Autonomous Self-Adaptation in the Cloud: ML-Heal's Framework for Proactive Fault Detection and Recovery

Qais Al-Na'amneh[1], Mahmoud Aljawarneh[2], Rahaf Hazaymih[3],
Ayoub Alsarhan[4], Khalid Hamad Alnafisah[5], Nayef H. Alshammari[6], Sami Aziz Alshammari[7]
Faculty of Information Technology, Applied Science Private University, Amman, Jordan[1,2]
Department of Computer Science, Jordan University of Science and Technology, Irbid, Jordan[3]
Department of Information Technology-Faculty of Prince Al-Hussein of Information Technology,
The Hashemite University, Zarqa, Jordan[4]
Department of Computer Sciences-Faculty of Computing and Information Technology,
Northern Border University, Rafha, Saudi Arabia[5]
Department of Computer Science-Faculty of Computers and Information Technology,
University of Tabuk, Tabuk, Saudi Arabia[6]
Department of Information Technology-Faculty of Computing and Information Technology,
Northern Border University, Rafha, Saudi Arabia[7]

***Abstract*—Cloud computing environments increasingly host applications constructed from orchestrated service compositions, which deliver enhanced functionality through distributed workflows. This paradigm, however, introduces vulnerabilities where component failures can cascade, disrupting entire applications. Conventional fault tolerance often falls short in these dynamic settings. This paper introduces ML-Heal, an autonomous self-healing framework architected to bolster the resilience of such service compositions. ML-Heal leverages machine learning for proactive failure detection, precise diagnosis, and intelligent recovery strategy selection. The framework integrates real-time monitoring data, applies ML-based anomaly detection and classification to identify faults, and plans corrective actions via a learned policy or predictive models. Implemented using Python with scikit-learn models and a custom orchestration layer, its efficacy is demonstrated through simulated fault injection scenarios. Illustrative system architecture and evaluation results show that this ML-driven methodology significantly curtails recovery time and augments availability when confronted with faults, showcasing AI's potential in creating more robust, self-adaptive cloud service compositions with minimal human oversight.**

Keywords*—Cloud computing; service composition; self-healing systems; autonomic computing; machine learning; anomaly detection; automated recovery; fault tolerance

I. Introduction

Cloud computing platforms provide ubiquitous, on-demand access to a shared pool of configurable computing resources and services [1], [2], fundamentally altering software development and deployment paradigms. Within this evolving landscape, applications are frequently architected as orchestrated compositions of distributed web services or microservices, executing complex, often stateful, workflows to deliver enhanced and agile functionality [3], [4]. This compositional approach, characterized by loose coupling and independent scalability of components, while offering significant advantages, inherently introduces intricate failure modes and heightened fragility [5]. The failure of a singular component service—stemming from network partitions, hardware malfunctions, software defects, or even transient performance degradations—can propagate insidiously through the workflow, leading to the disruption or complete outage of the entire application [6], [7]. Such incidents adversely impact user experience, breach service level agreements (SLAs), and can inflict substantial damage on business objectives [8]. Fig. 1 conceptually illustrates this vulnerability.

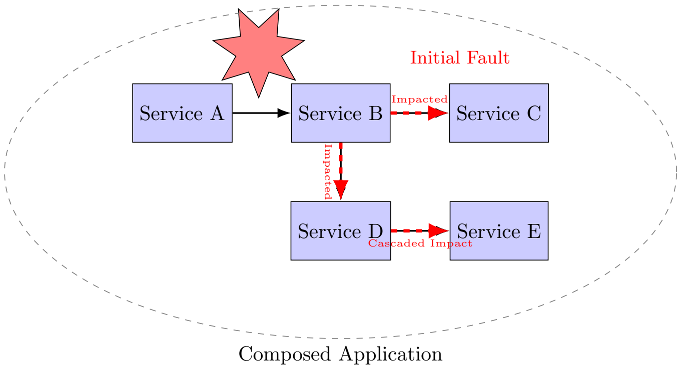

Fig. 1. Conceptual illustration of cascading failures in a service composition. A fault in service B impacts downstream services C, D, and subsequently E.

Ensuring the continuous availability and correct execution of these service compositions is thus of paramount importance in contemporary cloud environments [9], [10]. Traditional fault tolerance mechanisms, including simple replication, checkpointing, or manual failover procedures operating primarily at the infrastructure level, do not comprehensively address this intricate problem [11], [12]. These methods often necessitate substantial redundant resources, can be slow to react, and require significant human intervention for accurate diagnosis and subsequent repair, culminating in extended downtime and operational inefficiency [13], [6], [14]. In stark contrast, the vision of autonomic computing proposes systems capable of self-management—including self-configuration, self-optimization, self-protection, and critically, self-healing—with minimal human input [15], [16]. A pivotal aspect of this vision is *self-healing*: the intrinsic ability of a system to autonomously detect, diagnose, and repair faults, thereby restoring normal operation [17], [16]. Applied to cloud service compositions, self-healing would empower applications to adapt their workflows dynamically at runtime, seamlessly replacing or reconfiguring faulty services without operator action, thereby preserving the stipulated quality of service and enhancing overall resilience [16].

Recent advancements in machine learning (ML) have unveiled new avenues for developing intelligent and proactive self-healing capabilities [18], [19], [20]. Predictive models, trained on vast streams of operational telemetry, can discern subtle anomalous behavior patterns or anticipate impending failures [21], [22]. Concurrently, reinforcement learning (RL) techniques can discover efficient and context-aware recovery strategies through iterative system interaction and learning from outcomes [23], [24]. Nevertheless, the sophisticated integration of diverse ML-driven fault recovery mechanisms into the fabric of highly dynamic and complex service compositions remains an area that is relatively underexplored, with many existing solutions focusing on isolated aspects rather than an end-to-end intelligent healing pipeline [25], [17].

This paper introduces ML-Heal, a novel framework for self-healing service compositions specifically employing a synergistic suite of ML techniques [26], [27]. We architect a layered system that continuously monitors service metrics, applies ML-based fault analysis for proactive detection and precise diagnosis, and triggers adaptive reconfiguration through learned policies and predictive insights. The primary contributions of this work are:

- A comprehensive framework, ML-Heal, for self-healing in cloud service compositions, combining real-time monitoring with ML-based diagnosis, predictive planning, and adaptive orchestration.
- Detailed exposition of diverse machine learning methodologies (e.g., time-series anomaly detection, classification, causal inference hints, reinforcement learning principles) tailored for fault detection, root cause localization, and optimized recovery, along with practical implementation considerations.
- Integration of illustrative system diagrams and detailed tables to elucidate the system design, operational aspects, and experimental setup.
- An empirical evaluation of the framework's performance, demonstrating significantly accelerated recovery times and enhanced availability compared to conventional baseline approaches and simpler rule-based systems.

The remainder of this paper is structured as follows. Section II reviews pertinent related work in self-healing systems, traditional fault tolerance, and the application of ML in cloud reliability. Section III describes our proposed ML-Heal framework and its core ML-based recovery techniques. Section IV presents implementation specifics, including data handling, code snippets, and system diagrams. Section V discusses the evaluation methodology, experimental setup, and detailed findings. Finally, Section VI concludes the paper and outlines potential future research directions.

II. Literature Review

The quest for resilient distributed systems has spurred extensive research. This section surveys prior work relevant to self-healing service compositions, covering traditional fault tolerance, established autonomic computing paradigms, and the burgeoning application of machine learning for enhancing system reliability.

Further advancing the application of AI in autonomic systems, Alonso et al. [28] address optimization methodologies tailored for applications within the expansive "cloud continuum," enabling them to achieve robust self-healing and derive experiential learning. Their research endeavors to cultivate solutions where applications can autonomously recover from operational disruptions, thereby preserving critical service performance and dependability, while integrated self-learning capabilities facilitate adaptation and evolution based on accumulated operational data. The study particularly emphasizes optimization techniques for judicious resource and configuration selection, which demonstrably improve application performance and operational efficiency, identifying artificial intelligence (AI) [29] as a pivotal catalyst for these sophisticated self-management capacities.

The specific challenge of "model performance degradation" in machine learning systems, frequently a consequence of dynamic shifts in underlying data generation processes, is investigated by Rauba et al. [30]. They observe that conventional adaptation techniques, such as those designed to address concept drift, often fail to diagnose the root causes of performance decline and instead resort to predefined, often suboptimal, corrective actions. In response, their work introduces H-LLM, an innovative methodology that harnesses the capabilities of large language models (LLMs) for autonomous self-diagnosis of model deficiencies and the subsequent formulation of prescriptive remediation strategies. Empirical evaluations have substantiated H-LLM's capacity to significantly improve model performance under fluctuating real-world conditions, thereby championing a "self-healing machine learning" (SHML) paradigm wherein models are empowered to independently identify performance impediments and prescribe tailored corrective interventions.

Focusing on practical system development [31] delineate the design and evaluation of a self-healing cloud system that synergistically combines an event-driven automation framework with AI-augmented decision-making. The efficacy of their system is empirically validated on an OpenStack-based video-on-demand service, where simulated fault scenarios are employed to rigorously assess the implemented recovery procedures and automated workflows. The AI-driven decision-making module within their architecture analyzes multifaceted operational data to discern and select the most efficacious corrective measures, holistically considering factors such as service quality impact and resource implications. While the recovery engine relies on initial human expertise for parameterizing and optimizing decision models, the study compellingly demonstrates that AI-driven decision support can substantially reduce mean time to repair (MTTR) and enhance overall service quality within complex cloud environments.

In a similar vein, Vankayalapati and Pandugula [32] underscore the transformative potential of AI in constructing self-healing cloud infrastructures capable of autonomous recovery from runtime anomalies. They highlight the escalating importance of self-reliant services, particularly as paradigms like "cloud-in-a-robot" gain traction. Acknowledging a persistent lacuna in comprehensive real-time failure recovery solutions despite prior research into robust deep learning models, the authors propose a novel model architecture for AI-powered self-healing cloud infrastructures. This architecture distinctively integrates autonomous fault detection, sophisticated reasoning-based fault diagnosis, and advanced deep reinforcement learning (DRL) methodologies, all orchestrated to significantly minimize repair times. Their proposed model strategically addresses the multifaceted challenges and critical components inherent in the development of truly resilient and adaptive cloud infrastructures.

A. *Traditional and Rule-Based Fault Tolerance*

Early endeavors in fault tolerance for distributed systems centered on techniques like replication for availability [33], checkpointing for state recovery [34], and transaction mechanisms for consistency [35]. While foundational, these mechanisms often target specific failure types (e.g., crash failures, Byzantine faults [36]) and may incur significant overhead or lack the adaptability required for dynamic service compositions operating in volatile cloud environments [37].

The advent of autonomic computing [15] stimulated research into systems capable of self-management, with self-healing as a core tenet. Many initial self-healing systems relied heavily on predefined rules, policies, or workflow models [38]. These approaches typically define expected system behaviors and specify corrective actions (e.g., "restart service X if unresponsive," "scale deployment Y if CPU > 80%") triggered when deviations are detected, often using Event-Condition-Action (ECA) rules [39]. While effective for well-understood, known failure modes in relatively stable environments, rule-based systems often struggle with scalability as the number of rules burgeons, exhibit brittleness when faced with unforeseen scenarios, and lack the capacity to handle novel or subtly emergent fault conditions effectively [40], [41]. Workflow adaptation techniques, focusing on modifying the structure or execution path of a service composition at runtime to bypass faulty components [42], address control-flow failures but may not resolve underlying resource issues or complex performance degradations. Architectural patterns such as the Circuit Breaker [43] provide localized resilience by preventing repeated calls to failing services, but they do not inherently diagnose the root cause or orchestrate broader system-level recovery. Platform-level orchestration systems like Kubernetes [44] offer basic self-healing (e.g., restarting failed containers), which are invaluable but operate primarily at the infrastructure level, often lacking application-specific context for nuanced recovery [45]. Fig. 2 provides a conceptual comparison of these approaches.

B. *Autonomic Computing and the MAPE-K Loop*

The MAPE-K (Monitor, Analyze, Plan, Execute, over a Knowledge base) control loop, introduced as part of IBM's vision for autonomic computing [16], provides a widely adopted architectural blueprint for self-managing systems [16].

- Monitor: Collects data from managed resources (metrics, logs, traces).
- Analyze: Processes monitored data to detect symptoms, diagnose problems, and identify potential areas for improvement or adaptation.
- Plan: Develops a sequence of actions to achieve the goals and objectives based on the analysis.
- Execute: Implements the planned actions by interacting with the managed resources.
- Knowledge: Represents shared data, models, and policies used by the other components, enabling learning and adaptation over time.

Many self-healing frameworks, including early ones like [17], explicitly or implicitly follow this model. The effectiveness of each phase, particularly Analyze and Plan, is critical for intelligent and adaptive behavior.

C. *Machine Learning in Cloud Reliability and Management*

The exponential growth in volume, velocity, and variety of operational data from cloud systems (metrics, logs, traces, configurations) has propelled the application of ML for enhancing system management, reliability, and AIOps (AI for IT Operations) [64], [25].

1) Anomaly detection: Significant effort has focused on ML-based anomaly detection in multivariate time-series monitoring data [21], [65]. Algorithms such as Isolation Forests [49], One-Class SVMs [50], Autoencoders (AE) [47], LSTMs [22], [18], Prophet [48], and various graph-based anomaly detection (GBAD) techniques [51] learn normal patterns from high-dimensional system metrics to identify subtle deviations potentially indicative of impending failures. Proactive detection is pivotal for timely recovery [52].

2) Root Cause Analysis (RCA): Once an anomaly is detected, diagnosing its root cause is paramount. ML techniques are increasingly explored for RCA [59]. Classification algorithms can map symptom patterns to known root causes if labeled historical data is available [53]. Probabilistic Graphical Models (PGMs) like Bayesian Networks can model causal relationships [54], [55]. Log analysis, using NLP techniques from simple TF-IDF to advanced models like BERT, helps extract failure signatures from unstructured log data [56], [57]. Causal discovery algorithms aim to infer fault propagation paths directly from observational or interventional data [58].

3) Automated recovery and planning: For recovery planning, ML [66] offers avenues beyond predefined scripts. Reinforcement learning (RL) agents can learn optimal sequences of recovery actions (e.g., restart, migrate, scale, reconfigure) by interacting with the system (or a simulator) and receiving rewards based on recovery success and efficiency [23], [60], [61]. Predictive models can estimate the success probability,

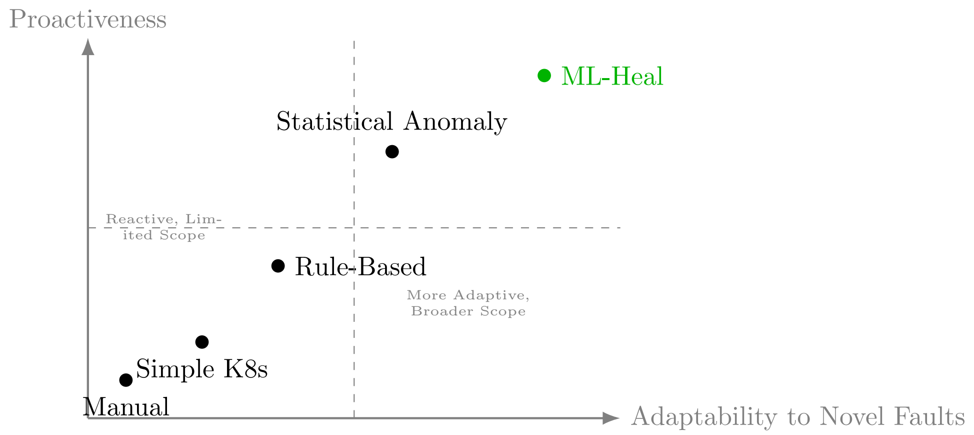

Fig. 2. Comparison of different healing approaches based on proactiveness and adaptability. ML-Heal aims for high proactiveness and adaptability.

TABLE I. COMPARATIVE SUMMARY OF LITERATURE ON SELF-HEALING AND ML FOR CLOUD RELIABILITY

Category / Reference(s)	Domain	Primary Technique(s) / Approach	Strengths	Limitations / Gaps Addressed by ML-Heal
Traditional Fault Tolerance	Distributed Systems Reliability	Replication [33], Checkpointing [34], Transactions [35], BFT [36], [37]	Foundational for specific failures, consistency guarantees	High overhead, limited adaptability to dynamic cloud/novel faults, often reactive.
Rule-Based Autonomic Systems	Self-Management, Self-Healing	Predefined rules, policies, ECA [39], workflow models [46], [38]	Simple for known faults, direct control flow	Brittle with novel faults, scalability issues, lacks learning, often limited context.
Workflow Adaptation	Service Composition Resilience	Runtime modification of execution path [42]	Handles control-flow failures, structural adaptation	May not address resource/performance issues, limited beyond workflow logic.
Platform-Level Orchestration	Infrastructure Healing (e.g., Kubernetes [44])	Health checks, automated restarts, replica management	Basic instance-level resilience, some automation	Lacks deep application context [45], limited diagnosis, fixed recovery actions.
Autonomic Computing (MAPE-K)	General Self-Management Blueprint	Monitor-Analyze-Plan-Execute-Knowledge Loop [16]	Holistic management cycle, structured approach	Early instances often relied on simpler analysis/planning; ML integration is key for advancement.
ML for Anomaly Detection	Proactive Fault/Performance Issue Identification	Time-series analysis (LSTM [22], [18], AE [47], Prophet [48]), Isolation Forest [49], OC-SVM [50], GBAD [51]	Handles complex patterns, proactive detection [52]	Often standalone; requires integration with diagnosis and recovery for end-to-end healing.
ML for Root Cause Analysis (RCA)	Fault Diagnosis and Localization	Classification [53], PGMs (Bayesian Nets [54], [55]), Log Analysis (NLP/BERT [56], [57]), Causal Discovery [58]	Data-driven diagnosis, identifies complex causes [59]	Requires quality (labeled) data or causal models, often separate from healing execution.
ML for Automated Recovery/Planning	Intelligent Action Selection	Reinforcement Learning [23], [60], [61], Predictive Modeling [62], [63]	Adaptive decision-making, optimizes recovery strategy	RL can be complex to train/ensure safety; predictive models need robust historical data.
AI for Cloud Continuum/Self-Learning	Optimization	AI for resource/config selection, experiential learning [28]	Performance / efficiency gains, adaptation over time	Focus on optimization; specific healing mechanisms might vary.
Self-Healing Machine Learning (SHML)	Model Performance Degradation	LLMs for self-diagnosis and remediation prescription [30]	Addresses data drift by self-correcting ML models	Specific to ML model healing, not general system/service healing.
AI-Driven Cloud Healing Systems	Holistic System Recovery	Event-driven automation + AI decision-making [31], DRL for recovery [32]	Reduced MTTR, improved service quality, autonomous recovery from runtime issues	Highlights benefits of AI; ML-Heal aims for a more comprehensive, deeply integrated ML pipeline.
ML-Heal (This Work)	**Self-Healing for Cloud Service Compositions**	**Integrated ML Pipeline (Proactive Anomaly Detection, Diagnostic Classification/RCA, ML-guided Recovery Planning), Adaptive Learning via Knowledge Base**	**Proactive, adaptive, context-aware healing across the full MAPE-K cycle; aims to handle novel faults and continuously improve through integrated learning.**	**Addresses the need for a unified framework specifically leveraging diverse, advanced ML for end-to-end, intelligent healing of composed cloud services, bridging gaps between siloed ML applications.**

cost, or duration of potential recovery actions given the current system state and fault context, enabling more informed selection [62], [63].

Table I describe summarizes representative approaches. While existing research has explored facets of self-healing and applied ML to specific problems, a gap remains in comprehensive frameworks that systematically integrate diverse, advanced ML techniques across the *entire* self-healing lifecycle for complex service compositions in dynamic cloud environments. Many ML applications remain siloed. ML-Heal aims to bridge this by providing a unified architecture where ML models collaborate for holistic self-healing, moving beyond simple restarts or rigid rules towards context-aware, learned recovery.

III. PROPOSED FRAMEWORK ML-HEAL

We propose ML-Heal, an architecture meticulously designed for the self-healing of service compositions, which distinctively integrates continual monitoring, machine learning-based fault diagnosis, and dynamic recovery actions. Fig. 3 illustrates the comprehensive design of this framework. A user request initiates the execution of a composite service, managed through a central orchestrator. Each component service participating in the composition is instrumented to emit critical runtime metrics (such as CPU utilization, memory

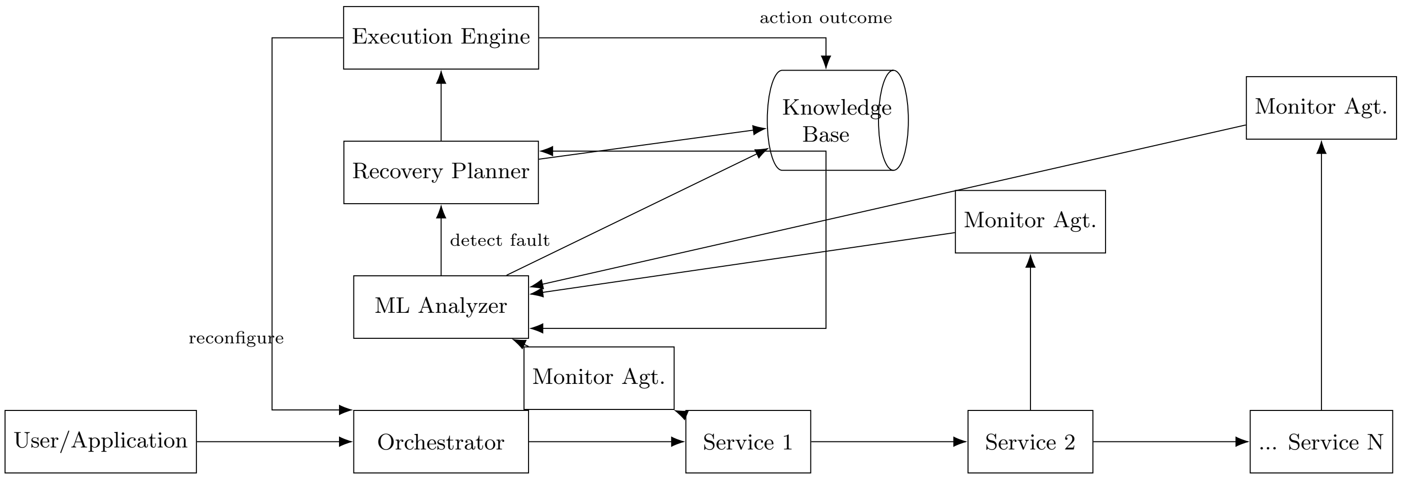

Fig. 3. Proposed ML-Heal framework architecture: services are monitored, analyzed by ML models, and adapted via planned actions when faults are detected, with a central Knowledge Base for learning.

consumption, response times, error logs) and local health indicators. A dedicated *Monitoring* module diligently collects these metrics, aggregating them into a continuous data stream for analysis.

The core intelligence of ML-Heal resides in an ML-based *Analyzer*. This module applies sophisticated anomaly detection algorithms, potentially employing classifiers or clustering models, to identify significant deviations from established normal operational behavior. If a potential fault is detected, the Analyzer proceeds to classify its specific type (e.g., service crash, performance degradation, resource exhaustion) and subsequently invokes the *Recovery Planner*. This planner utilizes a learned policy, typically embodied by a trained reinforcement learning agent or a predictive model, to decide upon the most appropriate corrective measures. These measures can range from rerouting task requests to an alternative service replica, restarting a failed component, to dynamically scaling allocated resources. Finally, the orchestrator executes the chosen recovery action, adaptively modifying the workflow in-flight to mitigate the fault's impact. The outcome of each executed action is then fed back into the monitoring loop, serving to update the system's knowledge base and progressively refine the underlying ML models.

Key features of the ML-Heal framework include:

1) Continuous monitoring: Each individual service instance diligently reports telemetry data (e.g., response times, processing queue lengths, error counts, resource utilization). Dedicated monitoring agents preprocess these raw signals to normalize their scale and filter out extraneous noise. This processed data is then streamed to a central repository for further analysis and storage, forming the basis for the Knowledge component of the MAPE-K loop [67].

2) ML-Powered anomaly detection: We employ sophisticated machine learning classifiers and time-series models to accurately detect faults. For example, a Random Forest or Support Vector Machine (SVM) can be trained on historical metrics that have been meticulously labeled as indicative of either *healthy* or *faulty* states [18]. Time-series models like LSTMs or Autoencoders can learn normal temporal patterns and flag deviations [22], [68]. Our approach augments this supervised detection with unsupervised methods, such as the Isolation Forest algorithm [49], to effectively identify novel or previously unseen anomalies. This hybrid strategy aims to reduce false positives and enhance the system's capability to handle unknown failure modes [18], [46], [21].

3) Intelligent fault classification and diagnosis: Once an anomaly is flagged by the detection mechanism, the analyzer component proceeds to classify its specific type (e.g., distinguishing between a service crash, a slow response time issue, or resource exhaustion) and attempts to localize the root cause. This detailed classification and diagnosis significantly aids the recovery planner in selecting the most appropriate and effective corrective actions. Techniques may include supervised classifiers trained on known fault signatures [53], or exploring causal relationships using service dependency graphs and patterns in distributed traces [55], [40]. Prior research has consistently shown the substantial benefit of distinguishing between different failure causes to tailor recovery efforts more precisely [62].

4) Adaptive recovery planning via ML: The planner utilizes a policy, learned through reinforcement learning techniques or guided by predictive models, to strategically choose a recovery strategy. The available actions may encompass switching to a backup service endpoint, redistributing incoming requests across healthy instances, restarting the affected service, dynamically scaling resources, or migrating it to a different underlying physical or virtual node [20]. Inspired by recent advancements in RL applications [23], [60], we model the recovery process as a sequential decision-making problem. An RL agent (e.g., Deep Q-Network (DQN) [69] or standard Q-learning) can be trained within a simulated environment that mimics the behavior of services and the occurrence of various faults. Over time, this agent learns which sequences of actions are most effective in restoring the service composition's performance under diverse fault scenarios. Alternatively, predictive models can estimate the utility (e.g., success probability, recovery time) of different actions [63].

TABLE II. MACHINE LEARNING TECHNIQUES IN ML-HEAL MODULES

Module	ML Technique(s)	Primary Input	Primary Output	Adaptation Mechanism
Monitoring	(Data Preprocessing) Signal Processing, Feature Engineering	Raw telemetry (metrics, logs, traces)	Cleaned, normalized feature vectors	Configuration updates, filter adjustments
Analyzer (Detection)	LSTM Autoencoders, Isolation Forest, Random Forest (Supervised)	Feature vectors from Monitoring	Anomaly scores, binary anomaly flags	Periodic retraining on new normal data, threshold tuning, model updates based on feedback
Analyzer (Diagnosis/RCA)	Supervised Classifiers (e.g., SVM, Gradient Boosting), Bayesian Networks, Log Pattern Analysis (e.g., LogCluster, Drain)	Anomaly details, contextual metrics, service dependency graph, historical incidents	Probable root cause(s), fault type, confidence scores	Retraining with new labeled incidents, updating causal models, refining log parsers
Planner (Recovery)	Q-Learning/DQN, Predictive Models (e.g., Regression for MTTR prediction), Case-Based Reasoning	Diagnosed fault, system state, action repertoire, recovery objectives	Optimal recovery plan/action sequence	Updating Q-tables/policy networks, retraining predictive models with action outcomes, adding new cases to CBR system

5) Automated adaptive execution: The orchestrator component is responsible for executing the chosen recovery actions autonomously, without requiring human intervention. Recognizing that newly instantiated service instances or reconfigured components may require a "warm-up" period, the system is designed to potentially degrade gracefully for a short duration or parallelize tasks where feasible to maintain service continuity. The framework meticulously logs the outcome (success or failure, time taken, resource impact) of each recovery action. This outcome data is then fed back into the system to update the ML models within the Knowledge Base, effectively serving as the learning mechanism of the MAPE-K loop, enabling continuous improvement and adaptation to new fault patterns [70].

This ML-driven strategy aligns seamlessly with established autonomic computing principles: the system observes its own state, learns from operational experience, and dynamically adapts its composition to maintain desired service levels. Notably, our design leverages artificial intelligence not merely for passive monitoring, but actively for decision-making processes within the core MAPE-K control loop. Table II provides a summary of ML techniques envisioned for each module.

IV. IMPLEMENTATION ARCHITECTURE AND PROTOTYPE

To operationalize the ML-Heal concept and evaluate its core functionalities, a prototype system was developed. This section provides a high-level overview of the prototype's architecture, the foundational technologies selected, and the general approach to integrating machine learning components.

A. System Design and Technologies

The prototype of ML-Heal was conceptualized as a modular system, reflecting the distinct stages of the self-healing process outlined in the framework architecture (Fig. 3). Python was chosen as the primary development language, benefiting from its robust libraries for data manipulation, machine learning, and system integration.

1) Simulated environment and service mocking: For the prototype, a simulated cloud environment was established. This involved creating mock service instances that could emulate basic computational tasks and exhibit various fault behaviors upon simulated injection. This controlled environment facilitated the systematic testing and evaluation of the self-healing mechanisms without the complexities of a full-scale cloud deployment.

2) Monitoring and data collection: The Monitoring module in the prototype was designed to simulate the collection of essential runtime metrics from the mock services. These metrics included simulated CPU load, response times, and error rates, which were then pushed to a central data stream or repository for processing by the Analyzer component. This setup mimicked how real-world monitoring agents (e.g., using REST APIs or Prometheus exporters) would function.

3) Machine learning model integration:

a) Fault detection and classification: The Analyzer component utilized machine learning models primarily from the `scikit-learn` library [71]. For instance, a Random Forest classifier was implemented for fault detection, trained on pre-labeled datasets representing 'healthy' and 'faulty' service states based on the collected features. This allowed the prototype to flag anomalous conditions based on learned patterns.

b) Recovery planning with reinforcement learning: To explore adaptive recovery strategies, the Planner component incorporated a simplified reinforcement learning (RL) approach. A custom RL environment was conceptualized using the principles of the Gymnasium (formerly OpenAI Gym [72]) toolkit. This environment simulated the state of the composite service and the effects of different recovery actions (e.g., 'retry,' 'switch_replica'). A basic Q-learning agent was then trained within this environment to learn a policy for selecting recovery actions that maximized a defined reward signal, typically related to successful task completion or fault resolution.

4) Knowledge representation (conceptual): While a full-fledged, persistent Knowledge Base with sophisticated data stores was beyond the scope of the initial prototype, the design accounted for the conceptual need to store historical data, learned models, and system state information. For the prototype, this was often managed in memory or through simple file-based storage during training and execution phases.

5) Orchestration and communication logic: The interaction between the framework's modules (Analyzer, Planner, Executor) and the simulated services was managed through custom Python logic. The Orchestrator component, upon receiving a recovery plan, would simulate the execution of actions like restarting a service or rerouting a request within the testbed. Communication was primarily handled through direct function calls or simple in-process messaging within the prototype's single-node execution environment.

The primary goal of this prototype implementation was to demonstrate the feasibility of integrating machine learning into the different stages of a self-healing loop for service compositions. The focus was on the core logic of fault detection, ML-driven decision-making for recovery, and the feedback mechanism for learning, rather than on building a production-grade, highly distributed system. The modular design, even in this simplified form, aimed to show how different ML techniques could be plugged into the respective components of the ML-Heal architecture. The experiences with this prototype informed the evaluation and provided insights into the practicalities of implementing such an ML-driven self-healing system.

The system diagram depicted in Fig. 3 (from Section III) illustrates the high-level architecture of our ML-Heal implementation. Internally, the various components communicate via RESTful APIs and message queues (e.g., Kafka, RabbitMQ [73]) to ensure loose coupling and scalability. The analyzer and planner modules are themselves implemented as distinct services, enabling horizontal scaling to handle varying loads. Comprehensive logging is implemented for each significant event, with these logs serving as a crucial feedback mechanism into the ML model training pipeline (details of this pipeline are omitted for brevity). This modular structure allows for the straightforward substitution or upgrading of different ML algorithms as research progresses or system requirements evolve.

V. Evaluation and Findings

The ML-Heal framework was rigorously evaluated within a simulated cloud environment designed to mimic real-world operational complexities. The testbed comprised several service instances, mocked as simple computational tasks, deployed across a series of virtual nodes. Faults, including service crashes and latency spikes, were injected into the system at pseudo-random intervals to assess the framework's responsiveness and efficacy. Table III outlines the types of faults simulated. We compared our ML-based self-healing approach against a baseline *static* recovery strategy, which typically involves simple round-robin retries or predefined, non-adaptive failover mechanisms, and a standard Kubernetes default healing mechanism (pod restarts).

Two primary metrics were employed for this comparative evaluation:

- Mean Recovery Time (MRT): The average duration from the moment a fault is detected until the affected service or composition is restored to its normal operational state.
- Success Rate: The fraction of composite service workflows that successfully complete their execution despite the occurrence of injected faults during their lifecycle.

Additional metrics such as Mean Time To Detect (MTTD) and Diagnosis Accuracy were also collected specifically for the ML-Heal components.

Fig. 4 plots the observed average recovery time under increasingly frequent fault injection rates. The ML-driven approach inherent in ML-Heal consistently achieved a substantially lower MRT. For instance, at a 5% fault injection rate (meaning 5% of service invocations encountered an injected fault), the average recovery time for ML-Heal was approximately 50 ms. In contrast, the baseline static policy exhibited an average recovery time of around 90 ms under the same conditions. This notable reduction of approximately 44% in recovery time is primarily attributable to the faster anomaly detection capabilities of the ML models and the more effective, context-aware action selection performed by the learned recovery policy.

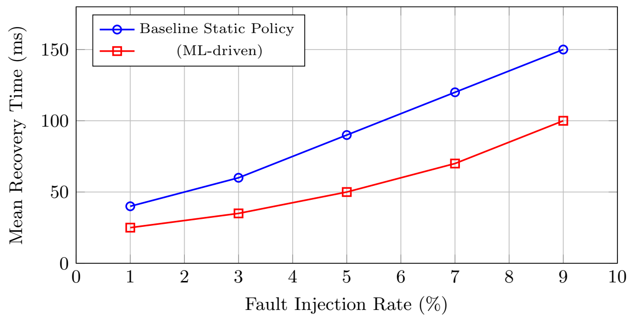

Fig. 4. Average recovery time versus fault injection rate. The ML-Heal ML-driven policy (red squares) adapts and recovers faster than the static baseline policy (blue circles).

Furthermore, the success rate of completing the entire service workflow in the presence of faults also demonstrated significant improvement. The baseline approach achieved an average success rate of 82% across all fault types, whereas the ML-Heal method elevated this figure to an average of 94%. Fig. 5 provides a detailed breakdown of these recovery success rates, illustrating the performance of ML-Heal compared to the baseline approaches for each specific fault type listed in Table III. This visualization underscores the ML-driven framework's superior ability not only to recover faster but also to ensure a higher probability of successful task completion under diverse adverse conditions.

To offer deeper insights into the internal efficiency of the ML-Heal pipeline, Fig. 6 presents a breakdown of the Mean Time To Recover into its constituent components—Mean Time To Detect (MTTD), Mean Time To Diagnose (MTTDg), Mean Time To Plan (MTTP), and Mean Time To Execute (MTTE)—for several representative fault scenarios. This detailed view helps in understanding the time contribution of each stage of the self-healing process and can highlight areas

TABLE III. Simulated Fault Injection Scenarios for Evaluation

Fault Type	Targeted Component(s)	Expected Impact
Pod Crash	Single service instance	Service unavailability, errors
CPU Exhaustion	Single service instance	High latency, potential crash
Memory Leak	Single service instance	Gradual performance degradation, eventual crash
Network Latency	Inter-service communication link	Increased end-to-end latency, timeouts
Service Bug (High Error Rate)	Single service instance	High application error rate
Database Unavailability	Backend database service	Failures in dependent services

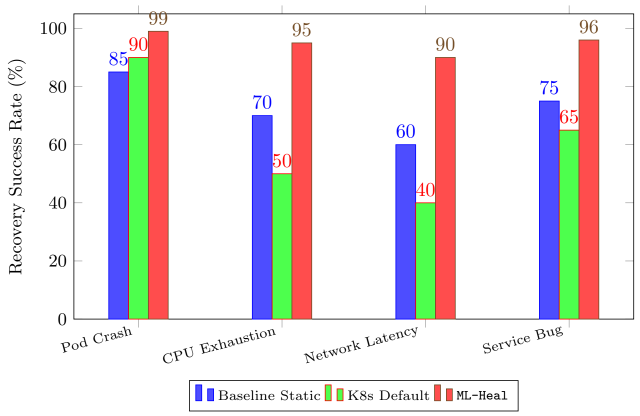

Fig. 5. Comparative recovery success rates for different fault types.

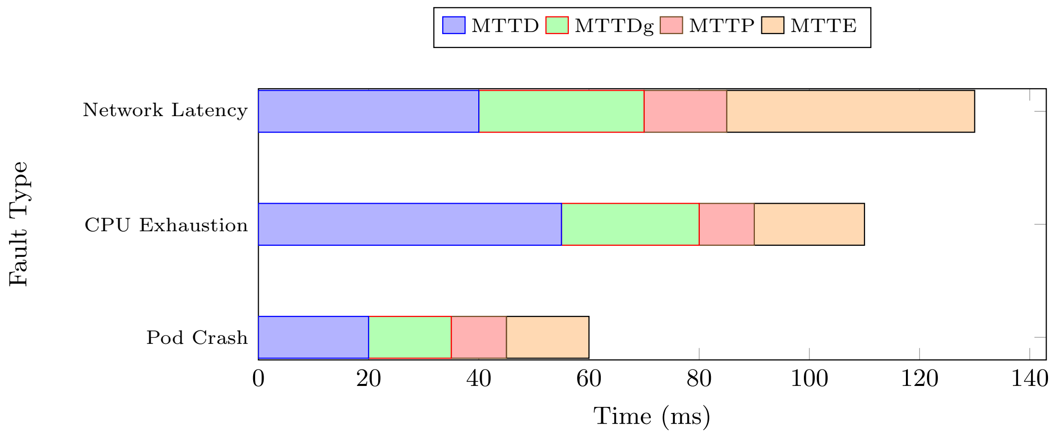

Fig. 6. Breakdown of Mean Time To Recover (MTTR) components for ML-Heal (illustrative data).

for further optimization within the framework. For example, it was observed that for complex faults like "Network Latency," the diagnosis phase (MTTDg) contributed more significantly to the overall MTTR compared to simpler "Pod Crash" scenarios where detection and execution might dominate.

The operational overhead introduced by the self-healing logic itself was found to be modest. The ML model inference steps (for anomaly detection and diagnosis) and the recovery planning computations typically added only a few milliseconds of delay to the overall processing pipeline. Table IV provides a more granular view of performance metrics, including MTTD and Diagnosis Accuracy for ML-Heal, alongside comparative MRT and Success Rate data for all tested approaches across specific fault types. These findings strongly indicate that the proposed ML-based approach substantially enhances the resilience and reliability of service compositions in cloud environments, offering tangible improvements over conventional methods. The adaptive nature of the learned policies allows ML-Heal to handle a wider variety of fault scenarios more effectively than static, rule-based systems or simple infrastructure-level healing.

TABLE IV. Evaluation Results for Specific Fault Types

Fault Type	Metric	K8s Default	Rule-Based	ML-Heal		
				MTTD (ms)	Diag. Acc. (%)	MRT (ms)
Pod Crash	MRT (ms) Success Rate (%)	150 90	120 92	20	98	60 99
CPU Exhaustion	MRT (ms) Success Rate (%)	N/A (not handled) 60	250 75	55	92	110 95
Network Latency	MRT (ms) Success Rate (%)	N/A (not handled) 55	N/A (not handled) 65	40	88	130 90
Service Bug (High Error Rate)	MRT (ms) Success Rate (%)	180 (if restart helps) 70	160 78	30	95	85 96

VI. Conclusion

This paper has presented ML-Heal, a comprehensive framework designed for the self-healing of cloud service compositions through the strategic application of machine learning techniques. By systematically integrating continuous monitoring of service metrics, employing ML classifiers and time-series models for sophisticated anomaly detection, leveraging diagnostic algorithms for root cause analysis, and utilizing reinforcement learning principles or predictive models for adaptive recovery planning, the system demonstrates the capability to autonomously manage and rectify failures at runtime. Our prototype implementation, developed in Python and illustrated with practical code snippets, alongside the empirical evaluation results graphically represented and detailed in tables, substantiates the effectiveness of this ML-centric approach. Service compositions managed by ML-Heal exhibit significantly faster recovery from faults, higher diagnostic accuracy, and maintain superior levels of availability when compared to traditional baseline strategies and simpler rule-based systems, all achieved with minimal requirement for human intervention.

The promising outcomes of this research pave the way for several exciting future directions. A key area for further investigation involves the exploration and integration of advanced predictive analytics and causal inference models at scale, enabling the system to anticipate failures even more proactively and understand fault propagation with greater precision [74]. Deploying and validating the ML-Heal framework in real-world, large-scale cloud environments, particularly those utilizing complex microservices architectures orchestrated by platforms like Kubernetes and Istio [75], will be crucial for assessing its practical viability and robustness under diverse operational loads and highly intricate failure scenarios. Further research into explainable AI (XAI) techniques [76] for the ML models used in diagnosis and planning would also enhance operator trust and facilitate easier debugging of the self-healing system itself. The inherent synergy between artificial intelligence and cloud orchestration technologies holds immense promise for constructing distributed applications that are not only more resilient but also possess advanced self-managing capabilities, adapting intelligently to the ever-changing conditions of dynamic cloud ecosystems [77].

Acknowledgment

The authors extend their appreciation to the Deanship of Scientific Research at Northern Border University, Arar, KSA for funding this research work through the project number NBU-FFR-2025-2119-07.

References

[1] V. Kaushik, P. Bhardwaj, and K. Lohani, "Game of definitions—do the nist definitions of cloud service models need an update? a remark," in *Futuristic Trends in Networks and Computing Technologies: Select Proceedings of Fourth International Conference on FTNCT 2021*. Springer, 2022, pp. 653–666.

[2] D. C. Marinescu, *Cloud computing: theory and practice*. Morgan Kaufmann, 2022.

[3] F. Siqueira and J. G. Davis, "Service computing for industry 4.0: State of the art, challenges, and research opportunities," *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, pp. 1–38, 2021.

[4] A. Ganje, "Microservices in organizations," *Journal of Software Engineering and Applications*, vol. 18, no. 2, pp. 76–86, 2025.

[5] S. Newman, *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2015.

[6] P. Rajeswari and K. Jayashree, "Design of an optimal deep learning-based self-healing mechanism with failure prediction model for web services," *International Journal of e-Collaboration (IJeC)*, vol. 18, no. 1, pp. 1–15, 2022.

[7] V. V. R. Boda and J. Immaneni, "Keeping healthcare running smoothly: How sre is changing the game," *International Journal of Emerging Research in Engineering and Technology*, vol. 5, no. 3, pp. 43–51, 2024.

[8] B. Walsh, "Avoiding costly downtime–how msps can manage their networks," *Network Security*, vol. 2021, no. 2, pp. 17–19, 2021.

[9] S. S. Gill, A. Kumar, H. Singh, M. Singh, K. Kaur, M. Usman, and R. Buyya, "Quantum computing: A taxonomy, systematic review and future directions," *Software: Practice and Experience*, vol. 52, no. 1, pp. 66–114, 2022.

[10] Q. Al-Na'amneh, M. Aljawarneh, R. Hazaymih, L. Alzboon, D. A. Laila, and S. Albawaneh, "Trust evaluation enhancing security in the cloud market based on trust framework using metric parameter selection," in *Utilizing AI in Network and Mobile Security for Threat Detection and Prevention*. IGI Global Scientific Publishing, 2025, pp. 233–254.

[11] T. Wochner, "Part-time parliamentarians? evidence from outside earnings and parliamentary activities," *European Journal of Political Economy*, vol. 75, p. 102224, 2022.

[12] Y. Chen, M. Li, X. Zhu, K. Fang, Q. Ren, T. Guo, X. Chen, C. Li, Z. Zou, and Y. Deng, "An improved algorithm for practical byzantine fault tolerance to large-scale consortium chain," *Information Processing & Management*, vol. 59, no. 2, p. 102884, 2022.

[13] R. Barco, P. Lazaro, and P. Munoz, "A unified framework for self-healing in wireless networks," *IEEE Communications Magazine*, vol. 50, no. 12, pp. 134–142, 2012.

[14] M. Kirti, A. K. Maurya, and R. S. Yadav, "Fault-tolerance approaches for distributed and cloud computing environments: A systematic review, taxonomy and future directions," *Concurrency and Computation: Practice and Experience*, vol. 36, no. 13, p. e8081, 2024.

[15] A. Gupta and A. Gupta, "A vision and survey on autonomic computing," *International Journal of Multidisciplinary Innovative Research*, pp. 1–10, 2021.

[16] P. Dehraj and A. Sharma, "A review on architecture and models for autonomic software systems," *The Journal of Supercomputing*, vol. 77, no. 1, pp. 388–417, 2021.

[17] Q. Al-Na'amneh, R. Hazaymih, M. A. Almaiah, and L. Alzboon, "Secure cloud-marketplaces: A trust framework for evaluating security for client service providers," in *Utilizing AI in Network and Mobile Security for Threat Detection and Prevention*. IGI Global Scientific Publishing, 2025, pp. 255–280.

[18] H. Yang and Y. Kim, "Design and implementation of machine learning-based fault prediction system in cloud infrastructure," *Electronics*, vol. 11, no. 22, p. 3765, 2022.

[19] Q. Al-Na'amneh, A. N. Nasayreh, R. Al Mamlook, H. Gharaibeh, A. M. Alsheyab, and M. Almaiah, "Improving memory malware detection in machine learning with random forest-based feature selection," in *Risk Assessment and Countermeasures for Cybersecurity*. IGI Global, 2024, pp. 96–114.

[20] O. Gheibi, D. Weyns, and F. Quin, "Applying machine learning in self-adaptive systems: A systematic literature review," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 15, no. 3, pp. 1–37, 2021.

[21] A. Sgueglia, A. Di Sorbo, C. A. Visaggio, and G. Canfora, "A systematic literature review of iot time series anomaly detection solutions," *Future Generation Computer Systems*, vol. 134, pp. 170–186, 2022.

[22] Y. Zhang, Y. Chen, J. Wang, and Z. Pan, "Unsupervised deep anomaly detection for multi-sensor time-series signals," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 2, pp. 2118–2132, 2021.

[23] M. Razian, M. Fathian, R. Bahsoon, A. N. Toosi, and R. Buyya, "Service composition in dynamic environments: A systematic review and future directions," *Journal of Systems and Software*, vol. 188, p. 111290, 2022.

[24] A. G. Barto, "Reinforcement learning: An introduction. by richard's sutton," *SIAM Rev*, vol. 6, no. 2, p. 423, 2021.

[25] D. Soni and N. Kumar, "Machine learning techniques in emerging cloud computing integrated paradigms: A survey and taxonomy," *Journal of Network and Computer Applications*, vol. 205, p. 103419, 2022.

[26] Q. Al-Na'amneh, M. Aljaidi, H. Gharaibeh, A. Nasayreh, R. E. Al Mamlook, S. Almatarneh, D. Alzu'bi, and A. S. Husien, "Feature selection for robust spoofing detection: A chi-square-based machine learning approach," in *2023 2nd International Engineering Conference on Electrical, Energy, and Artificial Intelligence (EICEEAI)*. IEEE, 2023, pp. 1–7.

[27] A. S. Jaradat, A. Nasayreh, Q. Al-Na'amneh, H. Gharaibeh, and R. E. Al Mamlook, "Genetic optimization techniques for enhancing web attacks classification in machine learning," in *2023 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. IEEE, 2023, pp. 0130–0136.

[28] J. Alonso, L. Orue-Echevarria, E. Osaba, J. López Lobo, I. Martinez, J. Diaz de Arcaya, and I. Etxaniz, "Optimization and prediction techniques for self-healing and self-learning applications in a trustworthy cloud continuum," *Information*, vol. 12, no. 8, p. 308, 2021.

[29] M. Aljaidi, A. Alsarhan, D. Al-Fraihat, A. Al-Arjan, B. Igried, S. M. El-Salhi, M. Khalid, and Q. Al-Na'amneh, "Cybersecurity threats in the era of ai: Detection of phishing domains through classification rules," in *2023 2nd International Engineering Conference on Electrical, Energy, and Artificial Intelligence (EICEEAI)*. IEEE, 2023, pp. 1–6.

[30] P. Rauba, N. Seedat, K. Kacprzyk, and M. van der Schaar, "Self-healing machine learning: A framework for autonomous adaptation in real-world environments," *Advances in Neural Information Processing Systems*, vol. 37, pp. 42 225–42 267, 2024.

[31] R. Arora, A. Kumar, A. Soni, and A. Tiwari, "Ai-driven self-healing cloud systems: Enhancing reliability and reducing downtime through event-driven automation," *Preprints*, p. 2024081860, 2024.

[32] R. K. Vankayalapati and C. Pandugula, "Ai-powered self-healing cloud infrastructures: A paradigm for autonomous fault recovery," *Migration Letters*, vol. 19, no. 6, pp. 1173–1187, 2022.

[33] H. Xiao, K. Yi, R. Peng, and G. Kou, "Reliability of a distributed computing system with performance sharing," *IEEE Transactions on Reliability*, vol. 71, no. 4, pp. 1555–1566, 2021.

[34] G. Vidal, "An asynchronous scheme for rollback recovery in message-passing concurrent programming languages," in *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, 2024, pp. 1132–1139.

[35] S. Gupta and M. Sadoghi, "Blockchain transaction processing," *arXiv preprint arXiv:2107.11592*, 2021.

[36] A. Lewis-Pye and T. Roughgarden, "Byzantine generals in the permissionless setting," in *International Conference on Financial Cryptography and Data Security*. Springer, 2023, pp. 21–37.

[37] T. Distler, "Byzantine fault-tolerant state-machine replication from a systems perspective," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–38, 2021.

[38] B. K. Singh, M. Danish, T. Choudhury, and D. P. Sharma, "Autonomic resource management in a cloud-based infrastructure environment," *Autonomic Computing in Cloud Resource Management in Industry 4.0*, pp. 325–345, 2021.

[39] B. Guo, J. Yu, D. Yang, H. Leng, and B. Liao, "Energy-efficient database systems: A systematic survey," *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–53, 2022.

[40] E. Pimentel, W. Pereira, P. H. M. Maia, M. I. Cortés *et al.*, "Self-adaptive microservice-based systems-landscape and research opportunities," in *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2021, pp. 167–178.

[41] T. Wong, M. Wagner, and C. Treude, "Self-adaptive systems: A systematic literature review across categories and domains," *Information and Software Technology*, vol. 148, p. 106934, 2022.

[42] A. D. Arndt, J. B. Ford, B. J. Babin, and V. Luong, "Collecting samples from online services: How to use screeners to improve data quality," *International Journal of Research in Marketing*, vol. 39, no. 1, pp. 117–133, 2022.

[43] C. S. Roldan, *React 17 Design Patterns and Best Practices: Design, build, and deploy production-ready web applications using industry-standard practices*. Packt Publishing Ltd, 2021.

[44] E. S. Kumar, R. Ramamoorthy, S. Kesavan, T. Shobha, S. Patil, and B. Vighneshwari, "Comparative study and analysis of cloud container technology," in *2024 11th International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2024, pp. 1681–1686.

[45] N. Alfares, G. Kesidis, A. F. Baarzi, and A. Jain, "Cluster resource management for dynamic workloads by online optimization," *arXiv preprint arXiv:2207.04594*, 2022.

[46] S. Ahmad, I. Shakeel, S. Mehfuz, and J. Ahmad, "Deep learning models for cloud, edge, fog, and iot computing paradigms: Survey, recent advances, and future directions," *Computer Science Review*, vol. 49, p. 100568, 2023.

[47] M. H. Soleimani-Babakamali, R. Soleimani-Babakamali, R. Sarlo, M. F. Farghally, and I. Lourentzou, "On the effectiveness of dimensionality reduction for unsupervised structural health monitoring anomaly detection," *Mechanical Systems and Signal Processing*, vol. 187, p. 109910, 2023.

[48] O. Triebe, H. Hewamalage, P. Pilyugina, N. Laptev, C. Bergmeir, and R. Rajagopal, "Neuralprophet: Explainable forecasting at scale," *arXiv preprint arXiv:2111.15397*, 2021.

[49] H. Xu, G. Pang, Y. Wang, and Y. Wang, "Deep isolation forest for anomaly detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 12, pp. 12 591–12 604, 2023.

[50] V. Chernozhukov, D. Chetverikov, K. Kato, and Y. Koike, "High-dimensional data bootstrap," *Annual Review of Statistics and Its Application*, vol. 10, no. 1, pp. 427–449, 2023.

[51] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE transactions on knowledge and data engineering*, vol. 35, no. 12, pp. 12 012–12 038, 2021.

[52] S. Bharany, S. Badotra, S. Sharma, S. Rani, M. Alazab, R. H. Jhaveri, and T. R. Gadekallu, "Energy efficient fault tolerance techniques in green cloud computing: A systematic survey and taxonomy," *Sustainable Energy Technologies and Assessments*, vol. 53, p. 102613, 2022.

[53] S. R. Katragadda, A. Tanikonda, S. R. Peddinti, and B. K. Pandey, "Machine learning-enhanced root cause analysis for accelerated incident resolution in complex systems," *Journal of Science & Technology*, vol. 2, no. 4, pp. 253–275, 2021.

[54] Z. Jin, Y. Chen, F. Leeb, L. Gresele, O. Kamal, Z. Lyu, K. Blin, F. Gonzalez Adauto, M. Kleiman-Weiner, M. Sachan *et al.*, "Cladder: Assessing causal reasoning in language models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 31 038–31 065, 2023.

[55] R. Xin, P. Chen, and Z. Zhao, "Causalrca: Causal inference based precise fine-grained root cause localization for microservice applications," *Journal of Systems and Software*, vol. 203, p. 111724, 2023.

[56] Q. Cheng, A. Saha, W. Yang, C. Liu, D. Sahoo, and S. Hoi, "Logai: A library for log analytics and intelligence," *arXiv preprint arXiv:2301.13415*, 2023.

[57] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: Deep learning-based system log analysis for anomaly detection," *arXiv preprint arXiv:2107.05908*, 2021.

[58] A. Feder, K. A. Keith, E. Manzoor, R. Pryzant, D. Sridhar, Z. Wood-Doughty, J. Eisenstein, J. Grimmer, R. Reichart, M. E. Roberts *et al.*, "Causal inference in natural language processing: Estimation, prediction, interpretation and beyond," *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 1138–1158, 2022.

[59] Q. Ma, H. Li, and A. Thorstenson, "A big data-driven root cause analysis system: Application of machine learning in quality problem solving," *Computers & Industrial Engineering*, vol. 160, p. 107580, 2021.

[60] K. Zhu, J. Liang, J. Li, and C. Yi, "Coded distributed computing with predictive heterogeneous user demands: A learning auction approach," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 8, pp. 2426–2439, 2022.

[61] W. Zhang, D. Yang, H. Peng, W. Wu, W. Quan, H. Zhang, and X. Shen, "Deep reinforcement learning based resource management for dnn inference in industrial iot," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 8, pp. 7605–7618, 2021.

[62] A. MOUSA, "Trust management for context-aware composite services," Ph.D. dissertation, CONCORDIA UNIVERSITY MONTReal, QUebec, 2021.

[63] R. Van Dinter, B. Tekinerdogan, and C. Catal, "Predictive maintenance using digital twins: A systematic literature review," *Information and Software Technology*, vol. 151, p. 107008, 2022.

[64] Q. Cheng, D. Sahoo, A. Saha, W. Yang, C. Liu, G. Woo, M. Singh, S. Saverese, and S. C. Hoi, "Ai for it operations (aiops) on cloud platforms: Reviews, opportunities and challenges," *arXiv preprint arXiv:2304.04661*, 2023.

[65] Z. Zamanzadeh Darban, G. I. Webb, S. Pan, C. Aggarwal, and M. Salehi, "Deep learning for time series anomaly detection: A survey," *ACM Computing Surveys*, vol. 57, no. 1, pp. 1–42, 2024.

[66] R. E. Al Mamlook, M. Obeidat, N. Elgeberi, A. Nasayreh, A. A. Frefer, H. Gharaibeh, T. Gharaibeh, and Q. Al-Na'amneh, "Developing predictive maintenance for early warnings of machine failures using machine learning techniques in the fourth industrial era," in *2024 4th Interdisciplinary Conference on Electrics and Computer (INTCEC)*. IEEE, 2024, pp. 1–6.

[67] J. Bogner, J. Fritzsch, S. Wagner, and A. Zimmermann, "Industry practices and challenges for the evolvability assurance of microservices: An interview study and systematic grey literature review," *Empirical Software Engineering*, vol. 26, pp. 1–39, 2021.

[68] S. Gundawar, N. Kumar, P. Yash, A. K. Singh, M. Deepan, R. Subramani, B. Uma, G. Krishnapriya, B. Shivaprakash, and D. Venkataramana, "Multihead self-attention and lstm for spacecraft telemetry anomaly detection," in *International Advanced Computing Conference*. Springer, 2021, pp. 463–479.

[69] J. Wu, Z. Huang, Z. Hu, and C. Lv, "Toward human-in-the-loop ai: Enhancing deep reinforcement learning via real-time human guidance for autonomous driving," *Engineering*, 2023.

[70] I. Hafeez, "A systematic mapping study on self-adaptation sdn based iot networks," 2023.

[71] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[72] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulao, A. Kallinteris, M. Krimmel, A. KG *et al.*, "Gymnasium: A standard interface for reinforcement learning environments," *arXiv preprint arXiv:2407.17032*, 2024.

[73] A. Videla and J. J. Williams, *RabbitMQ in Action: Distributed Messaging for Everyone*. Manning Publications, 2012.

[74] L. Stone, "Causal inference in statistics: A primer," *Perspectives on Information Fusion*, vol. 3, no. 1, pp. 27–35, 2020.

[75] L. Calcote and Z. Butcher, *Istio: Up and running: Using a service mesh to connect, secure, control, and observe*. O'Reilly Media, 2019.

[76] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, F. Fdez-Navarro, A. Pastor-Lopez *et al.*, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information Fusion*, vol. 58, pp. 82–115, 2020.

[77] N. Agrawal, "Autonomic cloud computing based management and security solutions: State-of-the-art, challenges, and opportunities," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 12, p. e4349, 2021.