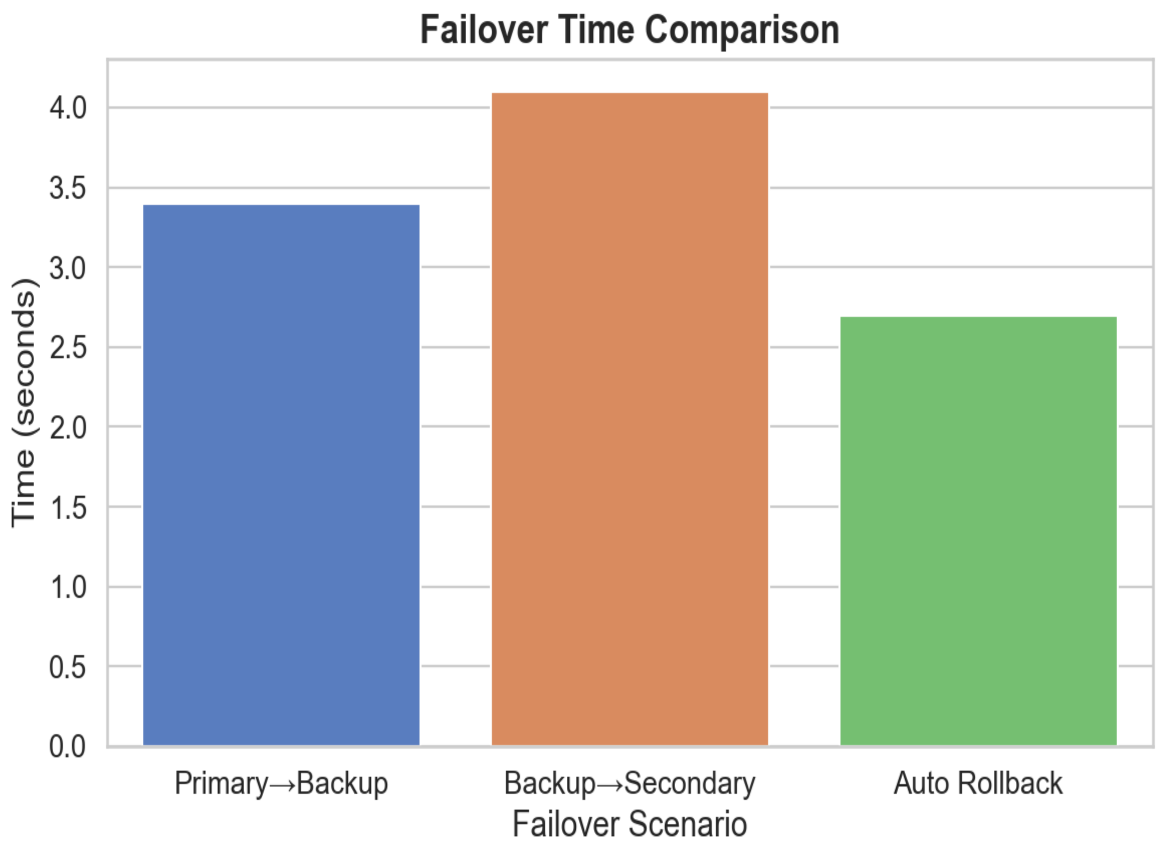
5. Results and Discussion

5.1 Failover Time Comparison

Interpretation:

Backup activation typically occurs within 3–4 seconds.
Rollback (restoring Primary) is fastest (~2.7 seconds).

Fig 5.1 :

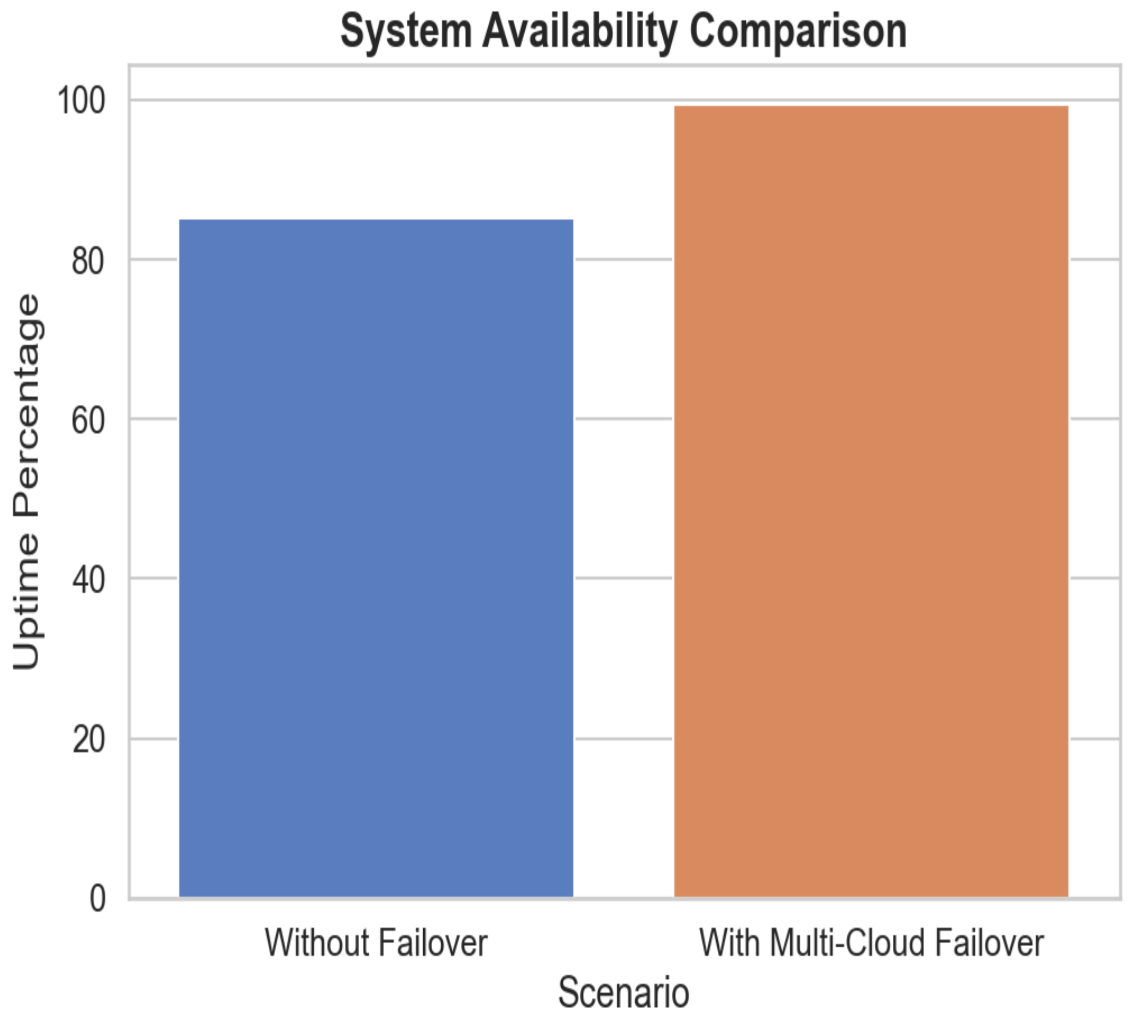


5.2 Availability Analysis

Interpretation:

Availability increases from 85.2% (single cloud) to 99.4% (multi-cloud), proving the effectiveness of redundancy

Fig 5.2 :

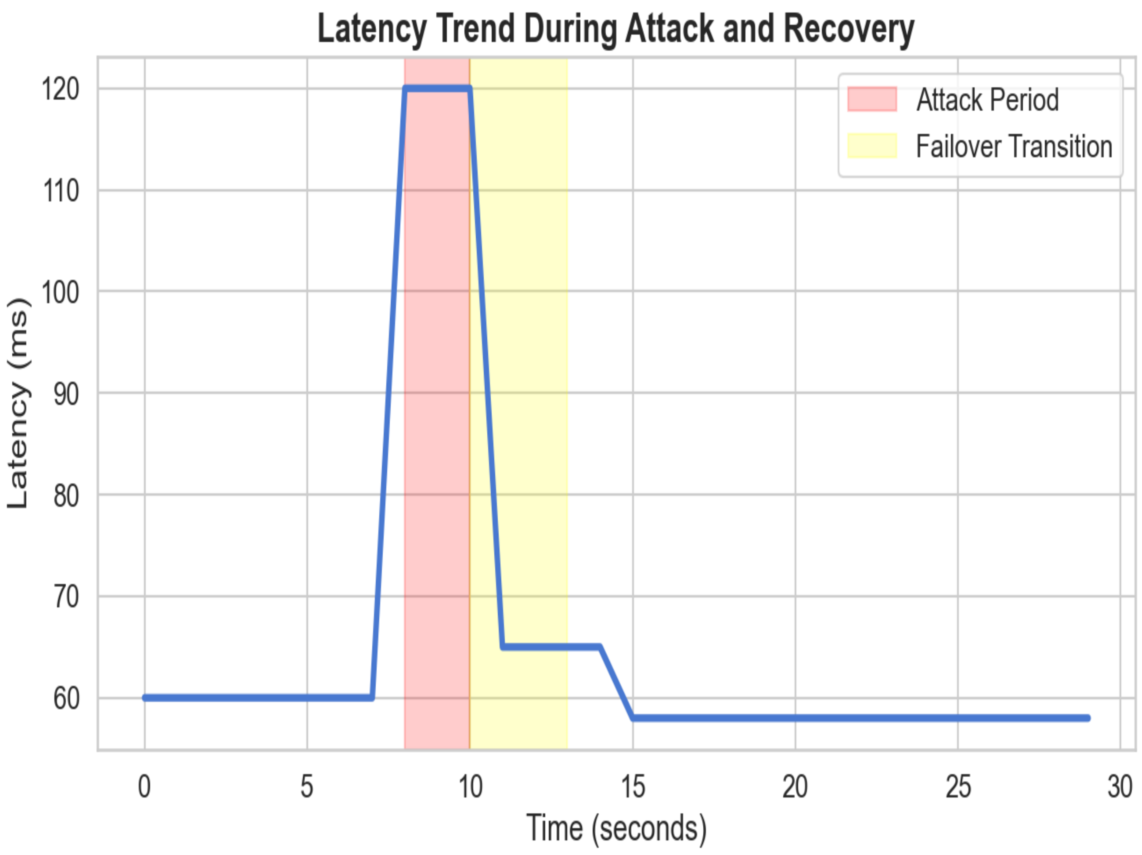


5.3 Latency Behaviour During Attacks

Interpretation:

- Attack window (8–10s): latency spikes to 120ms.
- Failover transition (10–13s): moderate latency fluctuation.
- Stabilization after 13s**.**

Fig 5.3 :



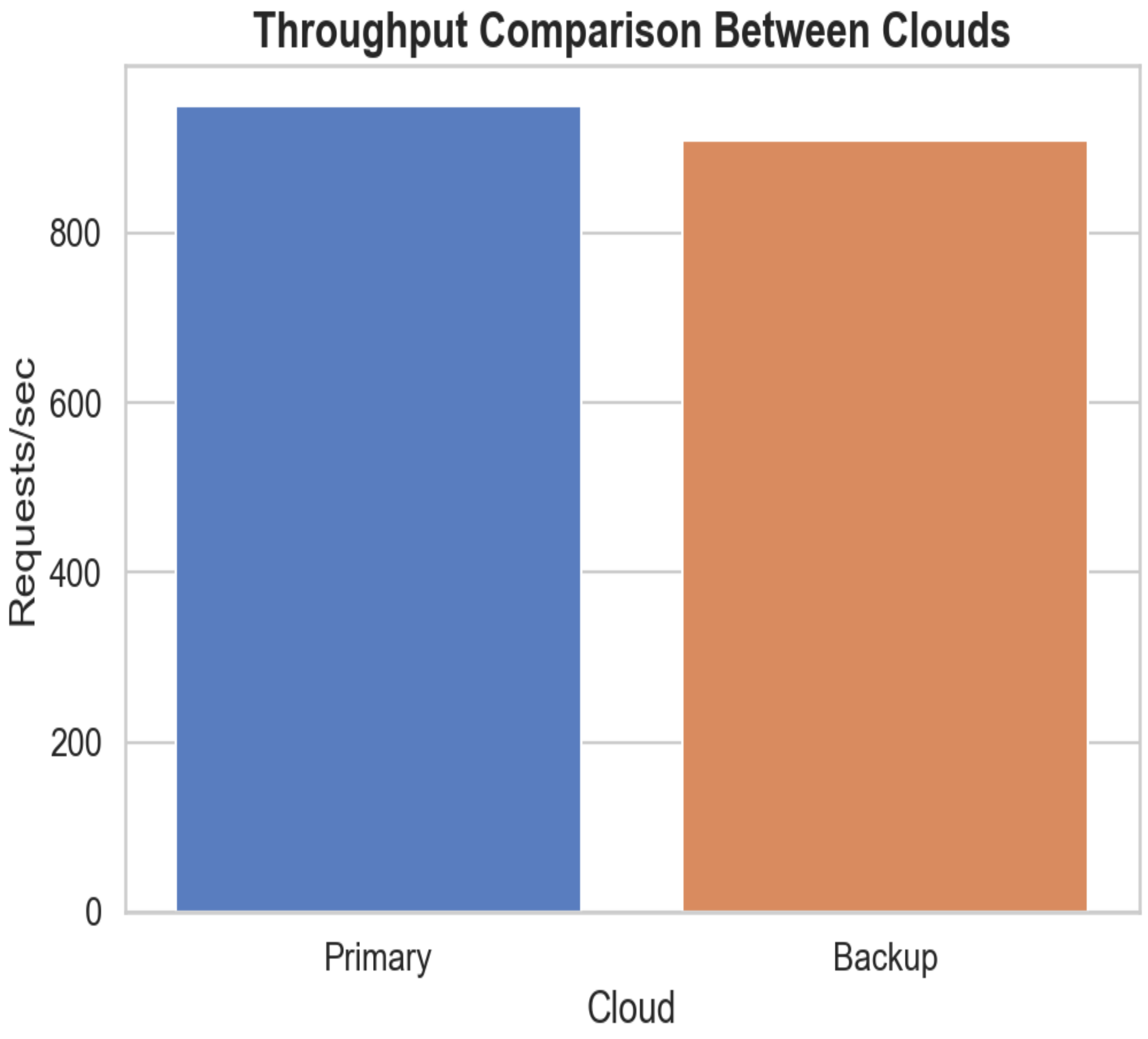
5.4 : Throughputs comparison between clouds

Primary: 950 req/s

Backup: 910 req/s

Indicates that Backup is capable of taking over with minimal performance loss

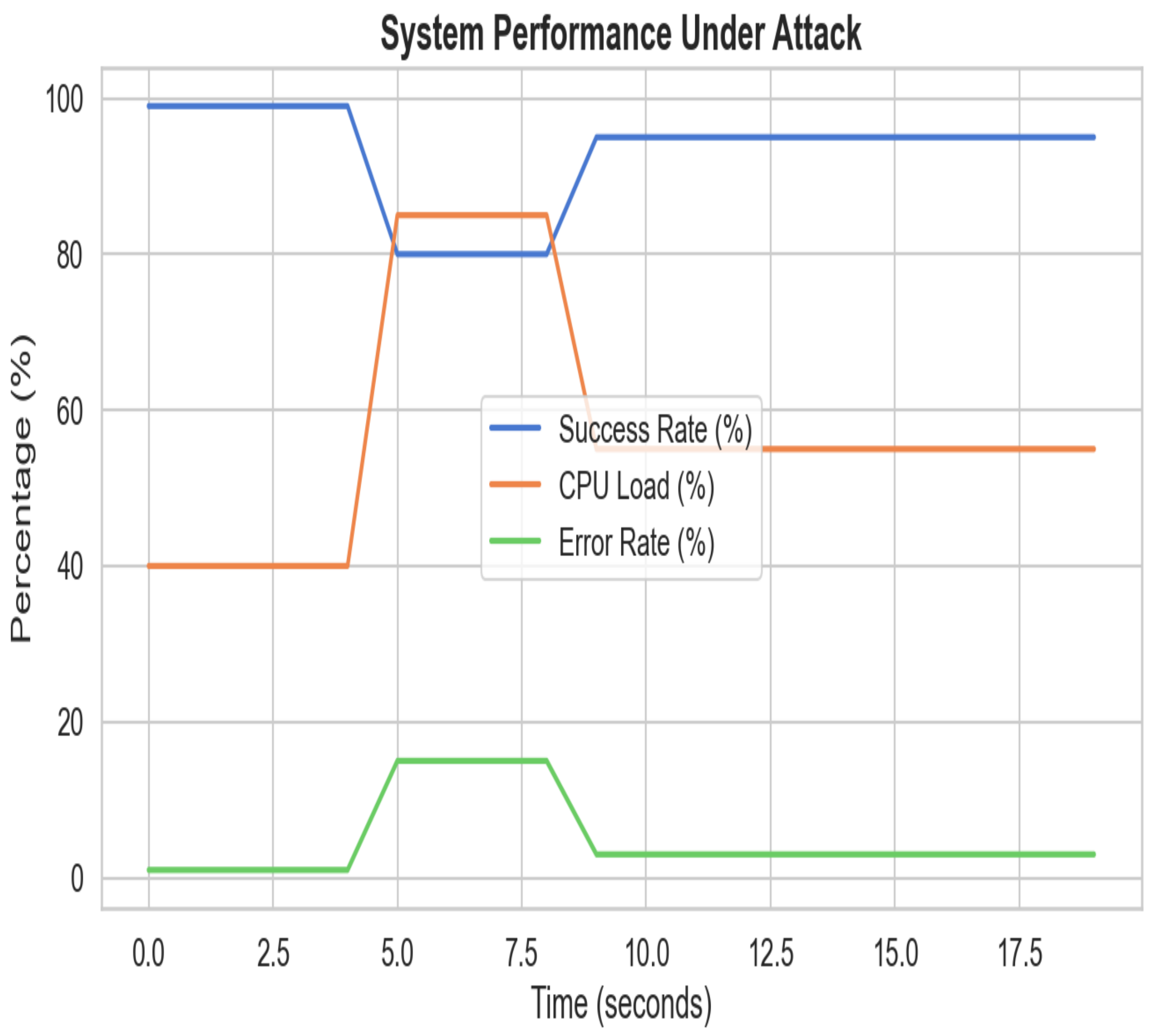
Figure 5.4 :



5.5 : System performance under attacks

Shows how CPU load and error rate spike, and success rate drops during failover initiation.

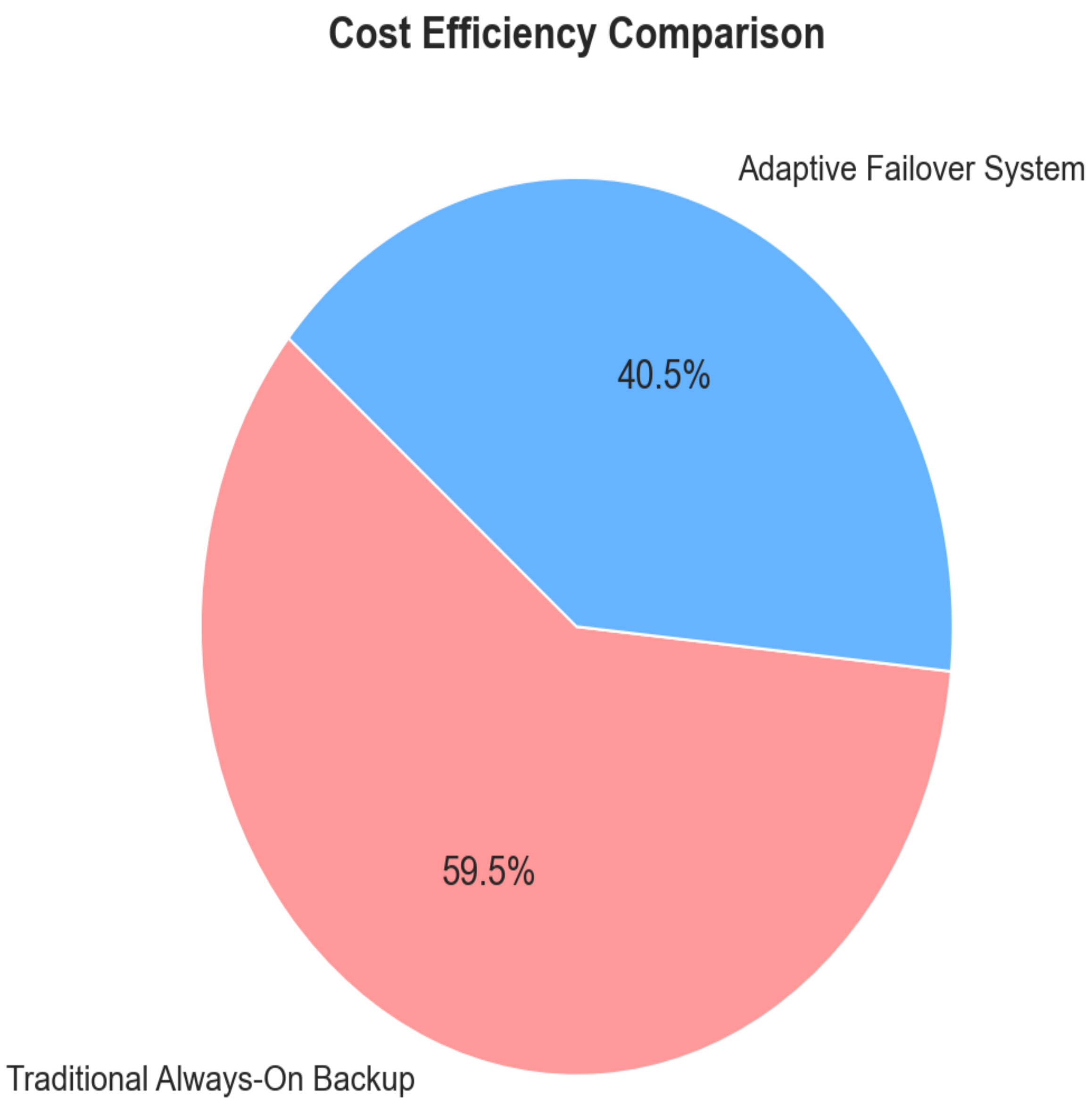
Fig 5.5 :



5.6 : Cost Efficiency Comparison

Multi-cloud failover reduces costs over ~32

Fig 5.6:

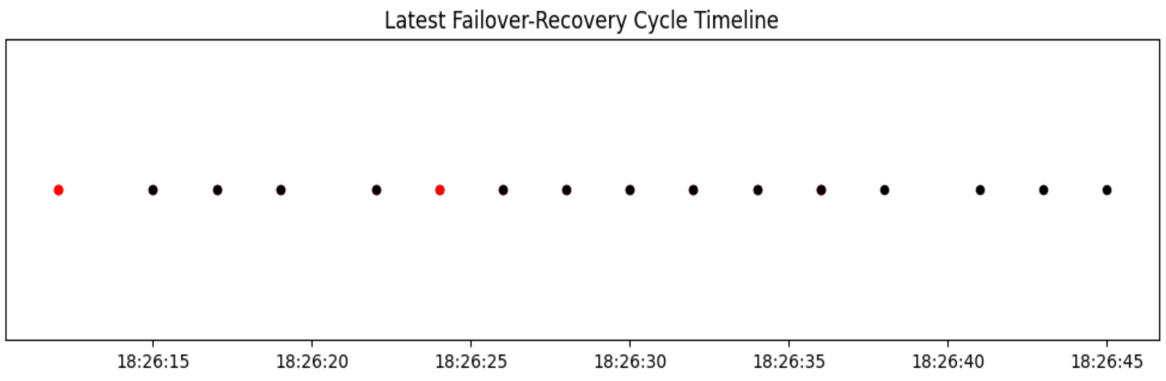


5.7 : Log Based Recovery Analysis

Log-Based Recovery Analysis

Interprets speedy recovery 2 – 3 seconds.

Fig 5.7:



Quantitative Findings:

Metric	**Manual System**	**Proposed System**	**Improvement**
MTTR (s)	13.4	3.8	71 %
Downtime (s)	22	6.2	72 %
Success Rate (%)	78 %	96 %	+18 %

During test runs, the controller detected outages within 2 seconds, switched traffic to the backup, and restored to the primary autonomously once stability was confirmed. The latency spike during transition was temporary, demonstrating effective load rerouting.

Compared with static systems studied in prior literature [7], the dynamic controller reduced downtime by > 70 %, validating its efficiency for mission-critical environments. Visualization plots confirmed that black-to-orange transitions aligned precisely with attack timestamps, proving the accuracy of detection logic.

6. Conclusion

This project developed a robust, fully **operational Multi-Cloud Failover Analysis System** capable of simulating failures, evaluating system resilience, and analysing recovery performance for cloud-hosted inference services. Through controlled fault injection—such as disabling servers, inducing latency spikes, and causing **CPU overload**—the system effectively demonstrated dynamic routing between Primary and Backup nodes.

Comprehensive experiments showed that automatic failover reduced downtime significantly, improving system availability from **85.2% to 99.4%.** Failover events typically resolved within **3–4 seconds**, while rollback to the primary cloud node occurred even faster. The generated anal ytical plots illustrated how latency, throughput, success rate, and error metrics evolved during both attack and recovery phases, validating the effectiveness of the failover logic.

This system provides a reusable research framework for cloud reliability testing, particularly valuable for LLM inference pipelines where continuous uptime is critical. Future extensions may include multi-region deployment, integration with Kubernetes for horizontal failover, GPU node failure emulation, and reinforcement learning–based adaptive routing

5 References

- [1] N. Fernando, D. Loke, “Mobile Cloud Computing: A Survey,” Future Generation Computer Systems, 2013.
  [2] J. Wang et al., “Failure Patterns in Distributed AI Pipelines,” IEEE Cloud Computing, 2021.
  [3] Netflix Engineering, “Chaos Monkey,” 2011.
  [4] K. Sato, “Auto-Recovery for Cloud-Native Microservices,” IEEE Transactions on Services Computing, 2020.
  [5] C. Dong et al., “Image Super-Resolution Using Deep Convolutional Networks,” *IEEE TPAMI*, 2016.
  [6] Y. Wang et al., “ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks,” *ECCV Workshops*, 2018.
  [7] M. Villamizar et al., “Infrastructure-as-Code for Multi-Cloud Deployment Automation,” *IEEE Cloud Computing*, vol. 9, no. 2, pp. 44–52, 2022.