Migrating from Monolithic to Microservice Architectures: A Systematic Literature Review

Hossam Hassan, Manal A. Abdel-Fattah, Wael Mohamed
Information Systems Department, Helwan University, Egypt

***Abstract*—Migration from monolithic software systems to modern microservice architecture is a critical process for enhancing software systems' scalability, maintainability, and performance. This study conducted a systematic literature review to explore the various methodologies, techniques, and algorithms used in the migration of monolithic systems to modern microservice architectures. Furthermore, this study underscored the role of artificial intelligence in enhancing the efficiency and effectiveness of the migration process by examining recent literature to identify significant patterns, challenges, and optimal solutions. In addition, it emphasizes the importance of migrating monolithic systems into microservices by synthesizing various research studies that enable greater flexibility, fault tolerance, and independent scalability. The findings offer valuable insights for both researchers and practitioners in the software industry. In addition, it provides practical guidance on implementing AI-driven methodologies in software architecture evolution. Finally, we highlight future research directions in providing an automation technique for the software architecture migration.**

***Keywords*—Software migration; software evolution; monolithic architecture; microservice architecture; systematic literature review**

I. Introduction

Over the last decade, software architecture has significantly changed. Software was initially monolithic, integrating all components into a single unit. However, as software systems grew more complex, limitations in scalability, flexibility, and maintainability appeared. This encourages an investigation of intermediary architectural patterns like micro-kernel architecture and service-oriented architecture (SOA). Micro-kernel architecture has a minimal core system with only the most important functions while delegating additional functionalities to modular, user-space components, whereas SOA focuses on breaking software down into loosely coupled services. These evolutions paved the way for a new microservices architecture, which divides software into tiny, autonomous, single-need components.

Most software businesses worldwide have adopted the modern microservice architecture because of its significantly increased scalability, maintainability, flexibility, and ease of development [1], [2]. In addition, many software businesses, like Amazon, Uber, Netflix, and Spotify, are adopting this architectural approach, and the transition to microservices is well underway [3]. Microservice architecture can be represented by a collection of tiny services, each operating in its own process and interacting using lightweight protocols like HTTP (Hypertext Transfer Protocol), developed around business needs and delivered independently [2], [4].

The microservices architecture solves many monolithic system issues. Some of the major benefits of microservices include scalability, which improves resource utilization and peak performance, and flexibility, which allows development teams to pick the best tools and technologies for each service, resulting in more customized and efficient solutions, as well as fault tolerance, which prevents system failures from affecting all services, improving resilience and reliability [3].

During the migration to the microservice architecture, some software warehouses that have monolith-based systems attempt to decompose them into coherent microservice-based implementations. The purpose of this decomposition is to occasionally assist software architects in identifying microservice candidates by analyzing the application's domain, business needs, source code, and version-related information [5], [6], [7].

Nevertheless, software vendors have been exerting efforts to manually migrate from monolithic to microservice-oriented application ecosystems. Scalability, component independence, data management, service communications, deployment, and monitoring are some of the efforts [1], [8]. This migration process is subjective, requires human judgment, and is prone to errors. Expert opinions are required for this process, as is software engineers' proficiency in microservice extraction and system quality preservation [9]. Besides, extracting microservices is becoming a complex process because there is no clear or straightforward method for defining the boundaries of microservices. The first and most challenging stage in breaking down a monolithic application involves identifying microservice boundaries. Insufficient identification may lead to more complex systems with lower quality [1].

To address this gap, there is an increasing interest in using AI and ML algorithms to facilitate the migration process. AI-driven methods, such as search-based techniques and clustering algorithms, make it possible to automatically find microservice components and improve the decomposition of systems. However, the majority of the studies fail to identify relevant and potential microservices, and they struggle to determine the appropriate number of candidate microservices while also ensuring their granularity and loose coupling [10].

This paper aims to conduct a comprehensive and systematic literature review to analyze and identify the most important methods and techniques used for facilitating the migration process of software from monolithic systems to modern microservices. The review also aims to compare the results in a detailed manner through a systematic literature review (SLR),

which can serve as a foundation for developing effective solutions.

This paper's subsequent sections follow this structure: Section II encompasses the background and motivation, while Section III presents the research methodology. Section IV provides an overview of the current state of knowledge and understanding regarding the process of software migration. Section V presents the findings of this literature review. Section VI covers the conclusion of this literature review and offers suggestions for future work and improvements.

II. Background

Monolithic and microservice architectures are two different methodologies for designing and constructing software systems. A variety of factors influence the choice between them. Each architecture possesses its own advantages and disadvantages, and the determination should be made considering the particular requirements and limitations of the project. In the following subsections, we will provide additional information regarding the similarities, differences, advantages, and disadvantages of the two architectural styles.

A. Monolithic Architecture

A single deployed unit is referred to as a monolith. A monolithic system necessitates the simultaneous deployment of all functionalities. Deploying all code as a unified process, consolidating all code into a single process, characterizes a monolithic architecture [2].

The modular monolith, a new version of the single-process monolith, divides the single process into multiple distinct modules developed separately; however, deployment requires the combination of these modules. It might be an optimal solution for several enterprises and software warehouses because it defines module boundaries well, provides a significant amount of parallel work, overcomes the complexities associated with the distributed microservice architecture, and adopts a simpler deployment topology [2], [11]. Differences between monolithic and modular monolithic software architectures can be shown in Fig. 1. Shopify is an example of a company that uses this technique as a substitute for microservice deconstruction, and it appears to be quite effective for them.

Unfortunately, the monolith has recently become a symbol of avoidance and is often associated with legacy systems. However, it is actually a viable option based on the system's requirements and specifications. Some of the advantages of monolithic architecture are listed below [2], [12]:

1) Faster and rapid deployment topology: Avoids numerous distributed system issues.

2) Workflows for developers are simpler to manage.

3) Simpler monitoring and troubleshooting: End-to-end testing is simplified.

4) Code reuse is simple enough, without any duplication.

Nevertheless, an important issue with the modular monolith architecture is that the database does not have the same degree of decomposition as the code, making it difficult to separate the monolith in the future [2].

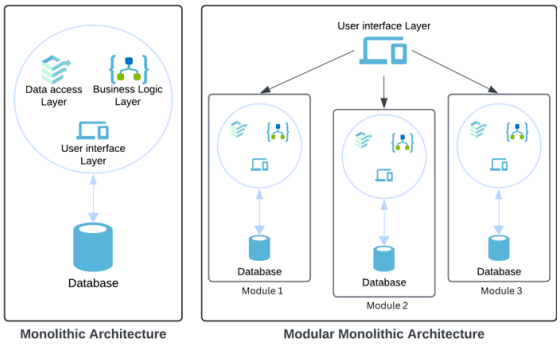

Fig. 1. Demonstrating the distinctions between monolithic and modular software architecture.

B. Microservice Architecture

Microservices are designed to be loosely coupled and independently deployable, but they may still rely on different coordination patterns to handle distributed transactions and inter-service communication, which are designed based on a specific business domain and potentially released separately. A service is a module that has specific functionality and allows other services to access it over networks. By combining these modules, the software warehouses could create more intricate and enterprise systems. Each microservice might represent a specific aspect of the system. When combined, these microservices form a complete enterprise system. In other words, they are a kind of service-oriented architecture that has a certain viewpoint on how service boundaries should be defined with the ability to deploy independently. Fig. 2 showed an example of a microservice architecture construction.

A single microservice appears as a black box. One or more network endpoints, such as a SOAP or REST API, host business functionality using appropriate protocols. Through these networked endpoints, consumers—microservices or programs—access this capability. The outer world conceals implementation elements like service technology and data storage. In most cases, microservice designs encapsulate their own databases instead of using common databases.

Microservices conceal information within their components and deliver minimal information through external interfaces. The hidden microservice implementation can be freely updated as long as the changes do not introduce incompatible modifications to the network interfaces it exposes. Changes made within a microservice boundary should not affect upstream consumers, allowing for separate functionality releases. Clear, consistent service boundaries that don't alter with internal implementation lead to looser coupling and greater cohesiveness. Some of the advantages of microservice architecture are listed below:

1) Deployability independence: Allows for modification and deployment without relying on other microservices.

2) Business domain-based model: Makes it easy to bring out new functionality and recombine microservices to provide consumers with new capabilities.

3) Owning their own state: Microservices must avoid relying on shared databases. Instead, it needs to request data from another microservice in order to access it.

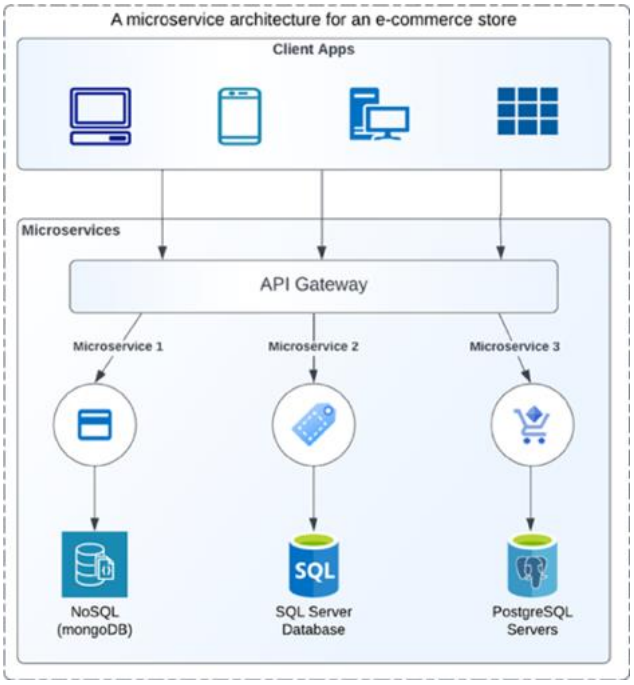

Fig. 2. A microservice architecture for an online shopping system.

III. Research Methodology

To achieve our goal of automatically migrating software architecture from monolithic to microservice, SLR was employed to analyze the migration process. This investigation encompassed the use of AI and other techniques to automate the process. SLR was chosen in this study due to its advantages over conventional literature reviews in terms of accuracy, effectiveness, and organization. It aids in the identification of our objectives, the assessment of their outcomes, and the classification of them into categories. This SLR is comprised of three primary steps, which are as follows [13], [14], and [15]:

1) Planning and preparing the review.
2) Performing the review.
3) Reporting the review.

A. Phase 1: Review Planning

Starting with these steps, this phase covers the core of SLR:

1) Clearly state questions the study will address:

RQ1: Which AI methods are effective for automated microservice architecture migration?

RQ2: What are the primary obstacles and impediments that organizations face during the migration process, and how can AI-driven solutions overcome these challenges?

RQ3: What are the criteria used to assess the effectiveness and success of AI-driven migration strategies?

2) Choose appropriate research repositories: Within the SLR, various online digital libraries were used, including but not limited to IEEE Xplore, Google Scholar, Science Direct, Springer, and MDPI.

3) Establishing research criteria: The search strings were chosen based on the SLR keywords and the alternatives to those keywords found in Software Architectural Evolution. As can be seen in Table I, these search strings were divided into two distinct categories.

4) Software Migration Process.
5) Monolithic to Microservice.

TABLE I. Search Strings for the Systematic Literature Review

GROUP	Search String
Software Migration	(“Software Migration” OR “Software Architectural Evolution” OR “Software Architecture Transformation” OR “Software Evolution” OR "Software Architecture")
	AND
monolithic to Microservice	(("Legacy", "Monolithic", "Monolithic system", "Single-layer application", "Modular Monolithic")) AND (("Microservices", "Microservices pattern", "Microservice architecture", "Service-oriented architecture (SOA)")

6) Providing clear definitions of inclusion & exclusion: The guidelines for the inclusion and exclusion criteria for this SLR were established by Kitchenhem [13].

The following criteria for inclusion were listed:

IC1: A journal publication or conference presentation are required for the research selection.

IC2: Studies should focus on software migration process, especially the migration from monolithic to microservice.

IC3: AI, ML, and other algorithms or solutions for software migration should be applied.

IC4: Studies published between 2018 and 2024. Microservices were introduced earlier, but the latest studies show the recent implementations, challenges, and innovations.

The following criteria for exclusion were listed:

EC1: Studies that failed to address the research questions.

EC2: Studies that ignored software migration process.

EC3: Studies that didn’t include microservices.

EC4: Publications published prior to 2018.

7) Establishing standards for measuring quality: This phase was significant because we compared and checked the quality of the selected studies to our objectives, research questions, and goals.

B. Phase 2: Performing the Review:

1) Primary data selection: During this stage, filtration methods were employed to apply search criteria and determine inclusion and exclusion criteria. The process of selecting the primary studies has started. The Tollgate approach was implemented to enhance the effectiveness and efficiency of the selection process in a systematic and organized manner [13].

2) Data extraction: The study selection was guided by specific criteria, including research methodology, publication

year, kind of study, and any restrictions or limitations imposed on the studies.

3) Data synthesis: The collected studies were assessed and compared with our research questions and study objectives.

C. Phase 3: Reporting the Review:

During this phase, selected studies were verified and compared against quality criteria. Fig. 3 shows the process of SLR, indicating a well-organized collection of studies available for discussion and investigation, with R representing the number of research studies in each step.

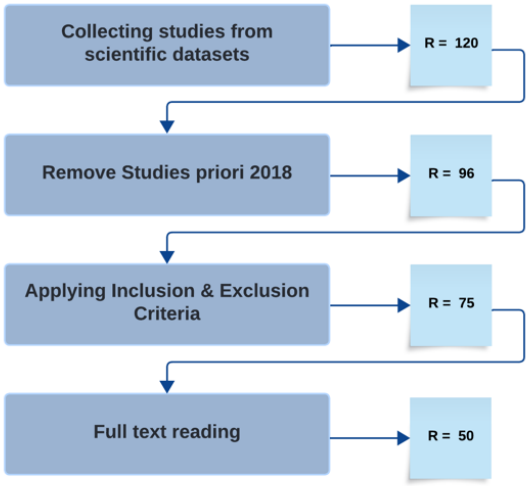

Fig. 3. The SLR selection process.

IV. State-of-the-Art

This section examined the studies selected for analysis, reviewing, and discussion.

Jin et al. [16] proposed a functionally-oriented microservice extraction (FoME) method that uses clustering of execution traces and classifies source code entities according to their functionality. The authors evaluated their method against three methods (LIMBO, WCA, and MEM) across four open-source projects. Their findings showed that FoME produces microservices with comparable cohesiveness to other methods and achieves much looser coupling. However, their method relies on high-quality test cases, with future efforts geared toward full automation of the process.

Sellami et al. [1] proposed a method called MSExtractor, which identifies microservices as multi-objective optimization problems using an indicator-based evolutionary algorithm (IBEA) while considering structural and semantic dependencies in the source code. They conducted a benchmark of seven software systems to assess the effectiveness of their method. Their findings demonstrated that MSExtractor outperformed other clustering algorithms like FoME and MEM mentioned in [16], [17]. Nevertheless, their method was limited to four web applications and lacked generalization. Future work should consider non-functional evaluation metrics.

Velepucha et al. [18] concentrated on breaking down microservice architectures. They compared different concepts, compiled a list of microservice architecture patterns as shown in Table II, and discussed the advantages and disadvantages of microservices over monolithic architectures. Future work and limitations include adapting micro-frontends, automatically migrating the decomposition process, restricting the results to an object-oriented approach, and further evaluating the literature.

TABLE II. List of Patterns used in Microservice Decomposition Process

Pattern / Description
Domain-Driven: Developing software systems focusing on business logic.
Service discovery: Addressing service interactions and communications.
Data-driven: Design systems around data behavior or structure. Data is key.
Backend for frontend: Developing services tailored to frontend clients.
Adapter microservice: Transforming microservices functionality data.
Strangler-application: Incrementally migrate to microservice.
Shared data microservice: Sharing and managing microservice data.
Aggregator microservice: Aggregate data from multiple microservices.

On the other hand, Kazanavičius et al. produced the conceptual model, which aims to migrate a monolith database into a multi-model polyglot persistence system [19]. They assessed their proof of concepts using the ISO/IEC 25012:2008 standard's definition of quality attributes. Their findings indicate that their proposed method can effectively transition data storage from a monolithic to a microservice architecture. Future direction includes the need to automate the solution and the fact that their model's adoption depends on single app.

Nordli et al. [20] focused on monolithic solution vendors; they struggle to convert monolithic products into multi-tenant cloud-native SaaS solutions because many clients, especially large enterprises, want customized products. The authors presented a proof of concept that outlines a combined approach for transforming monoliths into microservices-based, customizable cloud-native SaaS. Their findings demonstrated that a customization-driven migration approach can guide a monolith towards becoming a SaaS. Their limitations included the unavailability of datasets, the need for expert evaluation and the use of real-world systems to generalize the results, and the requirement for an automatic migration process.

Velepucha et al. [21] performed a SLR and provided a list of challenges, and benefits that arise when carrying out the migration process, as shown in Table III. Some limitations and future work include the necessity to analyze the pros and cons of each architecture, as well as automate the migration process.

TABLE III. Challenges Encountered Throughout the Software Migration

Problem/Challenges	
Utilizing an appropriate tool during the migration process.	Perform the entire microservices migration without breaking it down.
Reorganizing stakeholders is required when implementing microservices.	Challenges in identifying and designing microservices.
Desiring to transition all monolithic programs to microservices	Ensure the consistency when transitioning across databases.
Attempting to integrate new technologies into a monolithic application.	

Faustino et al. [11] performed a case study of transforming systems to microservices using a modular monolithic architecture. They discovered that a modular monolithic architecture could simplify the migration process. Additionally, it addresses issues related to performance optimization, eventual consistency, and inter-microservice communication. However, their method's limitation to a single migration case study hinders its generalizability.

Blinowski et al. [12] compared the performance and scalability of monolithic and microservice architecture by implementing a reference web application with two different technologies and architectural styles. Moreover, they selected three distinct deployment scenarios. Their findings indicated that monolithic architectures outperform microservices on a single machine, with Java handling computation-intensive tasks more efficiently than .NET. For non-computational services on machines with limited power, Azure's vertical scaling proves more cost-effective than horizontal scaling. In future studies, the authors intend to enhance the complexity of their benchmarking system, broaden its application across cloud platforms, and incorporate more performance metrics.

Bastidas Fuertes et al. [22] used Transpiler in the software architecture design model's back-end layer to automatically transform business logic from one source code to several equivalent versions. They tested the model for performance, scalability, and reliability in various scenarios and compared it with existing software design models to identify its pros and cons. Their result showed that the proposed model seeks to save costs, optimize the development process, and enhance the effectiveness of multi-programming language platforms. Nevertheless, the authors acknowledged the need for refining the model's effectiveness for generalizing the result.

Mazzara et al. [8] presented a case study to illustrate the advantages of transforming a monolithic into a microservice on scalability. The case study focuses on the FX Core system, which is critical for Danske Bank. They compared the two architectures, and the results showed that microservices have resulted in enhanced scalability and effectively resolved the significant issues posed by the monolithic. However, their findings lack insights, validation, and verification. Furthermore, the case study utilizes agile methodologies, which limits the generalizability of the results.

Assunção et al. [10] adopted an approach to redesigning features into microservices by employing search-based techniques to quantitatively assess potential redesign possibilities for monolithic features as microservices based on four criteria: coupling, cohesion, network overhead, and feature modularization. To evaluate their findings, an interview with eight monolithic system developers was conducted to get their feedback. Their findings showed that their approach demonstrates positive results and encourages more exploration. To justify configurability, future directions include generalizing results and suggesting specific criteria associated with variability.

Teguh Prasandy et al. [23] presented a method of modularizing application source code, databases, and cloud servers to identify the necessary preparations needed to make a successful transition to microservices. Their findings demonstrated that migrating to microservices can present challenges and affect stockholders, particularly system analysts and developers. Moreover, it's crucial to isolate program blocks during deployment and determine the upload time to prevent any failures. They utilized the Postman tool to assess the REST API as both a REST client and an application. Future research will include assessing the capacity of cloud servers, and evaluation is necessary to generalize the findings.

By calculating the metrics (latency, throughput, scalability, CPU usage, memory usage, and network usage) needed to compare the source and target applications, Fondazione et al. [24] developed a method to determine if migrating to microservices is beneficial. Their findings showed that monolithic works well with small to medium systems, which are typically defined by the project's overall size and complexity. The substantially higher scalability ratio of the microservice system supports the hypothesis that it performs better than a monolithic design for systems with too many concurrent users, especially when it comes to handling more traffic. Additionally, researchers conducted benchmarking experiments to evaluate their results. Future work includes testing on a real-life system, utilizing an alternative programming language, and ensuring security.

Abgaz et al. [25] conducted a SLR by examining 35 studies. Their results showed that the process of breaking down a monolith into microservices is still in its early stages, and there is a lack of techniques for integrating static, dynamic, and evolutionary data. The lack of adequate tool support is also apparent. The author conducted their SLR using Barbara Kitchenham's principles as a guide, as we illustrated in Section III. The authors suggested focusing on microservice deployment and standardizing analysis measurements.

Tapia et al. [26] assessed the performance and correlation of monolithic and microservices applications. They stress-tested their results using the same characteristics and hardware specifications. Furthermore, a mathematical model using the non-parametric regression method verified their studies' findings. Their results showed that monolithic and microservice can serve various technological situations. However, microservices improve hardware resource efficiency, cost savings, and productivity. Future directions include enhancing information security and combating cyberattacks. Moreover, automation tools for migration are required.

Kuryazov et al. [27] proposed a conceptual model for solving the issue of migrating from a monolithic architecture to microservices, especially the decomposition steps. Their conceptual model is still in its early stages and needs more evaluation and testing in a real industry. They need to develop a method for evaluating software using cohesion and coupling measurements, which will simplify the analysis of monolithic systems and estimate the migration effort. Additionally, they should create a tool that facilitates the extraction of software metadata and business logic.

Auer et al. [28] presented a decision support framework for software companies seeking to transition to microservices. Their framework is based on an examination of a set of characteristics and metrics, which they collected and reviewed through interviews with experts. Their findings provided data

and measurements that companies could use to assess microservices adoption. Future work includes validating the framework, identifying automatically applicable measures that can easily reduce decision subjectivity, and adding cloud-native technologies and micro-frontend architecture.

Daoud et al. [7] used a business process to identify needs, data, and semantics to capture dependencies between these processes, as well as a collaborative clustering technique to recommend microservices. Results showed that the approach outperformed similar ones for microservices identification and highlighted the importance of business processes. Future directions are as follows: generating new activity relationships utilizing powerful machine learning, including NLP, and evaluating various activity dependence models for microservice identification. The identification of microservices may be impacted by security concerns, which could be an intriguing development.

Hasan et al. [29] presented a collection of software architecture metrics, including coupling, complexity, cohesion, and size, to assess the maintainability of microservice architectural designs. Results showed that the suggested metrics criteria are more applicable for implementation in industrial settings. On the other hand, case studies from the real-world industrial sector need to be analyzed and applied to the suggested metrics to assess their efficacy. Furthermore, a tool-based methodology must be developed for evaluating the architectural quality of potential microservices.

Oumoussa et al. [30] performed a systematic literature review that highlighted critical areas requiring more attention, such as enhanced automated identification tools and standardized evaluation standards. Their findings showed that many techniques exist for identifying microservices; however, they often focus on particular challenges and abandon others. In addition, there is a dearth of studies that concentrate on a solution to address the migration problem.

Abdellatif et al. [31] conducted a comprehensive analysis of 41 studies. Their research aimed to identify the various inputs, processes, outputs, and usability of service identification methodologies in order to modernize monolithic software. Their findings demonstrate that the categorization aligns with the experiences of industry professionals and provides valuable assistance to practitioners in real-world industrial settings. Future directions include proposing an approach for identifying services based on their types, which enhances the potential for reusing them across several levels: application, enterprise, and business, besides generalizing their results.

Li et al. [32] proposed a technique to identify microservices by utilizing the unified model language (UML), which is derived from the source code. Then, the classes and sequence diagrams were analyzed, using them as input for clustering techniques to identify potential microservices. In addition, experiments were conducted to evaluate the proposed model and compare it to recent methods. Their findings revealed that the proposed model outperformed existing models. However, their results are not generalizable because their proposed model disregards microservice distributions and quality criteria.

The primary focus of Gomes Barbosa et al. [33] was to identify potential microservices from database procedures, specifically targeting monolithic applications developed in the 1980s and 1990s that utilized database procedures. Their proof-of-concept contributed to identifying duplicated code, improving system maintainability. For future directions, the author recommended using machine learning algorithms to fully automate the process and blackbox/whitebox testing methods to verify and validate the extracted microservices.

Al-Debagy et al. [34] decomposed monolithic into microservices architecture using a neural network model (code2vec). Their findings showed better results compared to other algorithms. Besides, authors validate their results by using quality measuring such as message level (CHM) and cohesion at domain level (CHD). Future directions may include further development and testing of the proposed model with other programming languages, as well as training.

Jin et al. [35] proposed the Functionality-oriented Service Candidate Identification (FoSCI) method, which uses a search-based functional atom grouping technique to identify service candidates from a monolithic system's execution traces. The authors assessed their method with an 8-metric service evaluation suite to analyze functionality, modularity, and evolvability. Additionally, the authors evaluated FoSCI against other methods (LIMBO, WCA, and MEM) to evaluate the impact of execution trace coverage on performance. Their results indicate that FoSCI outperforms the compared methods. However, their method prioritizes functionality over other quality attributes such as performance, security, and reliability, which could potentially benefit from future expansion.

Desai et al. [36] introduced a Graph Neural Network method for refactoring monolithic systems, termed COGCN (Clustering and Outlier-Aware Graph Convolution Net). This COGCN combines node representation, outlier detection, and clustering into a cohesive framework. To judge the quality of the clusters, four structural and desired metrics: modularity, structural modularity, non-extreme distribution (NED), and interface number (IFN) were used. Their results showed that COGCN works better than other methods like GCN, Node2Vector, and DeepWalk by improving the quality of clusters and finding outliers. Future work will include automatically determining microservice numbers and addressing procedural programming languages.

Kalia et al. [37] developed an approach called Mono2Micro. It used well-defined business use cases, spatio-temporal decomposition, and run-time call relations to functionally separate application classes. This method works better than other methods in well-defined metrics that are specific to the domain, such as FoSCI, MEM, CoGCN, and Bunch [17], [35], [36], [38] . Their tool was evaluated by using multiple criteria to verify its efficiency. Future directions focus on elaborating on the quality criteria, providing further assistance to develop effective use cases for practitioners, and generalizing the findings by supporting different programming languages.

Francisco et al. [39] performed a review of 20 papers addressing the migration process to microservices. Their results indicated that the majority of solutions rely on design

aspects, system dependency graphs, and clustering algorithms. Moreover, the majority of the studies depend on graphs and categorize them into microservices; 70% focus on web-based systems, mostly using Java as the main programming language. However, their study faces constraints due to its reliance on a single search engine and individual researcher bias. Furthermore, it advocates for a wider range of migration methodologies and database migration investigations.

Justas et al. [40] concentrated on the obstacles and approaches for migrating software to microservices by reviewing and analyzing different migration techniques; however, they didn't mention any particular evaluation metrics used to assess their review. Their results emphasized that the migration process is complex and expensive, with no one-size-fits-all solution. Future research includes a focus on developing standardized migration techniques to address the various issues posed by legacy systems.

Ren et al. [41] integrated static and dynamic analysis to examine the characteristics of monolithic systems. They employed function clustering to facilitate migration and hierarchical clustering to identify microservice candidates. They tested their method by comparing performance across four benchmark applications and validating the migration algorithm with 12 industrial and open-source applications. Their findings showed that their proposed method effectively migrates monolithic applications to microservices with high accuracy and low performance cost. However, the incomplete static analysis due to missing function invocations and user interactions may still limit the completeness of the migration.

Fritzsch et al. [42] conducted a qualitative study using semi-structured interviews in the context of migration to microservices. The authors conducted 16 comprehensive interviews with specialists from 10 software companies across 14 migration cases. Their finding indicated that the biggest motivations for migration were maintainability and scalability. Furthermore, many organizations choose a complete rebuild instead of a codebase breakdown. Principal problems included identifying the appropriate service cut and developing proficiency in emerging technologies. However, the sample procedure, which focuses only on 14 instances and individuals located in Germany, constrains their research and affects its generalizability.

Kalske et al. [43] performed a literature review focusing on architectural migration and associated challenges. Their findings indicate that organizations use microservices to mitigate complexity, enhance scalability, and resolve code ownership challenges. Nonetheless, restructuring and decoupling the tightly coupled monolith remain a significant challenge. However, their review lacks credibility and verification of their results, and there were no guidelines for conducting the review. Their future directions include understanding how various organizational structures influence the effectiveness of microservice adoption.

Vainio et al. [44] combined a case study and literature review to extract functionality from a monolithic system, then used microservices to demonstrate realistic real-world benefits and challenges for the migration process. Their findings showed significant advantages in scalability, maintainability, and the ability to use several programming languages for different services. Furthermore, the independent deployment of microservices improved system stability and performance. However, their case study did not thoroughly examine the architectural complexity and security implications of managing several microservices in extensive systems, indicating a need for additional research on these topics.

Eski et al. [45] employed a graph-based clustering methodology utilizing both static and evolutionary code coupling to derive microservices from monolithic applications. They evaluated their method by assessing two projects, which revealed an 89% success rate; however, certain services necessitated manual sub-clustering owing to their size. They planned to continue their research by adding weighted graph edges to their method, automating sub-clustering thresholds, and testing it on more projects using different algorithms.

Su et al. [46] conducted a systematic literature review by examining 32 studies to obtain insights into how software companies migrated from microservices back to monolithic architectures. Their study identified cost, complexity, scalability, performance, and organizational factors as the primary reasons for reverting. Future directions entail investigating the phenomenon across various industries and conducting empirical assessments to generalize the results.

Bandara et al. [47] developed a toolkit that utilized a fitness iterative function to identify microservices within monolithic systems. Their toolkit employed service quality metrics, including functionality, composability, self-containment, and usage. They compared the tool to manual microservice identification. Their results showed that their tool produced microservices like manual identification. Future directions include support for more programming languages and architectural patterns, machine learning models, and context knowledge to improve microservice extraction.

Michael Ayas et al. [48] examined the migration to microservices through interviews with 19 individuals from 16 organizations. Their results showed that the migration process is iterative and takes place on two levels: architectural and system-level. Moreover, they categorize key activities into four phases: designing architecture, altering the system, setting up supporting artifacts, and implementing technical artifacts. The study acknowledges the researchers' bias and the sample's representativeness. Future directions include examining more migration paths and validating results across organizations.

Taibi et al. [49] introduced a process-mining framework to aid in the decomposition of monolithic systems. Their framework identifies business processes based on log traces, clusters similar processes, and uses metrics to evaluate decomposition quality. A software architect validated their framework by comparing their results to a manual decomposition. Their framework revealed decomposition alternatives and software issues that manual analysis missed in an industrial case study. Future directions include automating the process, validating the methodology, and integrating patterns for microservices connectivity.

Silva et al. [50] conducted two case studies. These studies identified migration steps and challenges from legacy to

microservices. Their findings revealed four main issues: 1) identifying functionalities in large modules; 2) defining optimal microservice boundaries; 3) choosing features to convert into microservices; and 4) analyzing candidate microservice granularity and cohesion. Future steps include refining and supporting practical guidelines for migrating to microservice architectures, as well as surveying industry practitioners to learn more about migration challenges.

Kholy et al. [51] introduced the managing database for microservices architecture (MDMA) framework. Their framework indicates that MDMA significantly reduces execution time and data transfer size compared to a centralized approach and exhibits superior performance as the volume of requests increases. Furthermore, it improved flexibility and resilience, reducing the impact of service failures and facilitating data transfer during service deployment. Subsequent research may entail evaluating the framework in different real-world environments and broadening its applicability to various microservice architectures.

Antunes et al. [52] investigated the migration of a real-world application to a micro-frontend architecture. Results showed that micro-frontends enhance flexibility, team scalability, and incremental migration. Nonetheless, they observed increased complexity in the management of dependencies, environments, debugging, and testing. Future studies should investigate other projects and implementation methodologies, with a focus on simplifying dependency management and integration testing.

Maria et al. [53] investigated the migration towards micro-frontends. Their study demonstrates how to successfully re-implement a single-page application (SPA) using micro-frontends architecture by adopting Webpack to bundle modules and Cypress for testing. Their findings indicated improvements in team collaboration, independent deployment, and performance. Further studies include managing dependencies, enhancing integration, and evaluating performance.

Fritzsch et al. [54] used SLR to classify 10 existing approaches based on decomposition techniques. Their findings identified a lack of universally applicable approaches with adequate tool support. Future research will combine static code analysis with runtime data, create decomposition quality metrics, and automate the migration process.

Lauretis [55] presented a strategy for migrating to microservice. The author highlighted potential benefits such as improved scalability, maintainability, and evolvability for companies. Their strategy consists of five steps, including function analysis, business identification, business analysis, assigning functionalities, and creating microservices. However, their strategy is still in its early stages. Future directions include automating migration process and testing the strategy on real systems to gather performance and statistical data.

Nunes et al. [56] used transactional contexts to propose a microservices migration strategy. They used static code analysis to identify domain entities and applied a clustering algorithm to group them. Comparison with expert decompositions yields promising results. Future work will enhance the method and broaden the results, while limiting its applicability to specific frameworks and tools.

Santos et al. [57] proposed a complexity metric to measure the transition to microservices based on four similarity measures that examine entity decomposition. These measures focus on read and write sets, access sequences, and the cost of relaxing transactional consistency. Their metric was evaluated using three monolithic systems. Their complexity metric successfully identified the most complex decompositions. Future directions include using a combination of dynamic and static data, as well as further experimentation with a wider variety of systems.

Ma et al. [58] proposed Microservices Identification using Analysis for Database Access (MIADA) to account for the importance of "Database Per Service" in microservices design. Their approach facilitates clustering service endpoints for microservice identification. Two service-oriented software projects (PlanApproval and CoCoME) evaluated their results, demonstrating that MIADA can successfully recommend service endpoint clusters for microservice. Future directions include generating results and enhancing MIADA.

Ghofrani et al. [59] categorized migration challenges into inertia, anxiety, and context from 17 semi-structured expert interviews. The most significant barrier was migration anxiety, followed by inertia and context. Furthermore, they provided suggestions to overcome these obstacles. Future directions include evaluating their solutions and creating quality metrics.

Ghofrani et al. [60] conducted a survey of industry experts to identify challenges in microservices architecture. They identified critical challenges, including lack of notations, methods, and frameworks for microservices design, as well as insufficient tools for selecting third-party artifacts. In addition, they prioritized security, response time, and performance over resilience and fault tolerance. Future work includes expanding the scope and suggesting solutions for these gaps.

Razzaq et al. [61] provided a systematic mapping study on the migration towards microservices. Their review emphasized key benefits such as: including independent deployment, scalability, and lightweight mechanisms after migration. Besides, identifies migration challenges and success factors that help guide migration strategies. Future directions include, identify effective solutions for these areas, providing more in-depth research for microservice in emerging technologies.

V. RESULTS

As shown in Fig. 4, this SLR analyzed 50 studies published between 2018 and 2024 that focused on the migration process from monolithic to microservice. Furthermore, Table IV provides an overview of the findings from this literature research, covering the following topics:

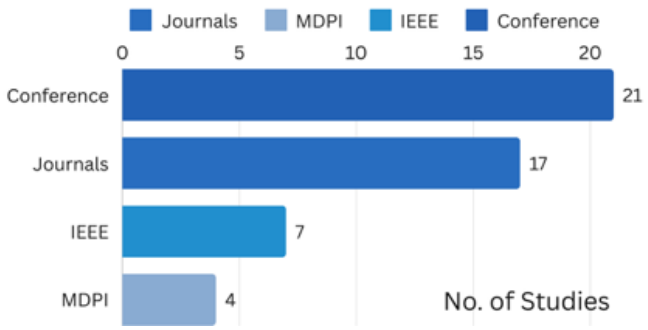

Fig. 4. Classification of studies examined in this SLR.

A. AI Algorithms and other Techniques:

Few authors emphasized providing proofs of concepts to support their ideas and provide insights for the migration process. Some studies conducted real-world industry case studies, attempting to assess and track the differences in performance, effort, scalability, and maintainability between monolithic and microservice architectures and drawing comparisons between them. Researchers are conducting extensive research on AI and other algorithmic techniques, including the indicator-based evolutionary algorithm (IBEA), neural networks, search-based algorithms, clustering algorithms, coupling, and cohesion. Additionally, some studies use SLR to improve their understanding of the microservice architecture. Fig. 5 presents the study distribution, demonstrating the use of AI and other clustering techniques in 18 studies, SLR in 13 studies, proof of concepts in 8 studies, and case studies in the remaining studies.

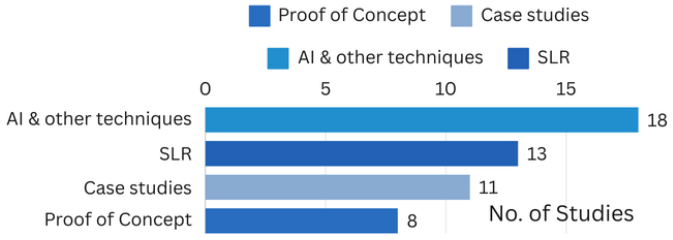

Fig. 5. Distribution of studies based on methodology.

B. Evaluation Techniques:

Major studies were evaluated and validated using a variety of techniques, including empirical assessment using a software system benchmark, the ISO/IEC 25012:2008 standard's definition of quality attributes, performance, scalability, and reliability comparisons, and architecture design comparisons. Other studies utilized cohesion at the domain level (CHD), conducted interviews with software industry experts, and utilized Postman tools for evaluation and testing. Fig. 6 presents the evaluation category.

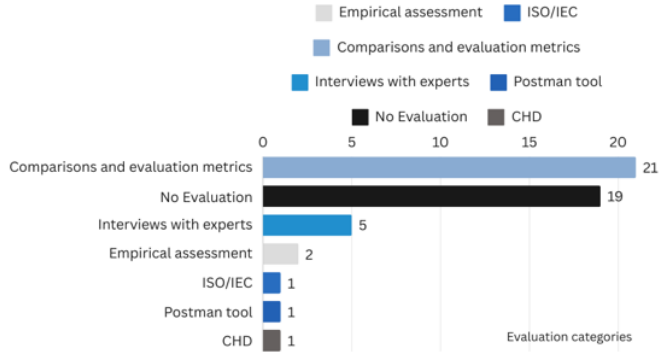

Fig. 6. Categorization of evaluations methods examined in this SLR.

C. Limitation and Future Work

Most studies in the software migration field, particularly those focusing on microservices, lack validation and verification of their models for real-world software systems, leading to a deficiency in generalizing the results of these studies. This gap emphasizes the need for further empirical research and actual implementations to test migration models in varied, real-world scenarios to ensure their robustness and reliability for wider use. The following is a list of future directions and areas for analysis and investigation:

1) Consider non-functional criteria.
2) Perspectives of software architects and developers.
3) Automated tools are needed for manual decomposition.
4) Generalizing results by testing on real-world industry.
5) The impact of cloud servers on the migration process is significant. Hyperscale's such as Azure, AWS, and GCP offer unique features like scalability options, flexible resource allocation, and Devops tools.
6) Adopt micro-frontend alongside microservices.

The subsequent discussion and answers correspond to the research questions outlined in Section III:

RQ1: There is a significant shortage of research on AI and ML methods in the software migration to microservices domain. Few studies are employing algorithms like the indicator-based evolutionary algorithm (IBEA), search-based techniques, and clustering algorithms. Other studies rely on comparisons and expert inputs to assess software quality criteria. The remaining studies offer proof-of-concepts but do not provide a comprehensive solution. A comprehensive solution would involve an automated method or algorithm that utilizes AI to simplify the transition from monolithic to microservice processes while maintaining performance, scalability, and availability. Additionally, real-world industry software systems must evaluate the solutions.

RQ2: The most significant challenges and obstacles in this domain arise from how software warehouses and architects

break down the logic and databases of software systems into dozens of interconnected microservices, which complicates the assurance of effective communication, data consistency, and the preservation of the system's overall integrity. Furthermore, it's crucial to consider the manner in which these services will interact and coordinate with each other. In addition, microservice architecture must meet non-functional criteria like performance, scalability, security, and reliability. As suggested in [7], AI can assist by using powerful ML techniques such as NLP to analyze the code, documentation, and system requirements, as well as complex clustering techniques to decompose the system logics into multiple services. To address non-functional criteria, it may be beneficial to monitor and utilize technologies, such as cloud computing and DevOps methodologies.

RQ3: The evaluation of software migration from monolithic to microservices is mostly based on comparing the two architectures and soliciting experts' opinions and recommendations. Despite this, AI models and other algorithms need to be rigorously evaluated and tested in the real industry to gather feedback from organizations and end users, enabling us to monitor performance and obtain relevant insights.

VI. Conclusion

This study conducted a systematic literature review to explore the key challenges, obstacles, and improvements in the software industry, specifically the migration from monolithic to microservice architecture. While there is growing interest in developing AI and other algorithmic techniques to facilitate this migration process, the field remains in its early stages, lacking comprehensive, end-to-end solutions that address the full complexity of the process. Most existing studies focus on theoretical frameworks, proof-of-concepts, and expert judgments rather than verifying and validating these approaches through real-world implementation.

Table IV provides a summary of this literature review, which includes a list of studies, each describing the methods used, the evaluation technologies, the results, their limitations, and future work. Section V addresses detailed answers to the three research questions. Both would assist software developers and other authors by providing the foundation for a comprehensive solution for addressing migration challenges.

The results of this literature study highlight a significant gap in the creation of effective migration tools and the validation of these solutions within the software industry. This emphasizes the need for future research to focus on the development, evaluation, and validation of automated tools in real-world environments. Furthermore, the migration process must emphasize non-functional requirements such as performance, scalability, reliability, and security. Integrating DevOps methodologies with cloud computing concepts is essential for optimizing the advantages of microservice architectures.

By identifying these gaps, this review emphasizes the importance of addressing both the theoretical and practical dimensions of the migration process, providing a roadmap for future research aimed at enhancing the efficiency, effectiveness, and scalability of software migrations. Resolving these challenges will ultimately enhance the industry's ability to migrate from monolithic systems to modern microservice-based architectures.

TABLE IV. A Concise Summary of the Research Findings from the Literature

R	Techniques used	Results	Limitations and future work
[1]	IBEA	Empirical testing showed the proposed approach is more effective.	Considering non-functional and software evaluation using.
[19]	Proof-of- concept	Migrate monolithic data storage to microservice can be applied.	Automation is needed, and the results are not generalizable.
[18]	Literature Review	Patterns for microservices. Microservices vs. monolithic pros/cons.	There is a need for evaluation, automatic decomposition.
[20]	Combined approaches	Customization-driven migration can guide monoliths into SaaS.	Generalizing results and automating the migration process.
[21]	Literature Review	List of challenges and benefits that arise in the migration process.	Analyze architectures pros and cons and automate migration.
[11]	Case Study	Modular monoliths aid migration, consistency, and communication.	Generalize and apply results to other case studies.
[12]	Case Study	Monolithic outperforms microservices, Java is better at computation.	Increase system complexity and deploy to various clouds.
[22]	Case Study	Their model optimizes development processes.	Generalizing the result required model improvement.
[8]	Case Study	Microservices improved scalability and solved the monolithic issues.	Expanding proof of concepts to generalize the results.
[10]	Search-based techniques	Promising strategy could be applied to other monolithic systems.	The model needs to be configurable and generalized.
[23]	Case Study	The migration affects stakeholders. Upload time affects deployment.	Cloud servers' application support and generalize the findings.
[24]	Case Study	Monolithic suits small-medium apps, but microservices are scalable.	Real-world testing. Ensuring security among microservices.
[25]	Literature Review	Microservices migration in its early stages, with few methods exists.	Consider deployment and microservice identification metrics.
[26]	Case Study	Microservices boost hardware efficiency, productivity, and cost.	Develop migration tools is needed and secure microservices.
[27]	Proof-of- concept	The conceptual model solve system breakdown migration steps.	Software business logic extraction tool need.

[28]	Decision framework	A complete microservices adoption measuring set for enterprises.	Validating framework and adding cloud-native technologies.
[7]	Clustering techniques	Microservices work better and stress business processes.	Using AI to identify microservices and security concerns.
[29]	Coupling and cohesion	The suggested metrics criteria are better for industrial use.	Evaluates microservice assessment using real-world case studies.
[30]	Literature Review	Emphasize automated identification and evaluation standards.	Studies focused on specific challenges and lack migration tools.
[31]	Literature Review	The study shows that categorization helps industrial professionals.	Grouping services by type may help generalize results.
[32]	Clustering techniques	Results showed the proposed model outperformed others.	Generalizing the results.
[33]	Proof-of- concept	The model found duplication and improved maintainability.	Automation using ML and black/white box for validation.
[34]	Neural network	The proposed algorithm performed better than others.	Testing the model in other languages and training on source codes.
[37]	Clustering techniques	Mono2Micro outperforms other methods.	Elaborate on the quality criteria and develop effective use cases.
[35]	Search-based techniques	FoSCI is better in service quality, functionality, and modularity.	Performance, security, and reliability are ignored.
[36]	Clustering techniques	COGCN improves cluster quality and outlier identification.	Determining microservice numbers and procedural languages.
[16]	Clustering techniques	FoME provides cohesive microservices with looser coupling.	Ensure the use of high-quality test cases and automate the process.
[39]	Literature Review	Most research used design, dependency graphs, and clustering.	Biased results and more migration methods are advocated.
[40]	Literature Review	Migration is complicated and expensive, with no single solution.	Standard migration methods to address legacy system issues.
[41]	Clustering techniques	Their microservice migration method was accurate with low cost.	Generalizing the results by adopting user interactions.
[42]	Qualitative study	Maintenance and scalability encouraged migration.	Generalizability is limited by the 14-person German sample.
[43]	Literature Review	For simplicity, scalability, and ownership, firms use microservices.	Their results lack credibility and verification.
[44]	Case Study	Microservices are scalable, maintainable, stable, and multilingual.	Security and architectural complexity need more research.
[45]	Clustering techniques	Their method revealed 89% success rate of extracted microservice.	Testing on different clustering algorithms to enhance the model.
[46]	Literature Review	Cost, complexity, and organization drive monolithic Reverting.	Examining the phenomenon across industries and generalizing.
[47]	Toolkit	Toolkit produced microservices like manual identification.	Supporting more languages, patterns, ML models, and context.
[48]	Qualitative study	Migration occurs on two levels: architectural and system-level.	Researcher bias and sample representativeness affect findings.
[49]	Process-mining	Framework outperforms manual analysis in industrial case studies.	Automating the process, validating, and integrating more patterns.
[50]	Case Study	Module's size, boundaries, features, and cohesion are challenges.	Survey industry specialists and refine microservice migration.
[51]	MDMA framework	MDMA improve runtime, data transfer, flexibility.	Testing the framework in real-world and expanding its use.
[52]	Case Study	Micro-frontends improve migration, scalability, and flexibility.	Dependencies, debugging, and testing require further research.
[53]	Case Study	SPA enhances deployment, collaboration, and performance.	Managing dependencies, enhancing integration and performance.
[54]	Literature Review	Lack of universally approaches with adequate tool support.	Automate migration with static code analysis and runtime data.
[55]	Proof-of- concept	Improved Company scalability, maintainability, and evolution.	Automating migration process and testing on real systems.
[56]	Clustering techniques	Comparison with expert decompositions yields promising results.	Improve the method and generalize the results.
[57]	Clustering techniques	Their metric successfully identified the complex decompositions.	Using dynamic and static data, along with further investigation.
[58]	Clustering techniques	MIADA can successfully recommend service endpoint clusters.	Future directions include generating results and enhancing MIADA.
[59]	Qualitative study	Anxiety, inertia, and context were the most barriers in migration.	Evaluating their proposed solutions and creating quality metrics.
[60]	Literature Review	Microservices design notations and method were critical challenges.	Expanding the scope and suggesting solutions for these gaps.
[61]	Literature Review	Emphasized independent deployment, scalability, and lightweight.	Provide research in emerging technologies to find solutions.

REFERENCES

[1] K. Sellami, A. Ouni, M. A. Saied, S. Bouktif, and M. W. Mkaouer, "Improving microservices extraction using evolutionary search," Inf Softw Technol, vol. 151, Nov. 2022, doi: 10.1016/j.infsof.2022.106996.

[2] S. Newman, "Building Microservices SECOND EDITION Designing Fine-Grained Systems."

[3] F. Ponce, J. Soldani, H. Astudillo, and A. Brogi, "Smells and Refactorings for Microservices Security: A Multivocal Literature Review," Apr. 2021, [Online]. Available: http://arxiv.org/abs/2104.13303

[4] M. G. de Almeida and E. D. Canedo, "Authentication and Authorization in Microservices Architecture: A Systematic Literature Review," Applied Sciences (Switzerland), vol. 12, no. 6, Mar. 2022, doi: 10.3390/app12063023.

[5] I. Saidani, A. Ouni, M. W. Mkaouer, and A. Saied, "Towards Automated Microservices Extraction Using Muti-objective Evolutionary Search," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer, 2019, pp. 58–63. doi: 10.1007/978-3-030-33702-5_5.

[6] S. Li et al., "A dataflow-driven approach to identifying microservices from monolithic applications," Journal of Systems and Software, vol. 157, Nov. 2019, doi: 10.1016/j.jss.2019.07.008.

[7] M. Daoud, A. El Mezouari, N. Faci, D. Benslimane, Z. Maamar, and A. El Fazziki, "A multi-model based microservices identification approach," Journal of Systems Architecture, vol. 118, Sep. 2021, doi: 10.1016/j.sysarc.2021.102200.

[8] M. Mazzara, N. Dragoni, A. Bucchiarone, A. Giaretta, S. T. Larsen, and S. Dustdar, "Microservices: Migration of a Mission Critical System," IEEE Trans Serv Comput, vol. 14, no. 5, pp. 1464–1477, 2021, doi: 10.1109/TSC.2018.2889087.

[9] P. Di Francesco, P. Lago, and I. Malavolta, "Migrating Towards Microservice Architectures: An Industrial Survey," in Proceedings - 2018 IEEE 15th International Conference on Software Architecture, ICSA 2018, Institute of Electrical and Electronics Engineers Inc., Jul. 2018, pp. 29–38. doi: 10.1109/ICSA.2018.00012.

[10] W. K. G. Assunção et al., "A Multi-Criteria Strategy for Redesigning Legacy Features as Microservices: An Industrial Case Study," in Proceedings - 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2021, Institute of Electrical and Electronics Engineers Inc., Mar. 2021, pp. 377–387. doi: 10.1109/SANER50967.2021.00042.

[11] D. Faustino, N. Gonçalves, M. Portela, and A. Rito Silva, "Stepwise migration of a monolith to a microservice architecture: Performance and migration effort evaluation," Performance Evaluation, vol. 164, May 2024, doi: 10.1016/j.peva.2024.102411.

[12] G. Blinowski, A. Ojdowska, and A. Przybylek, "Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation," IEEE Access, vol. 10, pp. 20357–20374, 2022, doi: 10.1109/ACCESS.2022.3152803.

[13] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering - A systematic literature review," Jan. 2009. doi: 10.1016/j.infsof.2008.09.009.

[14] Y. Xiao and M. Watson, "Guidance on Conducting a Systematic Literature Review," Mar. 01, 2019, SAGE Publications Inc. doi: 10.1177/0739456X17723971.

[15] Z. Bai and T. Wasson, "Conducting Systematic Literature Reviews in Information Systems: An Analysis of Guidelines," 2019. [Online]. Available: https://www.researchgate.net/publication/340686721

[16] W. Jin, T. Liu, Q. Zheng, D. Cui, and Y. Cai, "Functionality-Oriented Microservice Extraction Based on Execution Trace Clustering," in Proceedings - 2018 IEEE International Conference on Web Services, ICWS 2018 - Part of the 2018 IEEE World Congress on Services, Institute of Electrical and Electronics Engineers Inc., Sep. 2018, pp. 211–218. doi: 10.1109/ICWS.2018.00034.

[17] G. Mazlami, J. Cito, and P. Leitner, "Extraction of Microservices from Monolithic Software Architectures," in Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017, Institute of Electrical and Electronics Engineers Inc., Sep. 2017, pp. 524–531. doi: 10.1109/ICWS.2017.61.

[18] V. Velepucha and P. Flores, "A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges," IEEE Access, vol. 11, pp. 88339–88358, 2023, doi: 10.1109/ACCESS.2023.3305687.

[19] J. Kazanavičius, D. Mažeika, and D. Kalibatienė, "An Approach to Migrate a Monolith Database into Multi-Model Polyglot Persistence Based on Microservice Architecture: A Case Study for Mainframe Database," Applied Sciences (Switzerland), vol. 12, no. 12, Jun. 2022, doi: 10.3390/app12126189.

[20] E. T. Nordli, S. G. Haugeland, P. H. Nguyen, H. Song, and F. Chauvel, "Migrating monoliths to cloud-native microservices for customizable SaaS," Inf Softw Technol, vol. 160, Aug. 2023, doi: 10.1016/j.infsof.2023.107230.

[21] V. Velepucha and P. Flores, "Monoliths to microservices-Migration Problems and Challenges: A SMS," in Proceedings - 2021 2nd International Conference on Information Systems and Software Technologies, ICI2ST 2021, Institute of Electrical and Electronics Engineers Inc., Mar. 2021, pp. 135–142. doi: 10.1109/ICI2ST51859.2021.00027.

[22] A. Bastidas Fuertes, M. Pérez, and J. Meza, "Transpiler-Based Architecture Design Model for Back-End Layers in Software Development," Applied Sciences (Switzerland), vol. 13, no. 20, Oct. 2023, doi: 10.3390/app132011371.

[23] Teguh Prasandy, Titan, Dina Fitria Murad, and Taufik Darwis, "Migrating Application from Monolith to Microservices," in 2020 International Conference on Information Management and Technology (ICIMTech), Bandung, Indonesia: IEEE, 2020, pp. 726–731. doi: 10.1109/ICIMTech50083.2020.9211252.

[24] A. B. Fondazione et al., "Migration from Monolith to Microservices : Benchmarking a Case Study", doi: 10.13140/RG.2.2.27715.14883.

[25] Y. Abgaz et al., "Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review," IEEE Transactions on Software Engineering, vol. 49, no. 8, pp. 4213–4242, Aug. 2023, doi: 10.1109/TSE.2023.3287297.

[26] F. Tapia, M. ángel Mora, W. Fuertes, H. Aules, E. Flores, and T. Toulkeridis, "From monolithic systems to microservices: A comparative study of performance," Applied Sciences (Switzerland), vol. 10, no. 17, Sep. 2020, doi: 10.3390/app10175797.

[27] D. Kuryazov, D. Jabborov, and B. Khujamuratov, "Towards Decomposing Monolithic Applications into Microservices," in 14th IEEE International Conference on Application of Information and Communication Technologies, AICT 2020 - Proceedings, Institute of Electrical and Electronics Engineers Inc., Oct. 2020. doi: 10.1109/AICT50176.2020.9368571.

[28] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, "From monolithic systems to Microservices: An assessment framework," Inf Softw Technol, vol. 137, Sep. 2021, doi: 10.1016/j.infsof.2021.106600.

[29] M. H. Hasan, M. Hafeez Osman, N. I. Admodisastro, and S. Muhammad, "From Monolith to Microservice: Measuring Architecture Maintainability," 2023. [Online]. Available: www.ijacsa.thesai.org

[30] I. Oumoussa and R. Saidi, "Evolution of Microservices Identification in Monolith Decomposition: A Systematic Review," IEEE Access, vol. 12, pp. 23389–23405, 2024, doi: 10.1109/ACCESS.2024.3365079.

[31] M. Abdellatif et al., "A taxonomy of service identification approaches for legacy software systems modernization," Journal of Systems and Software, vol. 173, Mar. 2021, doi: 10.1016/j.jss.2020.110868.

[32] J. Li, H. Xu, X. Xu, and Z. Wang, "A Novel Method for Identifying Microservices by Considering Quality Expectations and Deployment Constraints," https://doi.org/10.1142/S021819402250019X, vol. 32, no. 3, pp. 417–437, Apr. 2022, doi: 10.1142/S021819402250019X.

[33] M. H. Gomes Barbosa and P. H. M. Maia, "Towards Identifying Microservice Candidates from Business Rules Implemented in Stored Procedures," in Proceedings - 2020 IEEE International Conference on Software Architecture Companion, ICSA-C 2020, Institute of Electrical and Electronics Engineers Inc., Mar. 2020, pp. 41–48. doi: 10.1109/ICSA-C50368.2020.00015.

[34] O. Al-Debagy and P. Martinek, "A Microservice Decomposition Method Through Using Distributed Representation Of Source Code," Scalable Computing, vol. 22, no. 1, pp. 39–52, 2021, doi: 10.12694:/scpe.v22i1.1836.

[35] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service Candidate Identification from Monolithic Systems Based on Execution Traces," IEEE Transactions on Software Engineering, vol. 47, no. 5, pp. 987–1007, May 2021, doi: 10.1109/TSE.2019.2910531.

[36] U. Desai, S. Bandyopadhyay, and S. Tamilselvam, "Graph Neural Network to Dilute Outliers for Refactoring Monolith Application," Feb. 2021, [Online]. Available: http://arxiv.org/abs/2102.03827

[37] A. K. Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, and D. Banerjee, "Mono2Micro: A practical and effective tool for decomposing monolithic Java applications to microservices," in ESEC/FSE 2021 - Proceedings of the 29th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, Inc, Aug. 2021, pp. 1214–1224. doi: 10.1145/3468264.3473915.

[38] B. S. Mitchell and S. Mancoridis, "On the Automatic Modularization of Software Systems Using the Bunch Tool."

[39] P. Francisco, M. Gastón, and A. Hernán, Migrating from monolithic architecture to microservices: A Rapid Review. IEEE, 2019. doi: 10.1109/SCCC49216.2019.8966423.

[40] K. Justas and M. Dalius, "Migrating Legacy Software to Microservices Architecture," Institute of Electrical and Electronics Engineers, 2019, p. 32. doi: 10.1109/eStream.2019.8732170.

[41] Z. Ren et al., "Migrating web applications from monolithic structure to microservices architecture," in ACM International Conference Proceeding Series, Association for Computing Machinery, Sep. 2018. doi: 10.1145/3275219.3275230.

[42] J. Fritzsch, J. Bogner, S. Wagner, and A. Zimmermann, "Microservices Migration in Industry: Intentions, Strategies, and Challenges," in Proceedings - 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019, Institute of Electrical and Electronics Engineers Inc., Sep. 2019, pp. 481–490. doi: 10.1109/ICSME.2019.00081.

[43] M. Kalske, N. Mäkitalo, and T. Mikkonen, "Challenges When Moving from Monolith to Microservice Architecture," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer Verlag, 2018, pp. 32–47. doi: 10.1007/978-3-319-74433-9_3.

[44] M. Vainio and T. Antti-Pekka, "The benefits and challenges in migrating from a monolithic architecture into microservice architecture," 2021. [Online]. Available: http://www.cs.helsinki.fi/

[45] S. Eski and F. Buzluca, "An automatic extraction approach - Transition to microservices architecture from monolithic application," in ACM International Conference Proceeding Series, Association for Computing Machinery, 2018. doi: 10.1145/3234152.3234195.

[46] R. Su, X. Li, and D. Taibi, "From Microservice to Monolith: A Multivocal Literature Review †," Apr. 01, 2024, Multidisciplinary Digital Publishing Institute (MDPI). doi: 10.3390/electronics13081452.

[47] C. Bandara and I. Perera, "Transforming monolithic systems to microservices - An analysis toolkit for legacy code evaluation," in 20th International Conference on Advances in ICT for Emerging Regions, ICTer 2020 - Proceedings, Institute of Electrical and Electronics Engineers Inc., Nov. 2020, pp. 95–100. doi: 10.1109/ICTer51097.2020.9325443.

[48] H. Michael Ayas, P. Leitner, and R. Hebig, "The Migration Journey Towards Microservices," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer Science and Business Media Deutschland GmbH, 2021, pp. 20–35. doi: 10.1007/978-3-030-91452-3_2.

[49] D. Taibi and K. Systä, "From monolithic systems to microservices: A decomposition framework based on process mining," in CLOSER 2019 - Proceedings of the 9th International Conference on Cloud Computing and Services Science, SciTePress, 2019, pp. 153–164. doi: 10.5220/0007755901530164.

[50] H. H. O. S. Da Silva, G. F. De Carneiro, and M. P. Monteiro, "Towards a roadmap for the migration of legacy software systems to a microservice based architecture," in CLOSER 2019 - Proceedings of the 9th International Conference on Cloud Computing and Services Science, SciTePress, 2019, pp. 37–47. doi: 10.5220/0007618400370047.

[51] M. El Kholy and A. El Fatatry, "Framework for Interaction between Databases and Microservice Architecture," IT Prof, vol. 21, no. 5, pp. 57–63, Sep. 2019, doi: 10.1109/MITP.2018.2889268.

[52] F. Antunes, M. J. Dias De Lima, M. Antônio, P. Araújo, D. Taibi, and M. Kalinowski, "Investigating Benefits and Limitations of Migrating to a Micro-Frontends Architecture," 2024.

[53] A. Maria and C. Fulvio, "Exploring Software Architectural Transitions: From Monolithic Applications to Microfrontends enhanced by Webpack library and Cypress Testing," Jul. 2024.

[54] J. Fritzsch, J. Bogner, A. Zimmermann, and S. Wagner, "From Monolith to Microservices: A Classification of Refactoring Approaches," 2018. doi: arXiv:1807.10059.

[55] L. De Lauretis, "From monolithic architecture to microservices architecture," in Proceedings - 2019 IEEE 30th International Symposium Software Reliability Engineering Workshops, ISSREW 2019, Institute of Electrical and Electronics Engineers Inc., Oct. 2019, pp. 93–96. doi: 10.1109/ISSREW.2019.00050.

[56] L. Nunes, N. Santos, and A. Rito Silva, "From a monolith to a microservices architecture: An approach based on transactional contexts," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2019. doi: 10.1007/978-3-030-29983-5_3.

[57] N. Santos and A. Rito Silva, "A complexity metric for microservices architecture migration," in Proceedings - IEEE 17th International Conference on Software Architecture, ICSA 2020, Institute of Electrical and Electronics Engineers Inc., Mar. 2020, pp. 169–178. doi: 10.1109/ICSA47634.2020.00024.

[58] S. P. Ma, T. W. Lu, and C. C. Li, "Migrating Monoliths to Microservices based on the Analysis of Database Access Requests," in Proceedings - 16th IEEE International Conference on Service-Oriented System Engineering, SOSE 2022, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 11–18. doi: 10.1109/SOSE55356.2022.00008.

[59] J. Ghofrani and A. Bozorgmehr, "Migration to Microservices: Barriers and Solutions," 2019, pp. 269–281. doi: 10.1007/978-3-030-32475-9_20.

[60] J. Ghofrani and D. Lübke, "Challenges of Microservices Architecture: A Survey on the State of the Practice," 2018. [Online]. Available: http://ceur-ws.org/Vol-2072

[61] A. Razzaq and S. A. K. Ghayyur, "A systematic mapping study: The new age of software architecture from monolithic to microservice architecture—awareness and challenges," Mar. 01, 2023, John Wiley and Sons Inc. doi: 10.1002/cae.22586.