Computer Standards & Interfaces 97 (2026) 104116

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

Chaos experiments in microservice architectures: A systematic literature review

Emrah Esen [a], Akhan Akbulut [a], Cagatay Catal [b,*]

[a] Department of Computer Engineering, Istanbul Kültür University, 34536, Istanbul, Turkey
[b] Department of Computer Science and Engineering, Qatar University, Doha 2713, Qatar

ARTICLE INFO

Keywords:
Chaos engineering
Microservice
Systematic literature review

ABSTRACT

This study analyzes the implementation of Chaos Engineering in modern microservice systems. It identifies key methods, tools, and practices used to effectively enhance the resilience of software systems in production environments. In this context, our Systematic Literature Review (SLR) of 31 research articles has uncovered 38 tools crucial for carrying out fault injection methods, including several tools such as Chaos Toolkit, Gremlin, and Chaos Machine. The study also explores the platforms used for chaos experiments and how centralized management of chaos engineering can facilitate the coordination of these experiments across complex systems. The evaluated literature reveals the efficacy of chaos engineering in improving fault tolerance and robustness of software systems, particularly those based on microservice architectures. The paper underlines the importance of careful planning and execution in implementing chaos engineering and encourages further research in this field to uncover more effective practices for the resilience improvement of microservice systems.

Contents

* Corresponding author.
E-mail address: ccatal@qu.edu.qa (C. Catal).

https://doi.org/10.1016/j.csi.2025.104116
Received 22 September 2024; Received in revised form 28 November 2025; Accepted 12 December 2025
Available online 15 December 2025

1. Introduction

In recent years, the adoption of microservice architecture has led to the transformation of application infrastructures into distributed systems. These systems are designed to enhance maintainability by decoupling services. The primary benefit of this architecture is the ease of maintenance of individual services within the microservice ecosystem due to their smaller and more modular nature [1]. However, despite these advantages, the distributed nature of microservices introduces significant challenges. Specifically, the complex management of services and their tight integration can considerably complicate software debugging. Debugging becomes complex in this architecture due to its distributed nature, the necessity to pinpoint the exact service causing the problem, and the dynamic characteristics of microservices. Consequently, debugging in microservice architecture demands a greater level of effort and specialized expertise compared to conventional monolithic architectures [2]. However, it becomes quite challenging to predict what will happen if there is an unexpected error or if a service on the network goes out of service. Service outages can be caused by anything from a malicious cyberattack to a hardware failure to simple human error, and they can have devastating financial consequences. Although such unexpected situations are rare, they can interfere with the operation of distributed systems and devastatingly affect the live environment in which the application is located [3]. It is necessary to detect points in the system before an error occurs and spreads to the entire system.

Microservice architecture applications undergo testing procedures to ensure their quality and dependability. These include unit testing, service test, end-to-end test, behavior-driven test, integration test, and regression test [4]. The comprehensive approach to microservices testing also encompasses live testing strategies for complex systems [5]. This thorough process emphasizes different aspects such as functionality, interoperability, performance of individual services within the architecture. It aims to detect and resolve issues early to ensure stable and high-quality microservice applications [1,6]. However, considering that microservices consist of multiple services, the application should not have an impact on the user experience in cases such as network failures and suddenly increased service loads. For example, if the microservice that adds the product to favorites on a shopping site fails or responds late, the user should be able to continue the shopping experience. Therefore, testing operations in production-like environments become inevitable. No matter how distributed or complex the system is, there is a need for a method to manage unforeseeable situations that can build trust in the system against unexpected failures. chaos engineering is defined as the discipline of conducting experiments in a live environment to test or verify the reliability of software [7].

The primary objective of this research is to conduct a thorough investigation into how chaos experiments are performed in the widely used microservices-based systems of today. Microservice architectures have come to the forefront in modern software development processes due to their advantages such as flexibility, scalability, and rapid development. However, these architectures also bring unique challenges due to complex service dependencies and dynamic operational environments. This study aims to comprehensively address the methodologies, application scenarios, and impacts of chaos experiments conducted to test the resilience of microservice systems and identify potential weak points. The research intends to present the current state of chaos engineering practices by analyzing them, highlighting best practices, challenges faced, and solutions. In addition, it will assess the effectiveness of chaos experiments in enhancing the reliability and robustness of microservice systems by using data obtained from real-world scenarios to develop strategic recommendations. This study is a critical step in understanding the applicability and impact of chaos engineering within the complexity of microservice architectures and aims to make significant contributions to the body of knowledge in this field. Recent research has applied chaos engineering for this architectural style, however, a systematic overview of the state-of-the-art on the use of chaos engineering in the microservice architecture is lacking. Therefore, a Systematic Literature Review (SLR) has been performed to provide an overview of how chaos engineering was applied.

This article primarily targets peer-reviewed research papers to maintain methodological consistency and ensure scholarly rigor. We specifically chose a systematic literature review (SLR) methodology because peer-reviewed academic studies are subject to rigorous validation processes, which enhance the reliability and validity of our findings [8,9]. Although excluding industry-specific, grey literature may restrict certain practical perspectives, this choice was deliberately made to avoid potential biases and uphold the scientific integrity of our review [10,11]. However, future studies could broaden the scope to incorporate industrial case studies and practical experiences, which would enrich our understanding of chaos engineering's applicability beyond the academic context.

The main contributions of this study are listed as follows:

1. To the best of our knowledge, this is the first study to employ a systematic literature review approach in the field of chaos engineering on microservice architecture applications [12]. The study provides an extensive systematic literature review of how chaos engineering can be applied to enhance the resilience of microservice architectures. It collates findings from various sources to provide insights into the current state of research and practice in this field.
2. The study categorizes and summarizes the range of chaos engineering tools and methods used in industry and academia, highlighting their functionalities in process/service termination, network simulation, load stressing, security testing, and fault injection within application code.
3. This research paper discusses contemporary techniques and approaches for implementing chaos engineering in microservice architectures. It also emphasizes the ongoing work in this field, offering a significant reference for future research endeavors. The paper systematically reviews existing literature to showcase how chaos engineering can enhance system resilience, laying a comprehensive groundwork for further exploration into chaos experimentation strategies and innovating new fault injection methods or tools within microservice architectures.

The rest of the paper is structured as follows: Section 2 explains the background and related work. Section 3 presents the methodology of the research. Section 4 presents the results and Section 5 comprehensively discusses the presented answers to research questions and validity threats. Lastly, the conclusion is presented in Section 6.

2. Background

The microservice approach breaks down a large application into a network of small, self-contained units, each running its own process and often communicating through web APIs. Unlike large, single-piece

monolithic systems, these small services are robust, easy to scale up or down, and can be updated individually using various programming languages and technologies. This structure allows development teams to be smaller and more agile, leading to faster updates and improvements. Yet, managing many interconnected services can become complicated, especially when something goes wrong. To enhance system reliability and resilience, a method known as chaos engineering is employed. This involves deliberately introducing problems into the live system to test its ability to cope and recover. This technique helps to uncover and rectify flaws, thereby making the system stronger overall. Regular and automated tests mimic real-life problems to ensure that the system can handle unexpected challenges and remain stable and efficient.

2.1. Microservice architecture

Microservice architectures have gained significant popularity in the software industry due to their ability to address the challenges and complexities of developing modern applications [6,13].

2.2. Microservice principles

Microservice architectures are based on the concept of decentralization, where each service is independently developed, deployed, and managed. This emphasizes autonomy and minimal inter-service dependencies. Each microservice is designed to focus on a single function or closely related set of functions and supports technology heterogeneity by allowing different services to use different technology stacks that best suit their needs. Resilience is a core aspect, with services built to withstand failures without affecting the entire system while scalability enables services to be scaled independently as per demand. Communication occurs through lightweight mechanisms like HTTP/REST APIs, supporting continuous delivery and deployment practices. Due to the distributed nature of microservice architecture, comprehensive monitoring and logging for observability becomes crucial. Additionally, there is often an alignment between the microservice architecture and organizational structure involving small cross-functional teams responsible for individual services [14].

It is helpful to compare the microservice architecture to the monolithic architecture. The main difference between them is the dimensions of the developed applications. The microservice architecture can be thought of as developing an application as a suite of smaller services, rather than as a single, monolithic structure. Enterprise applications usually consist of three main parts: a client-side user interface (i.e., containing HTML pages and Javascript running on the user's machine in a browser), a database (i.e., composed of many tables, common and often relational, added to database management), and a server-side application. In the server-side application, HTTP requests are processed, business logic is executed, HTML views are prepared that will retrieve data from the database and update it and send it to the browser. This structure is a good example of monoliths. Any changes to the system involve creating and deploying a new version of the server-side application [15]. The cycles of change are interdependent. A change to a small part of the application requires rebuilding and deploying the entire monolith [6].

Microservice architecture, on the other hand, has some common features, unlike monolithic architecture. These are componentization with services, organizing around job capabilities, smart interfaces and simple communication, decentralized governance, decentralized data management, infrastructure automation, and design for failure [16]. Today, although modern internet applications seem like a single application, they use microservice architectures behind them. Microservice architecture basically refers to small autonomous and interoperability services. It has emerged due to increasing needs such as technology diversity, flexibility, scaling, ease of deployment, organization and management, and provides various advantages in these matters. Its advantages are described as follows [17]:

Technology heterogeneity. They are treated as small services, each running independently and communicating with each other using open protocols. While monolithic applications are developed with a single programming language and database system, services included in a microservice ecosystem may use a different programming language and database. This allows the advantages of each programming language and database to be used.

Resilience. When an error occurs in the system in monolithic applications, the whole system is affected. In the microservice architecture, only the part under the responsibility of the relevant service is affected, the places belonging to other services are not affected and the user experience continues.

Scalability. While the scaling process on monolithic applications covers the entire application, the services that are under heavy load can be scaled in applications developed with microservice architecture. This prevents extra resource costs for partitions that do not need to be scaled unnecessarily and increases the user experience.

Deployment. Microservice architecture facilitates the autonomous deployment of individual services, enabling updates or changes without impacting others. Various deployment strategies, including blue–green, canary, and rolling deployment, minimize disruptions during the deployment process [18]. As a result, microservice architecture provides increased flexibility and resilience in deployment, distinguishing it from monolithic applications.

Organizational alignment. In software development processes, some challenges may be encountered due to large teamwork and large pieces of code. It is possible to make these challenges more manageable with smaller teams established. At the same time, this is an indication that microservices applications allow us to form smaller and more cohesive teams. Each team is responsible for its own microservice and can take action by making improvements if necessary.

2.3. Challenges/Troubleshooting/Failures in microservice architecture

Microservice architectures pose numerous challenges. As the number of services increases, the complexity of service interactions also grows. Network communication reliance leads to latency and network failure issues, while ensuring data consistency across multiple databases requires careful design and implementation of distributed transactions or eventual consistency models. Microservices bring typical distributed system challenges such as handling partial failures, dealing with latency and asynchrony, complex service discovery, load balancing in dynamic scaling environments, and managing configurations across multiple services and environments. Security concerns are heightened due to increased inter-service communications surface area. Testing becomes more complex involving individual service testing along with testing their interactions; deployment is challenging especially when there are dependencies between services; effective observability and monitoring become crucial for timely issue resolution; versioning management is critical for maintaining system stability; lastly assembling skilled teams proficient in DevOps, cloud computing, programming languages presents a significant challenge. Microservice architecture faces various challenges, troubleshooting, and failures. While adopting a distributed architecture enhances modularity, it inherently introduces operational complexities that differ significantly from monolithic structures. Recent research has also explored the use of hybrid bio-inspired algorithms to optimize this process dynamically. For instance, the Hybrid Kookaburra–Pelican Optimization Algorithm has been shown to improve load distribution and system scalability in cloud and microservice-based environments [19].

In conclusion, while microservices offer numerous advantages such as improved scalability, flexibility, and agility, they also introduce significant challenges in terms of system complexity, operational demands, and the need for skilled personnel and sophisticated tooling [20].

2.4. Chaos engineering

"Chaos engineering is the discipline of experimenting on a distributed system in order to build confidence in the system's capability to withstand turbulent conditions in production-like environment" [7,21]. It is the careful and planned execution of experiments to show how the distributed system will respond to a failure. It is necessary for large-scale software systems because it is practically impossible to simulate real events in test environments. Experiments based on real events are created together with chaos engineering [22]. By analyzing the test results, improvements are made where necessary, and in this way, it is aimed to increase the reliability of the software in the production environment.

Thanks to an experimental and systems-based approach, confidence is established for the survivability of these systems during collapses. Canary analysis collects data on how distributed systems react to failure scenarios by observing their behavior in abnormal situations and performing controlled experiments [23]. This method involves applying new updates or changes to a specific aspect of the system, enabling early detection of potential problems before they affect a larger scale.

Chaos experiments consist of the following principles [24,25]:

- Hypothesize steady state: The first step is to hypothesize the steady state of the system under normal conditions.
- Vary real-world events: The next step is to vary real-world events that can cause turbulence in the system.
- Run experiments in production: Experimenters should run the experiments in production-like environment to simulate real-world conditions.
- Automate experiments to run continuously: Experimenters should automate the experiments to run continuously, ensuring that the system can withstand turbulence over time.
- Minimize blast radius: The experiments should be designed to minimize blast radius, i.e., the impact of the experiment on the system should be limited to a small area
- Analyze results: Experimenters should analyze the results of the experiments to determine the system's behavior under turbulent conditions.
- Repeat experiments: The experiments should be repeated to ensure that the system can consistently withstand turbulence.
  When the experiment is finished, information about the actual effect will be provided to the system.

3. Review protocol

Systematic review studies must be conducted using a well-defined and specific protocol. To conduct a systematic review study, all studies on a particular topic must be examined [12]. We followed the systematic review process shown in Fig. 1 and took all the steps to reduce risk bias in this study. Multiple reviewers were involved in the SLR process, and in cases of conflict, a brief meeting was organized to facilitate consensus. The first step is to define the research questions. Then, the most appropriate databases were selected. Based on the selected databases, automated searches were conducted and several articles were identified. Selection criteria were then established to determine which studies should be included and excluded in this research. The titles and abstracts of all studies were reviewed. In cases of doubt, the full text of the publication was reviewed. Then, after the studies were analyzed in detail, selection criteria were applied. All selected studies were assessed using a quality assessment process. Subsequently, the results were synthesized, listed, and summarized in a clear and understandable manner.

3.1. Research questions

Research Questions (RQs) and their corresponding motivations are presented as follows:

- RQ1: How is Chaos engineering effectively applied in production environments to enhance the resilience of software systems?
  Motivation: Understanding the practical implementation of Chaos engineering in production environments is crucial for ensuring the resilience of software systems under real-world operating conditions.
- RQ2: Which platforms have been used for Chaos experiments?
  Motivation: Identifying the platforms provides insights into the technological landscape and tools available for conducting Chaos engineering practices.
- RQ3: How is Chaos engineering effectively applied to microservice architectures to ensure its successful implementation in enhancing system resilience?
  Motivation: Microservice architectures introduce new challenges in system design. Exploring the application of Chaos engineering in this context can help improve the resilience and fault tolerance of microservice systems.
- RQ4: To what extent can the centralized provision of Chaos engineering effectively facilitate the management of Chaos experiments across complex systems?
  Motivation: Understanding the feasibility of providing Chaos engineering as a centralized service enables organizations to coordinate Chaos experiments across complex systems.
- RQ5: What are the challenges reported in the relevant papers?
  Motivation: Identifying these challenges provides valuable insights into overcoming obstacles and advancing the adoption of Chaos engineering practices.

3.2. Search strategy

The primary studies were carefully selected from the papers published between 2010 and 2022 because the topic is only relevant in recent years. The databases are IEEE Xplore, ACM Digital Library, Science Direct, Springer, Wiley, MDPI and Scopus and Science Direct. The initial search involved reviewing the titles, abstracts, and keywords of the studies identified in the databases. The search results obtained from the databases were stored in the data extraction form using a spreadsheet tool. Furthermore, this systematic review was conducted collaboratively by three authors.

The following search string was used to broaden the search scope: ((chaos engineering) OR (chaos experiments)) OR (microservices)

The results of the searches made in the databases mentioned above are shown in Fig. 2.

3.3. Study selection criteria

After applying exclusion inclusion criteria, 55 articles were obtained. The exclusion criteria in our study are shown as follows:

- EC-1: Duplicate papers from multiple sources
- EC-2: Papers without full-text availability
- EC-3: Papers not written in English
- EC-4: Survey papers
- EC-5: Papers not related to Chaos engineering

The inclusion criteria in our study are shown as follows:

- IC-1: Primary papers discussing the use of Chaos experiments in a microservice architecture
- IC-2: Primary publications that focus on Chaos engineering

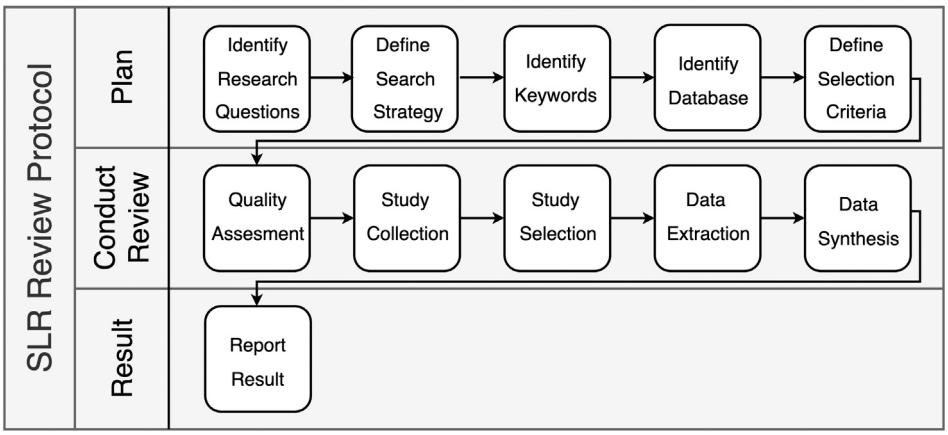

Fig. 1. SLR review protocol.
Source: Adapted from [26–28].

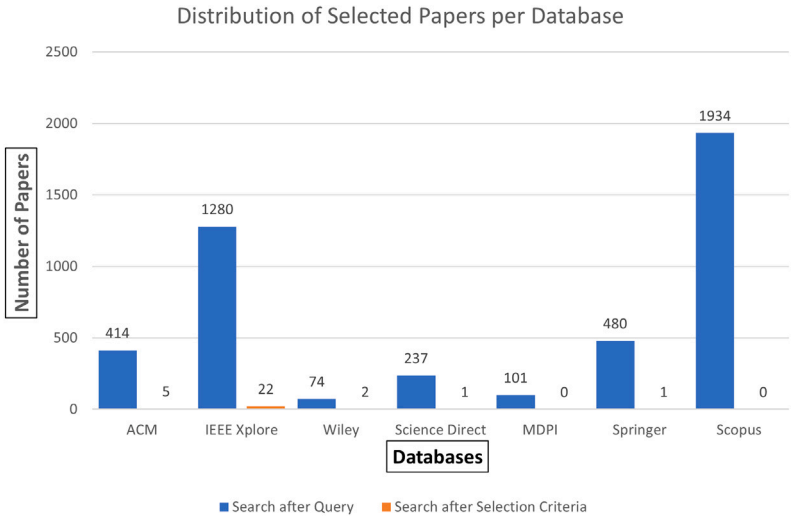

Fig. 2. Distribution of selected papers per database.

3.4. Study quality assessment

The assessment of each study's quality is an indicator of the strength of evidence provided by the systematic review. The quality of studies was assessed using various questions. Studies of poor quality were not included in the present study. These criteria based on quality instruments were adopted guide and other SLRs research [12]. The following questions were used to assess the quality of the studies.

- Q1. Are the aims of the study clearly stated?
- Q2. Are the scope and experimental design of the study clearly defined?
- Q3. Is the research process documented adequately?
- Q4. Are all the study questions answered?
- Q5. Are the negative findings presented?
- Q6. Do the conclusions relate to the aim of the purpose of the study and are they reliable?

In this study, considering all these criteria, a general quality assessment was performed for each paper. The rating was 2 points for the "yes" option, 0 points for the "no" option, and 1 point for the "somewhat" option. The decision threshold for classifying the paper as poor quality was determined based on the mean value, which corresponds to a total of 5 points.

Fig. 2 presents the distribution of papers based on databases where they were found at different selection stages. After the initial search, 4520 papers were retrieved, of which 55 remained after applying the selection criteria. After quality assessment, 31 papers were selected as primary studies. The 55 papers were carefully read in full and the required data for answering the research questions were extracted.

All the collected articles are listed in Table 1.

3.5. Data extraction

Data required for answering the Research Questions were extracted from the selected articles to answer the research questions. A data extraction form was created to answer the research questions. The data extraction form consists of several metadata such as the author's first and last name, the title of the study, the publication year, and the type of study. In addition to this metadata, several columns were created to store the required information related to the research questions. By employing a data extraction form, we ensured that the relevant data required to answer each research question were systematically captured from the selected publications. This approach facilitated the subsequent synthesis of the findings. The data extraction process involved meticulous attention to detail and ensured the reliability and integrity of the data used in our systematic literature review.

Table 1
Selected primary studies.

ID	Reference	Title	Year	Database
S1	[29]	Automating Chaos Experiments in Production	2019	ACM
S2	[25]	Getting Started with Chaos engineering—design of an implementation framework in practice	2020	ACM
S3	[30]	Human-AI Partnerships for Chaos engineering	2020	ACM
S4	[31]	3MileBeach: A Tracer with Teeth	2021	ACM
S5	[32]	Service-Level Fault Injection Testing	2021	ACM
S6	[33]	A Platform for Automating Chaos Experiments	2016	IEEE Xplore
S7	[34]	Automated Fault-Tolerance Testing	2016	IEEE Xplore
S8	[35]	Gremlin: Systematic Resilience Testing of Microservices	2016	IEEE Xplore
S9	[36]	Fault Injection Techniques - A Brief Review	2018	IEEE Xplore
S10	[37]	ORCAS: Efficient Resilience Benchmarking of Microservice Architectures	2018	IEEE Xplore
S11	[38]	The Business Case for Chaos engineering	2018	IEEE Xplore
S12	[39]	Use of Self-Healing Techniques to Improve the Reliability of a Dynamic and Geo-Distributed Ad Delivery Service	2018	IEEE Xplore
S13	[40]	Security Chaos engineering for Cloud Services: Work In Progress	2019	IEEE Xplore
S14	[41]	A Framework of Virtual War Room and Matrix Sketch-Based Streaming Anomaly Detection for Microservice Systems	2020	IEEE Xplore
S15	[42]	CloudStrike: Chaos engineering for Security and Resiliency in Cloud Infrastructure	2020	IEEE Xplore
S16	[43]	Identifying and Prioritizing Chaos Experiments by Using Established Risk Analysis Techniques	2020	IEEE Xplore
S17	[44]	Fitness-guided Resilience Testing of Microservice-based Applications	2020	IEEE Xplore
S18	[24]	A Chaos engineering System for Live Analysis and Falsification of Exception-Handling in the JVM	2021	IEEE Xplore
S19	[45]	A Study on Chaos engineering for Improving Cloud Software Quality and Reliability	2021	IEEE Xplore
S20	[46]	Chaos engineering for Enhanced Resilience of Cyber–Physical Systems	2021	IEEE Xplore
S21	[47]	ChaosTwin: A Chaos engineering and Digital Twin Approach for the Design of Resilient IT Services	2021	IEEE Xplore
S22	[48]	Platform Software Reliability for Cloud Service Continuity—Challenges and Opportunities	2021	IEEE Xplore
S23	[49]	Trace-based Intelligent Fault Diagnosis for Microservices with Deep Learning	2021	IEEE Xplore
S24	[50]	A Guided Approach Towards Complex Chaos Selection, Prioritization and Injection	2022	IEEE Xplore
S25	[51]	Chaos Driven Development for Software Robustness Enhancement	2022	IEEE Xplore
S26	[22]	Maximizing Error Injection Realism for Chaos engineering With System Calls	2022	IEEE Xplore
S27	[52]	On Evaluating Self-Adaptive and Self-Healing Systems using Chaos engineering	2022	IEEE Xplore
S28	[53]	Observability and chaos engineering on system calls for containerized applications in Docker	2021	ScienceDirect
S29	[54]	Scalability resilience framework using application-level fault injection for cloud-based software services	2022	Springer
S30	[55]	Chaos as a Software Product Line—A platform for improving open hybrid-cloud systems resiliency	2022	Wiley
S31	[56]	The Observability, Chaos engineering, and Remediation for Cloud-Native Reliability	2022	Wiley

3.6. Data synthesis

To answer the research questions, the data obtained are collected and summarized in an appropriate manner, which is called data synthesis. To perform the data synthesis, a qualitative analysis process was conducted on the data obtained. For instance, synonyms used for different categories were identified and merged in the respective fields. This comprehensive data synthesis approach allowed us to derive insights and draw conclusions from the collected information.

4. Results

The result section of the paper provides various insights into how chaos engineering is applied in production environments, particularly its use in improving the resilience and reliability of microservice architecture applications. The section discusses how fault detection is developed using chaos engineering tools and is mainly used in production for troubleshooting. Chaos Experiments are usually conducted in the production environment to provide realistic results. The section further enumerates several tools that have been used for Chaos experiments, as well as discussing general principles such as defining a steady state, forming a hypothesis, conducting the experiment, and proving or refuting the hypothesis. These principles and tools help detect problems like hardware issues, software errors network interruptions security vulnerabilities configuration mistakes within their respective contexts.

4.1. Main statistics

Fig. 3 shows the results of the quality assessment. The distribution of the years of publication is shown in Fig. 4. Most of the studies related to our study were conducted in the last year. This shows that researchers' interest in chaos engineering has increased in recent years. Most of the studies included were indexed in the IEEE Xplore database.

Fig. 5 presents the distribution of the type of publications and the corresponding databases. While there are many journal papers, conference proceedings also appear in the selected papers.

Chaos engineering involves several categories of functionality that serve distinct purposes in resilience testing. The first category involves intentionally terminating processes or services to evaluate system behavior and recovery from failures [7]. Another category is network simulation, which allows engineers to replicate adverse network conditions to assess system performance and reliability [25]. In the Stressing Machine category, engineers subject the system to extreme loads to identify limits and potential bottlenecks [7]. In security testing, engineers simulate breaches or attacks to assess the system's response and enhance defenses [7]. Lastly, engineers use fault application code to inject targeted faults or errors into the codebase, assessing system resilience and error-handling capabilities [24]. These categories help organizations proactively identify weaknesses, strengthen system robustness, and enhance reliability in complex technology landscapes [7]. Functionality categories of tools are presented in Fig. 6.

The tools utilized in industry settings are not comprehensively addressed in articles. To provide insights for future research, the identified tools from the additional examination were categorized based on their functionality, as presented in Tables 2 and 3. Table 2 displays the tools obtained from the study, while Table 3 presents additional tools that have been examined. Tools listed in the table with corresponding references indicate their inclusion in the referenced articles.

4.2. How is Chaos engineering effectively applied in production environments to enhance the resilience of software systems?

Table 4 examines the successful implementation of Chaos Engineering in operational settings, covering different aspects such as goals, techniques and resources, guiding principles, findings, limitations and substitutes, as well as the general strategy.

4.3. Which platforms have been used for chaos experiments?

Table 5 provides a concise summary of various tools and platforms used in Chaos experiments, along with their specific functionalities or characteristics. It offers comprehensive insights into each platform through detailed descriptions accompanied by the necessary references.

Fig. 3. Quality assessment scores.

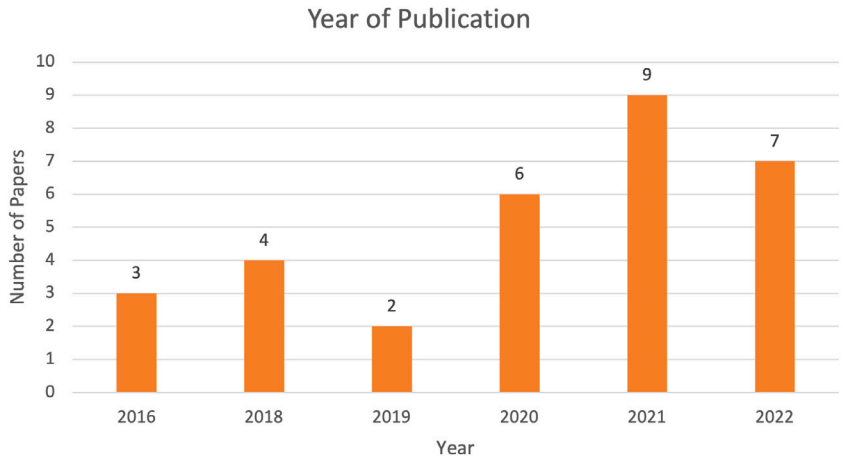

Fig. 4. Year of publication.

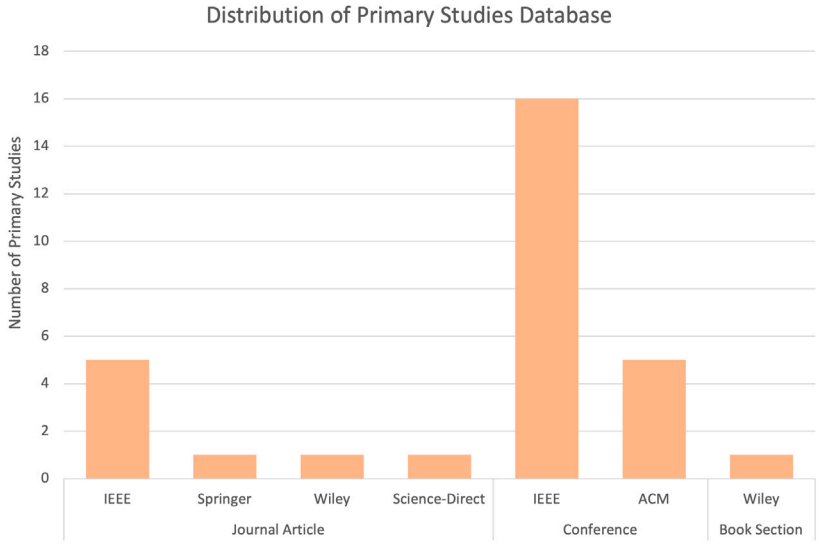

Fig. 5. Diagram of the distribution of studies per search database.

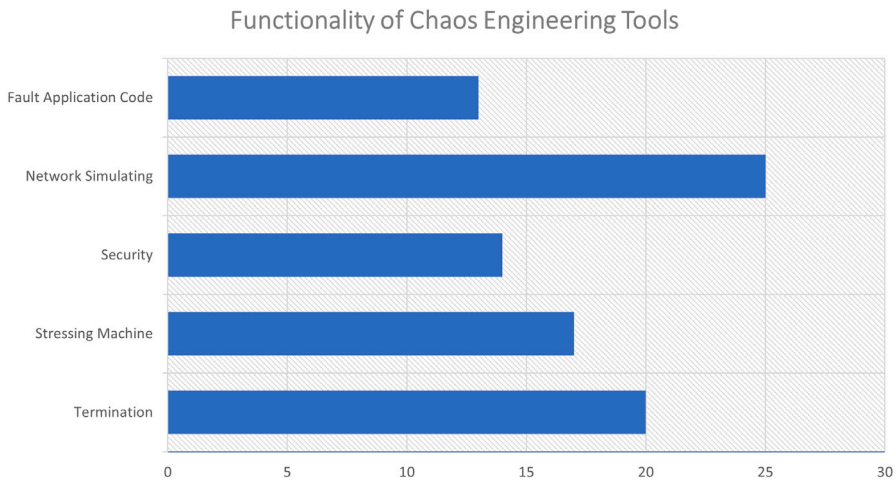

Fig. 6. Functionality of chaos engineering tools.

Table 2
Chaos engineering tools from studies.

Chaos engineering tool	Termination	Network simulating	Stressing machine	Security	Fault application code
Chaos Monkey [57]	×				
Gremlin [35]	×	×	×	×	×
Chaos Toolkit [45]	×	×	×	×	×
Pumba [55]		×	×		
LitmusChaos [45]	×	×	×	×	
ToxiProxy [45]		×		×	
PowerfulSeal [45]	×	×	×	×	
Pod Reaper [25]	×				
Netflix Simian Army [36]	×	×		×	
WireMock [25]		×			×
KubeMonkey [25]	×	×	×		
Chaosblade [45]	×	×	×		
ChaosTwin [47]	×	×	×		×
Chaos Machine [24]		×	×	×	
Cloud Strike [42]				×	
Phoebe [22]					×
Mjolnirr [58]					×
ChaosOrca [37]		×	×	×	
3MileBeach [31]		×			×
Muxy [25]		×	×		×
Blockade [25]		×			
Chaos Lambda [25]	×				×
Byte-Monkey [25]					×
Turbulence [25]	×		×	×	
Cthulhu [25]	×	×	×		×
Byteman [25]				×	×
ChaosCube [55]	×				
Chaos Lemur [25]	×				
Chaos HTTP Proxy [25]		×			
Chaos Mesh [45]	×	×	×		
Istio Chaos [45]		×			
ChAP [33]		×			×
IntelliFT [44]	×	×	×		×

Table 3
Chaos engineering tools from our search.

Chaos engineering tool	Termination	Network simulating	Stressing machine	Security	Fault application code
Pod Chaos	X	X	X		
DNS Chaos		X			
AWS Chaos	X		X	X	
Azure Chaos	X	X	X	X	
GCP Chaos	X	X	X	X	

Table 4
Chaos engineering in production environments.

Category	Description
Objective	The primary objective of applying chaos engineering in production environments is to enhance the resilience of software systems. This involves troubleshooting to identify and address potential malfunctions before they occur. The overarching goal is to minimize issues in production through the use of chaos engineering tools, enabling automatic fault detection [24,53].
Methods and tools	chaos engineering relies on specific tools to facilitate its effective application in production environments. These tools aid in automatic fault detection, a crucial aspect of troubleshooting to minimize potential issues in the production environment [24,53].
Principles and considerations	The effective application of chaos engineering is closely tied to key principles and considerations. These include continuous experimentation, serving as a form of robustness testing conducted in real-world operational conditions. Fundamental principles of Chaos Experiments involve defining a steady state, hypothesizing about its impact, conducting the experiment, and then demonstrating or refuting the hypothesis [53].
Insights and results	Chaos experiments conducted in the production environment provide valuable insights into the behavior of the system. This is particularly significant as the production environment may exhibit unpredictable behavior that differs from staging environments in some cases [24].
Constraints and alternatives	While conducting chaos experiments in production is ideal, it is acknowledged that legal or technical constraints may sometimes prevent this. In such cases, an alternative approach is considered, starting chaos experiments in a staging environment and gradually transitioning to the production environment [25].
Overall approach	The overall approach for the effective application of chaos engineering in production environments involves the systematic execution of chaos experiments. This includes leveraging chaos engineering tools and taking into account the constraints and challenges associated with conducting experiments in real-world operational settings. The aim is to proactively identify and address potential issues before they impact the production environment, ultimately enhancing the resilience of software systems.

Table 5
Chaos engineering tools identified from selected papers.

Platform/Tool	Description
The Chaos Machine	A tool for conducting chaos experiments at the application level on Java Virtual Machine (JVM), using exception injection to analyze try-catch blocks for error processing [24].
Screwdriver	An automated fault-tolerance testing tool for on-premise applications and services, creating realistic error models and collecting metrics by injecting errors into the system [34].
Chaos Monkey	Designed by Netflix, this tool tests the system's resilience by randomly killing partitions to check system functionality [7,45].
Cloud Strike	A security chaos engineering system for multi-cloud security, extending chaos engineering to security by injecting faults impacting confidentiality, integrity, and availability [42].
ChaosMesh	An open-source chaos engineering platform for testing the resilience and reliability of distributed systems by intentionally injecting failures and disruptions [55].
Powerfulseal	An open-source tool for testing the resilience of Kubernetes clusters by simulating real-world failures and disruptions [55].
IntelliFT	A feedback-based, automated failure testing technique for microservice applications, focusing on exposing defects in fault-handling logic [44].
The Chaos Toolkit	Open-source software that runs experiments against the system to confirm a hypothesis [25,55].
Phoebe	A fault injection framework for reliability analysis concerning system call invocation errors, enabling full observability of system call invocations and automatic experimentation [22].
Mjolnirr	A private cloud platform with a built-in Chaos Monkey service for developing private PaaS cloud infrastructure [58].
ChaosOrca	A tool for Chaos engineering on containers, perturbing system calls for processes inside containers and monitoring their effects [37].
Gremlin	Offered as a SaaS technology, Gremlin tests system resilience on various parameters and conditions, with capabilities for automation and integration with Kubernetes clusters and public clouds [35].
3MileBeach	A distributed tracing and fault injection framework for microservices, enabling chaos experiments through message serialization library manipulation [31].
ChAP	A software platform for running automated chaos experiments, simulating various failure scenarios and providing insights into system behavior under stress [29,33].
ChaosTwin	Utilizes a digital twin approach in Chaos Engineering to mitigate impacts of unforeseen events, constructing models across workload, network, and service layers [47].
Litmus Chaos	An open-source cloud-native framework for Chaos Engineering in Kubernetes environments, offering a range of chaos experiments and workflows [50].
Filibuster	A testing method in chaos engineering that introduces errors into microservice architecture to validate resilience and error tolerance [32].

Table 6
Chaos engineering in microservices: approaches, descriptions, and expected outcomes.

Approach	Description	Expected impact
Fault injection testing	This method involves intentionally introducing errors into the system to assess its response, particularly in microservices by simulating various failure modes such as network issues, service outages, or resource shortages within or between microservices, to evaluate the system's resilience and stability [52].	Evaluating and enhancing the system's resilience and stability.
Hypothesis-driven experiments	Key to chaos engineering is conducting experiments based on well-defined hypotheses about the normal state of the system and its expected behavior during failure scenarios. This strategic approach enables focused experiments that assess the resilience of both individual microservices and the overall system [45,53].	Identifying system weaknesses and increasing resilience.
Blast radius management	Managing the "blast radius" of experiments is crucial in microservices. It involves understanding the potential impact of introduced failures, starting with small experiments and then expanding, to manage failure impacts while identifying system vulnerabilities [45].	Better understanding and enhancing the system's resilience.
Resilience requirement elicitation	Utilizing chaos engineering to determine and analyze the resilience requirements of microservice architectures. This process involves observing the system's response to induced faults to identify specific resilience needs of each microservice and their interactions [52].	Understanding specific resilience needs of each microservice and their interactions.
Continuous testing and improvement	Regularly conducting chaos experiments as part of an ongoing testing process ensures that microservices remain resilient against unforeseen issues. This continuous approach aids in proactively finding and fixing potential system weaknesses [56].	Proactive identification and resolution of system weaknesses, leading to continual improvement and increased resilience.
Observability and remediation	Integrating chaos engineering with observability tools enhances the monitoring of microservices during fault injection, allowing for real-time tracking of responses to failures, aiding in the development of effective remediation strategies and overall system resilience improvement [56].	Real-time tracking of responses to failures and development of effective remediation strategies for overall system resilience improvement.

4.4. How can Chaos engineering be effectively applied to microservice architecture to ensure successful implementation and enhance system resilience?

Table 6 provides a comprehensive overview of the different facets and projected implications of implementing chaos engineering within microservice architecture.

By implementing these approaches and strategies, organizations can effectively integrate chaos engineering into their microservice architectures to uncover vulnerabilities and enhance the overall dependability of their systems.

4.5. To what extent can the centralized provision of Chaos engineering effectively facilitate the management of chaos experiments across complex systems?

Table 7 provides an overview of the ways in which centralized chaos engineering can simplify experiment management in intricate systems. It emphasizes advantages like standardization, resource utilization, risk mitigation, and more, resulting in enhanced system resilience and performance.

4.6. What are the challenges reported in the relevant papers?

Table 8 concisely presents the primary obstacles in the area of chaos engineering and their respective resolutions. These obstacles encompass system intricacy, hazards to live environments, resource demands, security issues, and automation complexities. The proposed resolutions involve phased implementation, risk assessment, knowledge enhancement, robust security protocols, and automation approaches.

5. Discussion

In the discussion section, we summarize answers to the research questions. They mention that chaos engineering can improve robustness by simulating real-world failure scenarios and exploring system reactions, especially in microservice architectures. Various tools for implementing chaos engineering were listed and compared. They conclude by stating that the application of chaos engineering requires careful planning due to inherent challenges but has the potential to greatly improve system resilience.

5.1. General discussion

In this article, we reviewed the literature on the application of chaos engineering in microservice architecture to understand the state-of-the-art. For this purpose, six research questions were defined and answered.

In RQ1, we aimed to understand how chaos engineering is applied to production environments. Chaos engineering, when adeptly applied in production settings, serves as a pivotal tool for augmenting the robustness of software systems. This approach entails conducting deliberate and controlled chaos experiments within the production environment, a strategy that is instrumental in uncovering and rectifying potential issues before they escalate into full-blown system failures, thereby bolstering system uptime [38]. Moreover, chaos engineering is characterized by the intentional injection of faults into systems. This methodology is crucial for identifying and addressing security flaws and risks, laying the groundwork for the development of resilient application architectures [56]. By replicating adverse conditions that could naturally arise in production settings, chaos engineering helps detect of inherent system vulnerabilities and structural deficiencies, fostering a proactive stance towards issue mitigation [38].

Additionally, this practice involves comprehensive testing of real-world scenarios on operational systems. Such testing is vital for assessing the complete spectrum of software systems, encompassing both hardware malfunctions and software glitches, within their actual deployment contexts. This approach significantly contributes to the enhancement of overall system resilience [38]. To effectively implement chaos engineering, it is recommended to initiate with less complex experiments, leverage automation for these experiments, and focus on areas with either high impact or high frequency of issues. Observing the system at its limits is also crucial for reinforcing resilience [25].

In RQ2, we discuss various platforms that aim to increase the flexibility and reliability of microservice architectures through chaos experiments. Tools like Gremlin, Chaos Monkey, Chaos Toolkit, Pumba, LitmusChaos, ToxiProxy and PowerfulSeal have been utilized in industry settings to simulate different failure scenarios. These tools provide functions such as terminating processes, simulating network conditions, applying stress tests security measures and injecting faults to proactively identify weaknesses and strengthen system robustness across different technology landscapes.

Table 7
Centralized provision in chaos engineering.

Approach	Description	Expected impact
Standardization	Centralized provision allows for the standardization of chaos engineering practices and tools across the organization. This ensures that all teams follow consistent processes and use approved tools, leading to better coordination and more reliable results [42].	Improved coordination and reliability of results.
Resource optimization	Centralized provision enables efficient allocation of resources for chaos experiments. It allows pooling of expertise, tools, and infrastructure, reducing redundancy and optimizing resource utilization [38].	Enhanced resource utilization and reduced redundancy.
Risk management	Centralized provision facilitates better risk management by providing oversight and governance for chaos experiments. It establishes clear guidelines, safety measures, and expected states for running experiments in production environments, ensuring controlled experimentation [42].	Controlled experimentation and effective risk management.
Automation and continuous testing	Centralized provision supports the automation of chaos experiments to run continuously. This ensures regular conduction of experiments, leading to ongoing validation of system resilience and identification of potential issues before they manifest as outages [38,42].	Ongoing validation of system resilience and early identification of potential issues.
Knowledge sharing and collaboration	A centralized approach encourages knowledge sharing and collaboration among teams. It facilitates the dissemination of best practices, lessons learned, and successful experiment designs, fostering a culture of continuous improvement and shared learning [25].	Promotion of a continuous improvement culture and shared learning.
Performance metrics and analysis	Centralized provision enables the establishment of standardized performance metrics and analysis methods for chaos experiments. This allows for consistent measurement of system health and identification of deviations from steady-state, leading to more effective decision-making and system improvements [43].	Consistent system health measurement and more effective decision-making.

Table 8
Challenges and solutions in chaos Engineering.

Category	Challenges	Possible solutions	References
Complexity	Designing and executing effective chaos experiments in large systems is complex due to intricate interdependencies within these systems.	To mitigate complexity, it is recommended to start with smaller, more manageable experiments and gradually expand the scope of chaos engineering practices.	[25,43]
Risk of impact	Concerns about causing disruptions in the production environment, affecting users and business operations.	Implementing risk analysis techniques can help prioritize experiments, focusing on less critical system components first to minimize potential impacts.	[45,50]
Resource intensiveness	Significant resources needed including time, expertise, and infrastructure, posing a barrier for many organizations.	Addressing resource intensiveness involves providing comprehensive training and education on chaos engineering best practices and tools to equip teams with the necessary skills and knowledge.	[7,47]
Security concerns	Introducing controlled failures can raise security issues, potentially exposing vulnerabilities or sensitive data.	To combat security concerns, robust security measures should be implemented during experiments to safeguard sensitive data and prevent unauthorized access.	[42,47]
Tooling and automation	Developing tools for automated chaos experiments is challenging in heterogeneous and dynamic environments.	Overcoming tooling and automation challenges requires the development and use of automated tools for Chaos experiments, which reduce manual efforts and facilitate continuous, unattended testing.	[7,33,38,40,42]

Recent studies have emphasized the growing intersection between artificial intelligence and cybersecurity within the context of chaos engineering. AI-driven techniques are nowadays used for real-time threat detection, anomaly prediction, and automated response mechanisms in enterprise systems. For example, generative AI models have been proposed to enhance cybersecurity frameworks by improving data privacy management and identifying potential attack vectors [59].

In RQ3, we focused on understanding how chaos engineering is implemented in microservice architectures. To enhance system resilience in microservice architectures through chaos engineering, organizations should utilize fault injection testing to replicate failures within microservices. They should also conduct hypothesis-driven experiments with a solid comprehension of the normal state and anticipated behavior during disruptions, while managing the scope of these experiments to minimize impact. Additionally, it is essential to identify and analyze resilience requirements, participate in continuous testing and improvement efforts, as well as integrate observability tools for real-time monitoring during fault injection tests. Moreover, organizations need to establish clear communication channels across teams involved in order to ensure effective collaboration and knowledge sharing.

The answer to RQ4, highlights the significance of centralized management and monitoring in conducting chaos experiments within large-scale microservices ecosystems. It discusses the utilization of software solutions like Netflix's Chaos Automation Platform (ChAP) and fault injection techniques such as service call manipulation. The emphasis is placed on the need for careful planning, effective communication, risk management, and continuous learning to ensure comprehensive and valuable chaos experiments for enhancing overall system resilience.

In response to RQ5, our discussion concludes that the practical implementation of chaos engineering, despite its promise to enhance system resilience, presents numerous challenges. These challenges include potential business impacts, difficulty in determining scope, the unpredictability of outcomes, time and resource constraints, system complexities, skill and knowledge prerequisites, interpretation of results, cultural readiness, and selection of appropriate tools. These all necessitate meticulous planning and skilled execution for effectiveness.

Recent studies explore the convergence of Chaos Engineering and Artificial Intelligence (AI). Large language models (LLMs) have been used to automate the chaos engineering lifecycle, managing phases from hypothesis creation to experiment orchestration and remediation [60]. Meanwhile, advances in applying chaos engineering to multi-agent AI systems suggest new directions: for example, chaos experiments applied to LLM-based multi-agent systems can surface vulnerabilities such as hallucinations, agent failures, or inter-agent communication breakdowns [61]. Together, these works show how intelligent,

adaptive chaos frameworks might evolve in microservice-based systems as well.

Recent research also discusses specific operational challenges such as load balancing and security in the context of chaos engineering. For example, an empirical study applies delay injections under different user loads in cloud-native systems to observe how throughput and latency change under stress, providing insights into how load balancing policies perform under fault conditions [62]. In parallel, several frameworks have begun integrating security-focused chaos tests that intentionally inject faults into authentication, identity management, and access control components to ensure that security mechanisms remain effective under stress conditions [63]. These studies highlight how chaos engineering can be extended beyond performance reliability to proactively strengthen both load distribution and security resilience in microservice environments.

The main challenges faced by previous researchers and possible solutions have been discussed in the paper. The collected challenges were mainly related to the correct interpretation of chaos experiments and making sense of them. There may be more challenges, but if they were not mentioned in these articles, we could not include them. We believe that chaos engineering is still in the early stages and the adoption in the software industry will take some time.

5.2. Threats to validity

Internal validity

The validity of this systematic literature review is threatened by issues related to defining the candidate pool of papers, potential bias in selecting primary studies, data extraction, and data synthesis. The application of exclusion criteria can be influenced by the researchers' biases, posing a potential threat to validity. We compiled a comprehensive list of exclusion criteria, and all conflicts were documented and resolved through discussions among us. Data extraction validity is crucial as it directly impacts the study results. Whenever any of us was uncertain about data extraction, the case was recorded for resolution through discussions with the team. Multiple meetings were held to minimize researcher bias.

External validity

The search for candidate papers involved using general search terms to minimize the risk of excluding relevant studies. Despite using a broad search query to acquire more articles, there remains a possibility that some papers were overlooked in electronic databases or missed due to recent publications. Furthermore, although seven widely used online databases in computer science and software engineering were searched, new papers may not have been included.

6. Conclusion

Our systematic literature review (SLR) on chaos engineering has explored its role in enhancing the resilience of software systems in production environments. Through our review, we have identified several crucial aspects that underline the effective application and challenges of chaos engineering [25].

Firstly, Chaos Engineering serves as a proactive troubleshooting approach in production environments [25]. By identifying and addressing potential malfunctions before they occur, it effectively preempts system disruptions. This proactive strategy is significantly implemented by chaos engineering tools that assist in automatic fault detection, thereby minimizing potential issues in these critical environments [50].

Secondly, the essence of chaos engineering is rooted in continuous experimentation and robustness testing under real-world operational conditions. The methodology involves a systematic approach: defining a steady state, hypothesizing its impacts, conducting controlled experiments, and subsequently confirming or refuting the hypotheses. These experiments are insightful, as they reveal system behaviors in production environments, which often differ unpredictably from staging environments [36,53].

Furthermore, the effectiveness of chaos engineering is contingent on the systematic execution of chaos experiments. These experiments, utilizing advanced chaos engineering tools, need to navigate the constraints and challenges inherent in real-world operational settings. The main objective is the enhancement of system resilience, achieved by proactively identifying and preemptively addressing potential issues [46].

However, it is acknowledged that conducting chaos experiments directly in production environments might be impeded by legal or technical constraints. In such scenarios, initiating experiments in a staging environment and then gradually transitioning to the production environment offers a viable alternative. This approach ensures that the benefits of chaos engineering can still be realized, but in a more controlled and possibly less direct manner.

Our review highlights that chaos engineering is a critical methodology for ensuring the resilience and robustness of software systems. By following continuous experimentation and proactive troubleshooting, it offers a pathway to address the challenges faced in complex production environments. This SLR contributes to the scientific community by discussing these methodologies and their applications, thereby providing a framework for future research and practical implementation in the field of software system resilience.

CRediT authorship contribution statement

Emrah Esen: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation. **Akhan Akbulut:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation. **Cagatay Catal:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] P. Jamshidi, C. Pahl, N.C. Mendonça, J. Lewis, S. Tilkov, Microservices: The journey so far and challenges ahead, IEEE Softw. 35 (3) (2018) 24–35, http://dx.doi.org/10.1109/MS.2018.2141039.

[2] I. Beschastnikh, P. Wang, Y. Brun, M.D. Ernst, Debugging distributed systems, Commun. ACM 59 (8) (2016) 32–37, http://dx.doi.org/10.1145/2909480.

[3] W. Ahmed, Y.W. Wu, A survey on reliability in distributed systems, J. Comput. System Sci. 79 (8) (2013) 1243–1255, http://dx.doi.org/10.1016/j.jcss.2013.02.006.

[4] D. Ma'ruf, S. Sulistyo, L. Nugroho, Applying integrating testing of microservices in airline ticketing system, Ijitee (Int. J. Inf. Technol. Electr. Eng.) 4 (2020) 39, http://dx.doi.org/10.22146/ijitee.55491.

[5] F. Dai, H. Chen, Z. Qiang, Z. Liang, B. Huang, L. Wang, Automatic analysis of complex interactions in microservice systems, Complexity 2020 (2020) 1–12, http://dx.doi.org/10.1155/2020/2128793.

[6] J. Lewis, M. Fowler, Microservices: a definition of this new architectural term (2014), 2014, URL: http://martinfowler.com/articles/microservices.html (cit. p. 26).

[7] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, C. Rosenthal, Chaos engineering, IEEE Softw. 33 (3) (2016) 35–41, http://dx.doi.org/10.1109/MS.2016.60.

[8] R.T. Munodawafa, S.K. Johl, A systematic review of eco-innovation and performance from the resource-based and stakeholder perspectives, Sustainability 11 (2019) 6067, http://dx.doi.org/10.3390/su11216067.

[9] J.M. Macharia, Systematic literature review of interventions supported by integration of ict in education to improve learners' academic performance in stem subjects in kenya, J. Educ. Pract. 6 (2022) 52–75, http://dx.doi.org/10.47941/jep.979.

[10] P. Gerli, J.N. Marco, J. Whalley, What makes a smart village smart? a review of the literature, Transform. Gov.: People Process. Policy 16 (2022) 292–304, http://dx.doi.org/10.1108/tg-07-2021-0126.

[11] R. Coppola, L. Ardito, Quality assessment methods for textual conversational interfaces: a multivocal literature review, Information 12 (2021) 437, http://dx.doi.org/10.3390/info12110437.

[12] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman, Systematic literature reviews in software engineering – A systematic literature review, Inf. Softw. Technol. 51 (1) (2009) 7–15, http://dx.doi.org/10.1016/j.infsof.2008.09.009, Special Section - Most Cited Articles in 2002 and Regular Research Papers.

[13] N. Dragoni, S. Giallorenzo, A.L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina, Microservices: yesterday, today, and tomorrow, 2017, arXiv:1606.04036.

[14] P.D. Francesco, I. Malavolta, P. Lago, Research on architecting microservices: Trends, focus, and potential for industrial adoption, in: 2017 IEEE International Conference on Software Architecture, ICSA, 2017, pp. 21–30, http://dx.doi.org/10.1109/ICSA.2017.24.

[15] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley Longman Publishing Co., Inc., USA, 2002.

[16] J. Lewis, M. Fowler, Microservices, 2014, https://martinfowler.com/articles/microservices.html.

[17] S. Newman, Building Microservices: Designing Fine-Grained Systems, " O'Reilly Media, Inc.", 2021.

[18] C.K. Rudrabhatla, Comparison of zero downtime based deployment techniques in public cloud infrastructure, in: 2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud), I-SMAC, 2020, pp. 1082–1086, http://dx.doi.org/10.1109/I-SMAC49090.2020.9243605.

[19] S.R. Addula, P. Perugu.P, M.K. Kumar, D. Kumar, B. Ananthan, R. R, S. P, S. G, Dynamic load balancing in cloud computing using hybrid Kookaburra-Pelican optimization algorithms, in: 2024 International Conference on Augmented Reality, Intelligent Systems, and Industrial Automation, ARIIA, 2024, pp. 1–7, http://dx.doi.org/10.1109/ARIIA63345.2024.11051893.

[20] M. Waseem, P. Liang, M. Shahin, A systematic mapping study on microservices architecture in devops, J. Syst. Softw. 170 (2020) 110798, http://dx.doi.org/10.1016/j.jss.2020.110798.

[21] C. Rosenthal, N. Jones, Chaos Engineering: System Resiliency in Practice, O'Reilly Media, 2020.

[22] L. Zhang, B. Morin, B. Baudry, M. Monperrus, Maximizing error injection realism for chaos engineering with system calls, IEEE Trans. Dependable Secur. Comput. 19 (4) (2022) 2695–2708, http://dx.doi.org/10.1109/TDSC.2021.3069715.

[23] Š. Davidovič, B. Beyer, Canary analysis service, Commun. ACM 61 (5) (2018) 54–62, http://dx.doi.org/10.1145/3190566.

[24] L. Zhang, B. Morin, P. Haller, B. Baudry, M. Monperrus, A chaos engineering system for live analysis and falsification of exception-handling in the JVM, IEEE Trans. Softw. Eng. 47 (11) (2021) 2534–2548, http://dx.doi.org/10.1109/TSE.2019.2954871.

[25] H. Jernberg, P. Runeson, E. Engström, Getting started with chaos engineering - design of an implementation framework in practice, in: Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM, ESEM '20, Association for Computing Machinery, New York, NY, USA, 2020, http://dx.doi.org/10.1145/3382494.3421464.

[26] A. Alkhateeb, C. Catal, G. Kar, A. Mishra, Hybrid blockchain platforms for the internet of things (IoT): A systematic literature review, Sensors 22 (4) (2022) http://dx.doi.org/10.3390/s22041304.

[27] R. van Dinter, B. Tekinerdogan, C. Catal, Predictive maintenance using digital twins: A systematic literature review, Inf. Softw. Technol. 151 (2022) 107008, http://dx.doi.org/10.1016/j.infsof.2022.107008.

[28] M. Jorayeva, A. Akbulut, C. Catal, A. Mishra, Machine learning-based software defect prediction for mobile applications: A systematic literature review, Sensors 22 (7) (2022) http://dx.doi.org/10.3390/s22072551.

[29] A. Basiri, L. Hochstein, N. Jones, H. Tucker, Automating chaos experiments in production, in: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP, 2019, pp. 31–40, http://dx.doi.org/10.1109/ICSE-SEIP.2019.00012.

[30] L.B. Canonico, V. Vakeel, J. Dominic, P. Rodeghero, N. McNeese, Human-AI partnerships for chaos engineering, in: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 499–503, http://dx.doi.org/10.1145/3387940.3391493.

[31] J. Zhang, R. Ferydouni, A. Montana, D. Bittman, P. Alvaro, 3MileBeach: A tracer with teeth, in: Proceedings of the ACM Symposium on Cloud Computing, SoCC '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 458–472, http://dx.doi.org/10.1145/3472883.3486986.

[32] C.S. Meiklejohn, A. Estrada, Y. Song, H. Miller, R. Padhye, Service-level fault injection testing, in: Proceedings of the ACM Symposium on Cloud Computing, SoCC '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 388–402, http://dx.doi.org/10.1145/3472883.3487005.

[33] A. Blohowiak, A. Basiri, L. Hochstein, C. Rosenthal, A platform for automating chaos experiments, in: 2016 IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW, 2016, pp. 5–8, http://dx.doi.org/10.1109/ISSREW.2016.52.

[34] A. Nagarajan, A. Vaddadi, Automated fault-tolerance testing, in: 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops, ICSTW, 2016, pp. 275–276, http://dx.doi.org/10.1109/ICSTW.2016.34.

[35] V. Heorhiadi, S. Rajagopalan, H. Jamjoom, M.K. Reiter, V. Sekar, Gremlin: Systematic resilience testing of microservices, in: 2016 IEEE 36th International Conference on Distributed Computing Systems, ICDCS, 2016, pp. 57–66, http://dx.doi.org/10.1109/ICDCS.2016.11.

[36] R.K. Lenka, S. Padhi, K.M. Nayak, Fault injection techniques - a brief review, in: 2018 International Conference on Advances in Computing, Communication Control and Networking, ICACCCN, 2018, pp. 832–837, http://dx.doi.org/10.1109/ICACCCN.2018.8748585.

[37] A. van Hoorn, A. Aleti, T.F. Düllmann, T. Pitakrat, ORCAS: Efficient resilience benchmarking of microservice architectures, in: 2018 IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW, 2018, pp. 146–147, http://dx.doi.org/10.1109/ISSREW.2018.00-10.

[38] H. Tucker, L. Hochstein, N. Jones, A. Basiri, C. Rosenthal, The business case for chaos engineering, IEEE Cloud Comput. 5 (3) (2018) 45–54, http://dx.doi.org/10.1109/MCC.2018.032591616.

[39] N. Brousse, O. Mykhailov, Use of self-healing techniques to improve the reliability of a dynamic and geo-distributed ad delivery service, in: 2018 IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW, 2018, pp. 1–5, http://dx.doi.org/10.1109/ISSREW.2018.00-40.

[40] K.A. Torkura, M.I. Sukmana, F. Cheng, C. Meinel, Security chaos engineering for cloud services: Work in progress, in: 2019 IEEE 18th International Symposium on Network Computing and Applications, NCA, 2019, pp. 1–3, http://dx.doi.org/10.1109/NCA.2019.8935046.

[41] H. Chen, P. Chen, G. Yu, A framework of virtual war room and matrix sketch-based streaming anomaly detection for microservice systems, IEEE Access 8 (2020) 43413–43426, http://dx.doi.org/10.1109/ACCESS.2020.2977464.

[42] K.A. Torkura, M.I.H. Sukmana, F. Cheng, C. Meinel, CloudStrike: Chaos engineering for security and resiliency in cloud infrastructure, IEEE Access 8 (2020) 123044–123060, http://dx.doi.org/10.1109/ACCESS.2020.3007338.

[43] D. Kesim, A. van Hoorn, S. Frank, M. H00E4ussler, Identifying and prioritizing chaos experiments by using established risk analysis techniques, in: 2020 IEEE 31st International Symposium on Software Reliability Engineering, ISSRE, 2020, pp. 229–240, http://dx.doi.org/10.1109/ISSRE5003.2020.00030.

[44] Z. Long, G. Wu, X. Chen, C. Cui, W. Chen, J. Wei, Fitness-guided resilience testing of microservice-based applications, 2020, pp. 151–158, http://dx.doi.org/10.1109/ICWS49710.2020.00027.

[45] S. De, A study on chaos engineering for improving cloud software quality and reliability, in: 2021 International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications, CENTCON, Vol. 1, 2021, pp. 289–294, http://dx.doi.org/10.1109/CENTCON52345.2021.9688292.

[46] C. Konstantinou, G. Stergiopoulos, M. Parvania, P. Esteves-Verissimo, Chaos engineering for enhanced resilience of cyber-physical systems, in: 2021 Resilience Week, RWS, 2021, pp. 1–10, http://dx.doi.org/10.1109/RWS52686.2021.9611797.

[47] F. Poltronieri, M. Tortonesi, C. Stefanelli, ChaosTwin: A chaos engineering and digital twin approach for the design of resilient IT services, in: 2021 17th International Conference on Network and Service Management, CNSM, 2021, pp. 234–238, http://dx.doi.org/10.23919/CNSM52442.2021.9615519.

[48] N. Luo, Y. Xiong, Platform software reliability for cloud service continuity - challenges and opportunities, in: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security, QRS, 2021, pp. 388–393, http://dx.doi.org/10.1109/QRS54544.2021.00050.

[49] H. Chen, K. Wei, A. Li, T. Wang, W. Zhang, Trace-based intelligent fault diagnosis for microservices with deep learning, in: 2021 IEEE 45th Annual Computers, Software, and Applications Conference, COMPSAC, 2021, pp. 884–893, http://dx.doi.org/10.1109/COMPSAC51774.2021.00121.

[50] O. Sharma, M. Verma, S. Bhadauria, P. Jayachandran, A guided approach towards complex chaos selection, prioritisation and injection, in: 2022 IEEE 15th International Conference on Cloud Computing, CLOUD, 2022, pp. 91–93, http://dx.doi.org/10.1109/CLOUD55607.2022.00025.

[51] N. Luo, L. Zhang, Chaos driven development for software robustness enhancement, in: 2022 9th International Conference on Dependable Systems and their Applications, DSA, 2022, pp. 1029–1034, http://dx.doi.org/10.1109/DSA56465.2022.00154.

[52] M.A. Naqvi, S. Malik, M. Astekin, L. Moonen, On evaluating self-adaptive and self-healing systems using chaos engineering, in: 2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS, 2022, pp. 1–10, http://dx.doi.org/10.1109/ACSOS55765.2022.00018.

[53] J. Simonsson, L. Zhang, B. Morin, B. Baudry, M. Monperrus, Observability and chaos engineering on system calls for containerized applications in Docker, Future Gener. Comput. Syst. 122 (2021) 117–129, http://dx.doi.org/10.1016/j.future.2021.04.001.

[54] A.A.-S. Ahmad, P. Andras, Scalability resilience framework using application-level fault injection for cloud-based software services, J. Cloud Comput. 11 (1) (2022) 1, http://dx.doi.org/10.1186/s13677-021-00277-z.

[55] C. Camacho, P.C. Cañizares, L. Llana, A. Núñez, Chaos as a software product line—A platform for improving open hybrid-cloud systems resiliency, Softw.: Pract. Exp. 52 (7) (2022) 1581–1614, http://dx.doi.org/10.1002/spe.3076.

[56] P. Raj, S. Vanga, A. Chaudhary, The observability, chaos engineering, and remediation for cloud-native reliability, in: Cloud-Native Computing: How To Design, Develop, and Secure Microservices and Event-Driven Applications, 2023, pp. 71–93, http://dx.doi.org/10.1002/9781119814795.ch4.

[57] M.A. Chang, B. Tschaen, T. Benson, L. Vanbever, Chaos monkey: Increasing sdn reliability through systematic network destruction, in: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, 2015, pp. 371–372.

[58] D. Savchenko, G. Radchenko, O. Taipale, Microservices validation: Mjolnirr platform case study, in: 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO, 2015, pp. 235–240, http://dx.doi.org/10.1109/MIPRO.2015.7160271.

[59] G.S. Nadella, S.R. Addula, A.R. Yadulla, G.S. Sajja, M. Meesala, M.H. Maturi, K. Meduri, H. Gonaygunta, Generative AI-enhanced cybersecurity framework for enterprise data privacy management, Computers 14 (2) (2025) http://dx.doi.org/10.3390/computers14020055.

[60] D. Kikuta, H. Ikeuchi, K. Tajiri, Y. Nakano, ChaosEater: Fully automating chaos engineering with large language models, 2025, arXiv preprint arXiv:2501.11107. URL https://arxiv.org/abs/2501.11107.

[61] J. Owotogbe, Assessing and enhancing the robustness of LLM-based multi-agent systems through chaos engineering, in: 2025 IEEE/ACM 4th International Conference on AI Engineering – Software Engineering for AI, CAIN, 2025, pp. 250–252, http://dx.doi.org/10.1109/CAIN66642.2025.00039.

[62] A. Al-Said Ahmad, L.F. Al-Qora'n, A. Zayed, Exploring the impact of chaos engineering with various user loads on cloud native applications: An exploratory empirical study, Computing 106 (2024) 2389–2425, http://dx.doi.org/10.1007/s00607-024-01292-z.

[63] K.A. Torkura, M.I. Sukmana, F. Cheng, C. Meinel, Security chaos engineering for cloud services: Work in progress, in: 2019 IEEE 18th International Symposium on Network Computing and Applications, NCA, 2019, pp. 1–3, http://dx.doi.org/10.1109/NCA.2019.8935046.