Exploring Alternative Microservice Decompositions using Data-driven Techniques and LLMs

Ana Martínez Saucedo
Instituto de Tecnología (INTEC)
Universidad Argentina de la Empresa (UADE)
Buenos Aires, Argentina
anmartinez@uade.edu.ar

J. Andres Diaz-Pace
ISISTAN Research Institute, CONICET & UNICEN University
Tandil, Buenos Aires, Argentina
andres.diazpace@isistan.unicen.edu.ar

Hernán Astudillo
Inst. Tecnología e Innovación para Salud y Bienestar (ITiSB), Universidad Andrés Bello
Viña del Mar, Chile
hernan.astudillo@unab.cl

Guillermo Rodríguez
ISISTAN Research Institute, CONICET & UNICEN University
Tandil, Buenos Aires, Argentina
guillermo.rodriguez@isistan.unicen.edu.ar

Abstract

The problem of migrating monolithic applications to microservices has become popular both in industry and academia, particularly when using automated tools to assist developers in the decomposition. However, deciding which is the most appropriate decomposition for a given monolith is challenging because the selected technique can return alternative decompositions depending on tool parameter configurations. This issue, often overlooked in the literature, makes developers have to resort to their intuition or use default parameters, leading to uncertain or opaque results. Based on a prior study of the parameters and variability of the decompositions generated an existing tool (*MicroMiner*), we propose an approach that leverages data-driven techniques to analyze the space of possible decompositions. These analytics are wrapped as predefined analysis mechanisms. Furthermore, we introduce a semantic layer based on an LLM, which connects developers' questions about the decompositions with predefined analyses, delivering textual answers and graphical charts. This enables user-friendly interactions between developers and decomposition tools' data. Our results demonstrate that our approach effectively identifies key parameters influencing decompositions and that the semantic layer provided relevant answers to 74% of possible practitioners' questions about the decomposition landscape, bringing insights into the tool parametrizations and also improving the interpretability of results by humans.

Keywords: microservices, monolith decomposition, sensitivity analysis, clustering, semantic layer, llms.

1 Introduction

The problem of decomposing an existing monolithic application into a collection of microservices has become popular both in industry and academia, particularly using automated tools to support developers in the de-

composition. A variety of techniques and tools have been proposed, such as *ServiceCutter* [1], *CARGO* [2], and *MicroMiner* [3], which seem to be effective for the problem. These approaches usually take the application source code as the main input and work as a black box that automatically generates a set of candidate microservices as the solution. The resulting decomposition can be seen as a kind of clustering of functional elements (e.g., classes) allocated to the monolith, which are re-grouped into candidate microservices. In this process, the developer can usually adjust parameters (as provided by the tool or technique under consideration) to affect the characteristics of the microservices decomposition, such as the granularity of each microservice, the relations among microservices, and the balance between cohesion and coupling metrics in the resulting solution, among others.

In this context, deciding which is the most appropriate (automated) decomposition for a monolith is challenging for the developer, because a technique can return different groups of functional elements and relations among them depending on its parameterization. For instance, a small variation in the *resolution* parameter of the clustering technique used by *MicroMiner* [4] can turn a given decomposition into another one with disparate characteristics. Although some tools suggest default parameterizations, they are still dependent on the monolith given as input. Therefore, when the developer wants to explore alternative decompositions, dealing with these black-box techniques becomes time-consuming and understanding the variations in the tool outputs is far from trivial.

In this work, we argue that understanding the effects of the input parameters of several techniques or tools on the space of potential decompositions for a monolith application is important for obtaining a decomposition that is comprehensible to the developer. Through a sensitivity analysis, a developer can make informed choices about the most suitable decomposition tool and the configuration of its parameters. In prior work [5], we performed a data-driven study using *MicroMiner* as the decomposition tool, and analyzed its parametrization with respect to microservices decompositions for two well-known applications. Our analysis showed that only a few decompositions generated by *MicroMiner* had a low variability in microservice-related properties, identifying the key parameters influencing those properties. Based on these results, we propose here an extension of the approach into two directions: i) a set of data analyses based on *clustering and heuristic search* that enable developers to get insights from the parameter and decomposition spaces, and ii) a semantic layer to simplify the interactions between a developer and the analysis techniques. The *parameter space* refers to the inputs of a (black-box) decomposition tool, whereas the *decomposition space* refers to its output, in terms of structural characteristics and quality properties. The semantic layer [6] is an abstraction on top of the data analyses that provides a user-friendly interface to access and understand analytics about the spaces. In particular, we provide a semantic layer powered by a Large Language Model (LLM), in which a developer can ask questions (in natural language), those questions are mapped to any of the available techniques triggering their execution, and finally their results are presented to the developer. The main contribution of our approach is the definition of such a semantic layer, along with its materialization in a prototypical implementation. The semantic layer is designed as a flexible framework in which additional tools, data sources, and query engines can be plugged in to enrich the support provided to developers.

We performed an in-depth empirical evaluation of our approach using the two case studies from [5] but incorporating another third-party tool, called *MIT* [4], to generate microservices decompositions. In this way, the space of candidate decompositions for the developer to examine is wider. Furthermore, we ran a set of tests to assess the capabilities of our semantic layer by creating a synthetic benchmark generator according to predefined developers' profiles. The results showed that our approach bridges the gap between practitioners' concerns regarding the landscape of possible decompositions and *parameter* and *decomposition spaces*. Specifically, the proposed semantic layer provided relevant answers to 74% of potential practitioners' questions about the landscape of possible decompositions obtained by running *MicroMiner* and *MIT* decomposition tools with different parametrizations.

The rest of the paper is organized into 8 sections as follows. Section 2 describes the main concepts behind our approach. Section 3 describes the main building blocks of our analysis approach. Section 4 presents the experimental setup of the approach. Section 5 analyzes the main findings of the study, discusses them, and presents the main threats to validity. Section 6 compares our approach with related works. Finally, Section 7 gives the conclusions and identifies future lines of research.

2 Background

This section sheds light on the main theoretical concepts related to microservice decomposition and the selected techniques to decompose a monolithic application into microservices.

2.1 Microservice decomposition

A monolith architecture is based on the encapsulation of the functional features of an application into a single and self-contained deployment artifact, even when some features can be internally organized into modules. This kind of architecture presents advantages when the size of the application remains manageable, such as easy deployment and implementation [7]. However, as an application grows in functionality and size, it becomes difficult to maintain its codebase, scale modules individually, and incorporate new technologies for certain features [8], among others. For these reasons, practitioners can decide to migrate to a microservice architecture, which promises benefits such as technological heterogeneity, scalability, autonomous services, and containerization, to cite a few.

Nonetheless, migrating from a monolithic architecture to microservices is not straightforward [9]. Key aspects to be addressed include: how to partition the monolith into smaller parts, how to handle communication among microservices, data migration (e.g., one database versus multiple databases), and DevOps adoption. A microservice decomposition involves identifying parts of the monolith (i.e., functional elements) to become microservices. Such microservices should be independent of each other, have adequate size and limited context. Furthermore, these microservices should adhere to the principle of single responsibility, when possible [10].

The process of microservice decomposition often departs from the codebase of the monolithic application and produces an initial (conceptual) partitioning of the monolith that a human must assess for feasibility. Several automated techniques and tools have been proposed to assist humans in the process [1–3]. These techniques often scan code, and design documents or data to (automatically) suggest a partitioning strategy. For instance, some tools analyze candidate microservices at the method level [11] while others do it at the class level [4]. The main goal of this analysis is to identify groups of cohesive functionality as microservices candidates, which is the perspective taken throughout this paper. Other necessary aspects to obtain a microservices application involve code refactorings and additional components (e.g., a gateway), which are left out of the scope of this work.

2.2 Motivating example: JPetStore

JPetStore [12] is a widely used application in the microservices literature. It is an open-source Java-based web application for a pet store where users can purchase pets online. *JPetStore* has a monolithic design that comprises 24 classes, which makes it a relatively small case study for evaluating automated decomposition techniques [13]. In this work, we rely on two existing techniques, *MicroMiner* and *MIT (Microservice Identification Tool)* [3, 4], to demonstrate our approach.

2.3 *MicroMiner*

MicroMiner [3] is a semi-automated decomposition technique based on Machine Learning (ML) algorithms and semantic code analysis for recommending microservice candidates at the class level. For example, Figure 1 shows two *JPetStore* decompositions suggested by *MicroMiner*. The classes on the left correspond to the monolithic application while the groups (or clusters) of classes on the right are the candidate microservices. Note that some classes might appear in more than one group (e.g., *Services #1, #2, and #5* related to the *Account* functionality), indicating that the class functionality should be split among several microservices. Overlapping classes among candidate services are highlighted with different colors.

In brief, the *MicroMiner* algorithm works in three stages. First, a source code analysis is performed based on tagging the classes according to their role (or type) in the system (e.g., entity, application, and utility). Second, a class dependency graph for the monolith application is built, in which dependencies among classes are derived from their method invocations. At last, a clustering procedure called community detection is executed on the graph. A relevant aspect is the *parameters* that influence the output of *MicroMiner*, as exemplified in Figure 1. There are parameters at work: i) clustering-related parameters such as $resolution$ and fuzziness (m_fuzzy), ii) class relationship type parameters such as α (static relationships weight) and β (semantic relationships weight), and iii) an expert-based threshold parameter such as $microservice\_threshold$ to decide whether an element belongs to a given microservice. If a developer (tool user) makes changes in these parameters, even small ones, the tool can generate a different partitioning of microservices, as shown in Figure 1.

2.4 *Microservice Identification Tool (MIT)*

MIT is an automated tool proposed by Brito et al. [4] that relies on topic modeling and clustering techniques to identify microservices candidates at the class level. The *MIT* algorithm involves three stages. First, lexical information is extracted from the source code in order to mine domain-specific topics. Second, a graph is

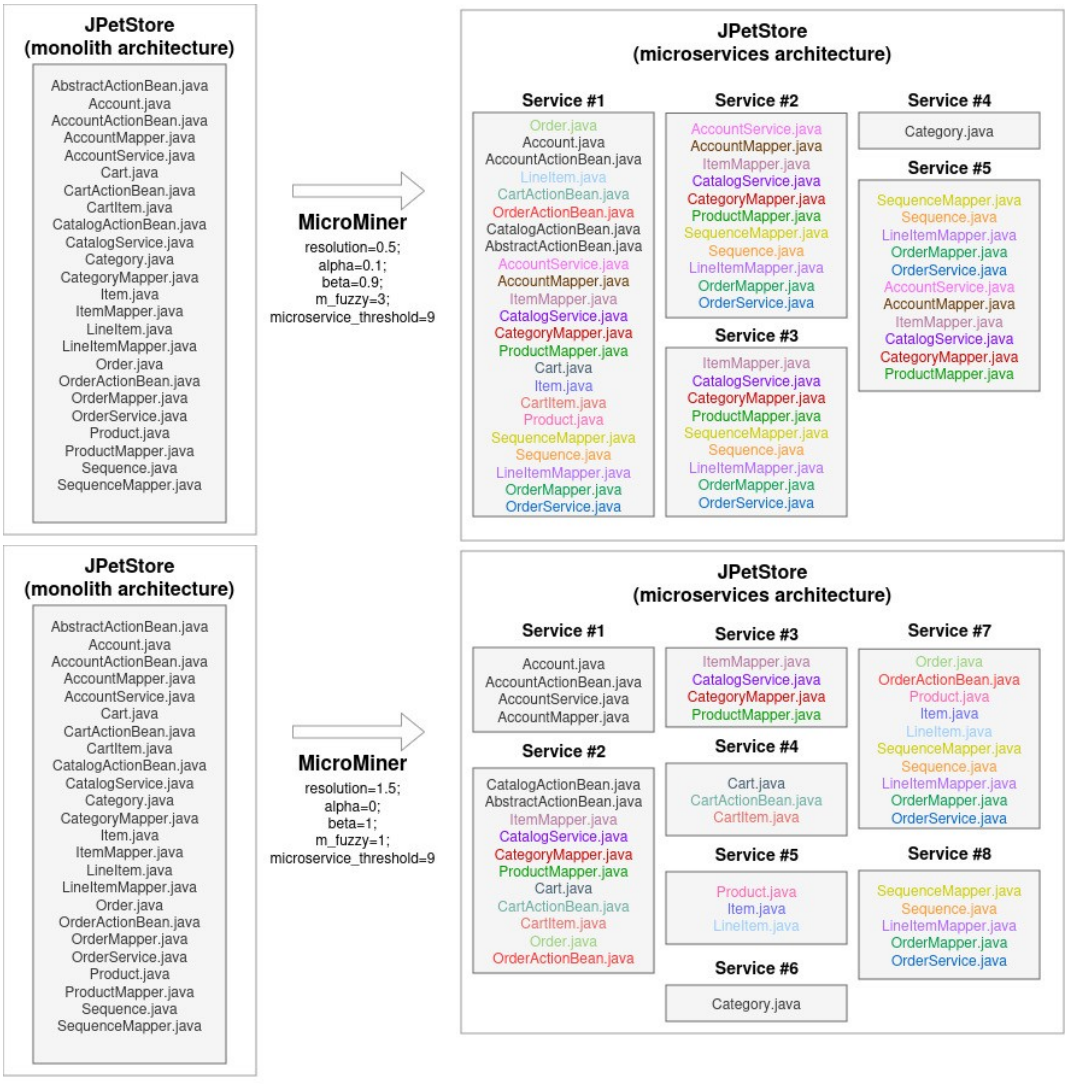


Figure 1: Alternative *JPetStore* decompositions suggested by *MicroMiner*

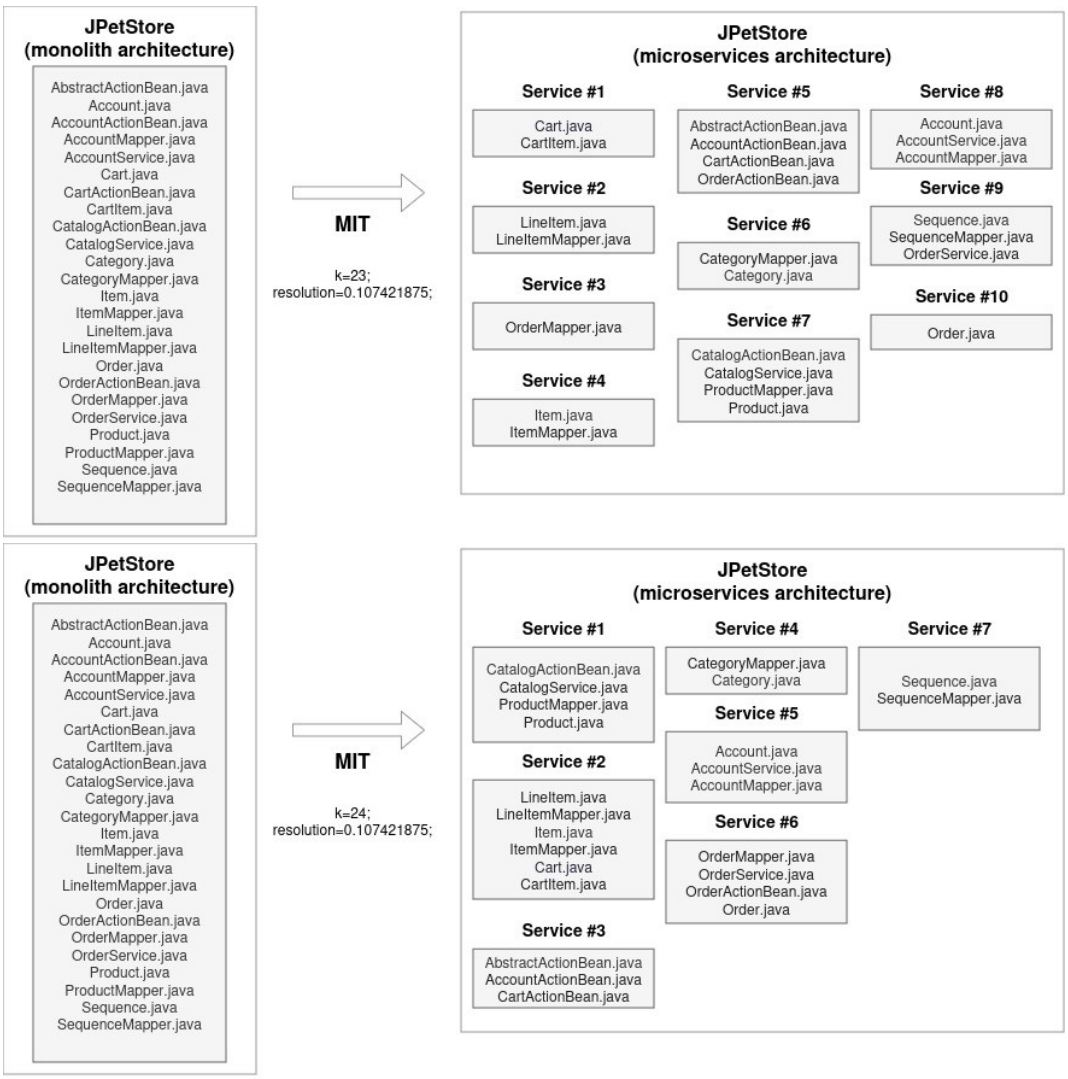


Figure 2: Alternative *JPetStore* decompositions suggested by *MIT*

built using the mined topics along with the structural information from the application. At last, a clustering procedure is executed on the graph. The *MIT* behavior depends on two clustering related parameters: i) *resolution*, and ii) k being the number of topics to be detected.

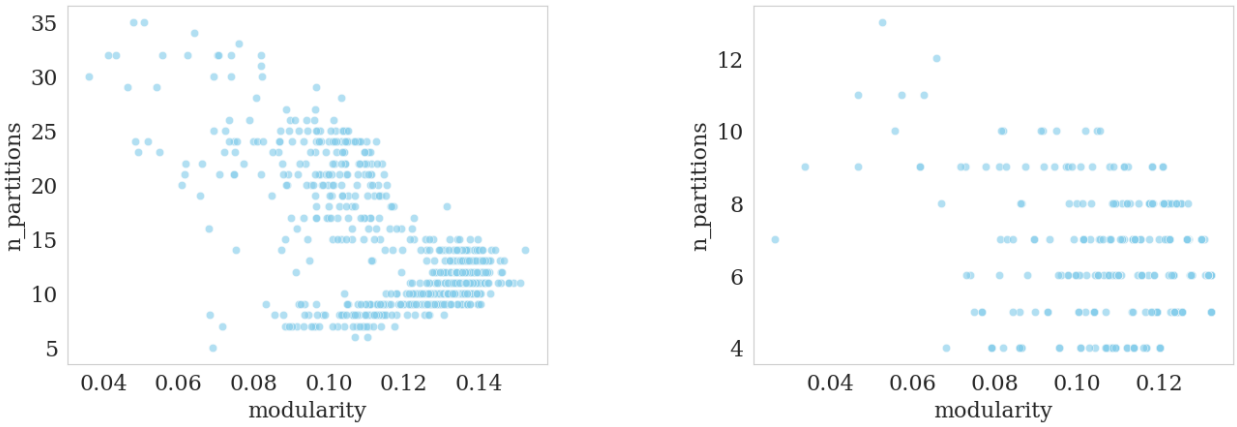
Figure 2 show the decompositions suggested by *MIT* for *JPetStore*. Note that each class is assigned to one microservice only; therefore, there are no overlapping classes as observed in the *MicroMiner* decompositions. However, a small variation of the k parameter produces a different partitioning.

2.5 Assessing decomposition with quality metrics

The outputs of most decomposition tools are evaluated in terms of cohesion and coupling to quantify the quality of the resulting decompositions [13]. Since a microservices partition is typically represented as a graph of interconnected nodes and clusters, some approaches employ graph metrics to quantify the partition quality. In particular, these metrics seek to evaluate size, graph partitioning, and cluster characteristics, among others. A common set of quality metrics include:

- *Modularity*: it calculates the difference between the actual intra-edges of a cluster and the expected intra-edges of the cluster in a randomly re-wired graph. Higher modularity values indicate partitions with better quality [14–17].
- *Number of partitions*: it quantifies the number of microservices in a decomposition [16, 18]. Ideally, a decomposition should lead to a manageable set of microservices, although this metric is dependent on the initial functional size of the monolith.

Given a variety of decomposition candidates, these metrics allow practitioners to understand which alternative is the most appropriate to select considering their migration drivers and organizational structure, among other criteria. For instance, while a decomposition with a large number of partitions might initially seem suitable for a large monolithic application, the allocation of development teams to the resulting microservices can be challenging. In this scenario, a practitioner could opt for a decomposition with fewer partitions but higher modularity as shown in Figure 3, which means that a suboptimal decomposition in terms of a particular metric could be better for the practitioner's technical and organizational context.



(a) *Cargo* (b) *JPetStore*

Figure 3: Variations between modularity and number of partitions in *MIT* decompositions

In general, a practitioner will make tradeoffs that weigh the metric values as well as other considerations or preferences [19]. This decision-making activity is not straightforward, due to both the sheer size of the decomposition space and the opaque nature of the existing tools and techniques for automated decomposition and analysis of the options.

3 Approach

To achieve a seamless exploration of a microservices decomposition space, we envision an approach in which a practitioner can interrogate a design space [20] formulating questions driven by her needs, goals, and concerns, and then an assistant maps those questions to specific database queries or analytical functions and returns an answer to the practitioner. To materialize this idea, we rely on a *semantic layer* [6] that is internally powered by a Large Language Model (LLM). In our approach, we assume that a software engineer or an architect performs the initial steps to configure the semantic layer with appropriate data and tools, enabling then practitioners (e.g., a developer) to make queries about decomposition alternatives in natural language.

Our proposed approach consists of several building blocks as depicted in Figure 4. The process begins with a software architect acquiring a target monolithic application. Next, one or more decomposition tools must be configured (1). These tools are usually provided by third parties. Depending on the selected tools, different kinds of inputs may be required. For simplicity, let us assume that the chosen tools work with the source code as the main information source. Second, different values for the input parameters of the tools need to be sampled. The selected tools are executed by the architect on the monolith application with those parameters, and the outputs are saved as candidate decompositions. We refer to the results of the parameter sampling as the *parameter space*. These decompositions often include groups (clusters) of related source code elements, i.e., potential microservices. Furthermore, the decompositions are used to compute quality metrics. We refer to the microservices decompositions along with their metrics as the *decomposition space*. The parameter and decomposition spaces are correlated, in the sense that changing certain parameter values is likely to affect the structure of the decompositions and their quality metrics thereof. We can think of both spaces as a *dataset* of alternatives which is obtained by running the tools on the monolithic application.

3.1 The parameter space

The first analysis centers on the parameter space, and its goal is to screen the tool parameters to detect the key parameters (2) for each decomposition tool, i.e., those parameters making a significant contribution to the values of the quality metrics. To this end, we perform a sensitivity analysis of the parameters using

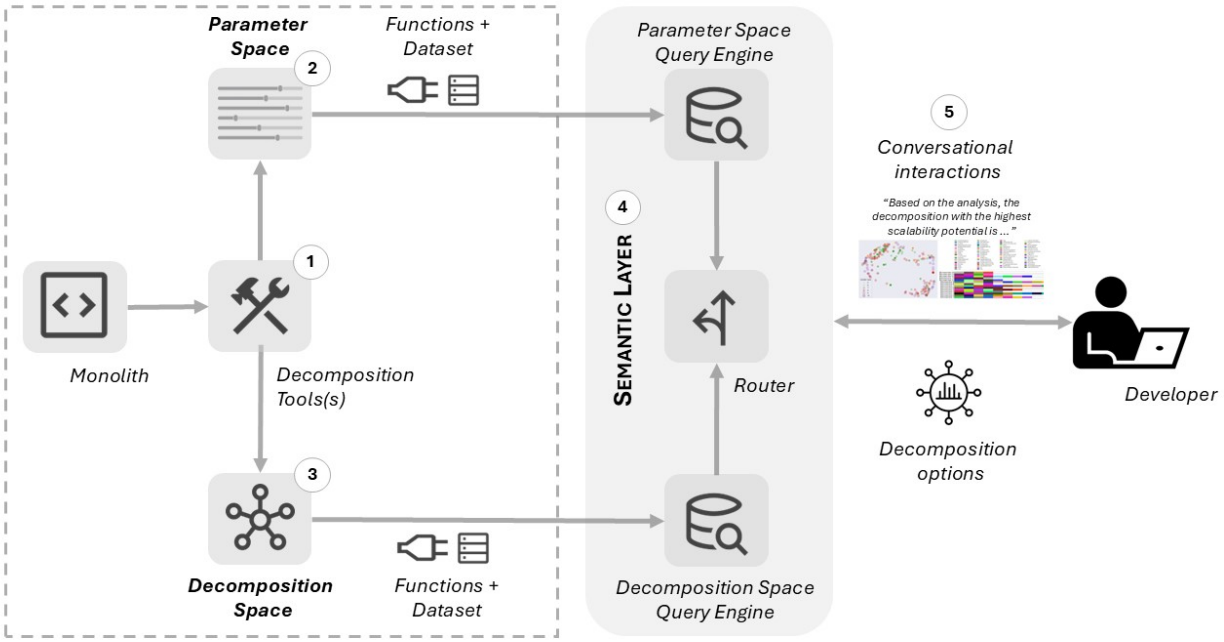


Figure 4: Main building-blocks of our semantic layer approach

the Sobol method [21]. The Sobol method measures the contribution of the input parameters of model or function to the output variance. As a result, a sensitivity score (known as the Sobol index) is assigned to each parameter. The higher the Sobol index for a parameter, the higher the chance that variations in that parameter will cause variations in a given metric.

For the dataset, we collect the parameter values for every execution of the decomposition tools, and then compute the Sobol index for each parameter. For instance, Figure 5 shows the Sobol indices for *JPetStore* (charts on the right) with respect to the modularity metric. In the case of the *MicroMiner* decompositions, the Sobol indices show that the $m\_threshold$ and m_fuzzy parameters influence more than 55% on the modularity variance, either through their direct contribution or interactions with other parameters. A clearer influence is observed for *MIT* decompositions (bottom), since the *resolution* parameter, both on its own and interacting with the k parameter, accounts for more than 90% of the modularity variance.

The sensitivity analysis of the parameter space is supported by the following two functions:

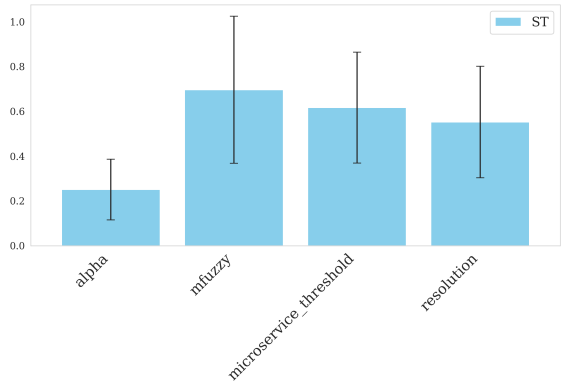
- `get_sensitivity_analysis_results(tool, metric)`: it determines the extent to which each parameter offered by a decomposition tool contributes to the variance of a target metric. The function output is a graphical chart, as shown in Figure 11.
- `get_influential_parameters(tool, metric)`: it complements the previous function by listing the parameters of a specific tool that contributes the most to the variance of a target metric.

The role of the functions above is to retrieve information (from the parameter space) that can be useful in answering the practitioner's questions.

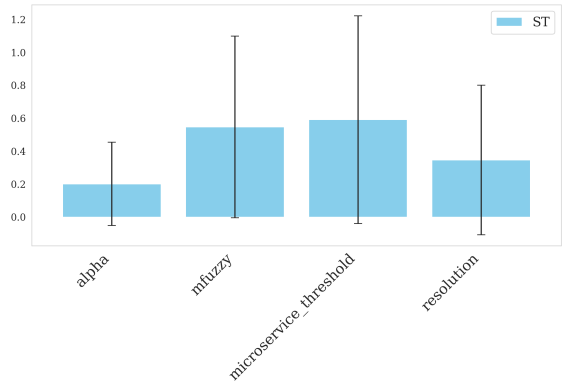
The analysis of the parameter space helps developers identify a handful of key parameters that impact a microservices quality metric, allowing developers to eventually adjust these parameters to obtain decompositions that best align with their needs. In this process, irrelevant parameters can be discarded. Furthermore, focusing on the most relevant parameters can simplify subsequent analyses, for example, about the design characteristics of the decompositions.

3.2 The decomposition space

The second analysis addresses the structural aspects of the decompositions. To do so, the decompositions are grouped according to their similarity (3) using clustering [22]. Note that, since this is an unsupervised setting, the decompositions dataset was not split into training and validation subsets. The model evaluation was performed instead using internal validation metrics. In particular, we rely on the *Omega* (ω) index, which measures the resemblance between two overlapping network clusters [23]. Moreover, a k-medoids

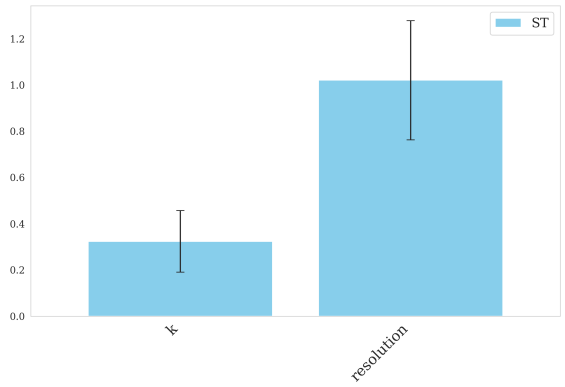


(a) *Cargo*

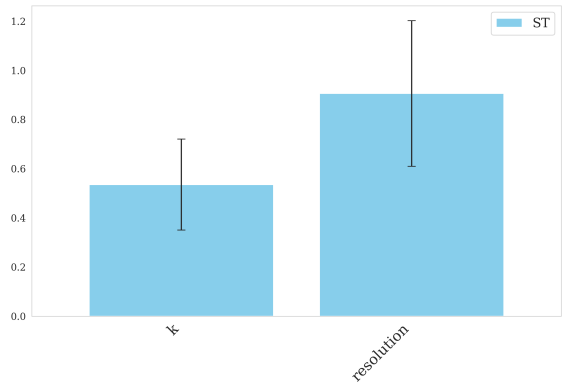


(b) *JPetStore*

MicroMiner decompositions



(c) *Cargo*



(d) *JPetStore*

MIT decompositions

Figure 5: Sobol total order index with respect to the modularity metric

algorithm is applied to infer groups of instances (i.e., decompositions), in such a way that instances in the same group (or cluster) are more similar to each other than to those in other groups. In this sense, k-medoids is a robust alternative to k-means, particularly because it is less sensitive to outliers and does not assume spherical clusters [24]. Furthermore, k-medoids generates centroids that are real instances of the dataset.

For instance, Figure 6 shows a clustering of the available partitions for *JPetStore* (right chart) using *5* k-medoids. Each medoid is the partition in a cluster whose sum of dissimilarities (in terms of the *Omega* index) to all the partitions in the cluster is minimal. Therefore, each medoid corresponds to a partition being structurally dissimilar from other medoids. We conjecture that these medoids are likely to suggest a handful of relevant partitions worth analyzing within the complex decomposition space, particularly regarding microservices quality metrics.

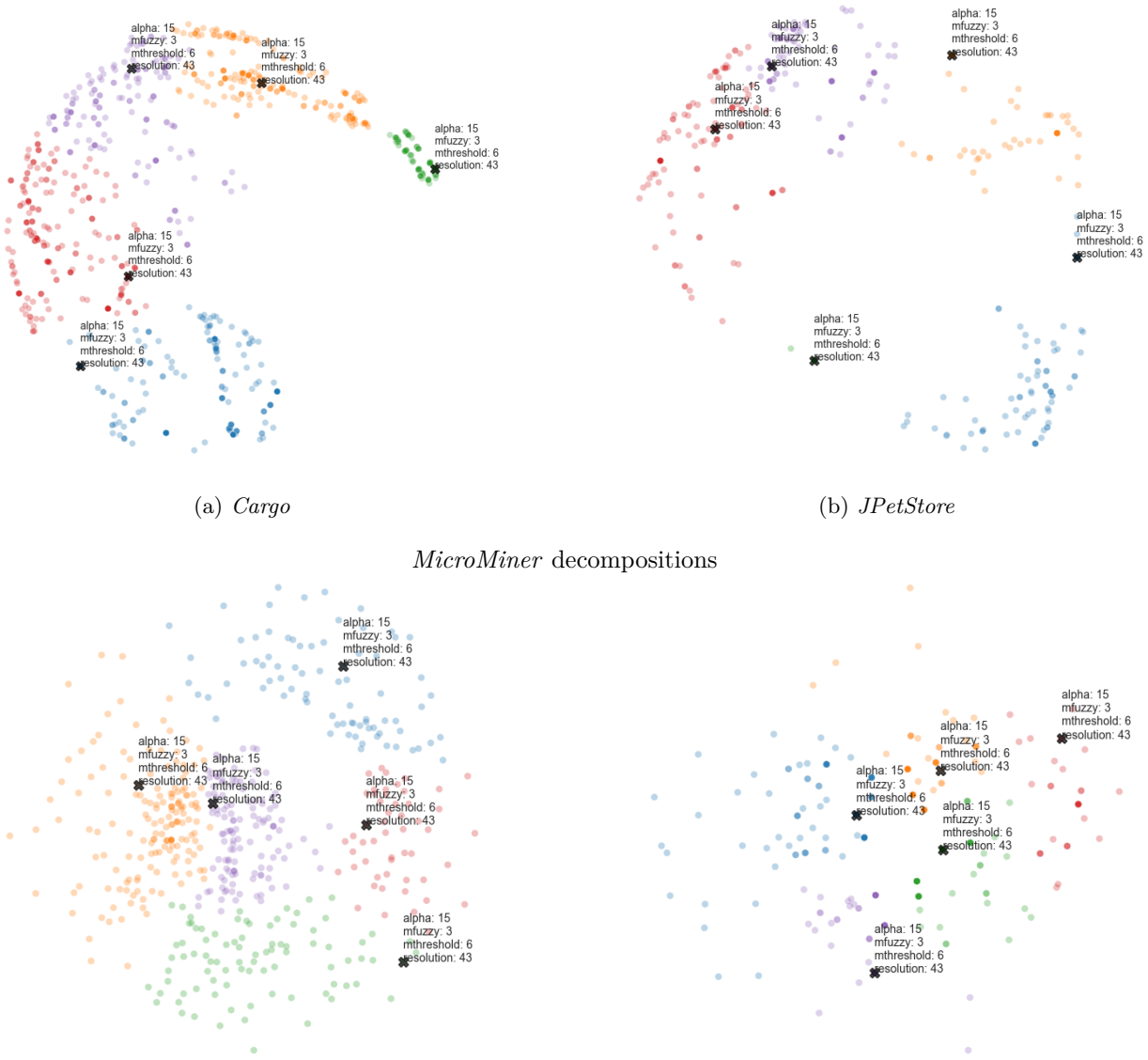
The analysis of the decomposition space is supported by the following functions:

- `get_decomposition_space()`: it generates a two-dimensional projection chart of the decomposition space, allowing practitioners map different groups of similar and dissimilar decompositions.
- `show_decomposition_structure()`: it shows a graphical representation of the classes each microservice is derived from in a candidate decomposition.

By defining the unit of analysis in terms of clusters, developers can identify the parameter configurations that led to similar decompositions, understand the characteristics of each cluster (through its centroid), and pinpoint the clusters that perform best according to the key metrics.

3.3 The semantic layer

The semantic layer is at the core of our approach, acting as a bridge between the developer's questions and the information provided by both the parameter and decomposition spaces. This information can be either fetched from tables (i.e., the dataset) or from the functions described in the preceding sections. The main



(a) *Cargo* (b) *JPetStore*

MicroMiner decompositions

(c) *Cargo* (d) *JPetStore*

MIT decompositions

Figure 6: Clustering of the partitions and their medoids using two-dimensional projection

goal is to provide insights about the decomposition alternatives for the monolithic application. Typical use cases are to quickly determine if parameter variations might lead to decompositions belonging to different clusters, or if a particular decomposition tool generates better decompositions regarding a metric of interest.

Conceptually, a semantic layer is an abstraction that aims to simplify interactions between complex data and users by providing a user-friendly interface that translates raw data into meaningful information. In this way, users focus on analysis rather than on the complexities of data [6]. In particular, our layer relies on an LLM-based agent to process a textual question from the developer and translate it into appropriate queries for structured data (i.e., tables) or calls to the provided functions. For instance, if the developer's question is about finding the decomposition with the highest potential for scalability, the agent could use the table that contains microservices quality metrics of each decomposition (available in decomposition space dataset) and try to rank them according to the modularity metric to generate a textual explanation. Furthermore, for the same question, the agent could invoke a function (with certain parameters) to supplement its response with a chart generated by the function. For instance, the chart could enrich the class structure of the suggested decomposition, as shown in Figure 7. In addition, the agent stores the history of questions and answers in a memory in order to maintain contextualized interactions with the developer [25].

The implementation of the semantic layer consists of two LLM-based engines built on top of the param-

Q16. Which decomposition has the highest scalability potential?

Based on the MIT analysis, the decomposition with the highest scalability potential is the one with a resolution of 1.992*, which is referred to as ẗresolution_1.992578125_k_1ïn Figure İt has a modularity of* 0.1523*, indicating a relatively high degree of independence among its components. Notably, it has 14 partitions, which can contribute to its scalability. Overall, this decomposition appears to have a strong potential for scalability due to its modular and partitioned structure.*

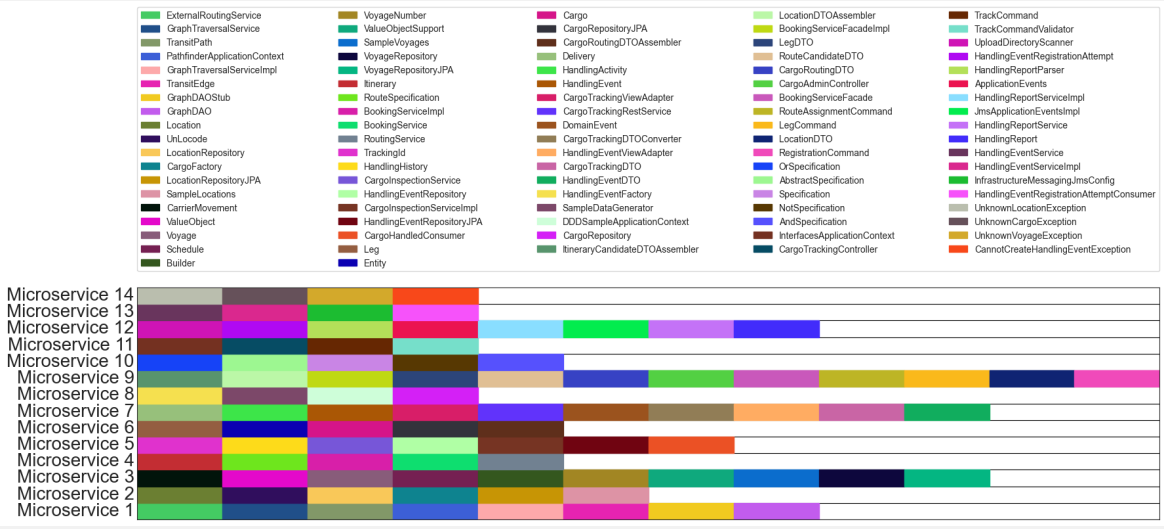


Figure 7: Example answer provided by the semantic layer to practitioner question concerning scalability

eter and decomposition spaces, respectively. Each query engine is capable of retrieving relevant pieces of information from its associated tables in the dataset, in response to a developer's question, and generating an answer in natural language (5). Since a query can require information from either the parameter space or the decomposition space, there is an LLM-based router for selecting the appropriate engine and executing it. This selection is based on metadata attached to each dataset, which describes its purpose and data types. If necessary, the router can forward the query to a function for generating a chart that enhances the (textual) query response with visual insights. For this function calling mechanism, the LLM needs to determine the right function to invoke as well as the parameter values for the function.

4 Experimental Setup

The end goal of the proposed approach is to provide practitioners with information about decompositions through the analysis of the parameter and decomposition spaces, enabling them to sift through a large collection of candidate decompositions in a comprehensive and user-friendly way. To evaluate the approach, we formulated the following research questions:

- **RQ#1:** How does the variability of decomposition tools' parameters affect the resulting decompositions?

  In this RQ, we aim to explore how the configuration of parameters in decomposition tools affects the microservices decomposition, measured by traditional microservices quality metrics. We seek to assess (potential) gaps between the desired decomposition characteristics and those resulting from the default parametrization suggested by tool maintainers, as well as to identify key parameters that drive a metric of interest.

- **RQ#2:** To what extent do the insights provided by the semantic layer answer practitioners' concerns about microservices decompositions?

  To understand the effectiveness of our approach in supporting practitioners to explore decomposition alternatives effectively and in an interactive manner, this RQ assesses the relevance of the responses produced by the semantic layer with respect to common practitioners' concerns about microservices. In this context, we aim to validate the semantic layer's ability to support informed decision-making.

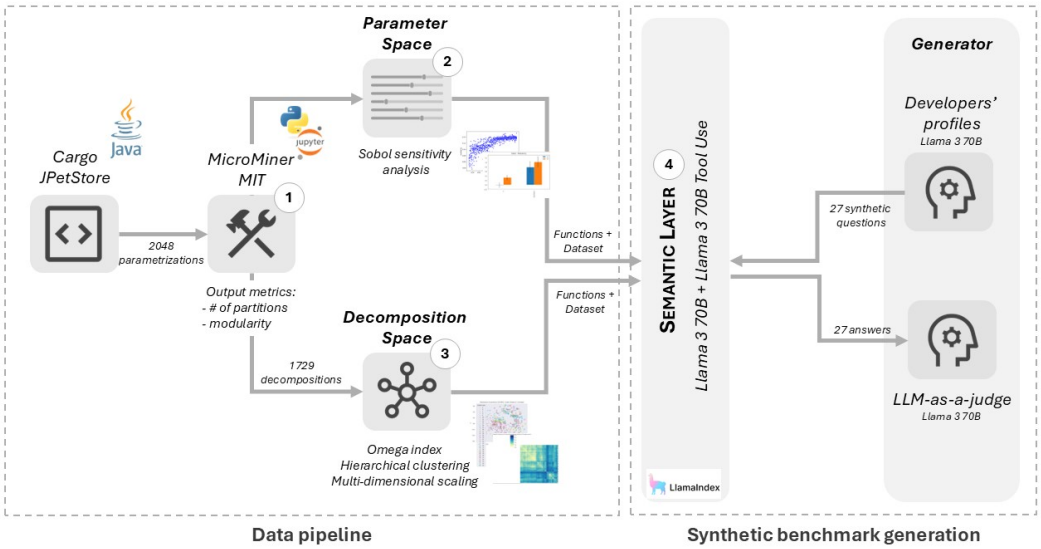


Figure 8: Experimental setup for the proposed data pipeline and synthetic benchmark generation

To answer the research questions we designed two separate experiments, as described in Figure 8. For **RQ#1**, we built a data-driven pipeline, executed it on two case studies from the literature, and assessed the quality and consistency of the results provided by the functions of the semantic layer. Furthermore, we included additional visualizations for parameter sensitivity, decomposition similarity, and decomposition distribution analyses.

For **RQ#2**, we created a synthetic benchmark of decomposition-related questions according to different developers' profiles and then applied an evaluation schema known as *LLM-as-a-judge*, which relies on an LLM to evaluate responses (from another LLM-based component) based on specific criteria, such as answer relevancy, correctness, or alignment. This evaluation schema was inspired by [26].

4.1 Case studies and Tools

We worked with the parameters and decompositions obtained through the *MicroMiner* and *MIT* tools for two Java web applications: *JPetStore* [12] and *Cargo* [27]. Both applications are structured as layered monoliths on top of the Spring framework, and consist of 24 and 100 classes, respectively.

The main feature of *Cargo* application is tracking and managing cargo shipments, including booking cargos, routing them through various transport legs, tracking their progress, and handling delays or disruptions. On the other hand, the *JPetStore* application includes typical e-commerce features such as a product catalog, user authentication, shopping cart functionality, and an order checkout process.

4.2 Data pipeline

Figure 8 (left side) shows the analysis process supported by the data pipeline, which involves three steps. First, we run the tools with different parameterizations on each case study. Second, we analyze the parameter space from the parameter sampling results. At last, we analyze the decompositions in the decomposition space.

As a preparation step (1), we extended the source code of *MicroMiner* and *MIT* to facilitate data generation about parameters and decompositions. In particular, we integrated the original *Jupyter* notebook of *MicroMiner* with the *EMA Workbench*[1]. This framework provides different sampling strategies, and also includes features for performing a Sobol sensitivity analysis. The input parameters for the tools were specified using predefined ranges and were randomly sampled via the *EMA workbench*. The parameters and numeric ranges for both *MicroMiner* and *MIT* are described in Table 1. In particular, we ran 1280 parametrizations over a grid of *MicroMiner* input parameters and 768 over *MIT* input parameters, resulting in a total of 2048 parametrizations for both study cases using Latin hypercube sampling.

Since *MicroMiner* and *MIT* construct a dependency graph of the decomposed application, we extracted and captured that graph using the *NetworkX* library[2]. As output, *MicroMiner* and *MIT* provide a collection

[1]https://emaworkbench.readthedocs.io/en/latest/index.html
[2]https://networkx.org/

Table 1: Input parameters and their ranges for *MicroMiner* and *MIT*.
Note: β is calculated as $1 - \alpha$

Technique	**Parameter**	**Bounds**
MicroMiner	α	[0, 1.0]
	m_fuzzy	[2,10]
	$microservice\_threshold$	[2,10]
	$resolution$	[0.01,1.0]
MIT	k	[1, # classes]
	$resolution$	[0.1, 2]

of clusters and a list of classes, representing the allocation of classes to microservices referred to as *partitions*. Each partition is modeled as a community, which is a group of densely interconnected nodes [28] (classes, in our context). Examples of such partitions were given in Figure 1. We calculated the modularity and number of partitions metrics for each decomposition candidate.

In step (2), we ran the Sobol analysis to understand which input parameters (or interactions among them) contribute the most to the variability of the outputs. The sensitivity of each parameter is represented by a Sobol index [21]. In particular, the first-order, second-order, and total-order indices were considered. A first-order index measures the degree to which an input affects the output variance, while a second-order index measures the degree to which the interactions between two inputs affect the output variance. The total index encompasses all *ith*-order indices for a given input.

To assess the distance between a pair of partitions A and B, we computed their *Omega* (ω) index. For instance, if A and B are the partitions given in Figure 1, $\omega(A, B) = 0.15$ indicates a low structural similarity between the partitions. We also kept track of the number of partitions, which is a metric ranging from 1 (a monolithic structure) to N, where N is the total number of classes in the application.

For step (3) we computed the distance $(1 - \omega)$ for the resulting partitions, which generated distance matrices of 1212×1212 dimensions and 1220×1220 for *MicroMiner* decompositions of *Cargo* and *JPetStore*, respectively. In the case of *MIT* decompositions, the dimensions of the distance matrices are 508×508 for *Cargo*, and are 480×480 for *JPetStore*. The disparity between the matrix dimensions arises from the exclusion of some duplicate parametrizations during the sampling process.

Afterward, we performed a hierarchical clustering [22] on 1729 decompositions using the *Omega* score as the distance criterion for the clusters. Obtaining clusters of structurally similar decompositions permits identifying decomposition groups of similar characteristics in a complex space. In effect, hierarchical clustering allowed us to analyze cluster structures at multiple levels of granularity, which is valuable when the optimal number of groups is not known in advance. Moreover, this method does not rely on Euclidean space assumptions, making it particularly suitable for our setting, where the distance metric (*Omega*) is not derived from geometric coordinates but rather from structural dissimilarities between partitions. Finally, we used multidimensional scaling (MDS) to embed the partition distance matrix and provide a 2D layout for the decompositions. MDS is a technique for creating a lower-dimensionality projection of a complex space that shows the dissimilarities among instances by trying to preserve the distances between them (in the original space) [29]. A lower-dimensional space helps the developer to better understand how the landscape of possible decompositions is organized.

4.3 Synthetic benchmark generation

To evaluate the ability of the semantic layer to handle possible questions from a practitioner about the decompositions, we developed an LLM-based *generator* (right side of Figure 8) that creates sets of synthetic questions according to predefined developers' profiles, in order to emulate real users. The LLM prompt was manually crafted and empirically adjusted to encourage the generation of practical, diverse, and realistic question sets. Note that the focus of this LLM and the information used for the generation differ from the LLM used within the semantic layer. Using these question sets, we can test the quality of the responses of the semantic layer for different scenarios.

We defined two question sets, one per case study, and generated $N = 27$ synthetic questions per set using `Llama-3-70B`[3] and employing the prompt shown in Figure 9. The specific questions used for *Cargo* application are listed in Table 2. Example answers to two of these questions are shown in Figures 7 and 10.

These questions were designed to address migration concerns reported in the literature, such as scalability and maintainability [13], which are suitable to be addressed with our approach. Each question is categorized according to a particular category according to the literature:

[3] `https://huggingface.co/meta-llama/Meta-Llama-3-70B`

```
You are an expert software developer that needs to decompose an objective-oriented
↪ monolithic application into microservices, while taking scalability and maintenance
↪ factors into consideration.
For this monolithic application, you have several decomposition alternatives at your
↪ disposal, and you have to choose the decomposition that best fits your needs.
Since the decompositions are generated by an automated tool, each decomposition includes
↪ information about the parameters used by the tool to obtain the decomposition,
such as: the class partitioning and several microservices quality metrics.

Monolithic application: {app_description}
---------------------
Factors:
- Scalability: Microservices are independent. Therefore, their sizing is easily
↪ accomplished. For instance, if a microservice is receiving several requests and
↪ consequently needs computational resources, it can be scaled individually using a
↪ container manager. Scalability then measures the ability of a system to cope with a
↪ load increase without sacrificing performance.
- Maintenance: Independence, reduced size, and limited context are characteristics that
↪ belong to microservices, which provide ease of maintenance. Therefore, it can be
↪ inferred that modification is straightforward, that is, it is possible to modify a
↪ microservice directly without major concerns about the impact that will be caused on
↪ the other microservices that make up the application, since that there is a low
↪ degree of coupling between them. In case of introducing errors to a microservice,
↪ they can be fixed more quickly.
---------------------
Your task is to generate {k} specific questions about the decompositions that should
↪ help you inform your selection of the best decomposition.

The documentation is as follows:
{query_str}

Please ensure that the questions are based on the documentation provided above and are
↪ not compound, and explore different analysis perspectives with respect to the
↪ decompositions.
The questions should be formatted as a JSON list.
```

Figure 9: Prompt employed to generate synthetic questions

- **Cohesion and coupling (CC)**: these metrics are widely used to quantify the quality of microservices decompositions. In particular, loose coupling and high cohesion are characteristics that practitioners seek in microservices architectures, as they enable independent deployment and easier scaling [30].
- **Domain model (DM)**: from a domain perspective, there is a trend among practitioners to map cohesive business functions or capabilities to microservices, adhering to domain-driven design principles. Therefore, understanding the allocation of business functions to microservices can be useful to assess whether microservices are modeled around domain boundaries, and thus achieving business alignment and scalability [13].
- **Quality attributes (QA)**: this category evaluates architectural metrics such as scalability, maintainability, complexity, and reusability [31]. The assessment of non-functional requirements of decomposition alternatives is possible through the metrics of this category.
- **Size and granularity (SG)**: questions in this category involve structural characteristics of decompositions, such as the number of microservices or the overall granularity of the decomposition. The choice of decomposition alternatives guided by a proper sizing can impact the performance and maintainability of the system. While coarse-grained microservices can become monoliths, fine-grained ones can introduce performance issues [32].

Finally, in the semantic layer, we leverage `Llama-3-70B` for the query engines and routing support, and

Q22. What is the number of microservices in each decomposition?

The output shows the number of microservices in each decomposition alternative. There are 508 different decomposition alternatives, each with a varying number of microservices. The number of microservices in each decomposition ranges from 13 to 35. The majority of the decompositions have between 28 and 32 microservices. Here are some key points:

- *The decomposition with the highest number of microservices has 35 microservices.*
- *The decomposition with the lowest number of microservices has 13 microservices.*
- *Most decompositions have between 28 and 32 microservices.*

Figure 10: Example answer provided by the semantic layer to a question about the number of microservices

`Llama-3-70B-Tool-Use`[4] to enable the function-calling feature of the semantic layer. Moreover, the semantic layer was built on top of the `LlamaIndex` [33] framework.

5 Evaluation

In this section, we report the main results for the research questions. The data and code used in this evaluation are publicly available for reproducibility purposes[5].

5.1 RQ#1: Effects of the tool parameters on the decompositions

We began analyzing how the variability of each tool parameter (α, m_fuzzy, $microservice\_threshold$, and $resolution$ in case of *MicroMiner*, k and $resolution$ in case of *MIT*) affects the variance of the metrics for modularity and number of partitions as shown in Figures 11 and 12. To this end, we employed a Sobol sensitivity analysis, and particularly looked at the first, second and total Sobol indices [21]. Additionally, we studied the extent to which the parameter interactions affect the outputs, as shown in Figures 5 and 13. These analyses are part of the parameter space.

Regarding modularity (Figure 11), the m_fuzzy parameter of *MicroMiner* explains 28.8% of the variance in *Cargo* decompositions, while the $m\_threshold$ parameter accounts for 35.1% of the variance in *JPetStore* decompositions. In *MIT*, the $resolution$ parameter stands as the main contributor to output variability in both study cases. Consequently, we infer that achieving high levels of modularity requires a segmented and cohesive grouping of the application classes into microservices. Conversely, a small modularity value can indicate a high interdependence among microservices and unclear boundaries between functional units. However, upon scrutinizing the parameter interactions of *MicroMiner* (Figure 5), we observed differences between the two applications: while *Cargo* mainly hinges on the m_fuzzy, *JPetstore* leans on $microservice\_threshold$ through its direct influence and interplay with other variables.

As depicted in Figure 12, $resolution$ is the most important parameter when measuring the number of partitions of the decompositions both in the *MicroMiner* and *MIT* tools. Specifically, it accounts for 77.5% of the variance in the *MicroMiner* decomposition of *Cargo*, 55.3% in the *MicroMiner* decomposition of *JPetStore*, 66.4% in the *MIT* decomposition of *JPetStore*, and 88.5% in the *MIT* decomposition of *Cargo*. For the *MicroMiner* decompositions of *Cargo*, the interaction of $resolution$ with the other parameters contributes to 96.4% of the variance of the number of partitions (Figure 13). Similarly, in the *JPetStore* decompositions of *MicroMiner*, the interaction of $resolution$ with other parameters contributes 93.9% to the output variance. However, in this case, we noticed that the interactions of $microservice\_threshold$, m_fuzzy, and α between them also contribute more than 40% to the output variance. Certainly, the number of microservices within an architecture is a pivotal factor in evaluating its quality. A surplus of microservices can indeed lead to communication overhead, potentially impeding system efficiency. Conversely, an inadequate number of microservices may lead to the formation of monolithic services (also known as distributed monoliths), which pose scalability and maintenance challenges. Thus, striking a balance in microservice granularity is imperative for achieving an architecture that deals with both performance and manageability.

Regarding the obtained decompositions, we analyzed them based on the *Omega* distances between partitions to then construct the decomposition space. The distributions of *Omega* distances among the partitions

[4] `https://huggingface.co/Groq/Llama-3-Groq-70B-Tool-Use`
[5] `https://github.com/amartinezsaucedo/microservices-decomposition-variability`

Table 2: Synthetic questions related to *Cargo* decompositions. CC: Cohesion and coupling, DM: Domain model, QA: Quality attributes, SG: Size and granularity

#	**Question**	**Category**
1	What is the cohesion metric for each microservice in each decomposition?	CC
2	What is the coupling metric between microservices in each decomposition?	CC
3	How many dependencies exist between microservices in each decomposition?	CC
4	What is the average number of interfaces per microservice in each decomposition?	CC
5	What is the average number of configuration files per microservice in each decomposition?	CC
6	How many Cargo-related classes are in each microservice of each decomposition?	DM
7	How many Location-related classes are in each microservice of each decomposition?	DM
8	How many RouteSpecification-related classes are in each microservice of each decomposition?	DM
9	How many Itinerary-related classes are in each microservice of each decomposition?	DM
10	How many CarrierMovement-related classes are in each microservice of each decomposition?	DM
11	How many HandlingEvent-related classes are in each microservice of each decomposition?	DM
12	How many Delivery-related classes are in each microservice of each decomposition?	DM
13	Which decomposition has the highest degree of alignment with business capabilities?	DM
14	Which decomposition has the lowest degree of alignment with business capabilities?	DM
15	What is the complexity metric for each microservice in the decomposition?	QA
16	Which decomposition has the highest scalability potential?	QA
17	Which decomposition has the lowest maintenance complexity?	QA
18	Which decomposition has the highest degree of independence between microservices?	QA
19	Which decomposition has the lowest degree of independence between microservices?	QA
20	Which decomposition has the highest degree of reusability across microservices?	QA
21	Which decomposition has the lowest degree of reusability across microservices?	QA
22	What is the number of microservices in each decomposition?	SG
23	What is the average size of each microservice of each decomposition in terms of number of classes?	SG
24	What is the largest microservice in each decomposition in terms of number of classes?	SG
25	What is the smallest microservice in each decomposition in terms of number of classes?	SG
26	Which decomposition has the most balanced distribution of classes across microservices?	SG
27	Which decomposition has the least balanced distribution of classes across microservices?	SG

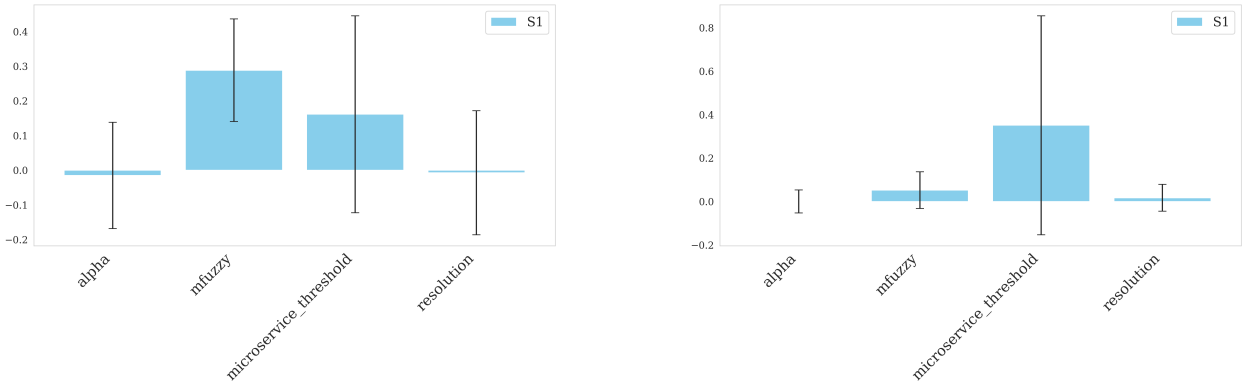
are shown as heatmaps in Figure 14. If $1 - \omega =$ is close to 0, the two partitions (row and column in the heatmap) are identical (light colors in the heatmap) and vice versa.

While both *JPetStore* and *Cargo* generated mainly similar matrix profiles for the *MicroMiner* decompositions, in *Cargo* the variability in terms of partition similarity was higher than in *JPetStore*. To further investigate these results, we looked at the clustering of the decomposition landscape.

We experimentally determined $k = 5$ as a good number of clusters for both applications and tools, and computed the medoids for each cluster to identify decompositions from which the sum of the *Omega* distances is minimal. The results are displayed in Figure 6. We conjecture that the cluster medoids correspond to decompositions whose similarity to other decompositions (within the same cluster) has low variability in the decomposition space. Thus, a guideline for the developer is to pick those solutions (and their parametrizations) as decompositions worth analyzing, as their quality metrics may vary due to being representatively dissimilar in structure to other medoids.

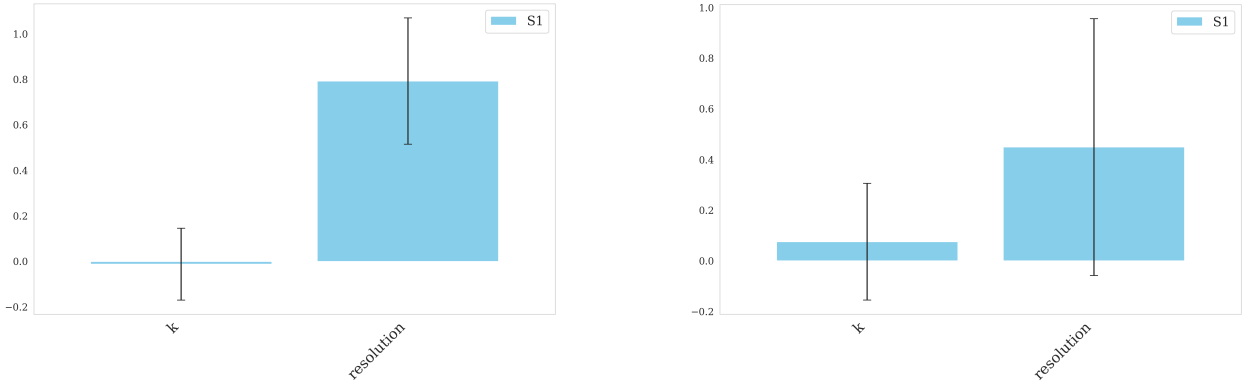
From a related perspective, Figure 15 presents the distributions of the partitions but highlights the number of microservices per partition (*n_partitions*). We noticed that the *JPetStore* decompositions obtained by *MicroMiner* and *MIT* exhibited more defined clusters than *Cargo* in terms of the similarity between partitions. The *JPetStore* partitions are more distributed in the decomposition space, which would indicate clear boundaries to separate decompositions based on the similarity of the partitions. For the developers, this implies a higher chance of finding partitions with low variability in the *JPetStore* space.

We noticed that the α parameter did not seem to influence the variability of the evaluated outputs. This differs from the claims made in the *MicroMiner* paper [3]. Since the β parameter is complementary to α, the static and semantic weighting schema for relationships in the application dependency graph seemed not relevant for the output variability in the two applications. Moreover, m_fuzzy, $microservice\_threshold$, and $resolution$ show varying effects on the outputs in the two applications. Except for the influence of $resolution$



(a) *Cargo* (b) *JPetStore*

MicroMiner decompositions



(c) *Cargo* (d) *JPetStore*

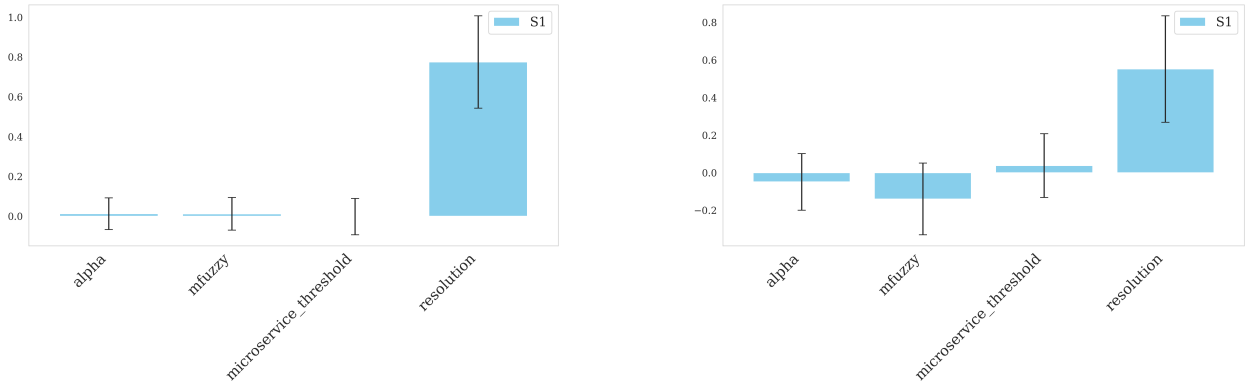
MIT decompositions

Figure 11: Values of 1st and 2nd order Sobol indices with respect to the modularity metric

on the number of partitions, the rest of the parameters presented different effects on the evaluated metrics. However, the *MIT* parameters exhibited a stable effect on the variance of both case studies, with a central influence of the *resolution* parameter on all the output metrics evaluated in our experiments.

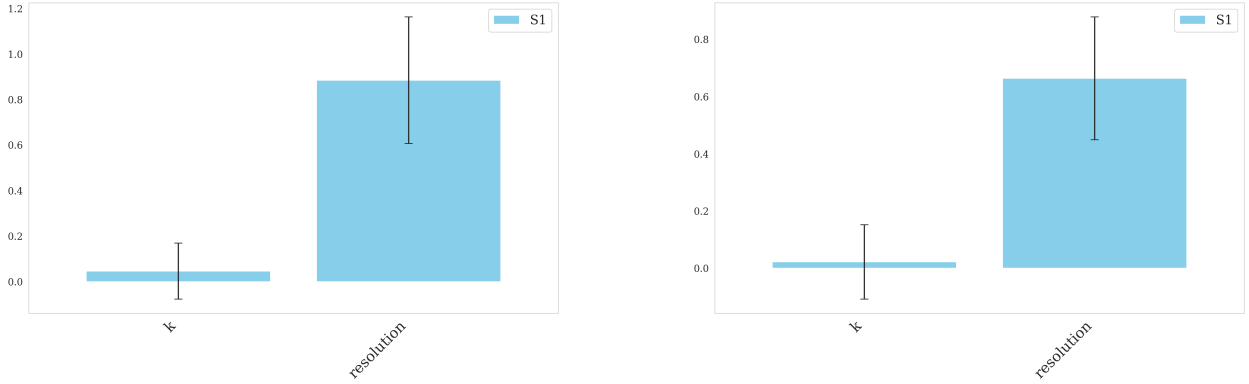
The varying results can be attributed to differences in application characteristics such as complexity, size, and design structure, among others. These differences can be explained in terms of how the tool processes a monolithic application to obtain microservices candidates. For instance, *MicroMiner* relies on the class dependency graph of the target application. Thus, the complexity and structure of the dependency graphs of both applications are the main contributors to the observed differences.

With respect to the decomposition space, our findings align with the distinct characteristics of the decompositions derived for both applications when the tool parameters are varied in small increments, as depicted in Figures 16 and 17. The color-coded schema in both figures shows how the Java classes from the monolith are assigned to the microservices for different decompositions. Specifically, in the case of the *MIT* decompositions of *JPetStore*, their decomposition structure varies in size, marked by the addition of three new clusters with each unit increase in k. Conversely, in the case of the *MicroMiner* decompositions of *Cargo*, the two decompositions exhibit variations in size (from 20 to 21 clusters for parametrizations a and b respectively). Furthermore, the composition of classes within each partition differs significantly. We hypothesize that the variability in class composition is influenced by the application size, since *Cargo* has five times as many classes as *JPetStore*. The intricate interactions between a large number of classes can lead to more convoluted decompositions. The chart gives insights on the difficulty of a decomposition in terms of refactoring efforts, if the developer were to take the microservices suggested by *MicroMiner*. That is, the more classes overlapping across microservices, the higher the potential implementation effort to split those classes. On the contrary, the *MIT* decompositions for *Cargo* show no overlap, which could imply simpler refactoring efforts in case of adopting the *MIT* suggestions.



(a) *Cargo* (b) *JPetStore*

MicroMiner decompositions



(c) *Cargo* (d) *JPetStore*

MIT decompositions

Figure 12: Sobol 1st and 2nd order indices regarding the number of partitions

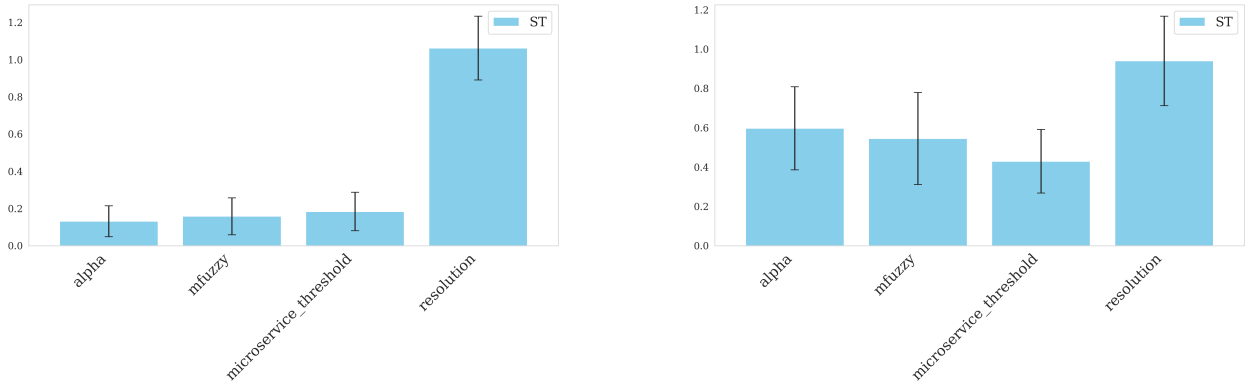
Summary The results suggest that analyzing both the parameter and decomposition spaces is useful for the developer to understand the variability of the tool parameters with regards to the resulting decompositions. Specifically, when assessing microservice decompositions is important to: (i) find relevant parameters in terms of output variability and discard irrelevant ones for the decomposition characteristics, and (ii) identify configurations of relevant parameters that lead to decompositions having low variability. We obtained promising results with the decompositions associated to the medoid parametrizations, which were structurally dissimilar according to the *Omega* distances. In sum, the conducted analyses show that, for both tools and case studies, a subset of relevant parameters significantly impacts the decompositions with respect to microservices metrics. Nonetheless, further analysis is required to choose the most suitable parametrization.

5.2 RQ#2: Semantic layer insights to answer practitioners' concerns about decompositions

In these experiments, we evaluate the extent to which the analyses enacted by the semantic layer address practitioners' concerns regarding microservices decomposition.

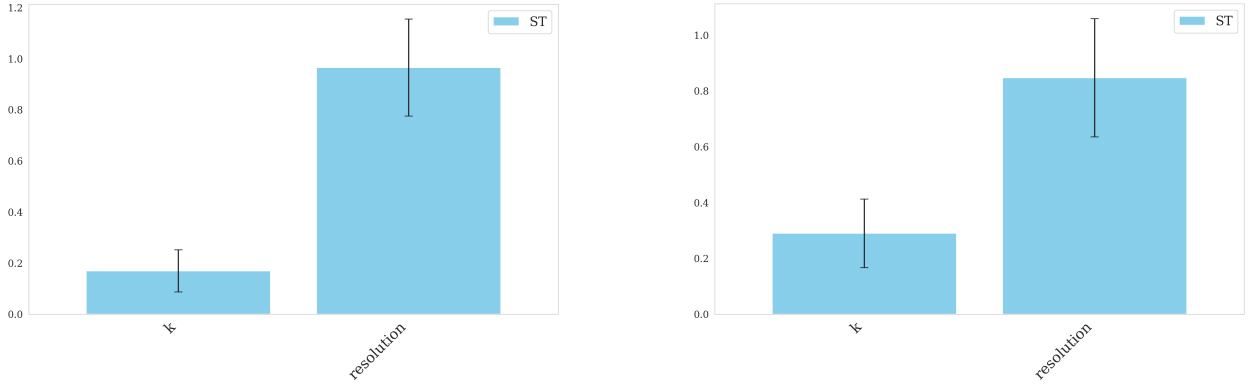
Given a benchmark of $N = 27$ synthetic questions, we prompted each of them to our semantic layer providing the context defined in Figure 18 and computed the answer relevancy metric as defined by Equation 1. This metric aims to assess the pertinency of a generated answer with respect to the given prompt. A low relevancy score indicates that an answer is incomplete or contains redundant information, while a high score is assigned to an answer that directly and appropriately addresses the question posed [34].

$$answer\_relevancy = \frac{1}{N}\sum_{i=1}^{N}\cos(E_{g_i}, E_o) \tag{1}$$



(a) *Cargo* (b) *JPetStore*

MicroMiner decompositions



(c) *Cargo* (d) *JPetStore*

MIT decompositions

Figure 13: Sobol total order index regarding the number of partitions

where:
E_{g_i} is the embedding of the generated question i,
E_o is the embedding of the original question,
N is the number of generated questions.

A sample of questions along with the answers from the semantic layer is shown in Figures 10 and 7. The complete set of results is included in the replication package.

The answer relevancy results for both case studies are presented in Table 3. On average, our approach reached a 74% relevancy rate in answering practitioners' questions about the landscape of possible decompositions. In the case of *Cargo*, the average answer relevancy score was 0.67, while for the *JPetStore* the score was 0.8. This difference in the scores suggests that the answers in the *JPetStore* case study were generally more relevant and better aligned with the expected responses.

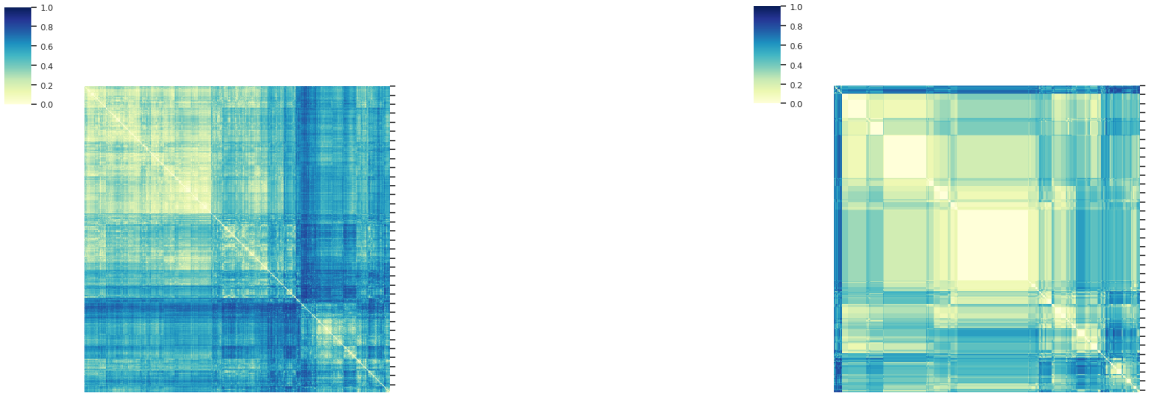
Table 3: Answer relevancy scores for each question set

Question set	**Case study**	**Number of questions**	**Average answer relevancy score**
Set 1	Cargo	27	0.67
Set 2	JPetStore	27	0.81

Upon analyzing the results grouped by categories, as detailed in Table 4, we observed that answers to questions related to size and granularity, as well as quality attributes, consistently achieved the lowest relevancy scores for both case studies. This performance can be attributed to the current quality metrics employed to assess microservice decomposition quality, such as the number of partitions and modularity. These metrics may be insufficient to answer some practitioners' questions regarding quality attributes such as reusability, or granularity. In contrast, most questions related to cohesion and coupling achieved a 100%

(a) *Cargo* (b) *JPetStore*

MicroMiner decompositions



(c) *Cargo* (d) *JPetStore*

MIT decompositions

Figure 14: Spatial arrangement (heatmap) of decompositions according to their *Omega* distances

relevancy rate, thus indicating that our approach, leveraging analyses on the parameter and decomposition spaces, is effective in addressing microservices dependencies (coupling) and the functionality agreement within microservices (cohesion).

Table 4: Answer relevancy scores for each question category

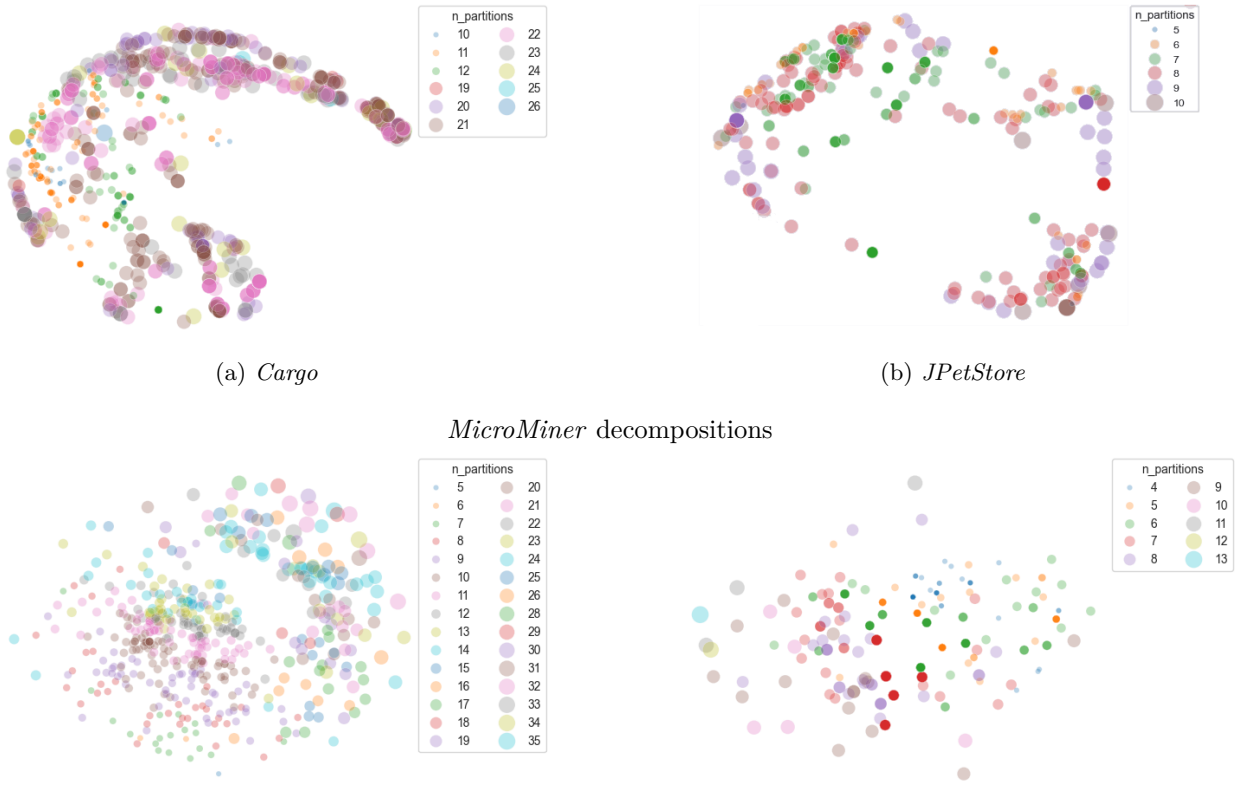
Question category	Relevant answers (Cargo)	Relevant answers (JPetStore)
CC	100%	100%
DD	61.1%	100%
QA	77.8%	66.7%
SG	66.7%	66.7%

Summary The use of a semantic layer to translate raw and complex (decomposition-related) data into relevant insights proved to be useful not only in addressing particular developer's concerns but also in consolidating the available data of both parameterizations and their corresponding decomposition characteristics. Moreover, the semantic layer promoted the interpretability of the presented results.

5.3 Discussion

Our findings highlight several practical insights for practitioners seeking to adopt a data-driven, smart approach to assess alternative microservices decompositions.

On the one hand, developers can readily implement our data-driven approach by conducting a variability analysis on their decomposition approach or tool of choice and linking the parameters to the decomposition landscape. By identifying decompositions that remain stable when varying tool parameters within a desired



(a) *Cargo* (b) *JPetStore*

MicroMiner decompositions

(c) *Cargo* (d) *JPetStore*

MIT decompositions

Figure 15: Distribution of the decompositions and their size using a 2D MDS projection (*n_partitions*)

threshold, developers can pinpoint the most relevant decompositions within the landscape of candidate microservices. Another finding is that, in addition to preferring a decomposition with low variability, a developer should also choose a decomposition in which the degree of class overlapping is low, like in the case of the *MIT* tool. Our current functions within the semantic layer need to be improved to incorporate visualizations in that regard.

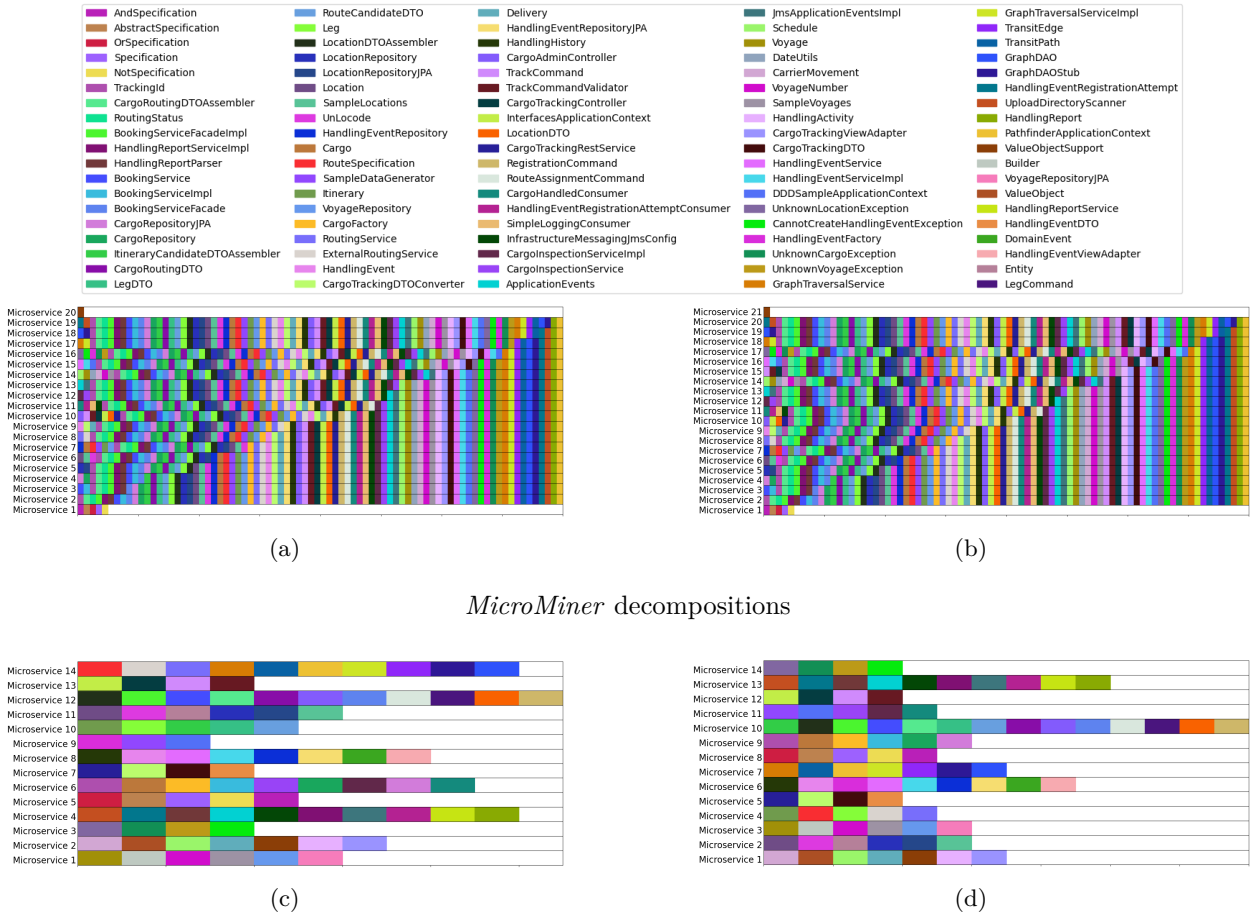
On the other hand, the introduction of the semantic layer appears to support the analysis of the parameter and decomposition spaces, potentially aiding developers in making decisions about candidate decompositions. Although we did not conduct a user study to test the semantic layer, we believe the developers' profiles used by the generator served to emulate human interactions. However, the discrepancies among case studies suggest that the benefits of the semantic layer may depend on the particular characteristics of the application under analysis, such as complexity or domain-specific aspects. Moreover, the stochastic nature of LLMs makes the question-answering process challenging. In this sense, expanding the number of tools, quality metrics, or case studies, as well as refining the metadata for the datasets, could enhance the effectiveness of the semantic layer and reduce hallucinations in the answers.

5.4 Threats to validity

The threats to validity can be divided into the following categories: construct, conclusion, internal, and external [35].

Construct validity refers to the extent to which the evaluation accurately reflects the concept it is intended to measure. The use of a predefined prompt to generate synthetic questions poses a threat to construct validity, since the inherent sensitivity of LLMs to prompt formulation might bias the types of questions generated in our benchmark. Although our prompt was designed to obtain diverse and relevant questions aligned with practitioners' concerns presented in the literature, the evaluation might have not fully captured the effectiveness of the semantic layer in real-world concerns. Experimenting with alternative prompts or practitioners' inputs in future work can help mitigate this limitation.

The internal validity is defined by the causal relations in the experiment. Since our approach is data-driven, it is dependent on the choice of the input parameters for the decomposition approach, and on the



(a) (b)

MicroMiner decompositions

(c) (d)

MIT decompositions

Figure 16: *Cargo* decomposition difference for parameters (a) $resolution = 0.35$, $\alpha = 0.89$, $m\_fuzzy = 9$, and $m\_threshold = 2$, (b) $resolution = 0.35$, $\alpha = 0.89$, $m\_fuzzy = 10$, and $m\_threshold = 2$ (varying m_fuzzy in one unit), (c) $k = 14$ and $resolution = 1.92$, and (d) $k = 15$ and $resolution = 1.92$ (varying k in one unit)

sampling strategy for generating the dataset for the parameter and decomposition spaces. Although we sampled a moderately large set of parametrizations, considering a wider set or additional parameters could affect the subsequent analyses, charts, and observations resulting from the pipeline.

Moreover, the process of connecting developers' questions about the decompositions with the execution of predefined analyses through the semantic layer could inadvertently introduce interpretation biases in cases in which the LLM-based mechanisms do not interpret a question accurately. To mitigate this threat, we included metadata in the query engines and functions to inform the routing and mapping of the developer's questions. Another threat to internal validity is the usage of the same LLM for both generating evaluation questions and implementing the semantic layer, which could introduce internal bias in our evaluation. While the two components are operationally separate -question generation uses prompting while the semantic layer relies on frozen embeddings— there is a possibility that shared linguistic or semantic priors could favor alignment between questions and retrieval outputs. We mitigated this threat by using a fixed, general-purpose prompt and by ensuring that no fine-tuning or feedback loop exists between components by design.

For conclusion validity, we analyze the possibility of drawing inaccurate results from the observations in the experiment. Given the amount of data to be analyzed, which involves the correlation between two multi-dimensional spaces, we mostly analyzed the relevant parameters and aspects of the phenomenon. However, we might have overlooked some aspects (or even configurations) when attempting to identify the decompositions with low variability. For instance, the usage of MDS is appealing for a human understanding of complex spaces, but so some distances among decompositions might have been distorted when mapping them to a 2D space. Furthermore, for the cluster medoids, we did not perform a systematic exploration in the parameter space to make sure that those parametrizations were in neighborhoods with low variability. This study is subject of future work.

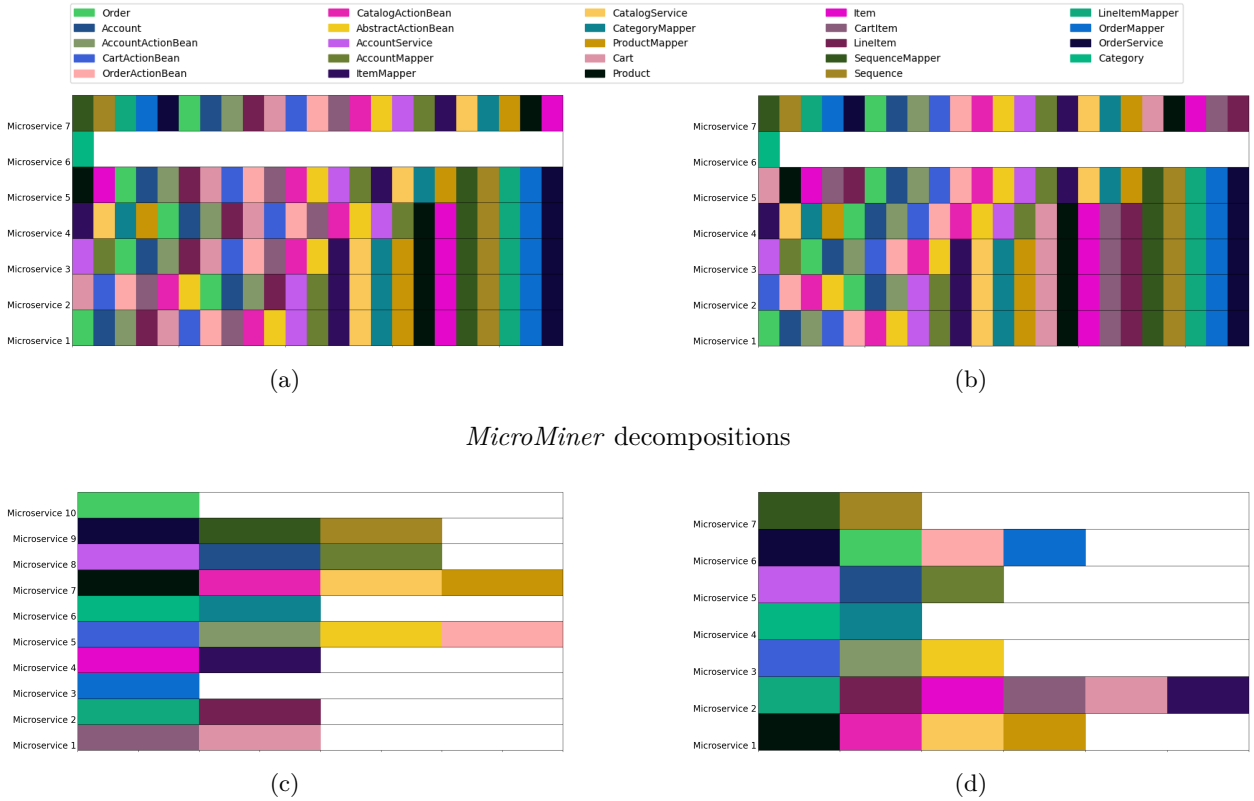


Figure 17: *JPetStore* decomposition difference for parameters (a) $resolution = 0.57$, $\alpha = 0.03$, $m\_fuzzy = 8$, and $m\_threshold = 6$, (b) $resolution = 0.35$, $\alpha = 0.03$, $m\_fuzzy = 8$, and $m\_threshold = 7$ (varying $m\_threshold$ in one unit), (c) $k = 23$ and $resolution = 0.11$, and (d) $k = 24$ and $resolution = 0.11$ (varying k in one unit)

```
k=27
app_description="Cargo Tracking. The main focus of Cargo Tracking is to move a
↪  Cargo (identified by a TrackingId) between two Locations through a
↪  RouteSpecification. Once a Cargo becomes available, it is associated with
↪  one of the Itineraries (lists of CarrierMovements), selected from existing
↪  Voyages. HandlingEvents then trace the progress of the Cargo on the
↪  Itinerary. The Delivery of a Cargo informs about its state, estimated
↪  arrival time, and being on track."
```

Figure 18: Context provided to the question generation prompt for the *Cargo* case study

The external validity is concerned with generalizing the results to other environments. Although our experiments covered two applications for two decomposition tools, and some of our findings and pipeline seem to apply to other parameter-driven decomposition approaches, we acknowledge that experiments with additional approaches and applications are necessary to validate our claims. Another limitation is that the answer relevancy of the semantic layer to practitioners' questions was assessed based on synthetic questions rather than with real participants. We plan to involve industry practitioners in a user study to refine our approach.

6 Related Work

The landscape of microservice architectures and variability management within software engineering has been discussed by various studies addressing diverse challenges and proposing innovative methodologies. In this section, we review relevant literature encompassing the utility of automatically generated microservice architectures, variability management in microservices, reconfigurable migration techniques for IoT systems,

and approaches for transitioning from monolithic to microservice architectures.

In [36] the authors investigate the efficacy and practical implications of automatically generating microservice architectures. By leveraging automated techniques for decomposition, service identification, and orchestration, the research explores potential benefits, such as reduced development effort, enhanced scalability, and improved maintainability. Through empirical evaluations and case studies, the authors assess the utility of automated approaches in diverse application domains, shedding light on their suitability and limitations in real-world scenarios.

In the context of re-engineering microservice-based webshops, this research [37] identifies six key challenges arising from the intersection of variability management and microservice architecture. By addressing issues such as service granularity, interface compatibility, and configuration management, the study offers insights into the complexities involved in managing variability within microservices ecosystems. Through qualitative analysis and practical examples, the authors propose strategies to mitigate these challenges and streamline the re-engineering process effectively.

In [38], the work presents *MSDeveloper*, a methodology that integrates variability management principles into microservice-based development processes. By adopting a guided approach to handle variability at different levels of granularity, the methodology aims to improve the flexibility and adaptability of microservices architectures. Through a combination of modeling techniques, tool support, and best practices, *MSDeveloper* offers a framework for managing variability and promoting modularization in distributed systems.

Addressing the challenges of migrating legacy IoT systems to microservice architectures, this study proposes a reconfigurable migration technique. By leveraging dynamic adaptation mechanisms and service orchestration strategies, the technique enables seamless transition while preserving system functionality and scalability. Through experimental validation and case studies, the research demonstrates the effectiveness of the approach in enhancing the agility and resilience of IoT deployments [39].

Comparing static and dynamic analysis approaches for transitioning from monolithic to microservice architectures, this research [40] offers insights into the trade-offs and considerations involved in migration strategies. By evaluating factors such as performance, maintainability, and deployment overhead, the study provides a comprehensive analysis of different migration techniques. Through empirical evaluations and case studies, the authors highlight the strengths and limitations of each approach, guiding practitioners in selecting suitable migration strategies.

In the realm of microservice variability and decomposition, our approach offers a contrasting perspective by focusing on the variability aspect within microservice architectures. Unlike the broader studies encompassing variability management and migration techniques, this research delves into the empirical analysis of variability patterns and their implications on decomposition strategies. By leveraging data-driven approaches, the study aims to uncover variability factors and their impact on architectural decisions, thereby providing valuable insights into the nuanced dynamics of microservice variability. Through quantitative analysis and empirical validation, we contribute to a deeper understanding of variability management within microservice ecosystems, offering practitioners actionable insights for informed decision-making and architectural design. Thus, while existing literature provides valuable methodologies and techniques for managing variability and transitioning to microservice architectures, this study offers a specialized perspective tailored to the intricacies of variability within microservice decompositions.

7 Conclusions

The exploration of effective microservices decompositions for a monolithic application is crucial for leveraging the advantages of microservice architectures. However, this process is not straightforward, involving technical and managerial considerations, which makes it challenging for developers to assess the pros and cons of different decomposition choices.

This paper employs a data-driven pipeline to analyze the variability of microservice decompositions, enhanced with a semantic layer that relies on an LLM to derive relevant insights. Our approach relies on a simple yet informative set of data techniques (e.g., sensitivity analysis, clustering, dimensionality reduction) that conceives the application being decomposed by a given technique as a data problem, and thus aims to extract relevant information from that data. The idea of having a semantic layer is to simplify the interactions of developers with the complex data sources resulting from the decomposition landscape.

The proposed approach generates a dataset of parameterizations and candidate microservices partitions from the execution of any number of decomposition tools to support parameter and decomposition spaces analysis through a semantic layer. It is important to note that for this reason, this approach is not necessarily tied to techniques such as Sobol o clustering in the analyses. In fact, it is designed to work with other decomposition tools (e.g., *ServiceCutter* [1]) or analyses (e.g., parameter correlation, alternative clustering algorithms), because it is agnostic of the inputs required by a decomposition tool. Whether source code,

runtime logs, database diagrams, or requirement documents are needed to perform decomposition, the approach treats each tool as a black box that generates microservices candidates, and it only requires configuring the tool parameters.

An initial evaluation with two real-world applications demonstrates the potential of our approach for providing guidelines to developers on how to select decompositions with low variability from the landscape of possible solutions. Furthermore, the tests performed on the semantic layer with our benchmark showed the effectiveness of the mechanism for answering common questions, thus promoting the interpretability of the results from our data pipeline.

In future work, we plan to borrow techniques from robust optimization to make the pipeline's sampling process more efficient, so as to approach larger spaces for parameters and decompositions. Furthermore, we expect to define a stability metric that can take into account the correlation between regions of the parameter and decomposition spaces having low variability based on *info-gap* metric from robustness theory [41]. Moreover, we plan to analyze the integration of reinforcement learning techniques, such as those introduced in the RLDec work [42], to explore the decomposition space. Specifically, it would complement our data-driven analyses and the semantic layer to provide decomposition recommendations based on quality metrics. Finally, we will investigate the (automated) generation of summaries from both spaces as user-level explanations of the candidate microservice options.

References

[1] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, *Service Cutter: A Systematic Approach to Service Decomposition*, Sep. 2016, pages: 200.

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

[3] I. Trabelsi, M. Abdellatif, A. Abubaker, N. Moha, S. Mosser, S. Ebrahimi-Kahou, and Y.-G. Guéhéneuc, "From legacy to microservices: A type-based approach for microservices identification using machine learning and semantic analysis," *Journal of Software: Evolution and Process*, vol. n/a, no. n/a, p. e2503, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2503

[4] M. Brito, J. Cunha, and J. a. Saraiva, "Identification of microservices from monolithic applications through topic modelling," in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, ser. SAC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1409–1418. [Online]. Available: https://doi.org/10.1145/3412841.3442016

[5] A. M. Saucedo, J. A. Diaz-Pace, H. Astudillo, and G. Rodríguez, "On the variability of microservice decompositions: A data-driven analysis," in *2024 L Latin American Computer Conference (CLEI)*, 2024, pp. 1–9.

[6] "What Is a Semantic Layer? | IBM," Aug. 2024. [Online]. Available: https://www.ibm.com/think/topics/semantic-layer

[7] A. Koschel, I. Astrova, and J. Dötterl, "Making the move to microservice architecture," in *2017 International Conference on Information Society (i-Society)*, Jul. 2017, pp. 74–79.

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

[9] A. Balalaie, A. Heydarnoori, and P. Jamshidi, *Migrating to Cloud-Native Architectures Using Microservices: An Experience Report*, Jul. 2015.

[10] S. A. Maisto, B. Di Martino, and S. Nacchia, "From monolith to cloud architecture using semi-automated microservices modernization," in *Advances on P2P, Parallel, Grid, Cloud and Internet Computing*, L. Barolli, P. Hellinckx, and J. Natwichai, Eds. Cham: Springer International Publishing, 2020, pp. 638–647.

[11] L. Cao and C. Zhang, "Implementation of domain-oriented microservices decomposition based on node-attributed network," in *2022 11th International Conference on Software and Computer Applications*, ser. ICSCA 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 136–142. [Online]. Available: https://doi.org/10.1145/3524304.3524325

[12] "mybatis/jpetstore-6: A web application built on top of MyBatis 3, Spring 3 and Stripes." [Online]. Available: https://github.com/mybatis/jpetstore-6

[13] A. Martínez Saucedo, G. Rodríguez, F. Gomes Rocha, and R. P. dos Santos, "Migration of monolithic systems to microservices: A systematic mapping study," *Information and Software Technology*, vol. 177, p. 107590, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584924001952

[14] A. Mathai, S. Bandyopadhyay, U. Desai, and S. Tamilselvam, "Monolith to microservices: Representing application software through heterogeneous graph neural network," 2022.

[15] T. Zhong, Y. Teng, S. Ma, J. Chen, and S. Yu, "A microservices identification method based on spectral clustering for industrial legacy systems," 2023.

[16] O. Al-Debagy and P. Martinek, "Dependencies-based microservices decomposition method," *International Journal of Computers and Applications*, vol. 44, no. 9, pp. 814–821, 2022. [Online]. Available: https://doi.org/10.1080/1206212X.2021.1915444

[17] Y. Li, Y. Zhang, Y. Yang, W. Wang, and Y. Yin, "Rm2ms: A tool for automatic identification of microservices from requirements models," in *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. Los Alamitos, CA, USA: IEEE Computer Society, oct 2023, pp. 50–54. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MODELS-C59198.2023.00018

[18] K. Sellami, A. Ouni, M. A. Saied, S. Bouktif, and M. W. Mkaouer, "Improving microservices extraction using evolutionary search," *Information and Software Technology*, vol. 151, p. 106996, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584922001264

[19] S. Hassan and R. Bahsoon, "Microservices and their design trade-offs: A self-adaptive roadmap," in *2016 IEEE International Conference on Services Computing (SCC)*, 2016, pp. 813–818.

[20] J. A. Diaz-Pace and D. Garlan, "The architect in the maze: On the effective usage of automated design exploration," in *2024 IEEE/ACM International Workshop on Designing Software (Designing)*, 2024, pp. 9–14.

[21] J. Nossent, P. Elsen, and W. Bauwens, "Sobol'sensitivity analysis of a complex environmental model," *Environmental Modelling & Software*, vol. 26, no. 12, pp. 1515–1525, 2011.

[22] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Annals of Data Science*, vol. 2, pp. 165 – 193, 2015.

[23] G. Murray, G. Carenini, and R. T. Ng, "Using the omega index for evaluating abstractive community detection," in *EvalMetricsNAACL-HLT*, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:266092455

[24] T. S. Madhulatha, "Comparison between k-means and k-medoids clustering algorithms," in *International Conference on Advances in Computing and Information Technology*. Springer, 2011, pp. 472–481.

[25] "Agents - LlamaIndex." [Online]. Available: https://docs.llamaindex.ai/en/stable/use_cases/agents/#agents

[26] B. Li, K. Mellou, B. Zhang, J. Pathuri, and I. Menache, "Large language models for supply chain optimization," 2023. [Online]. Available: https://arxiv.org/abs/2307.03875

[27] "citerus/dddsample-core: This is the new home of the original DDD Sample app (previously hosted at sf.net).." [Online]. Available: https://github.com/citerus/dddsample-core

[28] A. Lázár, D. Ábel, and T. Vicsek, "Modularity measure of networks with overlapping communities," *Europhysics Letters*, vol. 90, no. 1, p. 18001, apr 2010. [Online]. Available: https://dx.doi.org/10.1209/0295-5075/90/18001

[29] M. A. A. Cox and T. F. Cox, *Multidimensional Scaling*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 315–347. [Online]. Available: https://doi.org/10.1007/978-3-540-33037-0_14

[30] S. Panichella, M. I. Rahman, and D. Taibi, "Structural coupling for microservices," in *Proceedings of the 11th International Conference on Cloud Computing and Services Science - CLOSER*, INSTICC. SciTePress, 2021, pp. 280–287.

[31] S. Li, H. Zhang, Z. Jia, C. Zhong, C. Zhang, Z. Shan, J. Shen, and M. A. Babar, "Understanding and addressing quality attributes of microservices architecture: A systematic literature review," *Information and software technology*, vol. 131, p. 106449, 2021.

[32] F. H. Vera-Rivera, C. Gaona, and H. Astudillo, "Defining and measuring microservice granularity—a literature overview," *PeerJ Computer Science*, vol. 7, p. e695, Sep. 2021, publisher: PeerJ Inc. [Online]. Available: https://peerj.com/articles/cs-695

[33] J. Liu, "LlamaIndex," 11 2022. [Online]. Available: https://github.com/jerryjliu/llama_index

[34] "Answer Relevance | Ragas." [Online]. Available: https://docs.ragas.io/en/v0.1.21/concepts/metrics/answer_relevance.html

[35] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[36] L. Carvalho, T. E. Colanzi, W. K. Assunção, A. Garcia, J. A. Pereira, M. Kalinowski, R. M. de Mello, M. J. de Lima, and C. Lucena, "On the usefulness of automatically generated microservice architectures," *IEEE Transactions on Software Engineering*, 2024.

[37] W. K. Assunção, J. Krüger, and W. D. Mendonça, "Variability management meets microservices: six challenges of re-engineering microservice-based webshops," in *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A*, 2020, pp. 1–6.

[38] B. Kuruoglu Dolu, A. Cetinkaya, M. C. Kaya, S. Nazlioglu, and A. H. Dogru, "Msdeveloper: A variability-guided methodology for microservice-based development," *Applied Sciences*, vol. 12, no. 22, p. 11439, 2022.

[39] C.-a. Sun, J. Wang, J. Guo, Z. Wang, and L. Duan, "A reconfigurable microservice-based migration technique for iot systems," in *Service-Oriented Computing–ICSOC 2019 Workshops: WESOACS, ASOCA, ISYCC, TBCE, and STRAPS, Toulouse, France, October 28–31, 2019, Revised Selected Papers 17*. Springer, 2020, pp. 142–155.

[40] B. Andrade, S. Santos, and A. R. Silva, "From monolith to microservices: Static and dynamic analysis comparison," *arXiv preprint arXiv:2204.11844*, 2022.

[41] T. Roach, Z. Kapelan, and R. Ledbetter, "Comparison of info-gap and robust optimisation methods for integrated water resource management under severe uncertainty," *Procedia Engineering*, vol. 119, pp. 874–883, 2015.

[42] K. Sellami and M. Saied, "Extracting microservices from monolithic systems using deep reinforcement learning," *Empirical Software Engineering*, vol. 30, 10 2024.