

Proactive Elastic Circuit Deconstructor: A Machine Learning-Driven Resiliency Pattern For Microservices

Norton Stanley S A^{1*} and Shoney Sebastian^{1†}

^{1*}Department of Computer Science, CHRIST(Deemed to be University), Hosur Road, Bangalore, 560029, Karnataka, India.

*Corresponding author(s). E-mail(s): norton.sa@res.christuniversity.in; Contributing authors: shoney.sebastian@christuniversity.in; †These authors contributed equally to this work.

Abstract

Adopting cloud-based architecture is essential for scalability and dependability in the changing and highly regulated application environment of today, especially through microservice architecture. Usually providing reactive ways to address issues post-occurrence, traditional resilience patterns that are implemented using libraries like Hystrix, Resilience4j based on error thresholds or timeouts in microservices could cause downtime and performance impacts. Similarly, service meshes like Istio and Linkerd offer L7-aware traffic control at the proxy level, but remain unaware of service functionality or domain-specific failure modes. This work presents a proactive method by including predictive machine learning models into microservice resilience patterns, with particular attention on the Circuit Breaker pattern to improve fault tolerance. This new method forecasts microservices' abnormalities ahead of time and then switches over to another service with less failure probability. Based on the historic logs we could predict an alternative pre-configured service with least probability of failure and great accuracy using supervised learning methods including Random Forest and Randomised SearchCV. Our tests showed that the predictive models could efficiently forecast and switch to an alternative service, therefore proactively reducing any downtime. Without large system overhauls, the proposed pattern easily connects with current monitoring systems to increase resilience. This work proposes a proactive resilience design pattern for microservices that forecasts service dependability in real-time, ranks services by failure likelihood, and proactively reroutes requests to improve operational reliability, therefore providing robust services with minimal downtime.

Keywords: Cloud Computing, Microservices, Cloud Native, Design Patterns, Circuit Breaker, Elastic Circuit Deconstructor, Proactive Resiliency, Zero Downtime, Kubernetes.

1 Introduction

The inception of cloud computing has modernized the way the applications are scaled, developed and deployed. Cloud-based architecture [1] has emerged as a cornerstone of modern application ecosystems with key considerations being latency, scalability and reliability. Its flexibility to scale up and down resources, aids in load balancing while also maintaining low latency and high throughput. The applications are typically developed using either a "monolithic" or "microservice" architecture. When addressing microservice architecture, it is crucial to understand monolithic architecture first. To comprehend this monolithic architecture, as implied by the name, "mono" describes a single, cohesive environment for developing software applications that are autonomous in nature. It has a few benefits, including improved performance, simplicity, ease of implementation, and reduced overhead costs. Though simple and efficient, it poses significant challenges when the application scales [2]. In this type of architecture the components are significantly interdependent on each other, and it often struggles with continuous integration and continuous development(CI/CD) ie., any minor changes in the application leads to redeployment of the whole application, simply put the components are tightly coupled [3]. There are also significant limitations with this, being issues with scaling the resources, addressing the failures and unreliable.

The intricacy of large applications is beyond the capabilities of this design. It is not possible to be flexible with modifications. To address these constraints we have microservice architecture that provides multiple benefits. Microservice architecture makes it possible to create scalable and dependable applications. One of the major focuses is on synchronous communication among microservices [4], where services wait for the response before moving forward. There is a comprehensive list of design patterns that lays the foundation to ensure the services aids in reducing the risks by implementation of loosely coupled services [5]. One of the primary benefits of utilizing microservice architecture is its resilience nature. This is ensured in the case of various resiliency patterns particularly circuit breaker patterns [6].

The intersection of cloud computing and machine learning is bringing transformative capabilities into modern applications [7]. Machine learning provides greater fault tolerance and operational efficiency since it allows an application to learn from historical data, thereby modifying its behavior over time [8]. Through failure patterns and service utilization metrics, it is possible for machine learning models to forecast disruption potential. For instance, the performance and reliability trends of microservices over their lifetime can be analyzed by ML algorithms for determining anomalies which might predict failure [9]. Such predictive capabilities will make the application divert requests to the most reliable service in real time thus ensuring a low downtime and continuous availability of service. Moreover, decision-making capabilities through ML introduce dynamic resource allocation based on real-time traffic patterns and load conditions hence optimal performance and reduced operational costs.

In our previous paper we introduced "The Elastic De-Constructor Design pattern" which presents a fault-tolerant resiliency pattern designed over the circuit breaker pattern to improve system resilience through efficient runtime failure management. In contrast to conventional circuit breaker states like "Open," "Closed," or "HalfOpen," the Elastic De-Constructor presents a novel state called "De-constructed" [10]. The system automatically switches to this mode when it detects a failure, choosing a different service from a pre-configured list to provide uninterrupted service. With this strategy, cascading failures are less likely to occur and service is maintained even when there are disruptions.

In this paper drawing inspiration from the domain of data storage systems, multipathing techniques which has proven to improve system resilience by providing multiple pathways for data access, by ensuring high availability and fault tolerance [11] [12] [13], this study extends the principle of multipathing into the microservices architecture. This paper argues that there is a need for an architectural abstraction that bridges predictive intelligence and proactive resiliency at service level communication. More specifically, there is a need for a reusable design pattern that integrates ML-based failure prediction with circuit breaker semantics to proactively maintain functional continuity rather than reactive to failures after they occur.

To address this gap, we propose the Proactive Elastic Circuit Deconstructor (PECD) a behavioral resiliency pattern, which is an extension of the classical Circuit Breaker pattern that incorporates machine learning-driven anomaly prediction and proactive multipath request routing. PECD introduces a novel De-constructed state that enables the system to temporarily decompose the primary execution path and redirect requests to alternative functional services when an elevated risk of failure is predicted. Unlike traditional CBs, PECD aims to prevent functional downtime by acting before failures are externally observable.

This is done by integrating machine learning models with the ECD resiliency pattern. The pattern proactively mitigates failures by merging with ML which significantly enhances fault tolerance. It is carried out by analyzing the failure histories of microservices, which will eventually analyze the potential failures and proactively suggest an best alternate

service rather than being reactive. This approach helps in rerouting the request to the best services. The emergence of ML [14] has laid an equally enhanced new possibility for seamless integration with cloud based architecture. This combination enables applications to respond in real time and prevent failures by offering the best alternative one. In order to ensure reliability and optimize performance, algorithms and historical data are employed. As these settings fundamentally alter over time, machine learning (ML)-driven service prediction will become an indispensable tool for maintaining competitive advantage in the service-driven digital world.

1.1 Our Contributions

This study presents important changes to the design and implementation of microservice architectures, improving their resilience and operational efficiency through predictive machine learning. Here are the main contributions of our paper:

1.1.1 Innovative Use of Predictive Machine Learning Models for Proactive Resilience

Similar to how storage multipathing dynamically selects the optimal path for data transfer to avoid congestion or failure, our Proactive Resiliency Pattern dynamically reroutes service requests based on predictive analysis into service health. This methodology ensures that potential points of failure are preemptively avoided, leveraging the robustness characteristic of storage multipathing but innovatively applied to microservice management. We introduced the integration of machine learning models into microservice resilience patterns, which is a new technique in this field. The models use historical and realtime operational data to predict future service outages before they occur. This proactive approach is a considerable improvement over the traditional reactive models, which address issues after they occur, our approach reduces possible downtime and increases the service reliability. This proactive service reliability is a must for those businesses with continuous reliance on the services, such as e-commerce, finance, and healthcare.

1.1.2 Dynamic Service Rerouting Based on Predictive Insights

Another significant contribution is the creation of a dynamic service rerouting system that employs predictive analytics to redirect requests away from potentially failing services and towards more dependable alternatives. This approach not only increases overall reliability, but it also provides continuous service availability, which is vital for preserving user experience in critical applications.

1.1.3 Strategy for implementing the proactive pattern

The strategy applied makes use of machine learning to anticipate potential outages and suggest the most suitable alternative service in case of potential breakdowns; hence ensuring continuous proper functioning of the system rather than waiting for failure to occur and react. This is performed by analyzing key metrics such as service response times, error rates, and resources consumption; machine learning models can establish early signs indicating service degradation. The dataset that exhibit different service behaviors and performance in various situations are utilized to train these models. In such a case then, it would be able to foresee which services may experience failure alongside indicating at each point an alternative one that would tend to be more dependable. When an outage or decline in service happens to be recognized, this system does not just change over to any other random substitute instead it selects the most optimal alternate path according to the state of all existing services. The choice taken is influenced continuously by machine learning program that consider several aspects like:

- Response Latency
- Temporal usage trends
- System error codes
- Processor utilization
- Memory consumption

This way the proposed strategy stays ahead of potential faults hence improving reliability and user experience by precisely predicting suitable alternative services before any impending fault occurs.

1.1.4 Evaluation of the proposed pattern

We assess the effectiveness of our suggested proactive resiliency pattern from a mixture of logs that was generated from a system we designed and implemented in a distributed environment that mirrors users requests and a substantial dataset that was generated from a Java based application designed to send notifications. This data was leveraged to integrate machine learning techniques that allows the model to continually enhance the prediction in it, hence its accuracy as well as fueling fault tolerance with time. Our experimental analysis demonstrates the efficacy of our proposed pattern. The ECD pattern proactively predicts failures in the primary path and switches over to an alternate path which has the least probability of failure even before the error occurs. This ensures that corrective measures are taken proactively rather than a reactive approach. This stands in contrast to the existing resiliency patterns, which lacks a proactive approach to ensure the availability of the service functionality. Table 1 depicts practical benefits from the infusion of machine learning into system design toward proactive fault tolerance.

1.1.5 Impact on the field

The findings of this study are expected to have a significant impact on the field of cloud computing, particularly in terms of improving the robustness and efficiency of microservices. This study advances more resilient cloud architectures by overcoming the constraints of present resilience patterns and creating a scalable, predictive method. Furthermore, the principles outlined here are applicable to other computing paradigms, such as the Internet of Things (IoT) and edge computing, where service reliability is equally important.

1.2 Research Questions

The central objective of this work is not to claim a fundamentally new failure prediction technique, but rather to formalize proactive resiliency as a reusable architectural pattern that integrates predictive analytics with service-level routing decisions. To rigorously investigate the effectiveness, feasibility, and limitations of this approach, this study is guided by the following research questions:

RQ1: Can machine learning models accurately predict impending microservice failures using operational runtime metrics and historical logs before failures are externally observable?

RQ2: Does integrating predictive failure detection into a circuit breaker-based design pattern reduce end-to-end latency and error rates compared to traditional reactive circuit breakers?

RQ3: Can proactive, prediction-driven multipath routing improve functional availability in microservice systems where alternative service implementations exist?

RQ4: What computational and runtime overhead does a machine learning-enabled proactive circuit breaker introduce, and is this overhead acceptable at microservice scale?

RQ5: Under what architectural conditions and service domains is a proactive multipath resiliency pattern such as PECD applicable and beneficial?

To answer these questions, we formally document the PECD pattern using POSA conventions, implement the pattern in a Kubernetes-based microservice environment, and evaluate its performance using a large-scale dataset generated from a distributed notification system. Multiple machine learning models are assessed, with Random Forest achieving the best predictive performance. Extensive experiments with repeated trials and statistical validation demonstrate that PECD significantly reduces latency and error rates while introducing minimal runtime overhead.

The remainder of this paper is organized as follows. Section II provides background and motivation for proactive resiliency in microservices. Section III reviews related work, including adaptive circuit breakers and reinforcement learning-based approaches. Section IV details the PECD pattern specification, the machine learning framework, methodology, and experimental evaluation. Section V discusses the results and compares PECD with state-of-the-art approaches. Section VI concludes the paper with directions for future work.

2 Background and Motivation

Microservice-based architectures break down complicated programs into small, independent services that may be deployed on their own. These services mostly talk to each other via lightweight protocols. This sort of architecture makes it easier to scale and speeds up development, but it also makes systems more vulnerable to distributed failure modes. When you deploy a lot of microservices, you often get network delays, partial service outages, resource contention, and cascading dependency failures. As systems get bigger, these kinds of failures happen more often and have a bigger effect. This makes resilience a top design priority instead of an afterthought.

A. Characteristics of Failure in Microservice Systems

In microservice environments, failures are not often binary or instantaneous. Instead, they usually show up as performance problems that get worse before they completely fail. These problems can include higher latency, timeouts that happen randomly, partial unavailability, or higher error rates. These early warning signs could last for a few seconds or minutes before crossing the line of failure. During this time, downstream services keep sending requests to the failing component, which makes latency worse and makes cascade failures more likely.

Most traditional resiliency patterns view failures as discrete events rather than evolving processes. These patterns are reactive in nature and respond only when the service starts to get worse and can be visibly experienced. This slow response time is especially bad for systems that interact with users and have strict service level goals where even small problems can have a big effect.

B. Patterns of Reactive Resiliency and Their Limits

Retries, timeouts, rate limiting, bulkheads, and circuit breakers are all examples of classical resiliency patterns that make up the basis of fault tolerance in microservices. The Circuit Breaker (CB) pattern is one of the most popular because it is easy to use and stops repetitive requests to services that are failing. CBs stop faults from spreading by switching between Closed, Open, and Half Open states according on failure levels.

Even if they work well circuit breakers have some built in problems:

- **Reactive behavior:** State changes only happen once a failure occurs.
 - **Functional downtime:** Requests are blocked or do not go through during the detection timeframe.
 - **Static configuration:** Thresholds are usually set in stone or adjusted based on experience.
 - **Instance-centric focus:** CBs separate instances that are failing, but they do not use functional alternatives.
 - **Limited contextual awareness:** Classical CBs lack predictive or probabilistic thinking.
- These restrictions necessitate the development of methods that intervene prior to the breach of failure criteria.

C. Functional Multipath in Real-World Systems

Instance-level replication is widespread, but many real-world microservice systems also have functional multipath features, which means that the same business function can be done in different ways by different services.

Some examples are:

- Sending notifications by email, text message, or messaging apps
- Processing payments through more than one payment service provider
- Federated identity providers for authentication
- Access to data through backends that are different in terms of area or provider

In these kinds of systems, if one service fails, the system can still work as long as requests can be processed through a different path. But current resiliency patterns do not take advantage of this chance ahead of time. Instead they use fallback logic that only kicks in when anything goes wrong.

D. Reasons for Predictive, Proactive Resilience

Recent progress in machine learning has shown that operational data, like latency trends, resource usage, and response code distributions, can be used to anticipate problems that are about to happen with a fair amount of accuracy. Adding these kinds of predictive features to resiliency systems makes it possible to go from reacting to failures to avoiding them. The motivation for this research is hence dual-faceted:

Metric	Traditional Resiliency Pattern	Proactive ECD with ML integration
Latency (ms)	Variable with spikes	Stable and consistent
Functionality Failure Rate	High	Negligible
Scalability	Limited	Enhanced
User Satisfaction	Low	High

1. To integrate predictive intelligence directly into service communication patterns instead of utilizing external control planes.
2. To make proactive multipath routing a reusable architectural abstraction.

3 Related Work

Microservices architecture has been a potent trend in software engineering over the past couple of years, owing to its promised scalability, flexibility, and resilience benefits.

Increasing literature has emerged in the field to address the challenges and solutions adopted in the deployment process of microservices in various domains. This section critically reviews studies about resilient microservices and their influences on system reliability.

Traditional Reactive Patterns

Resilience patterns in microservices such as the Circuit Breaker pattern is extensively discussed in the literature [8]. The circuit breaker pattern, functions by temporarily halting faulty services when an error occurs until they recover and thereby preventing system failures from seeping into other service. As discussed by A. Hlybovets et al. [15] the effectiveness and limitations of such patterns are notable in cloud-native environments. . Similarly, service meshes like Istio and Linkerd offer L7-aware traffic control at the proxy level which is based on the network, but remain unaware of service functionality or domain-specific failure modes [16].

Limitations of Reactive Approaches

While reactive resilience patterns are useful in preventing failures across the system, they react after the initial failure occurs and rather than preventing the failure from occurring which does have an impact on the user experience. Authors in [15] have discussed the limitations of these patterns, noting that they cause delay and lead to a significant downtime during peak load which can result in poor user experience.

Predictive and Proactive Approaches

Recent research has explored the integration of predictive analysis in anomaly detection of microservices that aims at forecasting potential failures in advance so that they do not impact service continuity. In [17] the authors devise an approach where they model the systems behaviour during normal operations and using traces and this model is then used to detect anomalies. D. Noetzold et al. [18] have proposed a methodology which predicts possible failures which can help developers or service owners take action before the error occurs. The proactive resiliency patterns we propose builds upon these foundational studies by not only detecting potential failures in advance but also automatically rerouting requests to the most reliable services configured and thus ensuring uninterrupted service delivery.

Integration of ML in Microservice Resiliency Patterns

One such research [19] was on improving the productivity of DevOps engineers by optimizing the debugging time through better understanding of the interactions of microservices. It was done by analyzing the log patterns and using the machine learning models to predict the interactions of microservices at certain moments. Yamnual et al. [20] addressed the failure proactively by proposing a proactive failure detection framework, integrating the Elasticsearch-Logstash-Kibana (ELK) stack for system performance monitoring, and applying machine learning techniques for failure detection, including SVM, Random Forest, and Naive Bayes. Authors in [[21] have analyzed large datasets of machine performance and environmental conditions, in order to identify failure patterns by using Machine learning techniques. The authors [22] have done an extensive study on anomaly detection in cyber-physical systems. They focused on the challenges of early fault detection and localization in smart manufacturing machines. This study compares the performance of supervised and unsupervised learning algorithms, showing that supervised learning, especially tree-based models, significantly outperforms unsupervised methods. In [23], the authors present an IPFT approach based on deep learning model-Recurrence Neural Networks to predict the usage of edge node resources.

The proposed method has better reliability and maintainability than reactive fault tolerance methods. [24] Asmawi et.al, presented a comprehensive comparison and model evaluation for predictive models for job and task failure has been carried out with Gradient Boosting being the best model in failure prediction. The authors in [25] have discussed a Proactive approach to fault tolerance in Distributed systems. The proposed methods used ML models like Random Forest, Gradient Boosting to predict hardware or network failures. This approach predicts task failures for users to take corrective action in advance. In our research the focus has been to proactively predict failures in microservices and also take corrective action to switch over to an alternative microservice and ensure zero downtime. Table 2 depicts the state of art analysis for proactive fault tolerance mechanism.

Table 2: State-of-the-Art Approaches: An Analysis

Reference	Technique	Findings
Raeiszadeh et al. (2025). [26]	Graph Neural Network (GNN), Positive and Unlabeled (PU) learning	This study proposes an Asynchronous Real-Time Federated Learning (ART-FL) approach for anomaly detection in cloudbased microservice systems by making the edge clients perform real-time learning with continuous streaming local data.
Haroon M et al. (2024). [27]	Random Forest Model	A Proactive Approach to Fault Tolerance Using Predictive Machine Learning Models in Distributed systems
Kemal Akta,s et al. (2024) [28]	Isolation Forest, Logistic Regression, SVM, Decision Tree, Random Forest, MLP	Interaction Prediction and Anomaly Detection in a Microservicesbased Telecommunication Platform
Kalaskar and Thangam (2023) [29]	Linear Gradient regression, XGBoost, Boosting, tree Decision	This study presents a proactive fault tolerance strategy for cloud-computing applications, focusing on systems by employing machine learning models on a comprehensive dataset.

Gururaj et al. (2023) [30]	KNN, SVM	This systematic survey categorizes and compares various faulttolerant solutions for distributed Network system.
Theodoros Theodoropoulos et al. (2023) [31]	RNN	Intelligent Proactive Fault Tolerance (IPFT) method to predict resource usage in edge computing environments.
Tengku Nazmi et al. (2022) [32]	Logistic Regression, Gradient Boosting, Random forest	Comparison and model evaluation for Machine learning models for job and task failure
Chakrabarty et al. (2018) [33]	Random forest, Support Vector Machine (SVM), Gradient Boosting Classifier (GBC) and k-nearest neighbour algorithm	This study focuses on analyzing and predicting the arrival delay of individual flight.
Khanasin Yamnual et al. (2017) [34]	SVM, Random Forest, Naive Bayes	A proactive failure detection method based on a monitoring system for the high performance computing systems

Table 3: Feature comparison of traditional circuit breakers, service mesh, and Proactive ECD.

Feature / Tool	Traditional CB (Hystrix, Resilience4j)	Service Mesh (Istio, Linkerd)	Proactive ECD
Layer	Application (Client Code)	Proxy (Sidecar, Envoy, L7)	Application (Client Code)
Trigger Mechanism	Reactive (thresholds)	Reactive (metrics) + External ML Integrations	Predictive (Built-in ML)
Failure Signal Source	Error rate, timeout	HTTP status, gRPC errors, latency	CPU, memory, business logic metrics
Silent Failure Detection	No	Partial (via custom integrations)	Yes

ML Integration	No	Supported via external control loops	Native + trainable
Failover Strategy	Static fallback or abort	Retry, mirror, traffic split	Dynamic rerouting with predictive context

The application of machine learning to improve the resilience of microservices is a relatively new area of research. InstantOps by Raphael Rouf et al. [35] uses microservices events (e.g. Image Pull Back Off, PVC Pending), and resource metrics in order to predict system failures and root cause analysis.

Our work extends these methodologies by integrating them into a microservice environment, enabling real-time prediction and management of service reliability. This not only addresses the gaps identified by previous studies in terms of delay and user impact but also sets a new benchmark on how machine learning can be integrated in order to achieve zero downtime using the resiliency pattern.

3.1 Research Gap

Existing research on proactive fault tolerance focuses on applying various machine learning models and algorithms to predict failures in the system. While it is crucial to use predictive methods to identify problems before they occur, they often do not suggest alternative methods to address the request for effective fault tolerance. Notably, this shows a significant gap in having integrated systems that will not only forecast failures but also suggest alternative paths to address these failures.

Our research extends beyond the traditional reactive scope and existing proactive methods, as it not only predicts failures but also recommends an alternative service from the configured set of services if a failure is anticipated. This capability to predict and respond proactively to failures in business systems significantly enhances their operational resilience. This strategy prevents downtime and improves overall performance and system stability.

4 The Proactive Elastic Circuit Deconstructor (PECD) Pattern

Design patterns provide reusable solutions to recurring problems within a given context that enables architects and practitioners to reason about complex systems at a higher level of abstraction. In the domain of microservice architectures several well established patterns such as Circuit Breakers, Bulkheads, Retries, and Rate Limiters have been widely adopted to improve system resilience. These patterns encapsulate best practices for fault isolation and failure containment however they are predominantly reactive in nature and assume that failures must first manifest before mitigation can be applied.

As discussed in the preceding sections, microservice failures often exhibit observable degradation signatures—such as increasing latency, elevated error rates, or resource saturation—prior to complete failure. At the same time, many real-world microservice systems exhibit functional multiplicity, where a given business capability can be fulfilled by multiple alternative services or execution paths. Existing resiliency patterns do not explicitly address how predictive knowledge of impending failure can be systematically incorporated into request-routing decisions to preserve functional continuity.

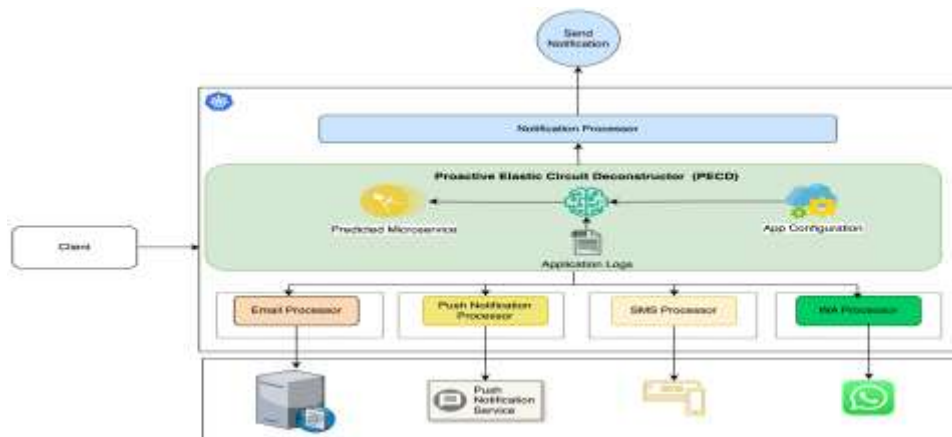
To address this gap, this paper proposes the Proactive Elastic Circuit Deconstructor (PECD) as a design pattern that extends the classical Circuit Breaker abstraction with predictive intelligence and proactive routing semantics. Rather than introducing a new

failure detection mechanism, PECD restructures the decision logic around service invocation by embedding machine learning-based failure prediction into the circuit breaker state machine. The pattern introduces a novel De-constructed state that enables temporary decomposition of the primary execution path and proactive redirection of requests to alternative functional services before failures become externally observable. PECD is not presented as a replacement for existing resiliency mechanisms but as a complementary architectural pattern applicable in systems where functional alternatives exist and proactive continuity is required. The proposed pattern provides a reusable technology agnostic blueprint that can be instantiated across different microservice platforms, programming languages, and deployment environments. Table 4 provides a GoF mapping [36] of the classical circuit breaker with the PECD pattern and Table 5 provides a comparison of state-of-the-art resiliency approaches with the proposed PECD pattern.

In this section, we present an advanced behavioral design pattern which integrates machine learning models with the Elastic De-Constructor pattern. The main goal of this framework is to predict service failures before they occur and automatically reroute requests to the service with the least probability of failure, thereby ensuring no interruption in the user experience. The framework consists of a predictor module which analyzes the historical logs and ensures that the current path to process the request has the least probability of failure. When the module identifies that the current path is failure prone based on the historical data, it switches over to the Deconstructed state and uses the predicted alternative path. This is depicted in Figure 1 with an added layer of machine learning to dynamically predict the best service, which is achieved by breaking the process into stages of data preprocessing to guarantee input data quality and feature engineering in extracting patterns from temporal and categorical data. The model is chosen for its ability to deal with a wide range of features while being efficient in the detection of non-linear relationships within the data.

The model functionality is analyzed with metrics like accuracy, precision, recall, F1 score, and ROC-AUC curve with the test data, which prove to be valid and generalizable. It is designed to scale up the training process with new data to enhance the projection accuracy of changing system conditions. This flexible approach opens more avenues toward further improvements, namely real-time integration of log data and continuous monitoring for the purposes of improving reliability predictions and proactive decision making towards service continuity. The methodology of the proposed solution includes the following key features:

Fig. 1: Proposed Pattern Flow



- Using machine learning techniques to detect and predict microservice failures based on historical data.
- Recommending alternative services to ensure seamless system performance and reliability which is used by the Elastic Circuit Deconstructor.

Table 4: GoF Mapping of Classical Circuit Breaker to Proactive ECD.

GoF Attribute	Classical Circuit Breaker	Proactive ECD (This Work)
Pattern Type	Behavioral	Behavioral (Extended)
Primary Intent	Prevent cascading failures	Preserve availability under failure
Core States	Closed, Open, Half-Open	Closed, **De-constructed**, Open Half-Open
Failure Handling	Reject requests	Reconstruct service path
Adaptivity	Reactive	Reactive + Proactive
Decision Logic	Threshold-based	Threshold + ML-driven

Table 5: Comparison of state-of-the-art resiliency approaches with the proposed PECD pattern

Approach / Study	Core Mechanism	Control Layer	Proactive	Multipath Routing	Strengths	Limitations Relative to PECD
Classical Circuit Breaker [37]	Threshold-based statemachine (Closed/Open/Half-open)	Application library / client-side	No	No	Simple and widely adopted; prevents cascading failures	Reactive; unavoidable downtime; no functional alternatives
Adaptive Circuit Breakers [38]	Runtime adaptation of circuit breaker thresholds and failure windows	Application library	No	No	Improved stability under workload variation	Still reactive; no prediction; no proactive rerouting
RL-based Service Mesh Resiliency [39]	Model-based reinforcement learning to optimize resiliency policies	Service mesh control plane	Partial	Indirect	Learns adaptive policies for routing, retry, and timeout control	High complexity and training overhead; not a reusable

						design pattern
Method- aware Circuit Breakers [40]	Contextual and methodlevel circuit breaker behavior customization	Runtime instrumentati on	No	No	Fine- grained control over circuit breaker semantics	Reactive; no predictive intelligenc e; no multipath routing
Self- adaptive Circuit Breaking and Retry Control [41]	Dynamic retry budgets and circuit breaker parameter tuning	Application library	No	No	Reduces retry storms and cascading amplificati on	Activates after failures occur; focuses on retries rather than routing
Self- healing / Autonom ic Systems [42] [43]	Detect and remediate failures via restart, rescheduling, or scaling	Orchestration / infrastructure layer	Partial	No	Automated recovery at the system level	Operates post- degradatio n; no service- level routing logic
ML- based Anomaly Detection [44] [45]	Machine learning-based prediction of anomalies from logs and telemetry	Monitoring / observability layer	Yes	No	Early detection of failure precursors	Detection- only; mitigation logic external to the model

In order to achieve the proposed approach, microservices data is gathered, then the data is processed and prepared to be understood by machine learning algorithms for training, testing purposes. These models are utilized for both fault detection purposes and alternative service suggestions. The prediction model framework is illustrated in Figure 2. The following mathematical equations represent how the proposed method works together.

Prediction Function

$$P(f_s) = \text{ML Model}(H_s)$$

where $P(f_s)$ is the probability of failure for service s , ML Model is the machine learning model, and H_s represents historical logs for service s .

Decision Function

$$D(s) = \begin{cases} \text{service } s' & \text{if } P(f_s) > \tau \\ s & \text{if } P(f_s) \leq \tau \end{cases}$$

where τ is the failure probability threshold, and s' is the alternative service chosen based on the lowest predicted failure probability.

Execution Function

$S = \text{Execute}(R, D(R))$

where S is the service response, R is the service request, and D is the decision based on the service rerouting logic.

Continuous Learning and Adjustment

$M' = \text{Update_Model}(M, N)$

where M' is the updated model, M is the current model, and N represents new incoming data.

Comprehensive Formula Representation

Combining all the components, the comprehensive formula is represented as:

$S = \text{Execute}(R, D(R = \text{ML_Model}(H)))$

This equation captures the approach of dynamically reordering service requests using predictive analytics to guarantee better dependability and reduce downtime in microservices systems.

4.1 Data Collection

Due to the lack of publicly-available datasets of labelled logs from production systems, we validate our approach by a mixture of logs that was generated from a system we designed and implemented in a distributed environment that mirrors users requests and a substantial dataset that was generated from a Java based application designed to send notifications. This application was packaged as a collection of microservices, deployed into a Kubernetes cluster. The data was collected using JMeter to simulate real time request generation to the different microservices in this case SMS, Email and Whatsapp microservices to send notifications. Simultaneously a python module was introduced to invoke APIs to induce errors into the system randomly. Logs were generated using the above approach by running the application for over a period of days at different intervals. By the end, a robust dataset was delivered which gives valuable insights into what the system would do when subjected to various conditions. A separate python script was executed to clean the generated logs and format the data in a fashion that can be processed by the machine learning models as depicted in Figure 3. To ensure the synthetic workload reflects real-world service patterns we compared our dataset's characteristics against industry-standard benchmarks such as DeathStarBench [46]. Our dataset includes over 380,000 request records across 4 microservices with injected anomalies mimicking faults observed in production (e.g., high latency, memory pressure, and CPU saturation). Metrics such as request latency (7ms–1500ms), CPU usage, memory footprint and labelled health

Fig. 2: Prediction Model workflow

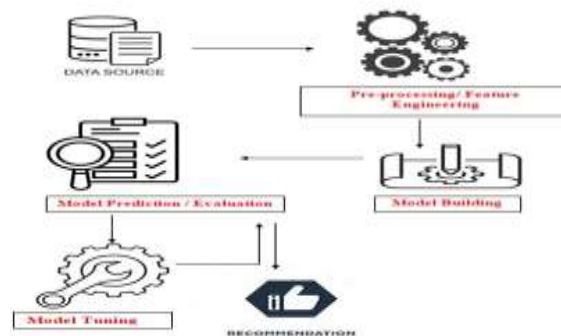


Table 6: Dataset used for Training the model

microservice	starttime	endtime	response code	latency	Memory Usage	CPU	Health
Email	1.73E+12	1.73E+12	200	15	432796064	133867000	TRUE
Email	1.73E+12	1.73E+12	200	15	432796064	100929000	TRUE
SMS	1.73E+12	1.73E+12	200	17	439039664	50324000	TRUE
SMS	1.73E+12	1.73E+12	200	17	439039664	47193000	TRUE
WA	1.73E+12	1.73E+12	200	7	440911912	37634000	FALSE
SMS	1.73E+12	1.73E+12	502	8	471375568	74386000	TRUE
SMS	1.73E+12	1.73E+12	502	9	471375568	44747000	TRUE
WA	1.73E+12	1.73E+12	502	5	319655784	162191000	FALSE
WA	1.73E+12	1.73E+12	200	5	301190504	231166600	FALSE

outcomes align closely with ranges reported in DeathStarBench social network traces. Unlike raw system traces, our dataset is structured for supervised learning tasks with explicitly annotated health states. This makes it more suitable for developing real-time predictive models. An extract from the generated dataset is depicted in Table 2. The dataset includes some essential metrics that describe the behavior and reliability of microservices. Each row captures information as follows:

Influential Factors:

- Microservice: The particular service involved in the interaction ie., (Email, SMS, WA (Whats App)).
- Timestamps: starttime and endtime in epoch and readable format to know when the service interaction started and ended.
- Latency: The time taken by the microservice to deliver the response. Its responsiveness and efficiency are interconnected.
- Memory consumption: A resource utilization metric
- CPU Usage: The processing taken by the service in handling the interaction. This aids in the evaluation of the system load and performance.

Target Variable

Response Code: HTTP response status codes such as 200 for success and 502 for Failure, it describes if the service was successful in processing the response or it failed. During the training phase, errorcode is used to train the model, while in testing the model's aim is to

predict this variable based on the input features.

This dataset will serve as the basis of the study through which machine learning models can be applied to proactively predict service failures, rank available services based on failure probabilities, and ensure a seamless and reliable user experience.

4.2 Data preprocessing

We started off by reading the data and examined its contents then addressed any redundancy or incomplete records. By using categorical encoding, we've encoded the categorical variables to facilitate easier conversion in the models. We then normalized the numerical features to bring down to the same scale ensuring consistency across the features by not giving heavy weightage to any of it in particular. Finally the data is split into training and testing, preparing it for the next steps in the process. These are presented in Table 4.

4.3 Model Building

We employed classification algorithms like Random Forest, Gradient Boosting, Logistic Regression and SVM to assess the model's capability to distinguish between failed and successful services. These algorithms have the ability to handle Non-Linear relationships as well which in our case is true . Following this the evaluation of the model is done to enhance the prediction capacity and reduce the overfitting/underfitting issues. Overfitting occurs when a model is too complex, capturing noise or fluctuations in the training data that do not generalize to new data. So we hypertuned

Fig. 3: Data Collection



Table 7: Feature extraction and transformation

Feature	Description
Starttime, Endtime	Converted to Datetime format
Memory Usage, CPU, latency	Power-transformed features to normalize their distributions for improved model performance.
target	Binary variable indicating success (1) or failure (0) of microservice requests.
hour, minute, second	Time-based features extracted from the starttime column to capture temporal patterns.
microservice encoded	Numerical encoding of the microservice categorical feature.

the model using RandomizedSearchCV. This approach performs an exhaustive search for the best parameters while incorporating K-fold cross-validation to ensure optimal model performance. Cross-validation helps in detecting overfitting by assessing model performance on multiple data subsets. Cross-validation is a technique which is used to assess the generalizability of this ensemble model. It typically splits our dataset into k-folds (in our case, k=5), training the model on k-1 folds [47] and testing it on the remaining

fold. This process is repeated for each fold to make sure the model is evaluated on different data splits. Algorithm 1 details the model building process.

Algorithm 1 Proactive ECD with ML Integration

Input: Training dataset $X_{\text{train}}, y_{\text{train}}$, Testing dataset $X_{\text{test}}, y_{\text{test}}$, Hyperparameters $\epsilon, n_{\text{estimators}}, \text{max depth}$

Output: Trained Model, Performance Metrics

Initialization: Model parameters, hyperparameter grid

Feature Engineering: Extract features {MemoryUsage, CPU, MicroserviceEncoded, Hour, Minute, Second} and set X, y

Data Splitting: Split X, y into $X_{\text{train}}, X_{\text{test}}, y_{\text{train}}, y_{\text{test}}$ with 80:20 ratio

Model Training: Initialize RandomForestClassifier with $n_{\text{estimators}}, \text{max depth}, \epsilon$
Train the model: $\text{model.fit}(X_{\text{train}}, y_{\text{train}})$

Hyperparameter Tuning (Optional): Use RandomizedSearchCV with cross-validation $k = 5$ to optimize hyperparameters

Prediction Phase: $y_{\text{pred}} \uparrow \text{model.predict}(X_{\text{test}})$

Evaluation: Compute accuracy, precision, recall, and F1-score y_{pred} indicates failure

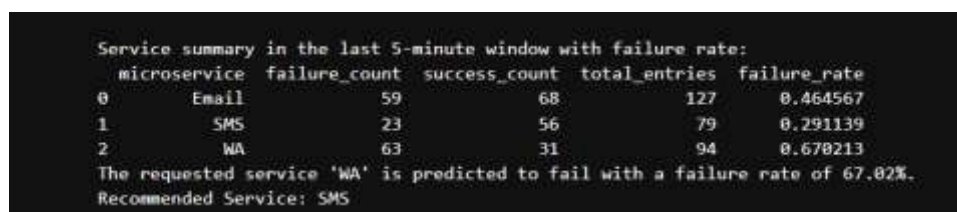
Trigger Service recommendation algorithm Proceed with normal operations

Output Results: Return best model hyperparameters and performance metrics

4.3.1 Alternative Service prediction

This framework is designed to make sure the service availability is uninterrupted when the user makes a request by proactively identifying the failures. This then begins by mapping each service into an encoded form. It identifies the defined time window to filter relevant service logs. Through this we ensure that the current operational index is being taken into consideration which reflects present service conditions. By utilizing the trained model, it predicts the potential failures for all services available. For each of those services, the failure rate is calculated based on the ratio of predicted failures to total observations within the time frame. If the service requested by the user, failure rate exceeds the threshold ie., (average failure rate of all microservices) we recommend the alternative service with least failure rate, in case it is lesser than the threshold, we will suggest it for continual use making sure the best alternative is provided as seen in Figure 4. The detailed flow of this is mentioned in Algorithm 2.

Fig. 4: Output from the model prediction



```

Service summary in the last 5-minute window with failure rate:
  microservice  failure_count  success_count  total_entries  failure_rate
0      Email             59             68             127      0.464567
1       SMS             23             56             79      0.291139
2        WA             63             31             94      0.670213
The requested service 'WA' is predicted to fail with a failure rate of 67.02%.
Recommended Service: SMS
  
```

5 Results

5.1 Ensemble Model evaluation

In this experiment, the Random Forest classifier achieved a testing accuracy of 85%, which indicates that it is well suited to capture data-driven patterns. Precision and recall were

equal at 85%, signifying a good ability to classify both positive and negative instances. Then, using RandomizedSearchCV, we performed hyperparameter tuning on the number of estimators, max tree depth, min samples per leaf, and criterion for splitting nodes. This was aimed at raising model complexity and reducing overfitting as well as improving stability in the cross-validation results.

To assess the effectiveness of different machine learning models for failure prediction, we compared multiple classifiers based on their accuracy scores. The results are summarized in Table 9.

Among these models, Random Forest Classifier demonstrated the highest accuracy (85%), making it the most effective model for failure prediction in our experiments. We used cross-validation to further probe generalization and found average accuracy to be 85%.

Algorithm 2 Service Recommendation Algorithm

Input: Requested Service S , Failure Threshold ω
Output: Recommended Service S_{best}
 Initialize Service Mapping: email $\rightarrow$ 0, sms $\rightarrow$ 1, wa $\rightarrow$ 2
Get current time as $t_{current}$
Find last entry time t_{last} from dataset
 Define time window $[t_{last} \leftrightarrow 5 \text{ minutes}, t_{last}]$ Filter dataset within the time window **if** No data exists in the time window **then**
Return: No data available for requested service S . **end if**
 Predict failure for all services in the window using model
 Calculate failure rates for each service Retrieve failure rate for requested service S **if** Failure rate of $S > \omega$ **then**
 Identify most reliable service with failure rate $\rightarrow \omega$ **if** No reliable services found **then**
 Recommend service with lowest failure rate **else**
 Recommend most reliable service **end if else**
 Recommend requested service S
end if
Return: Recommended Service S_{best}

Table 8: Comparison Of Accuracy

Model	Accuracy
Logistic regression	50%
KNN	63%
XGBClassifier	68%
Random forest Classifier	85%
Gradient Boosting Classifier	66%

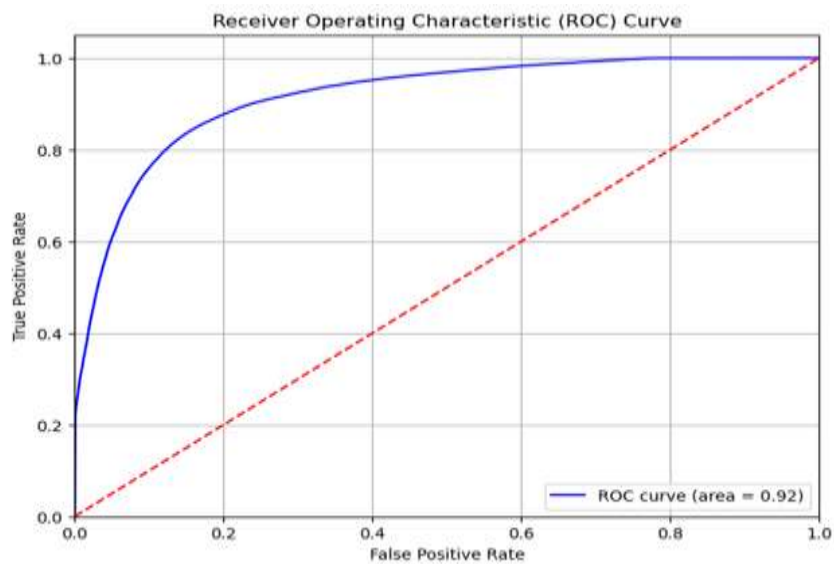
This has proved to be increasing the accuracy, precision and recall of both the metrics . The close performance between the validation scores (mean accuracy of 82.12%) and the test score (85.59%) demonstrates that the model has effectively learned the underlying patterns in the data without overfitting. The slight difference between the validation and test scores indicates that the model is robust, and it suggests that the model is neither overfitting nor underfitting, but rather generalizing well to both the validation and test datasets.

Classification Metrics:

Besides accuracy metric, one of the crucial evaluation tools when it comes to assess the classification between positive and negative classes at different thresholds is “The receiver operating characteristic tool”. As shown in Figure 5, The AUC-ROC curve has achieved 92% which demonstrates the model’s ability to reduce the classification

Table 9: Classification Metrics

Class	Precision	Recall	F1-Score
0	0.84	0.85	0.84
1	0.85	0.84	0.84
Accuracy	0.84		

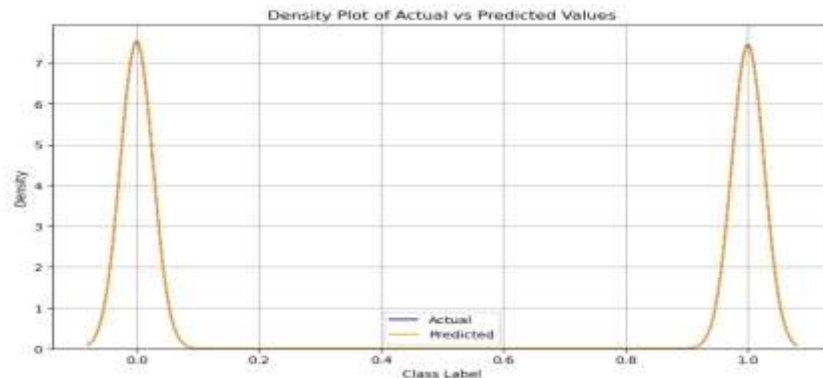
Fig. 5: The AUC-Roc curve of Random Forest Model

errors. The curve’s proximity to the top-left corner indicated a low false positive rate and high true positive rate, demonstrating a higher level classification. Mathematically, this is represented by the True Positive Rate (TPR) = $TP / (TP + FN)$ and False Positive Rate (FPR) = $FP / (FP + TN)$.

The final tuned model illustrated in Figure 6 shows very good predictive tendencies, minimal overfitting tendencies, and good generalization across data contexts. The fine-tuned model is suited for purposes where accurate classification is demanded.

5.2 Performance evaluation of the proposed design**Traditional Circuit Breaker vs. Proactive Approach**

As seen in Figure 7 in the traditional Circuit breaker pattern when the service goes down we notice that the requests start failing and the latency spikes as the requests

Fig. 6: Observed v/s Estimated plot

start piling over. The output of the reactive Elastic Circuit Deconstructor pattern is depicted in Figure 8. Here we notice that when the primary path fails on reacting to the failure the ECD switches over to the configured alternative path. This is seen with the dip before the application switches over to the adaptive microservice. These observations shed important light on the stability and limitations of reactive alternate paths used in reactive resiliency mechanisms and the importance of a proactive strategy in coping with sudden performance deviation.

As indicated in Figure 10, the final evaluation of the working model brings out the pioneering capability of the proactive failure detection system. Unlike conventional models, this system is intended to quickly identify any failure and easily switch over to other alternative paths. This proactive mechanism for failover ensures uninterrupted delivery of services by greatly improving user experience because service downtime is removed completely. The switch is instant and automatic, so even if there is a failure, the system can continue performing at full capacity, as the user is unaware of the shift in the underlying system. In this study the strategy proposed was evaluated by comparing the performance of the baseline reactive system and the proactive system employing machine learning for failure prediction. The evaluation focused on key metrics, including latency, failure handling efficiency, scalability, and user experience. Figure 9 shows the comparison of latency across different patterns. Under normal operation conditions, the application responds within 5ms without failing. However during our tests with a sample of 100 requests, we noticed that the latency increased to about 20ms when failures occurred without a circuit breaker. With the traditional circuit breaker having a faulty static alternative path, the latency increased to about

Fig. 7: Traditional - Circuit Breaker

Fig. 8: Reactive - Elastic De-constructor pattern



15ms on failure. The Elastic Circuit Deconstructor displayed a moderate increase in latency of around 10ms. The Proactive Elastic Circuit Deconstructor displayed optimal performance, showing only a slight increase in latency of around 8ms. This is depicted in Figure 9.

Fig. 9: Comparison of latency for different patterns

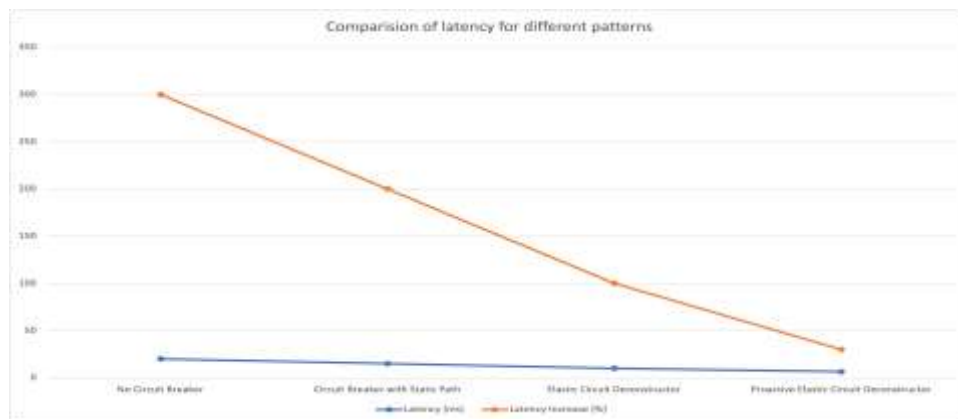
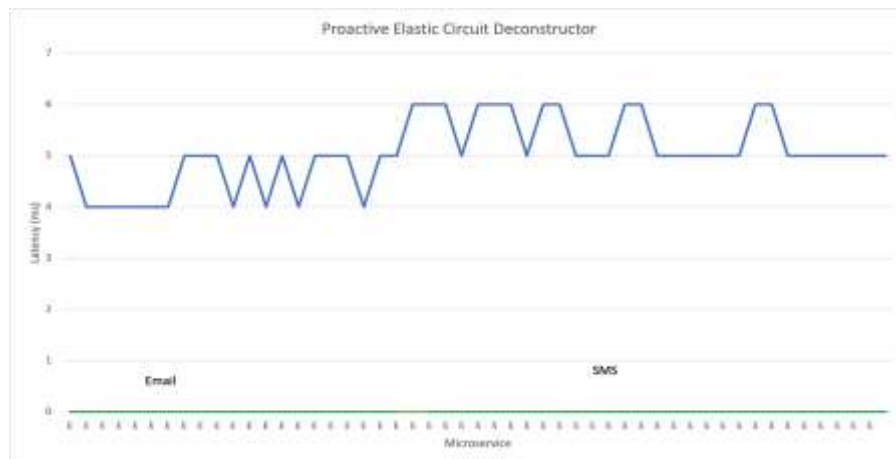


Fig. 10: Proactive - Elastic De-constructor pattern



Benchmark Comparison: Reactive vs. Proactive resiliency approaches

The comparison Table 10 highlights the key improvements offered by the Proactive approach over the Reactive approach in terms of the key performance measures. In the Reactive approach, due to the delay in detecting failures and inefficiency in retries, the success rate goes down with the rise in failure severity, resulting in higher latency and lower net promoter score (NPS) which results in lower customer satisfaction and customer retention. With respect to the Moderate Case, the Reactive approach has only 85% success due to sporadic failures, while the Proactive approach has 95% success by detecting and preventing failures in advance.

Similarly, the Reactive approach has 20 ms latency while the Proactive approach has 10 ms latency, which actually minimizes delays due to dynamically re-routing requests. With the rise in failure severity, the proactive mechanism performs better than the reactive system all the time, offering better reliability and user experience. The plot of reactive vs proactive approach under different scenarios is depicted in Figure 11.

Table 10: Reactive vs. Proactive resiliency approaches

Scenario	Request Success Rate (Reactive)	Request Success Rate (Proactive)	Latency (Reactive)	Latency (Proactive)	NPS (Reactive)	NPS (Proactive)
Best Case	100%	100%	5ms	5ms	100	100
Moderate Case	85%	95%	20ms	10ms	70	85
Degrading	65%	90%	70ms	20ms	30	75
Severe Failures	40%	85%	150ms	30ms	-10	60
Worst Case	20%	80%	250ms	40ms	-50	50

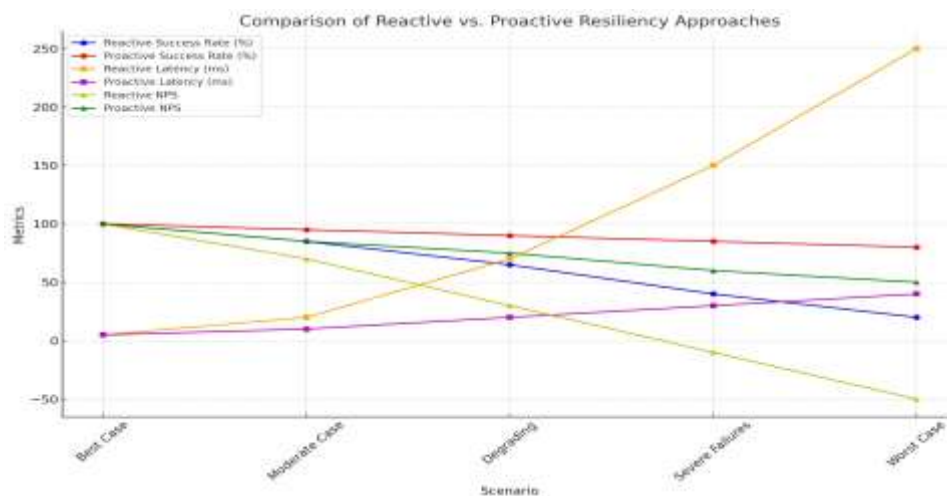
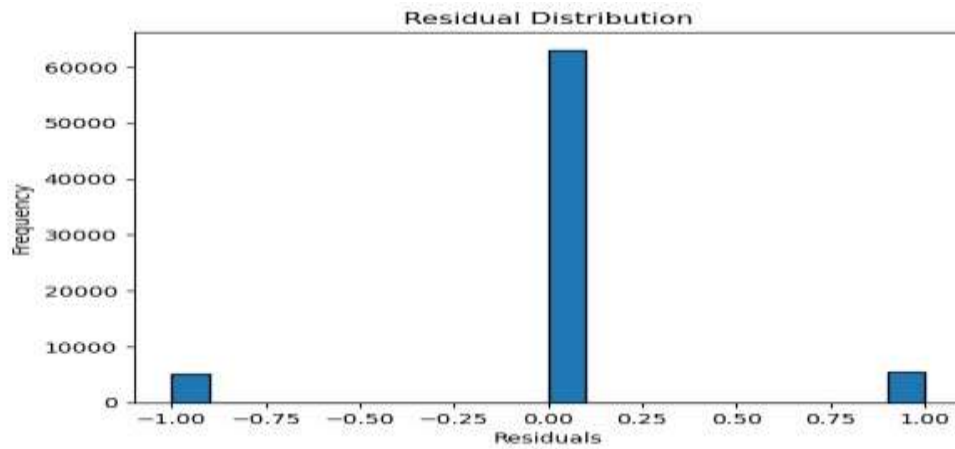
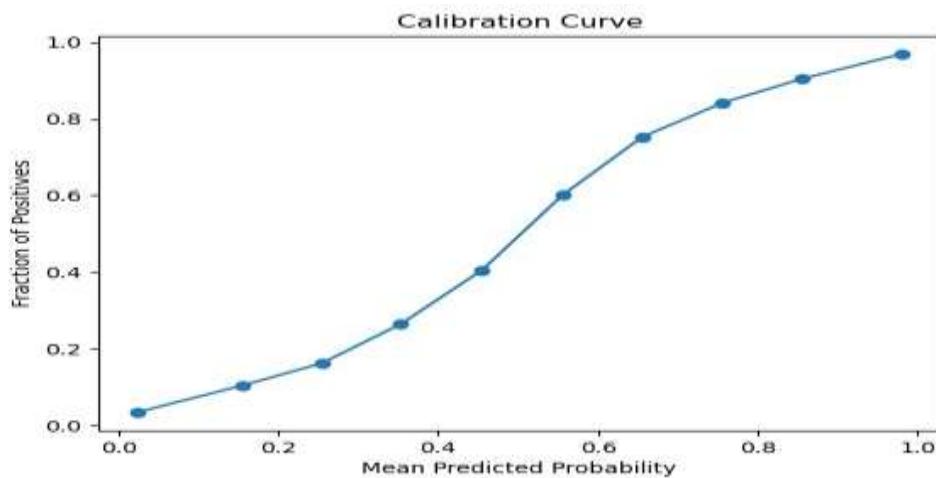
Fig. 11: A plot of Reactive vs Proactive approach

Fig. 12: Error rate analysis of Observed v/s Predicted**Limitations of the proactive failure detection approach:**

Although the failure prediction approach offers more excellent benefits over traditional approaches, there are a few drawbacks. The major drawback is a small overhead in performance while using mechanisms for anomaly detection and dynamic path-switching. While the failure prediction approach dramatically reduces latency, which was from 15ms in traditional systems to 8ms, there is an overhead compared with purely reactive models. The complexity of continuous risk assessment and real-time system adoption in hostile environments characterized by frequent or large anomalies. This may lead to higher resource usage. However, the model's performance and accuracy remain high, and more tuning can assist to address these concerns.

5.3 Residual Analysis For Model Performance Evaluation

The residual distribution plot Figure 12 reveals that the model correctly predicted most true positive cases (residual = 0), with fewer false negatives (residual = -1) and false positives (residual = 1). The large number of residuals at zero signifies good model performance, with a minimal number of misclassifications indicating potential areas for improvement.

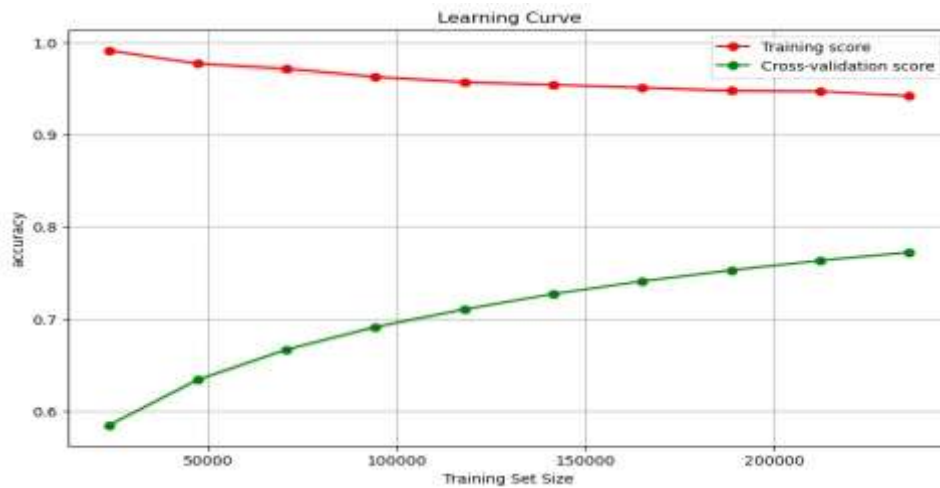
Fig. 13: Calibration curve

5.4 Model Validation and Future Potential

This analysis validates that our model's predictive ability will increase in the future. These studies will assist us improve when we opt for more sophisticated methods like online learning for continuous data updation and Deep learning methods.

This calibration curve explains that there is good probability, because the model's predicted probabilities closely match the observed results Figure 13. Given the curve seems relatively well-aligned with the diagonal, it shows that the model is making reliable predictions in terms of probability. To clarify on whether adding additional data to the model is improving the predicting accuracy or not as well as analyzing if the overfitting is reducing. The learning curve of our model demonstrates that when adding more observed data improves the accuracy meanwhile the overfitting is also seen reducing in the above given Figure 14. This serves as a proof in the validity of both the data as well as the model.

Fig. 14: Learning Curve Of Prediction Model



6 Conclusion and Future work

Inspired by the resilience mechanisms of storage multipathing, our Proactive Resiliency Pattern introduces an innovative approach to managing microservices in cloud environments. This study not only underscores the potential of multipathing principles in new domains but also sets a foundation for further innovations in applying these principles across broader IT infrastructure contexts. The study was able to propose a strong fault-tolerant proactive Elastic Circuit Deconstructor resiliency pattern utilizing machine learning with a Random Forest classification model used to predict the best service from historical failure data. The proposed pattern therefore enhances reliability in the services, and since the suggested pattern proactively chooses the highest probable service, the functionality of the application is not affected by the failure. This architecture enhances the robustness of microservices by the ability to build a system that can handle failures and reduce downtime through the use of the proactive Elastic Circuit Deconstructor pattern. Predictive modeling means one can predict service failure, and then one can dynamically reroute user requests to the most reliable service by using resilient design patterns. Since such an approach provides an accurate ranking of services with regard to their probability of failure, the proposed approach is effective for usage in the real-world application environments for the provision of continuous system availability and enhanced user satisfaction. Futurework, we can incorporate more adaptivity, using online learning methods that will enable the system to update with real-time data, hence making it more efficient and accurate in terms of predictions in a

changing environment. The system should be able to adapt to changes in patterns of failures in services so that near-instantaneous updates are maintained for selecting the most reliable services. Other potential models in machine learning could be reinforcement learning and deep learning methods which will enable the system to learn optimal policies with time, and further modification in the advance selection of services. Those can form a more robust, scalable, and fault-tolerant system to thrive well in dynamic cloud-based ecosystems.

7 Data Availability Statement:

The data that support the findings of this study are available from the corresponding author, Norton Stanley S.A, upon request.

8 Conflict Of Interest Statement:

On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

- [1] Gharibvand, V., Kolamroudi, Zeeshan: Cloud based manufacturing: A review of recent developments in architectures, technologies, infrastructures, platforms and associated challenges. *The International Journal of Advanced Manufacturing Technology* 131, 93–123 (2024) <https://doi.org/10.1007/s00170-024-12989-y>
- [2] Z. Ding, B.F. Y. Xu, Jiang, C.: Microservice extraction based on a comprehensive evaluation of logical independence and performance. *IEEE Transactions on Software Engineering* 50, 1244–1263 (2024)
- [3] Elgheriani, N.S., Ahme, N.D.A.S.: Microservices vs. monolithic architectures [the differential structure between two architectures]. *MINAR International Journal of Applied Sciences and Technology* 4, 500–514 (2022) <https://doi.org/10.47832/2717-8234.12.47>
- [4] Hanada, H., Ishibashi, K.: Hexagon: Generating asynchronous microservices benchmarks for the evaluation of anomaly propagation resistance. In: 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), pp. 2436–2441 (2024)
- [5] Stanley, N.S.A., Sebastian, S.: A systematic review on the identification and classification of patterns in microservices. In: 2023 4th International Conference for Emerging Technology (INCET), pp. 1–8 (2023)
- [6] Stanley, N.S.A., Sebastian, S.: Circuit breaker: A resilience mechanism for cloud native architecture. In: 2023 Global Conference on Information Technologies and Communications (GCITC), pp. 1–8 (2023)
- [7] Lopes, J., Scarfone, K.: Unleashing intelligence: The intersection of cloud computing, ai, and ml. *International Journal of Advanced Engineering Technologies and Innovations* 1 (2024)
- [8] B. Cline, D.H. R. S. Niculescu, Deckel, B.: Predictive maintenance applications for machine learning. In: 2022 Annual Reliability and Maintainability Symposium (RAMS), pp. 1–7 (2017)
- [9] J. Nobre, E.J.S.P., Reis, A.: Anomaly detection in microservice-based systems. *International Journal of Experimental Research and Review* 13, 7891 (2023)
- [10] N. Stanley, S., Sebastian, S.: Elastic circuit de-constructor: a pattern to enhance resiliency in microservices, 1–22 (2024)
- [11] J. Luo, J.-W.S., Xue., W.: Design and implementation of an efficient multipath for a san environment. *International Symposium on Parallel and Distributed Processing and Applications* (2005)

- [12] Khalil, M.: Storage design and implementation in vsphere 6: A technology deep dive. In: VMware Press Technology (2017)
- [13] Storage, P.: Content Studio, "What is MPIO? (2024). <https://www.purestorage.com/knowledge/what-is-mpio.html>
- [14] Singh, N.D.P.: Cloud-based machine learning: Opportunities and challenges. International Journal of Scientific Research in Computer Science Engineering and Information Technology 10(6), 267–270 (2024)
- [15] Hlybovets, A., Paprotskyi, I.: Increasing the fault tolerance in microservice architecture. Cybernetics and Systems Analysis 60 (2024)
- [16] A. Colman, Q.B.V. A. V. Uzunov, Chhetri, M.B.: Agent controlled service meshes for resilient, self-adaptive microservice systems. In: IEEE International Conference on Software Services Engineering (SSE), Chicago, IL, USA, 2023 (2023)
- [17] M. Panahandeh, M.H. A. Hamou-Lhadj, Miller, J.: Serviceanomaly: An anomaly detection approach in microservices using distributed traces and profiling metrics. Journal of Systems and Software 209 (2023)
- [18] D. Noetzdold, V.R.Q.L. A. G. D. M. Rossetto, Costa, H.J.D.M.: Enhancing infrastructure observability: Machine learning for proactive monitoring and anomaly detection. Journal of Internet Services and Applications 15 (2024)
- [19] Akta,s, K., Kilinc, H.H.: Interaction prediction and anomaly detection in a microservices-based telecommunication platform. In: ICSSP '24: International Conference on Software and Systems Processes, pp. 56–65 (2024)
- [20] K. Yamnual, P.P., Achalakul, T.: Failure detection through monitoring of the scientific distributed system. In: 2017 International Conference on Applied System Innovation (ICASI), pp. 568–571 (2017)
- [21] D. K. Yadav, A.K., Yadav, N.: Predicting machine failure using machine learning and deep learning algorithms. Sustainable Manufacturing and Service Economics 3, 100029 (2024)
- [22] W. Marfo, D.K.T., Moore, S.V.: Tcondition monitoring and anomaly detection in cyber-physical systems. In: 2022 17th Annual System of Systems Engineering Conference (SOSE) (2022)
- [23] Al., T.E.: Intelligent proactive fault tolerance at the edge through resource usage prediction. ITU Journal on Future and Evolving Technologies (3), 761–778 (2023)
- [24] Asmawi, T.N.T., Ismail, A., Shen, J.: Cloud failure prediction based on traditional machine learning and deep learning. Journal of Cloud Computing Advances Systems and Applications 11(1) (2022)
- [25] T, A.T.N., Ismail, A., Shen, J.: A proactive approach to fault tolerance using predictive machine learning models in distributed systems. International Journal of Experimental Research and Review 44, 208–220 (2024)
- [26] M. Raeiszadeh, R.H.G.J.E. A. Ebrahimzadeh, F. Mini, R.: Asynchronous realtime federated learning for anomaly detection in microservice cloud applications. IEEE Transactions on Machine Learning in Communications and Networking (2025)
- [27] Haroon, S.Z.A.H.M.A.A..A.T. M.: A proactive approach to fault tolerance using predictive machine learning models in distributed systems. International Journal of Experimental Research and Review (2024)
- [28] Akta,s, K., Kilinc, H.H.: Interaction prediction and anomaly detection in a microservices-based telecommunication platform. International Conference on Software and Systems Processes (ICSSP '24) (2024)
- [29] Kalaskar, .T.S. C.: Fault tolerance of cloud infrastructure with machine learning. Cybernetics and Information Technologies (2023)
- [30] Gururaj: Fault tolerance of network routers using machine learning techniques (2023)

- [31] Theodoropoulos: Intelligent proactive fault tolerance at the edge through resource usage prediction. *ITU Journal on Future and Evolving Technologies* (2022)
- [32] Asmawi: Cloud failure prediction based on traditional machine learning and deep learning. *Research Square* (2022)
- [33] Chakrabarty: Flight arrival delay prediction using gradient boosting classifier. In *Advances in intelligent systems and computing* (2018)
- [34] K. Yamnual, P.P., Achalakul, T.: Failure detection through monitoring of the scientific distributed system. *International Conference on Applied System Innovation (ICASI)* (2017)
- [35] al., R.R.: Instantops: A joint approach to system failure prediction and root cause identification in microservices cloud-native applications. *ICPE '24: Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering* (2024)
- [36] A. Ampatzoglou, S.C., Stamelos, I.: Research state of the art on gof design patterns: A mapping study. *Journal of Systems and Software*, vol. 86, no. 7 (2013)
- [37] Fowler, M.: *Patterns of Enterprise Application Architecture*, (2002)
- [38] Punithavathy, E., Priya, N.: Auto retry circuit breaker for enhanced performance in microservice applications. *International Journal of Power Electronics and Drive Systems/International Journal of Electrical and Computer Engineering*, vol. 14, no. 2 (2024)
- [39] F. Meng, L.J., Thottan, M.: Model-based reinforcement learning for service mesh fault resiliency in a web application-level. *arXiv (Cornell University)* (2021)
- [40] C. Meiklejohn, C.C.M.R. L. Stark, Miller, H.: Method overloading the circuit. In: *SoCC '22: Proceedings of the 13th Symposium on Cloud Computing* (2022)
- [41] M. R. S. Sedghpour, B.S.C.K. D. Garlan, Tordsson, J.: Breaking the vicious circle: Self-adaptive microservice circuit breaking and retry. In: *2023 IEEE International Conference on Cloud Engineering (IC2E)*, Boston, MA, USA (2023)
- [42] Kephart, J.O., Chess, D.M.: The vision of autonomic computing. *Computer*, vol. 36, no. 1 (2003)
- [43] Salehie, M., Tahvildari, L.: Self-adaptive software. *ACM Transactions on Autonomous and Adaptive Systems* (2009)
- [44] Tan, R., Li, Z.: Maad: A distributed anomaly detection architecture for microservices systems. In: *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2024)
- [45] al., M.J.: An anomaly detection algorithm for microservice architecture based on robust principal component analysis. *IEEE Access*, vol. 8 (2020)
- [46] al., Y.G.: An open-source benchmark suite for microservices and their hardware/software implications for cloud edge systems. In: *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems* (2019)
- [47] Scikit-learn: Cross-validation: evaluating estimator performance. https://scikit-learn.org/1.5/modules/cross_validation.html