Migration of Monolithic Systems to Microservices using AI: A Systematic Mapping Study

Ana Martínez Saucedo[1,2,3] and Guillermo Rodríguez[2,3]

[1]Universidad Argentina de la Empresa (UADE), Instituto de Tecnología (INTEC)
Buenos Aires, Argentina

[2]CONICET
Buenos Aires, Argentina

[3]ISISTAN Research Institute (CONICET - UNICEN)
Tandil, Buenos Aires, Argentina

anmartinez@uade.edu.ar, guillermo.rodriguez@isistan.unicen.edu.ar

Abstract. *The popularity of microservices architecture has been increasing considerably because of its capacity to alleviate monolithic architecture issues. Nonetheless, the migration of monolith systems to microservices is a complex task. This work aims to analyze and characterize the state of the art on migrating monolithic applications towards microservices (semi-) automatically using Artificial Intelligence (AI) techniques by applying a systematic mapping methodology. Results showed that clustering is the preferred IA technique to decompose a monolith, cited by 63% out of the 22 reviewed studies. Moreover, the most prevailing input type used in migration techniques was source code (36.4%).*

1. Introduction

The monolithic architecture is based on the integral encapsulation of the functionalities of a single application executed by a single process [Koschel et al. 2017]. This has been the most predominant architecture in the scope of software development due to the fact a monolithic application is easy to implement and deploy [Koschel et al. 2017]. Nonetheless, the problems of monolithic architecture appear as the application increases its size [Fritzsch et al. 2019b, Koschel et al. 2017], causing high code coupling, difficulties in comprehending code, arduous maintenance, slow compilation process, and increased communication effort among project's stakeholders [Li et al. 2019]. Therefore, the microservice architecture was designed to provide solutions for the drawbacks of monolithic architecture. The main components of such an architecture are the microservices, which are defined by several authors as small, independent, and autonomous services that are capable of efficiently performing a clearly defined task [Fowler and Lewis 2014, Newman 2015]. In this context, applications built according to microservice architecture have several advantageous aspects, mainly module-level scalability, faster delivery, and maintenance [Jamshidi et al. 2018]. This is why several companies, such as Netflix, SoundCloud, and Amazon, have chosen to migrate from monolithic architecture to microservice architecture [Li et al. 2019]. However, migration from monolithic to microservices architectures fails to be classified as a simple process, since it involves complex issues that require strategies to be solved [Balalaie et al. 2015, Christoforou et al. 2017].

This work aims to characterize the migration of monolithic systems to microservices using Artificial Intelligence (AI) techniques based on the state of the art. The information for carrying out this work was acquired through a systematic mapping study: we selected 22 studies from an initial set of 855 papers to answer questions that analyze the identification of microservices techniques. The main results of this work are:

- Migration of monolithic systems to microservices is complex and techniques to achieve migration are varied in terms of supported programming languages, types of input, and granularity;
- Clustering is the most addressed AI technique in the literature to migrate monolithic applications towards microservices.

The remainder of the article is organized as follows. Section 2 describes related work. In Section 3, the research method is presented. In Section 4, the results are presented based on the research questions. Section 5 discusses the findings of this study. Threats to Validity are described in Section 6. Finally, Section 7 concludes the article with remarks and future work.

2. Related Work

The migration of monolith applications to microservices has been reviewed from different perspectives. Most of the studies have focused on migration techniques or activities and factors that motivate migrating toward a microservice architecture. From the perspective of the migration process, Fritzsch et al. [Fritzsch et al. 2019b] address monolith's dissociation, which consists of fragmenting the monolithic application into small services. This is a substantial and high-level task in terms of complexity, which is the reason why it can be considered the most important task for migration. Thus, the authors perform a systematic literature review in order to collect techniques available in the literature to identify microservices, in which ten main techniques were found. Similarly, Ponce et al. [Ponce et al. 2019] reached a classification of migration approaches through a rapid review: model-driven, static and dynamic analysis of code.

In the same direction, Fritzsch et al. [Fritzsch et al. 2019a] investigate the migration process adopted in the industry, outlining the lack of a semi-automated process to support migration. Therefore, companies rely on non-systematic approaches or manual functional decomposition approaches to perform migration. For this reason, Lapuz et al. [Lapuz et al. 2021] gather from the literature dynamic data collection tools that were (or could be) employed to assist in the migration of monolithic applications to microservices through dynamic analysis. Finally, Di Francesco et al. [Di Francesco et al. 2018] focus on the industry as well by conducting an industrial survey that aims to characterize the activities and challenges faced by practitioners when migrating towards microservices.

However, to our knowledge, no work has focused on analyzing and characterizing migration techniques that have used AI techniques to support monolith decomposition. Considering the complexity involved in migrating towards a microservice architecture, the automation of (most of) the migration process would alleviate practitioners in the task of identifying loosely coupled and highly cohesive microservices.

3. Research Method

The purpose of this work is to analyze and characterize the state of the art on migrating monolithic applications towards microservices (semi-) automatically using AI techniques.

To do so, we performed a systematic mapping study. To assist in the execution of the research protocol, the tool StArt[1] was used, which tracks and documents each decision made throughout each phase of the protocol.

The research objective was defined using part of the model GQM (Goal-Question-Metric) [Basili and Weiss 1984]: the objective is to analyze migration from monolithic applications to microservices cases, with the purpose of characterizing, with respect to microservice identification techniques, from the researchers' point of view, in the context of theoretical and applied research.

With the purpose of complying with the protocol, criteria detailed by PICO (Population, Intervention, Comparison, and Outcome) were determined [Schugerl et al. 2009]. Since this work aims for characterization, only the attributes Population, Intervention, and Outcome were used, thus a PICO extract is shown in Table 1. Therefore, two research questions (Q) were designed as follows:

RQ1 – Which AI techniques are used to migrate monolithic applications to microservices?

RQ2 – Which aspects of a monolithic application are processed as inputs by AI models?

PICO criteria	**Description**
Population	Microservices, Monolith
Intervention	Decomposition, Migration, Transformation, Move, Modernization, Refactoring
Comparison	Not applicable
Outcome	Process / Technique / Model / Method / Approach / Features / Characteristics

Table 1. PICO criteria.

3.1. Sources Selection

The selection of the sources used for the execution of this work was made through a series of criteria and guided by the works of Petersen et al. [Petersen et al. 2015] and Kitchenham et al. [Kitchenham and Brereton 2013]. The criteria set out to select database and index systems were a) being a relevant scientific database in software engineering, and b) content access availability. A total of 855 studies were retrieved between 2015 and 2023 and composed the initial pool. The selected sources were: ACM (23.3%), IEEE Xplore (4.9%), Scopus (11.2%), SpringerLink (55%), and Science Direct (5.6%).

3.2. Search Strategy

The search strategy aims to find clauses that meet the main questions suggested in the research. First, the search was automated on the sources already established. For the development of this work, there was no restriction on the publication date of the bibliographic materials analyzed.

The search strategy is divided into the following steps:

(i) Criteria declaration, search clause extraction, and search execution. A total of 855 studies were included;

[1] `https://www.lapes.ufscar.br/resources/tools-1/start-1`

(ii) Removal of duplicates, in which 764 studies were kept;
(iii) Execution of the first qualitative analysis of the results based on the inclusion and exclusion criteria. As a result, a total of 158 studies were kept;
(iv) Snowballing activity by extracting for each study its relevant references, 7 studies were added;
(v) Conduction of the second qualitative analysis based on the analysis of the results, keeping 163 studies;
(vi) Data extraction to solve the research questions proposed, in which 22 out of 163 studies addressed migration answered the questions by a) presenting a migration technique and the steps needed to perform it, and b) the migration technique is based on an AI algorithm.

3.3. Search Terms

The search terms were gathered according to the following keywords: microservices, microservice, microservice architecture, monolith, monoliths, decomposition, migration, transform and move. These were used in a search string, which is a set of keywords combined with boolean operators that will execute the search.

The search string used to perform this search was:

*(((microservices **OR** microservice **OR** "microservice architecture") **AND** (monolith **OR** monoliths)) **AND** (decomposition **OR** migration **OR** transform **OR** move **OR** legacy **OR** modernization **OR** refactoring))*

Since the exclusion criteria define that only studies in English are accepted, the terms used were written in English only. To provide the best result possible, two logical tests were applied to the string. In addition, metadata specification could have been applied. However, since the vocabulary is very particular to the Computer Science subject, it was disregarded to define it as the subject of the search.

3.4. Study Selection Criteria

For the selection of the bibliography resulting from the search, a series of inclusion and exclusion criteria were defined. It is important to note that there was no restriction on the search time. The inclusion and exclusion criteria adopted in this review are described below.

Inclusion criteria

- IC01: Study analyses the migration of monolithic systems to microservices
- IC02: Study presents an AI technique to identify microservices.

Exclusion criteria

- EC01: Duplicated study.
- EC02: Study is not written in English.
- EC03: Study published only as abstracts or prefaces of journals and events.
- EC04: Study is not available for download.
- EC05: Study is not peer-reviewed.
- EC06: Study is not related to the migration of monolithic systems to microservices.

3.5. Studies Selection Procedure

Titles and abstracts of the retrieved studies were analyzed to verify whether they are suitable for this work. From the analysis performed, it was possible to classify the studies with the assistance of the StArt tool as accepted, rejected, or duplicated. It was found 158 (18%) accepted, 606 (71%) rejected, and 91 (11%) duplicates.

Afterward, full texts of accepted studies were read to find answers to the questions made in the research protocol. It is important to notice that all accepted studies conform to the inclusion criteria and contribute to answering the questions.

4. Results

Migration techniques classification schema

In order to evaluate the works reviewed and make a comparative description among them, we have identified common criteria regarding relevant issues in the field of monolith to microservices migration:

- AI technique. This feature represents the AI technique used to decompose a monolith.
- Language support. This criterion deals with the monolith programming language that is supported by each migration technique.
- Input type. This feature evaluates which aspects of a monolithic application are required to perform migration, such as source code, log traces, and requirements, among others.
- Granularity. This criterion evaluates the unit of analysis used by each technique to approach migration, attempting to discover the quality of each decomposition obtained in terms of granularity.

A total of 22 studies were reviewed and characterized according to the aforementioned criteria as described in Table 2.

Table 2. Migration techniques.

Study	AI technique	Input type	Language support	Granularity
[Al-Debagy and Martinek 2019]	Clustering Word Embeddings	Documentation based (O)	Agnostic	API
[Baresi et al. 2017]	Clustering	Documentation based (O)	Agnostic	API
[Kamimura et al. 2018]	Clustering	Static	Java Cobol	Class
[Nunes et al. 2019]	Clustering	Static	Java	Class
[Pigazzini et al. 2019]	Topic Detection	Static	Java	Class
[Sellami et al. 2022]	Clustering	Static	Agnostic (Java)	Class
[Trabelsi et al. 2022]	Clustering SVM Word Embeddings	Static	Agnostic (Java)	Class

Table 2. Migration techniques (continued).

Study	AI technique	Input type	Language support	Granularity
[Nitin et al. 2022]	Label propagation	Static	Java	Class
[Brito et al. 2021]	Clustering Topic Detection	Static	Agnostic (Java)	Class
[Mathai et al. 2022]	Graph Neural Network	Static	Agnostic (Java)	Class
[Li et al. 2022]	Clustering	Documentation based (O) Dynamic Static	Agnostic (Java)	Method
[Kalia et al. 2021]	Clustering	Dynamic	Java	Class
[Liu et al. 2022]	Genetic algorithm	Dynamic	Agnostic (Java)	Class
[Jin et al. 2021]	Genetic algorithm	Dynamic	Agnostic (Java)	Class
[Bajaj et al. 2020]	Clustering	Dynamic	Agnostic	URI
[Eski and Buzluca 2018]	Clustering	Source control based Static	Agnostic	Class
[Mazlami et al. 2017]	Clustering	Source control based Static	Agnostic	Class
[Gysel et al. 2016]	Label propagation Clustering	Documentation based (A&D)	Agnostic	Nanoentity
[Ren et al. 2018]	Clustering	Dynamic Static	Agnostic (Java)	Method
[Cao and Zhang 2022]	Clustering	Dynamic Static	Agnostic (Java)	Method
[Matias et al. 2020]	Clustering	Data based Dynamic Static	Python	Class
[De Alwis et al. 2018]	Clustering	Data based Dynamic Static	PHP	Class

Research questions results

In order to answer RQ1 and RQ2, we have analyzed the criteria as mentioned above independently as described below.

By AI technique Among the AI-based techniques, a wide variety of algorithms have been employed to migrate towards microservices (Table 3). Nevertheless, clustering has been the most popular AI technique (63%) to identify microservices. Yet, varying clus-

tering algorithms have been proposed for the decomposition problem, which is described below according to the classification presented by Xu et al. [Xu and Tian 2015]:

- Affinity propagation-based: Affinity Propagation [Al-Debagy and Martinek 2019]
- Density-based: ϵ-DBSCAN [Sellami et al. 2022]
- Graph theory-based: heuristics based [De Alwis et al. 2018] and MST-based [Mazlami et al. 2017]
- Hierarchical-based: Girven-Newman [Gysel et al. 2016, Matias et al. 2020], Louvain [Brito et al. 2021, Li et al. 2022], Fast Community [Eski and Buzluca 2018], Hierarchical K-means [Ren et al. 2018], Leiden [Cao and Zhang 2022], SArF [Kamimura et al. 2018] and SLINK [Kalia et al. 2021]
- Partitioning-based: Fuzzy C-means [Trabelsi et al. 2022] and K-means [Bajaj et al. 2020]
- Unclassified: [Nunes et al. 2019, Baresi et al. 2017]

Considering unsupervised techniques, the NSGA-II genetic algorithm is employed by the two studies in the category [Liu et al. 2022, Jin et al. 2021], while topic detection algorithms such as LDA [Pigazzini et al. 2019, Brito et al. 2021] or SLDA [Pigazzini et al. 2019] are used to group lexical and structural information from the monolith. The Word Embeddings models chosen in the reviewed studies are Word2Vec [Al-Debagy and Martinek 2019, Trabelsi et al. 2022], FastText [Al-Debagy and Martinek 2019] and CodeBERT [Trabelsi et al. 2022]

AI Technique	**Algorithm**	**Studies**	**Percentage**
Unsupervised	Clustering	[Al-Debagy and Martinek 2019, Kamimura et al. 2018, Nunes et al. 2019, Kalia et al. 2021, Gysel et al. 2016, Matias et al. 2020, Ren et al. 2018, Sellami et al. 2022, Brito et al. 2021, Li et al. 2022, Bajaj et al. 2020, Trabelsi et al. 2022, Eski and Buzluca 2018, Cao and Zhang 2022, Mazlami et al. 2017, De Alwis et al. 2018, Baresi et al. 2017]	63.0%
	Genetic algorithms	[Liu et al. 2022, Jin et al. 2021]	7.4%
	Topic detection	[Brito et al. 2021, Pigazzini et al. 2019]	7.4%
	Graph Neural Network	[Mathai et al. 2022]	3.7%
Self-supervised	Word Embeddings	[Al-Debagy and Martinek 2019, Trabelsi et al. 2022]	7.4%
Semi-supervised	Label propagation	[Nitin et al. 2022, Gysel et al. 2016]	7.4%
Supervised	SVM	[Trabelsi et al. 2022]	3.7%

Table 3. AI technique classification of AI-based migration techniques.

By input type All migration techniques need one or more aspects of a monolithic application to obtain microservices candidates. Nonetheless, the reviewed studies have used as input different outputs of a software development cycle. Consequently, a characterization of input types is proposed in Table 4.

Input type	Development phase
Documentation based (A&D)	Analysis & Design
Data based, Static	Implementation
Documentation based (O), Dynamic, Source control based	Operation

Table 4. Input type classification.

In the Analysis and Design (A&D) phases of an application, documentation is the principal output. However, several types of documentation might be used to migrate a monolithic application towards microservices, namely use cases [Gysel et al. 2016], design documents [Li et al. 2022, Gysel et al. 2016], and Entity-Relationship design models [Li et al. 2022, Gysel et al. 2016]. During the Implementation phase, source code (referred to in this study as Static) is developed and databases and tables (referred to in this study as Data based) are created and set up.

Finally, in the Operation (O) phase, the monolith application is deployed and running. In this phase, operation documentation such as OpenAPI specifications [Al-Debagy and Martinek 2019, Baresi et al. 2017] and hardware resource dependencies [Li et al. 2022] might be required as input to perform the migration. On the other hand, a dynamic aspect of a running monolith might be represented as log traces [Kalia et al. 2021, Liu et al. 2022, Jin et al. 2021, Ren et al. 2018, Cao and Zhang 2022, Matias et al. 2020, De Alwis et al. 2018], hardware utilization metrics [Li et al. 2022] or web access logs [Bajaj et al. 2020]. Source control-based inputs include repositories and Git history of commits.

As described in Table 5, in the reviewed studies the static aspect of a monolithic application is preferred when performing migration, followed by the dynamic aspect. To use source code as input to a migration technique is required to have access to those files. Therefore, when source code is not available, other migration techniques rely on the dynamic facet of an application, which corresponds to outputs generated when the application is up and running.

To rely on the static or dynamic aspect of a monolith application only has advantages and disadvantages. On the one hand, there are tools to support the static analysis of a monolith through source code. On the other, analyzing a monolith dynamically allows the identification of dead code and cyclic dependencies, which could affect the performance of the proposed microservices architecture. Nonetheless, the static facet of an application may not be sufficient to understand the true nature of dependencies in runtime. Conversely, to analyze the dynamic aspect of an application a complete runtime log traces from each operation performed by a user is needed, which generally conforms to the application test suite. In this case, high test coverage is fundamental to encompass most functionalities.

Due to the aforementioned impediments of focusing on a single aspect of an application to perform migration, 45.4% of the studies propose a multi-facet analysis in which

the static aspect is always included. The most prevalent aspect to supplement the static aspect is source control: by analysing code repositories and change history it is allowed to identify code partitions that change together in time. However, it is important to note that source control on its own has not been used as input to obtain microservices candidates.

Other studies incorporated documentation into the analysis to reflect the true purpose of an application, which may diverge during the Implementation and Operation development phases. No studies include data as the only input to guide migration.

Input type	**Studies**	**Percentage**
Static	[Brito et al. 2021, Kamimura et al. 2018, Mathai et al. 2022, Nitin et al. 2022, Nunes et al. 2019, Pigazzini et al. 2019, Sellami et al. 2022, Trabelsi et al. 2022]	36.4%
Dynamic	[Bajaj et al. 2020, Jin et al. 2021, Kalia et al. 2021, Liu et al. 2022]	18.2%
Data based + Dynamic + Static	[De Alwis et al. 2018, Matias et al. 2020]	9.1%
Documentation based (O)	[Al-Debagy and Martinek 2019, Baresi et al. 2017]	9.1%
Dynamic + Static	[Cao and Zhang 2022, Ren et al. 2018]	9.1%
Source control based + Static	[Eski and Buzluca 2018, Mazlami et al. 2017]	9.1%
Documentation (A&D) + Dynamic + Static	[Li et al. 2022]	4.5%
Documentation based (A&D)	[Gysel et al. 2016]	4.5%

Table 5. Input type classification of migration techniques.

By language support Although the majority of the reviewed techniques are designed to support a monolith developed in any programming language (65.2%), a subset of them (53.3%) currently support Java monoliths due to the fact those techniques rely on tools that are only available for the Java language, for instance Wala [Ren et al. 2018] and Java Call Graph [Cao and Zhang 2022].

21.7% of the studies migrate Java monoliths to microservices, thus becoming Java the most popular monolith language to develop migration techniques. However, varied frameworks technologies are supported apart from Vanilla Java [Pigazzini et al. 2019]: Spring Framework [Kamimura et al. 2018], Java EE [Nitin et al. 2022, Kalia et al. 2021] and Fenix Framework used in conjunction with Spring Framework [Nunes et al. 2019].

By granularity When obtaining microservices candidates, each technique suggests migrating different types of elements to compose a microservice. Granularity denotes the unit of the element to be migrated, and the reviewed studies propose varying levels as described in Table 6.

Similarly to the aforementioned classification criteria, migration techniques consider levels of granularity that correspond to different phases in the software development lifecycle. During the Analysis & Design phase no code is available. In this phase, Nanoentities [Gysel et al. 2016] are proposed as migration elements that encode not only functionality but also data and its stores.

However, the vast majority of migration elements correspond to components that are available in the Implementation phase (81.7%): classes and methods. Choosing method level granularity when migrating towards microservices implies that methods of different or the same classes will compose a microservice. In the same fashion, class granularity involves moving whole classes to a microservice candidate.

While method granularity might be the most accurate unit of migration to choose in order to obtain cohesive and low coupled microservices, adjustments and merging needed to perform due to incorporating methods from different classes may take much effort considering the few tools available to semi-automatically support the process. On the contrary. the task of processing and analyzing classes might be alleviated thanks to tools available to perform, for example, call graph generation. This may explain why class granularity is the most chosen among migration techniques.

The second minority of studies (13.7%) has focused on elements made available during the Operation development phase. Those elements include URIs and APIs, which generally convey user-relevant functionality.

Studies	**Granularity level**	**Associated development phase**	**Percentage**
[Brito et al. 2021, Kalia et al. 2021, Liu et al. 2022, Jin et al. 2021, Eski and Buzluca 2018, Mazlami et al. 2017, Matias et al. 2020, Kamimura et al. 2018, Nunes et al. 2019, Pigazzini et al. 2019, Sellami et al. 2022, Trabelsi et al. 2022, Nitin et al. 2022, De Alwis et al. 2018, Mathai et al. 2022]	Class	Implementation	68.1%
[Li et al. 2022, Ren et al. 2018, Cao and Zhang 2022]	Method		13.6%
[Gysel et al. 2016]	Nanoentity	Analysis & Design	4.6%
[Bajaj et al. 2020]	URI	Operation	4.6%
[Al-Debagy and Martinek 2019, Baresi et al. 2017]	API		9.1%

Table 6. Granularity classification.

5. Discussion

This work aims to characterize the migration of monolithic systems to microservices by analyzing the identification of microservices techniques that are supported by AI models. Similarly to [Fritzsch et al. 2019b], we present a characterization of techniques that have been addressed in the literature. However, instead of proposing a technique classification

that considers which aspects of a monolith may aid in the migration (static code, metadata, workload data, and runtime environment), we also defined language support and granularity criteria to decompose a monolith to provide for finer-granularity analysis.

Results show that techniques proposed in the literature are varied and have trade-offs in terms of technological limitations or the need for manual intervention in the process. In effect, to uniformly assess the quality of a monolith decomposition, each technique would require the usage of an industrial and/or academic accepted metric against which decompositions could be evaluated. In the field of monolith to microservices migration, little consensus has been identified concerning how to evaluate a candidate microservices decomposition: whether decomposition should be evaluated against cohesion and coupling metrics only or include performance or even organizational metrics depends on the study focus. For this reason, it is difficult to assess if different AI algorithms perform better in terms of cohesion, coupling, performance, or organizational improvements.

Unsupervised techniques have been widely adopted to address monolith to microservices migration (81.5%) and in particular clustering algorithms that rely on a graph representation of the monolithic application are the most used in the literature, perhaps due to being a natural representation of the underlying software structure. While Natural Language Processing (NLP) algorithms have not been as widely researched as clustering algorithms, the exploration of the ability of Large Language Models (LLM) to identify cohesive and loosely coupled microservices based on the underlying domain knowledge represented by classes and methods could be addressed without prior input preparation by prompting source code directly to a code-specific LLM such as Code Llama. In this sense, this type of LLM would remove the need to focus on a specific programming language, as these models support a wide range of programming languages out of the box. Moreover, monolith knowledge coming from design documents or operational logs could be wired into these models by incorporating Retrieval Augmented Generation. Nonetheless, practitioners should consider the costs associated with running LLM locally, both due to licenses or appropriate hardware acquisition, as well as LLM challenges such as bias, information hallucinations, and explainability, among others [Zhao et al. 2023].

In line with the variety of metrics used to evaluate the performance of AI algorithms for the monolith decomposition task, and related to the lack of practical systems for benchmarking microservice-based architectures reported in [Di Francesco et al. 2019], the scarcity of a variety of open source projects in terms of size and complexity to validate results should be considered by researchers who want to propose new approaches to migrate monolithic systems to microservices. In this sense, the incorporation of synthetic projects created by LLM could also alleviate the task of evaluating migration techniques on monolithic applications of different sizes and domains.

6. Threats to Validity

We have considered the most common threats to validity of systematic literature reviews in Software Engineering reported in [Zhou et al. 2016] to account for how we mitigated them.

Construct validity To mitigate the threat of using inappropriate or incomplete search terms in the automatic search, we have tested our search string before executing the research protocol to ensure results were relevant. Moreover, we have included multiple databases and the StArt tool, which executes queries on the different sources to reduce

subjective errors during the search phase and ensure replicability. We have also complemented our search results by including the snowballing method.

Internal validity To reduce bias in study selection and data extraction, two researchers performed these tasks independently and then cross-analyzed decisions made about each study suitability. Moreover, we have documented each accept/reject decision.

External validity To ensure the generalizability of our results, we have automated our search and complemented it with the snowballing method. Moreover, we have included peer-reviewed papers only, and have not restricted the publication date of bibliographic materials analyzed.

Conclusion validity To allow for replicable results, we have defined a protocol that was cross-validated by two researchers. Each researcher worked independently and individual findings were discussed and cross-validated.

7. Final Remarks

In this work, an analysis of studies referring to the current state of the art regarding migration from monolithic systems to microservices using AI techniques is presented. In addition, this work proposed a characterization of migration techniques in terms of AI technique, input type, language support, and granularity. Results show that no single technique is completely effective when migrating to microservices: aspects of a monolith to feed into AI models or technologies constraints make it difficult to obtain a general migration process which could be applied to any monolith system regardless of the technologies it was developed with or the domain it is contained. Notwithstanding the wide range of AI-based migration techniques supported, clustering is the preferred one to obtain candidate microservices decompositions, although little consensus is identified in the selected studies considering the inputs required by each technique.

Considering the vast amount of peer-reviewed papers addressing monolithic systems to microservices migration analyzed in this work, as future work we are further advancing our research by incorporating works that include gray literature, such as blogs and websites, among other sources, in order to extend it as a multivocal literature review.

Acknowledgments

We acknowledge the financial support provided by CONICET and UADE under doctoral grant id 14120210200034CO and project PIP no.11220200100430.

References

Al-Debagy, O. and Martinek, P. (2019). A New Decomposition Method for Designing Microservices. *Periodica Polytechnica Electrical Engineering and Computer Science*, 63(4):274–281. Number: 4.

Bajaj, D., Bharti, U., Goel, A., and Gupta, S. C. (2020). Partial migration for re-architecting a cloud native monolithic application into microservices and faas. In Badica, C., Liatsis, P., Kharb, L., and Chahal, D., editors, *Information, Communication and Computing Technology*, pages 111–124, Singapore. Springer Singapore.

Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2015). *Migrating to Cloud-Native Architectures Using Microservices: An Experience Report*.

Baresi, L., Garriga, M., and De Renzis, A. (2017). Microservices Identification Through Interface Analysis. In De Paoli, F., Schulte, S., and Broch Johnsen, E., editors, *Service-Oriented and Cloud Computing*, Lecture Notes in Computer Science, pages 19–33, Cham. Springer International Publishing.

Basili, V. R. and Weiss, D. M. (1984). A Methodology for Collecting Valid Software Engineering Data. *IEEE Transactions on Software Engineering*, SE-10(6):728–738. Conference Name: IEEE Transactions on Software Engineering.

Brito, M., Cunha, J., and Saraiva, J. a. (2021). Identification of microservices from monolithic applications through topic modelling. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, SAC '21, page 1409–1418, New York, NY, USA. Association for Computing Machinery.

Cao, L. and Zhang, C. (2022). Implementation of domain-oriented microservices decomposition based on node-attributed network. In *2022 11th International Conference on Software and Computer Applications*, ICSCA 2022, page 136–142, New York, NY, USA. Association for Computing Machinery.

Christoforou, A., Garriga, M., Andreou, A. S., and Baresi, L. (2017). Supporting the Decision of Migrating to Microservices Through Multi-layer Fuzzy Cognitive Maps. In Maximilien, M., Vallecillo, A., Wang, J., and Oriol, M., editors, *Service-Oriented Computing*, Lecture Notes in Computer Science, pages 471–480, Cham. Springer International Publishing.

De Alwis, A. A. C., Barros, A., Polyvyanyy, A., and Fidge, C. (2018). Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems. In Pahl, C., Vukovic, M., Yin, J., and Yu, Q., editors, *Service-Oriented Computing*, Lecture Notes in Computer Science, pages 37–53, Cham. Springer International Publishing.

Di Francesco, P., Lago, P., and Malavolta, I. (2018). Migrating Towards Microservice Architectures: An Industrial Survey. In *2018 IEEE International Conference on Software Architecture (ICSA)*, pages 29–2909.

Di Francesco, P., Lago, P., and Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. *Journal of Systems and Software*, 150:77–97.

Eski, S. and Buzluca, F. (2018). An automatic extraction approach: transition to microservices architecture from monolithic application. In *Proceedings of the 19th International Conference on Agile Software Development: Companion*, XP '18, pages 1–6, New York, NY, USA. Association for Computing Machinery.

Fowler, M. and Lewis, J. (2014). Microservices.

Fritzsch, J., Bogner, J., Wagner, S., and Zimmermann, A. (2019a). Microservices migration in industry: Intentions, strategies, and challenges. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE.

Fritzsch, J., Bogner, J., Zimmermann, A., and Wagner, S. (2019b). From Monolith to Microservices: A Classification of Refactoring Approaches. volume 11350, pages 128–141. arXiv:1807.10059 [cs].

Gysel, M., Kölbener, L., Giersche, W., and Zimmermann, O. (2016). *Service Cutter: A Systematic Approach to Service Decomposition*. Pages: 200.

Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., and Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software*, 35(3):24–35.

Jin, W., Liu, T., Cai, Y., Kazman, R., Mo, R., and Zheng, Q. (2021). Service Candidate Identification from Monolithic Systems Based on Execution Traces. *IEEE Transactions on Software Engineering*, 47(5):987–1007. Conference Name: IEEE Transactions on Software Engineering.

Kalia, A. K., Xiao, J., Krishna, R., Sinha, S., Vukovic, M., and Banerjee, D. (2021). Mono2micro: a practical and effective tool for decomposing monolithic java applications to microservices. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM.

Kamimura, M., Yano, K., Hatano, T., and Matsuo, A. (2018). Extracting Candidates of Microservices from Monolithic Application Code. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 571–580. ISSN: 2640-0715.

Kitchenham, B. and Brereton, P. (2013). A systematic review of systematic review process research in software engineering. *Information and Software Technology*, 55(12):2049–2075.

Koschel, A., Astrova, I., and Dötterl, J. (2017). Making the move to microservice architecture. In *2017 International Conference on Information Society (i-Society)*, pages 74–79.

Lapuz, N., Clarke, P., and Abgaz, Y. (2021). Digital transformation and the role of dynamic tooling in extracting microservices from existing software systems. In Yilmaz, M., Clarke, P., Messnarz, R., and Reiner, M., editors, *Systems, Software and Services Process Improvement*, pages 301–315, Cham. Springer International Publishing.

Li, S., Zhang, H., Jia, Z., Li, Z. E., Zhang, C., Li, J., and Gao, Q. (2019). A Dataflow-Driven Approach to Identifying Microservices from Monolithic Applications. *Journal of Systems and Software*, 157.

Li, Z., Shang, C., Wu, J., and Li, Y. (2022). Microservice extraction based on knowledge graph from monolithic applications. *Information and Software Technology*, 150:106992.

Liu, B., Xiong, J., Ren, Q., Tyszberowicz, S., and Yang, Z. (2022). Log2ms: a framework for automated refactoring monolith into microservices using execution logs. In *2022 IEEE International Conference on Web Services (ICWS)*, pages 391–396.

Mathai, A., Bandyopadhyay, S., Desai, U., and Tamilselvam, S. (2022). Monolith to microservices: Representing application software through heterogeneous graph neural network.

Matias, T., Correia, F. F., Fritzsch, J., Bogner, J., Ferreira, H. S., and Restivo, A. (2020). Determining microservice boundaries: A case study using static and dynamic software analysis.

Mazlami, G., Cito, J., and Leitner, P. (2017). Extraction of Microservices from Monolithic Software Architectures. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 524–531.

Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 1st edition.

Nitin, V., Asthana, S., Ray, B., and Krishna, R. (2022). Cargo: Ai-guided dependency analysis for migrating monolithic applications to microservices architecture.

Nunes, L., Santos, N., and Rito Silva, A. (2019). From a monolith to a microservices architecture: An approach based on transactional contexts. In Bures, T., Duchien, L., and Inverardi, P., editors, *Software Architecture*, pages 37–52, Cham. Springer International Publishing.

Petersen, K., Vakkalanka, S., and Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18.

Pigazzini, I., Arcelli Fontana, F., and Maggioni, A. (2019). Tool support for the migration to microservice architecture: An industrial case study. In Bures, T., Duchien, L., and Inverardi, P., editors, *Software Architecture*, pages 247–263, Cham. Springer International Publishing.

Ponce, F., Márquez, G., and Astudillo, H. (2019). Migrating from monolithic architecture to microservices: A Rapid Review. In *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–7. ISSN: 1522-4902.

Ren, Z., Wang, W., Wu, G., Gao, C., Chen, W., Wei, J., and Huang, T. (2018). Migrating Web Applications from Monolithic Structure to Microservices Architecture. In *Proceedings of the 10th Asia-Pacific Symposium on Internetware*, Internetware '18, pages 1–10, New York, NY, USA. Association for Computing Machinery.

Schugerl, P., Rilling, J., Witte, R., and Charland, P. (2009). A Quality Perspective of Software Evolvability Using Semantic Analysis. In *2009 IEEE International Conference on Semantic Computing*, pages 420–427.

Sellami, K., Saied, M. A., and Ouni, A. (2022). A hierarchical-dbscan method for extracting microservices from monolithic applications.

Trabelsi, I., Abdellatif, M., Abubaker, A., Moha, N., Mosser, S., Ebrahimi-Kahou, S., and Guéhéneuc, Y.-G. (2022). From legacy to microservices: A type-based approach for microservices identification using machine learning and semantic analysis. *Journal of Software: Evolution and Process*, n/a(n/a):e2503.

Xu, D. and Tian, Y. (2015). A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J., and rong Wen, J. (2023). A survey of large language models. *ArXiv*, abs/2303.18223.

Zhou, X., Jin, Y., Zhang, H., Li, S., and Huang, X. (2016). A map of threats to validity of systematic literature reviews in software engineering. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, pages 153–160.