USING TOPOLOGICAL FEATURES OF MICROSERVICE CALL GRAPHS TO PREDICT THE RESPONSE TIME VARIATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BARIŞ FINDIK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INFORMATION SYSTEMS

JANUARY 2024

Approval of the thesis:

USING TOPOLOGICAL FEATURES OF MICROSERVICE CALL GRAPHS TO PREDICT THE RESPONSE TIME VARIATION

submitted by **BARIŞ FINDIK** in partial fulfillment of the requirements for the degree of **Master of Science in Information Systems Department, Middle East Technical University** by,

Prof. Dr. Banu Günel Kılıç
Dean, Graduate School of **Informatics** _____

Prof. Dr. Altan Koçyiğit
Head of Department, **Information Systems** _____

Prof. Dr. Banu Günel Kılıç
Supervisor, **Information Systems, METU** _____

Assoc. Prof. Dr. Aysu Betin Can
Co-supervisor, **Information Systems, METU** _____

Examining Committee Members:

Prof. Dr. Altan Koçyiğit
Information Systems, METU _____

Prof. Dr. Banu Günel Kılıç
Information Systems, METU _____

Assoc. Prof. Dr. Ayça Tarhan Kolukısa
Computer Engineering, Hacettepe University _____

Date:19.01.2024

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Barış Fındık

Signature :

ABSTRACT

USING TOPOLOGICAL FEATURES OF MICROSERVICE CALL GRAPHS TO PREDICT THE RESPONSE TIME VARIATION

Fındık, Barış
M.S., Department of Information Systems
Supervisor: Prof. Dr. Banu Günel Kılıç
Co-Supervisor: Assoc. Prof. Dr. Aysu Betin Can

January 2024, 97 pages

Microservice architectures are increasingly gaining popularity in the field of software design. Research on the topology of graphs formed by communication between microservices is a subdomain within the broader scope of microservice architecture research. Although there have been studies examining the relationship between topology and response time, the variability in response time and its connection to topology has not been thoroughly explored. To ensure performance stability, architectures with low response time variation are needed. Low response time variation enables the creation of more predictable and easily testable systems. In this study, machine learning models trained with the topological features of microservice call graphs are used to explore the impact of topology on predicting response time variation. The feature importance of models achieving successful and significant results is examined. The centralization, modularity, node count, average degree, and loop count features obtained from 70,000 microservice call graphs are used to train random forest, LightGBM, and CatBoost classifiers. These models aim to predict whether the call graph belongs to the class of high response time variation or low response time variation. The fea-

ture importance of models achieving F1 score, accuracy, and precision higher than 0.8, along with statistically significant results from the McNemar test, is examined using SHAP values and dependence plots. Finally, models predicting response time variation based on topological features and understanding the impact of topological features on response time variation are obtained in this thesis.

Keywords: Microservices, machine learning, topological features, software architecture, graph algorithms

ÖZ

MİKROSERVİS ÇAĞRI AĞLARININ TOPOLOJİK ÖZELLİKLERİNİ KULLANARAK YANIT SÜRESİ DEĞİŞKENLİĞİNİN TAHMİN EDİLMESİ

Fındık, Barış
Yüksek Lisans, Bilişim Sistemleri Bölümü
Tez Yöneticisi: Prof. Dr. Banu Günel Kılıç
Ortak Tez Yöneticisi: Doç. Dr. Aysu Betin Can

Ocak 2024 , 97 sayfa

Mikroservis mimarileri, yazılım tasarım alanında giderek daha fazla popülerlik kazanmaktadır. Mikroservisler arasındaki iletişimden oluşan ağların topolojisi üzerine yapılan araştırma, mikroservis mimarisi araştırmalarının bir alt alanını oluşturur. Topoloji ile yanıt süresi arasındaki ilişkiyi inceleyen çalışmalar olmasına rağmen, yanıt süresindeki değişkenliğin topoloji ile olan bağlantısı tam anlamıyla keşfedilmemiştir. Mikroservis performans istikrarı, mimarinin kalitesini gösteren kritik bir metrik olarak hizmet verir. Performans istikrarını sağlamak için düşük yanıt süresi değişkenliğine sahip mimarilere ihtiyaç vardır, bu da daha öngörülebilir ve kolay test edilebilir sistemlerin oluşturulmasını sağlar. Deneysel gözlem, çağrı ağı topolojisinin yanıt süresi değişkenliği üzerinde önemli bir etkisi olduğunu ortaya koymakta, bu da bu ilişkinin araştırılmasının önemini vurgulamaktadır. Bu çalışmada, mikroservis çağrı ağlarının topolojik özellikleri ile eğitilen makine öğrenimi modelleri kullanılarak bu etkiyi keşfetmek ve yanıt süresi değişkenliğini tahmin etmek amaçlanmaktadır. Başarılı ve istatistiksel olarak anlamlı modellerin özellik önemi incelenir. Bu özellik önemi

bilgisi ile yanıt süresi değişkenliği açısından stabil olan topolojik tasarımların anlaşılması hedeflenmektedir. 70 bin mikroservis çağrı ağından elde edilen merkezilik, modülerlik, mikroservis sayısı, döngü sayısı ve ortalama derece özellikleri rastgele orman, LightGBM, CatBoost modellerini eğitmek için kullanılır. Bu modeller çağrı ağının düşük ve yüksek yanıt değişkenliği sınıflarından hangisine ait olduğunu tahmin etmeye çalışır. İstatistiksel olarak anlamlı ve 0.8'den yüksek F1 skor, doğruluk ve kesinlik elde eden modellerin SHAP değerleri ve bağımlılık grafikleri incelenerek özelliklerin model çıktısı üzerindeki etkisi incelenir. Sonuç olarak bu tezde, yanıt süresi değişkenliğini topolojik özellikleri kullanarak sınıflandıran modeller ve topolojik özelliklerin yanıt süresi değişkenliğindeki etkisi elde edilir.

Anahtar Kelimeler: Mikroservisler, makine öğrenimi, topolojik özellikler, yazılım mimarisi, ağ algoritmaları

To my family and friends...

ACKNOWLEDGMENTS

Firstly, I would like to express my gratitude to my advisor, Prof. Dr. Banu Günel Kılıç, and my co-advisor, Assoc. Prof. Aysu Betin Can. I could complete this thesis thanks to what they taught me, the motivation they provided, and the quick responses they gave when I needed them. Working with such advisors who make communication and obtaining the desired answers so easy was truly a great opportunity for me.

I would like to thank my family for the lifelong support they have given me. I want to express special thanks to my mother Serap Yıldız Fındık for always supporting and motivating me.

Finally, I would like to express my gratitude to my friends for their trust, to my colleagues for their support.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ML	Machine Learning
CV	Coefficient of Variation
RT	Response Time
RTV	Response Time Variation
RTVT	Response Time Variation Threshold
MS	Microservice
DB	Database
RPC	Remote Procedure Call
SHAP	Shapley Additive Explanations
LightGBM	Light Gradient Boosting Machine
RF	Random Forest
CSV	Comma Separated Values
UM	Upstream Microservice
DM	Downstream Microservice
API	Application Programming Interface
KS	Kolmogorov-Smirnov
GMAT	Graph-based Microservice Analysis and Testing
SCC	Strongly Connected Components
MCI	Microservice Coupling Index
DAG	Directed Acyclic Graph
AWS	Amazon Web Services
GAE	Google App Engine

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

Microservices architectures are becoming an increasingly popular concept in the software design world [1]. Since it was first discussed in a software design workshop by James Lewis and Martin Fowler in May 2011, various studies have been conducted, and different suggestions have been made regarding how microservices architectures should be designed [2]. According to Lewis and Fowler's definition, "Microservice Architecture" is a contemporary approach to designing software applications into independently deployable services, characterized by features such as organization around business capability, automated deployment, intelligence in endpoints, and decentralized control of languages and data [3].

The primary reason of the popularity of microservices is their advantages over monolithic architectures. These advantages include straightforward scalability, clear separation of responsibilities among teams, independent deployment capability, and faster response to customer feedback through quick releases [4]. Additionally, the architecture introduces flexibility in technology selection by creating technological heterogeneity among software development teams.

Research on microservices contains various critical domains, including design, testing, configuration management, migration strategies for legacy systems, architectural support through reference architectures and modeling, platform support with a focus on testing, deployment, and microservice identification tools [5]. The exploration of these research fields aims to advance our understanding and implementation of microservices, microservice call graph topology, quality assurance, deployment, and

domain-specific considerations to promote effective and scalable software development practices.

The topology of graphs consisting of communications between microservices is also an essential subdomain in microservices architecture design. Some studies use topology information to optimize microservices tests [6][7], for root cause analysis [8], anomaly detection [9], dealing with cyclic dependencies [10], and measuring architecture quality [11][12]. There is also a study that uses call graph topology information to measure coupling in microservices architectures where low coupling is aimed [13].

Researchers analyzing the Alibaba Cloud Platform's cluster with microservices and traces have contributed to studies on the graph topology and dependencies of Data Parallel Jobs [14]. Another study focuses on microservice dependencies and performance using call graph topologies [15]. Additionally, there is a study that analyzes the relationship between microservice call graphs and runtime performance along with topology information and allocated resources to microservices[16]. In addition to conducting these studies, researchers have shared the datasets they used to enable further research.

Researchers analyzing the Alibaba cluster trace have shared a dataset in their study [15] on microservice dependencies and call graph characteristics. This dataset includes calls between microservices, the microservices themselves, and the time when the calls occurred. In their study, they aim to predict the response time of calls between microservices using graph learning algorithms. One of their significant findings is that the response time highly depends on topology.

While it is shown that response time is significantly affected by call graph topology, their study does not provide information on which topological features are crucial due to the algorithm used. The algorithm aimed to achieve high prediction scores, and the study did not focus on identifying which topological features are influential and how they affect response time.

Although there is an advanced study on response time prediction [15], there is no comprehensive study in the literature using a dataset like the Alibaba cluster trace

dataset on response time variation, which has significant effects on the predictability and testability of systems in production [17].

Response time variation is a very important metric for the predictability and cost management of systems running on the cloud [18]. Even in the selection of cloud service providers, low response time variation is an essential factor [19]. Therefore, it is a valuable scientific research topic.

The lack of examination of the relationship between the topology of microservice call graphs and response time variation and the absence of a study using a comprehensive dataset to investigate how call graph topological features affect architecture are the main motivations for this thesis. Various research questions have been asked based on this motivation, and their answers are provided in this thesis.

These research questions can be listed as follows:

- Can the response time variation of call graphs be predicted using the topology of the microservice call graph? How successful can machine learning algorithms be in this prediction task?
- In microservice architectures, which are mentioned in the literature with decentralization and modularity frequently [3][20][21][22][23][24][25][26], are the centralization and modularity features obtained from the topology of call graphs effective on response time variation? If so, how are they effective?
- What is the impact of features such as node count, average degree, and loop count obtained from microservice call graphs on response time variation?

1.2 Proposed Methods and Models

To investigate the relationship between topological features and response time variation, a sample is extracted from the Alibaba dataset [15]. Alibaba dataset consists of 20 million call graphs. Before sampling, call graphs with a node count fewer than 40 are excluded, and only those with a node count greater than or equal to 40 are retained. In social network analysis, there have to be at least 30 or 40 nodes to reach

meaningful network modularity values. Specifically, using call graphs with a low node count for the modularity metric would lead to issues in its computation. The selection of a node count filter at 40 is based on the paper that shared the dataset because the paper shared that 10% of call graphs have a node count greater than 40. This number seems enough to reach statistically significant results.

Random call graphs from different time intervals within the 7-day Alibaba dataset are selected and divided into three subsets, and their response time variation distributions are calculated. To compute the response time variation of a call graph, the coefficient of variation for each call observed on each edge of the call graph is calculated. Eventually, the average of all calls is calculated to find the response time variation of the call graph. The dataset is expanded until each of the three subsets has a similar response time variation distribution. The Kolmogorov-Smirnov [27] test is applied to verify whether the response time variation distributions of the three subsets are similar.

Topological features believed to influence response time variation are identified as closeness centralization, betweenness centralization, harmonic centralization, modularity with the Louvain algorithm, modularity with label propagation algorithm, node count, loop count, and average degree. These features are calculated using the Python programming language's networkx[28], pandas[29], and numpy[30] libraries. Call graphs with node count, loop count, and average degree values in the lowest 5% quantile and above the 95% quantile are considered outliers and eliminated. In the final step, a sample containing 70,000 microservice call graphs is obtained. All call graphs are used in their undirected form.

When examining the Pearson correlation coefficient [31]between the calculated topological features and response time variation, no significant correlation is observed. To assess feature importance, a methodology inspired by a study on virality [32] in social media is adopted. Using different threshold values, the problem is transformed from a regression problem to a binary classification problem. This threshold value is referred to as the Response Time Variation Threshold (RTVT). Call graphs with response time variation above the RTVT are labeled as fluctuating, while those below are labeled as steady. The prediction algorithm aims to determine whether a call graph is steady or

fluctuating based on topological features. The experiment is repeated with different RTVT values, and the differences, along with the effects of features, are interpreted.

After determining the topological features and response time variation labels for microservice call graphs, the Random Forest classifier [33], CatBoost [34] classifier, and LGBM classifier [35] models are trained with these features and labels. Half of the dataset is used as test data, and accuracy, precision, and F1 scores are examined. These scores are compared with the scores of a dummy classifier that makes predictions based on label distribution. The McNemar [36] test is applied using the predictions of the dummy classifier and models. When a statistically significant difference is obtained from the McNemar test, the first research question, "Can the response time variation of call graphs be predicted using the topology of the microservice call graph? How successful can machine learning algorithms be in this prediction task?" is answered.

To learn the impact of topological features, the SHapley Additive exPlanations (SHAP) [37] method is used. This method helps understand which features ML models use to achieve successful predictions. Additionally, it allows the investigation of the ranges of these topological features that affect whether a microservice call graph is steady or fluctuating. Through SHAP, the second research question, "In microservice architectures, which are mentioned in the literature as advantageous for being decentralized and modular, are the centralization and modularity features obtained from the topology of call graphs effective on response time variation? If so, how are they effective?", and the third research question, "What is the impact of features such as node count, average degree, and loop count obtained from microservice call graphs on response time variation?" are answered.

In this thesis, while investigating the research questions, there are certain terminology that are commonly used in various domains with different meanings. Therefore, we want to clarify this terminology. In graph theory and social network analysis, the term "node" refers to a vertex in the graph, and in this thesis, when referring to a "node", it is meant in the sense of a vertex, not a physical machine. The terms "centralization" and "modularity" have multiple meanings in software architecture and microservices. In this thesis, their meanings from social network analysis and graph theory

are adopted. Network centralization refers to the extent to which control, decision-making, and communication within a network are concentrated in a specific location or a few nodes, network modularity is a measure of the strength of the division of a network into modules or subgroups. Networks with high modularity have dense connections between the nodes within modules but sparse connections between nodes in different modules.

1.3 Contributions

The contributions of the thesis are listed below.

- This thesis demonstrates empirical evidence that the topology of microservice call graphs can be used to predict the response time variations of call graphs through machine learning models. It explains the procedure outlining how these machine learning models are constructed.

- This thesis explains the impact of centralization and modularity features derived from the topology of microservice call graphs on response time variation through machine learning models and SHAP values. On the Alibaba dataset, the thesis identifies the advantageous intervals in terms of response time variation for these topological features.

- This thesis determines the effects of features such as node count, average degree, and loop count obtained from microservice call graphs on the response time variation of these call graphs through machine learning models and SHAP values. The intervals that are advantageous for response time variation are also identified.

- For designers aiming to create a system with low response time variation, this thesis experimentally highlights the importance of topology in achieving low response time variation. Various insights regarding the details of the designs with low response time variation are shared.

1.4 The Outline of the Thesis

Chapter 2 provides background information on microservice architectures and the algorithms used, along with related research. Chapter 3 is the methodology section, encompassing details about the created dataset, methods for calculating topological features, and response time variation labels from microservice call graphs. Additionally, it outlines the methodologies for using machine learning models. Chapter 4 contains the experiments conducted to answer the research questions and discusses the results. It also includes comments on the limitations of the thesis and threats regarding the validity of the results. Chapter 5 interprets the findings, explains future work related to the research topic, and concludes the thesis.

CHAPTER 2

BACKGROUND AND RELATED RESEARCH

2.1 Introduction

This chapter provides information that facilitates understanding of research similar to the thesis and explains the terms and methods used in the thesis. The research in the thesis investigates the relationship between the topological features of microservice call graphs and the response time variation of the call graph. In this context, related research includes the definition of microservice architecture and research on microservice call graph topology. Background information on these topics is also covered in this chapter. Other studies that have used the dataset to investigate the relationship between response time variation and microservice call graph topology are also included in the related research. Research on the importance of response time variation and how it is measured in different studies is also presented in this chapter. Additionally, this chapter provides information on inspiring research on machine learning algorithms and feature explanation, which constitute an important part of the methodology, to increase understanding of these methods.

Research on microservice architectures, which is the main topic of the thesis, and background information are found in Section 2.2. Sections related to response time variation are located in Section 2.3. Research and background information on machine learning and feature explanation are presented in Section 2.4.

2.2 Microservices

Lewis and Fowler introduced the concept of microservices. According to their definition, the term "Microservice Architecture" is a modern method of structuring software

applications into autonomously deployable services. This approach includes emphasis on organizing around business capability, implementing automated deployment, incorporating intelligence in endpoints, and adopting decentralized control over data management [3].

The main factor contributing to the adoption of microservices is its benefits compared to monolithic architectures. These advantages are scalability, division of responsibilities among teams, autonomous deployment capabilities, and the ability to quickly respond to customer feedback through rapid releases [4]. Also, the architecture introduces flexibility in technology selection by advancing technological diversity among software development teams.

Investigations of microservices span a range of areas, including design, testing, configuration, and management. They also include migration strategies for legacy systems, processes for identifying microservices, architectural suggestions, and platform support with an emphasis on testing, deployment, and microservice identification tools [5]. The exploration of these research domains enhances our comprehension and application of microservices, microservice call graph topology, quality assurance, deployment, and domain-specific considerations, with the ultimate goal of promoting efficient and scalable software development practices.

There are many topics studied on microservices, and this thesis focuses on the topologies of microservice call graphs. The diversity of topologies that can be used in microservice architectures and the effects of these topologies can be investigated in various ways. This thesis specifically investigates the relationship between the topological features of microservice call graphs and response time variation. Therefore, studies exploring the relationship between microservice topologies and various features of architectures are considered as related research in this thesis.

As the usage of call graph topology information has increased in the literature, there is an open microservice dependency dataset available for researchers [38]. This dataset extracts the dependencies of open-source projects, providing researchers with access to their call graphs. Additionally, Alibaba shares the cluster trace data in their cloud in online platforms [39].

Graph-based Microservice Analysis and Testing (GMAT) is one of the studies related to microservice topology [6]. It introduces a method to help the development of Microservice architectures. Addressing the challenge of handling complex call relationships between microservices, GMAT proposes a service dependency graph for analyzing and visualizing dependencies, enabling early anomaly detection and linkage tracing during development. GMAT has three primary research goals. They are visualizing dependencies, detecting cyclic references, and improving service test coverage. GMAT automates service dependency graph generation to detect risky services and increase developers' ability to trace cyclic dependencies. A team mostly comprised of researchers who developed GMAT has also conducted a study on microservice retrieval, using methods similar to GMAT along with vector space model and word2vec [7].

There is a research paper that focuses on service-oriented and microservice architectures, aiming to divide applications into loosely coupled services for rapid development and continuous feature deployment [8]. However, the complexity of architectures can prevent observability and maintenance, the paper introduces a root cause analysis framework based on graph representations, demonstrating its effectiveness compared to a machine learning method. The advantages of a graph-based representation include facilitating an exploration of connected components, helping decision-making in complex architectures, and offering anomaly detection. The incorporation of expert knowledge enhances understanding, especially in critical systems, where machine learning decisions may lack transparency. Semantic graphs extend possibilities for incorporating ontology, enabling user-friendly access and defining alerts with semantic meaning. The system's flexibility in using and tuning weights and thresholds provides a decision support system for troubleshooting and diagnosing problems in microservice architectures.

The topology of the call graphs can be useful to detect anomalies. There is a study that uses topology information by utilizing execution traces to automate the anomaly detection process [9]. The paper addresses the challenge of detecting anomalies and locating root causes in microservices, aiming to enhance their reliability. They state that the current methods often rely on the manual selection of observed metrics or expert analysis of execution traces, resulting in limitations in accurately locating faults

at the microservices and components level. The proposed approach utilizes dynamic tools to collect execution traces, using call trees to describe application execution. Anomalies are detected by calculating the anomaly degree using tree edit distance and locating faulty components. The approach is evaluated using a microservice-based application, demonstrating 81%–97% precision and 75%–99% recall in detecting anomalies caused by CPU, network, memory, and service faults. Contributions include reducing the number of analyzed traces, automatic anomaly detection through trace comparison, and experimental validation of the approach's effectiveness.

There is a paper that tries to solve the challenge of detecting cyclic dependencies, a common anti-pattern in microservices architecture that can prevent the desired independence and decoupling of services. The proposed graph-based solution employs the strongly connected components (SCC) algorithm to automatically detect cyclic dependencies at design time. The authors explain the difficulty in achieving decoupled independence in microservices and highlight the negative impact of cyclic dependencies on system reliability, maintenance, and scalability. The paper presents a process, starting with transforming microservices architecture into a graph representation, executing the SCC algorithm, and analyzing the results. Two case studies, Lakeside Mutual and a customized e-commerce application, are used to demonstrate the effectiveness of the approach.

Microservice call graph topology is also used in the architecture validation framework named MicroValid [11]. MicroValid offers a validation framework for microservice architectures by measuring the quality using various quality attributes. This validation framework evaluates quality attributes in three categories: granularity, coupling, and cohesion. For granularity, the number of nano entities and lines of code in microservices is calculated, and the coefficient of variation(CV) is determined based on these numbers. For coupling, the coefficient of variation of outdegrees of microservices is calculated, and another metric is obtained by dividing the number of strongly connected components in the architecture by the number of microservices. For cohesion, the coefficient of variation of entity numbers, the coefficient of variation of responsibility numbers, and a score obtained through semantic similarity algorithms are used. After calculating various metrics for these three categories, a validation score is obtained for each category. This study particularly claims that the usage of

strongly connected components in the coupling section and the usage of degrees of microservices are parts of the topology that need validation.

The coupling in microservice architectures is studied in another paper [13]. In this paper on software quality management, the authors try to solve the challenge of measuring and comparing the coupling of microservice architecture. They introduce a set of novel metrics called the Microservice Coupling Index (MCI) derived from relative measurement theory, aiming to assess the dependency and coupling among microservices. The dependency information comes from the topology of the call graphs. The study involves 15 open-source projects with 113 distinct microservices, and the results show that MCIs outperform existing metrics in discriminating high and low-coupled microservices. The metrics are also correlated with change impacts, indicating that higher MCIs are associated with greater difficulty in localizing changes and evolving individual microservices independently. The paper suggests practical implications for refactoring based on MCI values and discusses potential means to improve MCIs.

There are studies that focus on microservice call graph topology or utilize microservice call graph topologies. Datasets obtained from Alibaba clusters provide a comprehensive source for such studies. Consequently, microservice architectures in previous research have more limited resources compared to those offered by the Alibaba dataset. One of the diverse datasets obtained from Alibaba's clusters is based on directed acyclic graphs (DAGs). The topologies of these graphs are not as extensive in terms of diversity as the dataset used in this thesis. However, significant research has also been conducted on DAGs [14]. In the DAG research on Alibaba clusters, they say that the comprehension of DAG structures and their runtime behaviors in expansive production clusters plays an important role in scheduler design. The research helps scheduler design by conducting an investigation using a recently released cluster trace from Alibaba. Their analysis reveals that the DAGs of Alibaba jobs show sparsely connected vertices and can be approximated as multiple trees with limited depth. Additionally, they delve into the runtime performance of these DAGs, demonstrating significant variability in resource usage and duration among dependent tasks, even for recurrent tasks. In contrast to query jobs in standard benchmarks, they find that they inadequately represent the characteristics of production DAGs in both struc-

ture and runtime performance. To enhance the benchmarking of DAG schedulers at scale, they introduce a workload generator capable of accurately synthesizing task dependencies based on the production Alibaba trace. Their extensive evaluations confirm that the synthesized DAGs closely imitate the statistical attributes of production DAGs, and the scheduling outcomes with various schedulers remain consistent between synthesized and real workloads.

Apart from the study conducted on DAGs, there is another paper [15] focusing on microservice architectures, which shares the dataset used in this thesis. It also investigates the correlation between response time and call graph topology. The shared data is not only about the call graph topology, but also the resources of microservices. The paper presents a groundbreaking analysis of large-scale microservices deployed in Alibaba clusters, offering a comprehensive characterization of their structural properties and runtime performance. Notably, microservice call graphs show a dynamic nature with a heavy-tail distribution in size, displaying a tree-like topology where most nodes have in-degrees of one. The study identifies hot spots and multiplexing behaviors among microservices, showcasing dynamic call dependencies and highlighting the unique characteristics of microservice graphs compared to traditional DAGs. The paper introduces a stochastic model for simulating dynamic microservice call graphs. Moreover, the research emphasizes the sensitivity of microservice performance to CPU interference, providing insights for efficient scheduling and resource management. The study contributes an examination of existing microservice benchmarks and proposes roadmaps for future research, positioning itself as a pioneering exploration into the complex dynamics of large-scale microservice deployments. The authors also reveal important information about the call graph topology, stating that the response times of online microservices are strongly influenced by the call graph topology and there is a need for awareness of topology in order to enhance the efficiency of microservice architectures. They also say that the call graph topology is a significant challenge.

The dataset, shared in conjunction with Alibaba's papers, is used in other studies. Another study utilizing data from Alibaba clusters tries to predict latency by using its own graph neural network algorithm [40]. The input for their designed model is the topology of the call graph. In a separate study, a topology-aware scheduling

framework has been generated, with the goal of optimizing scheduling operations in the cloud [41]. Additionally, a study in the dataset automates the process of scaling resources using workload information [42].

2.3 Response Time Variation

The response time is the duration required to send the data, process it through the computer, and transmit the resulting response back to the upstream service. Response time is a crucial key performance indicator for cloud platforms, and fast systems are preferred [20]. While fast systems have their advantages, it is also important for the system to show stable performance in terms of response time. This ensures that the system is predictable and testable[17]. It is also important for reducing costs in cloud platforms [18]. Therefore, response time variation in cloud platforms is a worthwhile research field.

There is a research on performance variation of cloud service providers. The researchers state that the low-performance variation is an important metric for selection of cloud service providers [19]. Their research methodology involves characterizing the performance variability of cloud services. The researchers first create meaningful datasets from performance traces obtained from production clouds. They use performance indicators, defined as stochastic variables describing the performance of operations or sequences of operations over time. The response time is evaluated as the key performance indicator in their research. Performance traces for AWS and GAE are sourced from Hyperic's CloudStatus team [43], providing real-time values and weekly averages for various performance indicators. The analysis method consists of three steps for each trace: determining the presence of variability, identifying the main characteristics of variability, and analyzing variability time patterns. Variability is assessed using statistical measures, such as quartiles, mean, standard deviation, inter-quartile range, and coefficient of variation. The researchers also investigate variability over time, investigating yearly, monthly, weekly, and daily patterns. They reveal that the coefficient of variation above 1.10 means high variability for their dataset.

2.4 Machine Learning Models

Machine learning has become a highly popular field today. The development of new machine learning algorithms allows solutions to previously unsolvable problems, and significantly improved performance in existing problems is achieved through these algorithms [44]. According to Jordan and Mitchell, machine learning is a field that grows around two questions: How can computer systems be designed to enhance themselves autonomously through experience? What are the fundamental statistical, computational, and information-theoretic principles that govern all learning systems, encompassing computers, humans, and organizations? They claim that investigating machine learning is essential for solving these fundamental scientific and engineering questions.

The ability of machine learning algorithms to address complex problems is used in this thesis. Criteria such as the success of models, the ability to measure the impact of features, and the provision of a fast working environment are considered due to the existence of various machine learning algorithms. Research is conducted on performance, feature interpretation, and operational speed to achieve these objectives. Specifically, the research questions in the thesis, which focused on feature interpretability, highlighted certain models. The studies use the three models in the thesis. The models are CatBoost, Random Forest, and LightGBM, provided insights into the selection of these models. Random Forest, being an older model compared to the other two, is expected to perform less effectively. Information regarding whether hyperparameter optimization is correctly performed for the other two models could be handled by using a random forest model. While not precisely a baseline model, its presence alongside the other two models is advantageous for making comparisons. There are studies using Random Forest for feature interpretation [45][46][47], and similarly, LightGBM is a model used in the literature for feature interpretation [48][49]. CatBoost's documentation on its website already references a study where feature interpretation was performed using SHAP values[50]. Since SHAP values were considered during the development of the CatBoost model, it is successful in this area. Numerous examples in the literature also show the usage of CatBoost for feature interpretation[51][52][53][54]. Additionally, the CatBoost model has been

able to achieve state-of-the-art results in many tasks where the performance of other machine learning models has also been evaluated[55][34].

When using Machine Learning models, understanding their operational mechanisms is crucial. In this context, the working mechanisms of the models are examined and summarized briefly in this chapter.

Random Forest model is an ensemble of decision trees through bootstrap sampling and random feature selection [33]. This ensemble approach increases the model's performance by solving overfitting problems and improving generalization to new data. During training, each decision tree is built using a subset of the training data and a random subset of features at each node. In the prediction phase, the outcomes of individual trees are combined by voting for classification or averaging for regression, providing a robust and accurate final prediction. Random Forest is a good option for large datasets, it requires minimal preprocessing, and it is efficient in terms of both training and prediction. Moreover, the model offers a valuable feature importance ranking, helping in the interpretation of the significance of each feature in contributing to the model output.

LightGBM is a gradient boosting algorithm. It works by using a tree based learning algorithm to create an ensemble of decision trees [35]. Its working mechanism includes a depthwise tree growth strategy, prioritizing nodes that lead to larger gains in training loss. This approach contributes to faster training times and improved efficiency, especially with large datasets. LightGBM also supports categorical features naturally without the need for encoding. In terms of feature interpretation, LightGBM provides a feature importance score, aiding in understanding the contribution of each feature to the model's predictive power. Its ability to handle complex relationships, efficient computation, and feature interpretability makes LightGBM a powerful tool for various machine learning tasks.

CatBoost is a gradient boosting algorithm. It distinguishes itself through its working mechanism that incorporates an ordered boosting technique and careful handling of categorical features [34]. It builds an ensemble of decision trees in a sequential manner, giving preference to features with greater categorical cardinality. This ordered boosting strategy contributes to improved performance and robustness. Cat-

Boost automatically handles categorical features without the need for preprocessing, and it incorporates a dynamic learning rate adjustment for faster convergence. In terms of feature interpretation, CatBoost provides a feature importance ranking, aiding in understanding the relevance of each feature in predicting the target variable. Its ability to efficiently manage categorical data, coupled with its performance and interpretability, makes CatBoost a valuable choice for machine learning tasks.

After the model selection research, a research was conducted on the methodology for feature interpretation. Inspiration is drawn from the methodology of a study aimed at understanding the reasons behind viral tweets [32], supporting the methodology of this thesis. Once features that could influence virality are identified in this study, instead of predicting retweet counts, the retweet count is divided into two classes: viral and non-viral tweets, using a threshold value. By varying this threshold value, the results of classification problems with different thresholds are compared. In this study, the feature importances of models that achieved meaningful results for the classification problem using a generalized linear model are examined.

CHAPTER 3

METHODOLOGY

3.1 Introduction

The research questions in this thesis are related to understanding the effects of topological features of microservice call graphs on response time variation (RTV) and predicting whether a microservice call graph, based on its topological features, shows fluctuating or steady behavior in terms of response time variation. To answer these questions in a data-driven manner, a sufficient number of microservice call graphs, along with their calculated topological features and response time variation information, are required. For this purpose, the topological features and response time variations of microservice call graphs are computed on a sample taken from the Alibaba Dataset.

To understand the impact of topological features on RTV and measure their ability to predict RTV, the problem is formulated as a classification problem. Instead of directly predicting RTV, microservice call graphs are divided into two classes based on low and high RTV, treating it as a classification problem. Since the concept of low and high response time variation is relative, multiple classification problems are created with different threshold values, and the results for each threshold value are shared. The class with low RTV is called "steady," while the class with high RTV is called "fluctuating."

Figure 3.1 includes the steps of the research methodology. According to these steps, firstly the Alibaba dataset is read and sampled. Missing and unwanted parts are removed from the sampled data. The undirected topological features and RTV labels of the sampled microservice call graphs are calculated. Three different machine learning

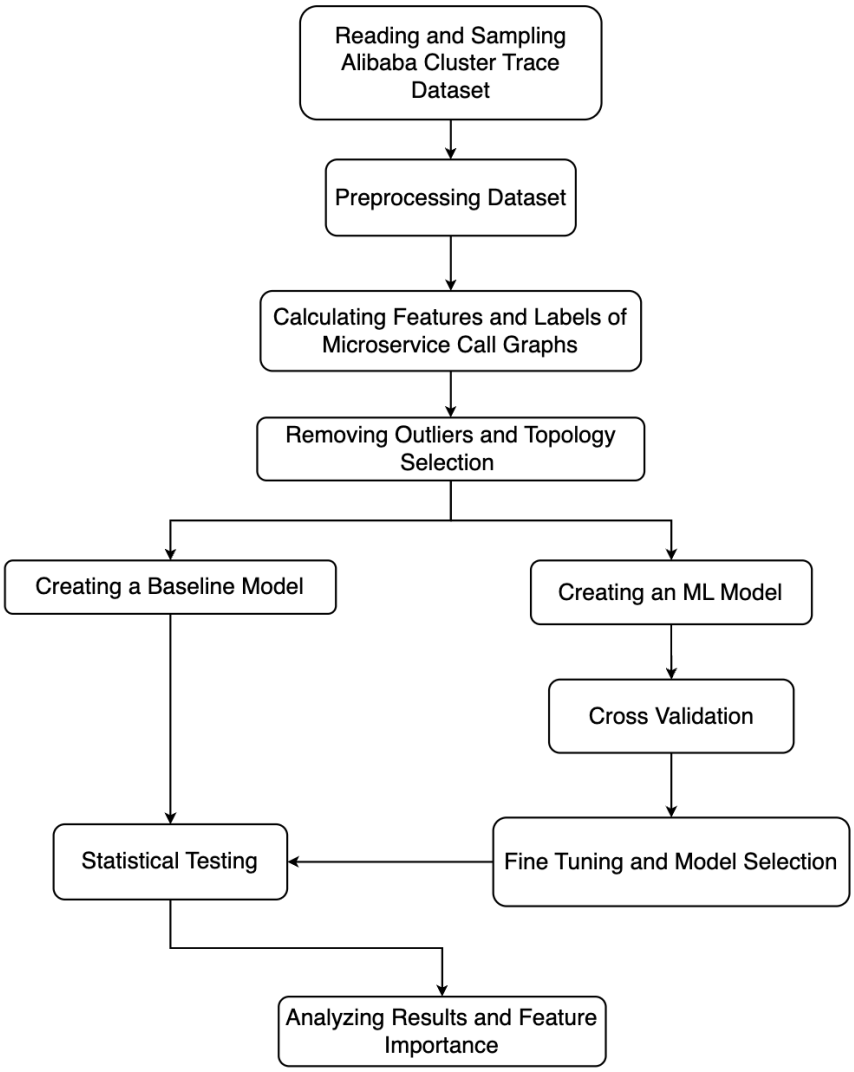

Figure 3.1: Overview of the Methodology

models are trained using these features and labels. Since it is a binary classification problem, the results of the models are statistically tested using the McNemar [36] test and a baseline model generating stratified random predictions. The feature importance of the models that pass the statistical test is computed using SHAP. Finally, the feature importance of the models is analyzed.

In this chapter, Section 3.2 explains the details of the Alibaba dataset. Section 3.3 provides information on how the topological features of microservice call graphs and response time variation labels are calculated and how the tabular dataset suitable for training machine learning models is created. Section 3.4 covers the training process

of machine learning models and the stages of statistical testing.

3.2 Dataset

The Alibaba dataset is the most comprehensive source in terms of creating call graphs from microservices traces. It contains 10 billion calls, 20k+ microservices, and 20 million call graphs.

10% of call graphs have more than 40 nodes. There are many calls between the same upstream and downstream microservices. All of these calls have various response time values.

Alibaba cluster trace data is shared in a repository. The repository and the dataset could be obtained by running the following commands in the terminal.

```
git clone https://github.com/alibaba/clusterdata.git
cd cluster-trace-microservices-v2021
bash fetchData.sh
```

Alibaba dataset contains 10 billion calls between microservices and 20 million call graphs. To reach statistically significant results, 70k call graphs are downloaded without loss of generalizability.

The fetchData.sh file in the repository consists of consecutive traces. To sample traces from different time intervals without downloading the entire dataset, it is necessary to change the fetchData.sh file in the repository. First of all, since MSRTQps and MSResource folders are not needed, the parts related to them can be removed from the sh file. Afterward, calling the wget command in the file one by one with different numbers, rather than in a loop, works to collect efficient samples from different time intervals and topologies.

```
wget -c --retry-connrefused
--tries=0 --timeout=50 ${url}
/MSCallGraph/MSCallGraph_${i}.tar.gz
```

One can change "i" in the code above to download the desired MSCallGraph.

Call graph data is in CSV file format. Each of these CSV files contains millions of transactions between microservices. Each transaction is in a separate row in CSV files. Table 3.1 shows an example record from the CSV files.

Table 3.1: An Example Call from Alibaba Dataset

timestamp	229255
traceid	0b133c3215919238292013000eb3c7
UM	6545e5559493a18497075bb949b955abf8642a62d20287
DM	9c2e1e0e2e9a71881cbbf95fab38cc978feee0ba1e7fed
rt	48
rpcid	0.1.2.32.8
rpctype	rpc

The explanations of the fields in Table 3.1 are listed below.

- timestamp: Timestamp of call records. Its unit is milliseconds. The start time of the records is 0.
- traceid: traceid is the id of a call graph. It is unique for call graphs
- UM: The name of upstream microservice. The upstream microservice is the service that creates a call by sending a request to the downstream microservice.
- DM: The name of downstream microservice. The downstream microservice is the service that receives a call from the upstream microservice.
- rpcid: Traces have multiple calls. Every call in a trace has a unique rpcid which includes information about the (UM, DM) pair. Note that, remote invocation calls have two duplicate records with the same rpcid in both upstream and downstream microservices.
- rpctype: rpctype is the method of communication. It can take "RPC" value for remote procedure calls. If the downstream microservice is database in inter-

process communication, rpctype is written as "DB". If the downstream microservice is cache in inter-process communication, rpctype is written as "MC"

- interface: The interface between UM and DM
- rt: rt is the response time of the call. Its unit is milliseconds. If the response time is less than 1 ms, its value is written as 0. The rt value can take both positive and negative values because the rt value can be recorded in both UM and DM.

Each traceid in Table 3.1 contains multiple calls. The same UM and DM pairs are observed in more than one call record within a trace. Microservice call graphs are constructed using these traceids, with each trace having its own call graph. Response time variation (RTV) is calculated within a trace when the same call is observed multiple times. To avoid creating a sample consisting of traces observed in the same time intervals, the upper and lower bounds of timestamp filters are adjusted, and calls from different time intervals are sampled. The sample size is enlarged slightly by checking the compatibility of values, such as the number of microservices, call depth, indegree, and outdegree distribution with those on the paper[15]. It is stopped when similar distributions and values are captured. Furthermore, the similarity of the response time variation distributions of the three samples is tested using the Kolmogorov-Smirnov test. According to the results of the Kolmogorov-Smirnov test, the statistical value D representing the largest difference between the distributions of the groups was 0.0001. The p value, which indicates the probability of this difference occurring randomly, was calculated as 0.45. This obtained p value suggests that there is no statistically significant difference between the groups. This allows for reaching enough examples for the representation of RTV and the creation of a sample that represents the entire dataset in terms of response time variation. Sampling and filtering are carefully done to ensure that these processes applied to the dataset do not alter the statistics provided in the paper where the dataset is shared. Firstly, the average depth of the dataset stands at 4.27, with a standard deviation of 3.25. Notably, over 10% of stateless microservices have an out-degree of at least 5, while the majority of microservices maintain an in-degree of one. The study also identifies that most of the graphs have a depth of three in Alibaba traces. Furthermore, the analysis highlights that more than 5% of

microservices showcase in-degrees of 16 in aggregate calls. These hot spot microservices are pervasive, appearing in nearly 90% of call graphs and handling a substantial 95% of total invocations in Alibaba traces. Additionally, the dataset includes approximately 10% of call graphs comprising more than 40 microservices, emphasizing the complexity and scale of certain network structures. Moreover, the research notes that over 4% of call graphs have call depths exceeding ten, underscoring the existence of deep and intricate call structures within the dataset. These findings collectively contribute to a comprehensive understanding of the characteristics and patterns present in the analyzed network dataset.

There are some missing and problematic examples in the dataset. All fields must be in the correct form to obtain topologies properly. RPC and interface fields are ignored while dropping missing value items because only traceid, UM, DM, and rt fields are used in this study. After dropping the null values of these four fields, the UM and DM values which are '(?)' are dropped.

Dropping calls changes the graph topologies dramatically. The problematic traces and calls are removed from the dataset. Graphs with missing edges are excluded from the dataset by removing these traces.

If the direction of the call is from downstream to upstream, the sign of the response time is negative. The negative RT values are converted to positive by taking their absolute values. Unless the negation problem is solved, calculating the response time variation by using the Coefficient of Variation(CV) might decrease the model performance.

Call graphs are sampled based on traces. Microservice call graphs that contain missing or problematic values are removed. Some graphs consist of the same microservice set. The duplicate microservices sets may cause overfitting in training the models. The reliability of the model scores might be suspicious if the model is tested with the graphs with the same microservices set. To prevent overfitting problems and increase the reliability of the test scores, the duplicate graphs in terms of the microservices set are removed from the dataset.

Graphs that are large in size are used to ensure that the dataset intended to be used

in this study is rich in topological features. For this reason, graphs with node counts less than 40 are removed from the dataset.

Since the metric to be measured in this study is response time variation, the aim is to have the same call seen at least 3 times between the same UM and DM. These calls and traces are cleaned from the dataset. Also, the graphs that have these calls are removed because the topologies of call graphs should not be broken.

All graphs in the dataset must be reachable because the purpose is to measure the topological features and use them to predict RTV. Although it is rare, some graphs are not reachable in the traces. To examine these graphs, the largest connected component is found and the graph is evaluated based on the largest connected component.

After the feature calculations are completed, the dataset is further reduced by eliminating some graphs. The Gaussian distributions of graphs are computed based on node count, average degree, and loop count. The graphs that are over the 95th percentile are removed from the dataset. The experiments are also done for the dataset that has outliers. The performance scores, which are F1 score, precision, and accuracy, are nearly 10% lower. At the end of the preprocessing and outlier elimination process, there are 70k microservice call graphs in the dataset. In addition to outlier elimination, call graphs that have the same topology are eliminated during the training of the machine learning model to ensure they are not included in both the test and train sets, and unique call graphs with distinct topologies are obtained. The number of call graphs with identical topologies is even less than 1% of the dataset, and it is unlikely to have any impact on model performance; however, they are still excluded to establish a more reliable experimental setup.

3.3 Features and Labels

The primary objective of this thesis is to understand which topological features are significant and to compare their relative importance. Therefore, the importance of calculating the topological features is crucial for this study. In contrast to other studies, the purpose is to predict response time variation instead of response time. It is vital for microservices architectures not only to have low response times, but also to

show minimal changes in response time, having a stable response time.

This section includes details regarding the calculation of topological features and the measurement of response time variation of microservice call graphs.

3.3.1 Features

There are some frequent keywords in the studies that focus on evaluating microservices architecture design. Decentralized architecture and modularity are two significant examples of these frequent keywords[21][3][22][23][20][24][25][26]. From a topological perspective, centralization and modularity metrics closely align with these two architectural design concepts.

There are numerous features that can be obtained from the microservice call graph topology. In this thesis, instead of preparing a universal set of topological features and selecting them based on performance, the features to be used are predetermined, and the experiments in the thesis are built upon these features. Due to the frequent association of microservice architectures with the concepts of decentralization and modularity, this thesis uses different features for centralization and modularity. The metrics for centralization and modularity are calculated using the networkx[28] library, and the metrics offered by this library are selected. While all metrics in this library for centralization and modularity are experimented with, not all of them are suitable for the thesis. This is because some may be too slow, taking over a minute even for a single call graph, while others have optimization algorithms limited to a certain number of iterations. Features suitable for continuous and fast experiments are included in this thesis. Also, node count, degree and loop count features are mentioned in the paper which shares the Alibaba dataset. Therefore, node count, degree and loop count features are added to the feature list.

Centralization and modularity cannot be expressed by a single mathematical equation because various centralization metrics are designed considering different priorities, as well as multiple modularity metrics that use different partitioning algorithms.

This subsection includes details on the centralization and modularity metrics used and how they are calculated. There is information about the calculation of important

features representing topology, such as node count, average degree, and loop count. The calculated features for each microservice call graph are provided as input to the machine learning models that classify response time variation. The subsection also provides insights into the specific aspects to consider during the calculation of these metrics.

Besides the calculation of the topological features, the distributions of the calculated topological features are shared through various plots in this subsection. Someone aiming to achieve similar results should prepare a dataset for machine learning models by taking into account these distributions. These distributions contain important information about the sample used from the Alibaba dataset.

3.3.1.1 Centralization

Centrality provides an evaluation of the significance of a node in terms of its effect on the connectivity or flow of information in the network. Centrality indicates the impact of an individual node. A high centrality value indicates a significant impact on both the information flow within that node and the connectivity of the network. While centrality provides important information for individual nodes, it does not provide information about the network as a whole. Finding how centralized a network is needs calculation of the centrality information of each node. Additionally, calculating the maximum possible centrality variation in that network is also needed.

$$C = \frac{\sum_{i=1}^{n} (C_{\max} - C_i)}{\sum_{i=1}^{n} (C_{\max\_star\_graph} - C_{i\_star\_graph})} \tag{3.1}$$

Centralization is calculated by using Equation 3.1. In Equation 3.1, C_{i} is the centrality of individual nodes, n is the number of nodes, $C_{\max}$ is the maximum centrality value in the graph. Therefore, the centralization can be calculated after calculating the centrality values of nodes and the maximum possible centrality variation. The denominator of Equation 3.1 is the star graph version of the formula in the numerator. This is because, to calculate centralization, the denominator needs to be the maximum sum of variations that can occur with the same number of nodes. The total variation is maximized by normalizing it with the maximum possible variation possibile in a

network of the same size, which is a star network for the four centralization metrics that are used in this thesis. Centralization means how centralized a network is and it gives information about the whole network.

There are multiple techniques for computing centrality and centralization. In this thesis, closeness centrality, betweenness centrality, degree centrality, and harmonic centrality values are used to compute network centralization.

Degree centrality measures the importance of a node in a network based on the number of edges connected to it.

$$C_D(v) = \frac{\deg(v)}{N-1} \tag{3.2}$$

The degree centrality of the nodes in the microservice call graph is calculated by using Equation 3.2. The terms that are used in the equation are listed below.

- $C_D(v)$: Degree centrality of node v.
- $\deg(v)$: Degree of node v (the number of edges connected to v).
- N: Total number of nodes in the network.

Closeness centrality assesses how quickly a node can interact with other nodes in the network.

$$C_C(v) = \frac{1}{\sum \text{shortest paths from node } v} \tag{3.3}$$

The closeness centrality of the nodes in the microservice call graph is calculated by using Equation 3.3. The terms that are used in the equation are listed below.

- $C_C(v)$: Closeness centrality of node v.
- $\sum$ shortest paths from node v: Sum of the reciprocals of the lengths of the shortest paths from node v to all other nodes.

Betweenness centrality identifies nodes that act as bridges along the shortest paths between other nodes.

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{3.4}$$

The betweenness centrality of the nodes in the microservice call graph is calculated by using Equation 3.4. The terms that are used in the equation are listed below.

- $C_B(v)$: Betweenness centrality of node v.
- $\sigma_{st}(v)$: Number of shortest paths between nodes s and t that pass through node v.
- σ_{st}: Total number of shortest paths between nodes s and t.

Harmonic centrality emphasizes nodes that are central to multiple other nodes.

$$C_H(v) = \sum_{u \neq v} \frac{1}{d(v,u)} \tag{3.5}$$

The harmonic centrality of the nodes in the microservice call graph is calculated by using Equation 3.5. The terms that are used in the equation are listed below.

- $C_H(v)$: Harmonic centrality of node v.
- $d(v,u)$: Length of the shortest path between nodes v and u.

After the computation of these four centrality values for every node in the microservice call graphs, star graphs are used to find the maximum possible centrality variation of a graph with the same number of nodes, which is used then for normalization.

Figure 3.2 shows an example star graph with 40 nodes. A star graph is the most centralized network for these four centrality metrics, where a single node is responsible for the connectivity of the graph and has the maximum centrality, while other nodes

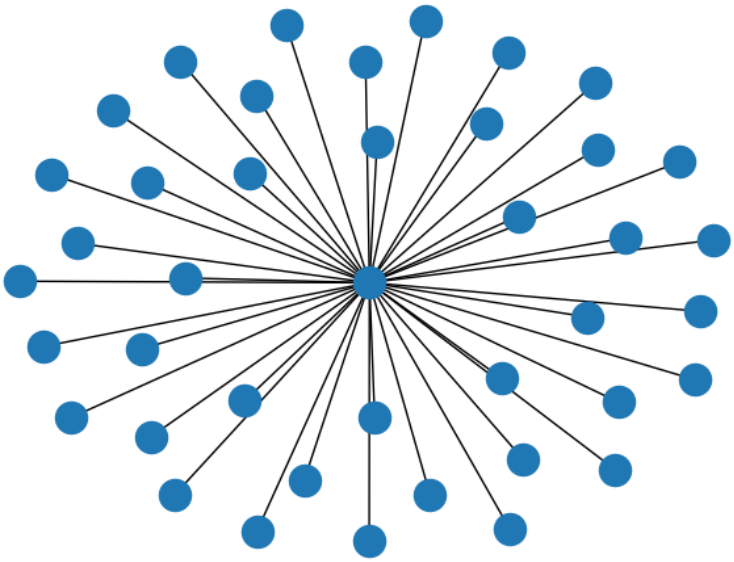

Figure 3.2: A star graph with 40 nodes

have the minimum centrality. Therefore, a star graph helps to calculate the maximum centrality variation of a graph with the same number of nodes.

In microservice call graphs, centralization features are computed for each node by using centrality values and the total difference in centrality between the center node of a star graph with the same number of nodes and other nodes. Betweenness, closeness, degree, and harmonic centralization values form four columns of the input in the tabular data format used during the training of the machine learning model.

The distribution of centralization values obtained from centralization calculations directly impacts the performance of the experiments to be conducted. It is essential to verify whether the distribution is skewed. When examining the impact of feature intervals on response time variation in experiment results, it is crucial to understand the skewness of the distribution, the intervals with the representation ability, and the intervals where values are not represented in the dataset. Therefore, distributions for the four calculated centralization metrics are examined using histogram plots.

In the closeness centralization histogram in Figure 3.3, there is a slightly positive skewness. Most of the centralization values are below 0.5, the call graphs can be

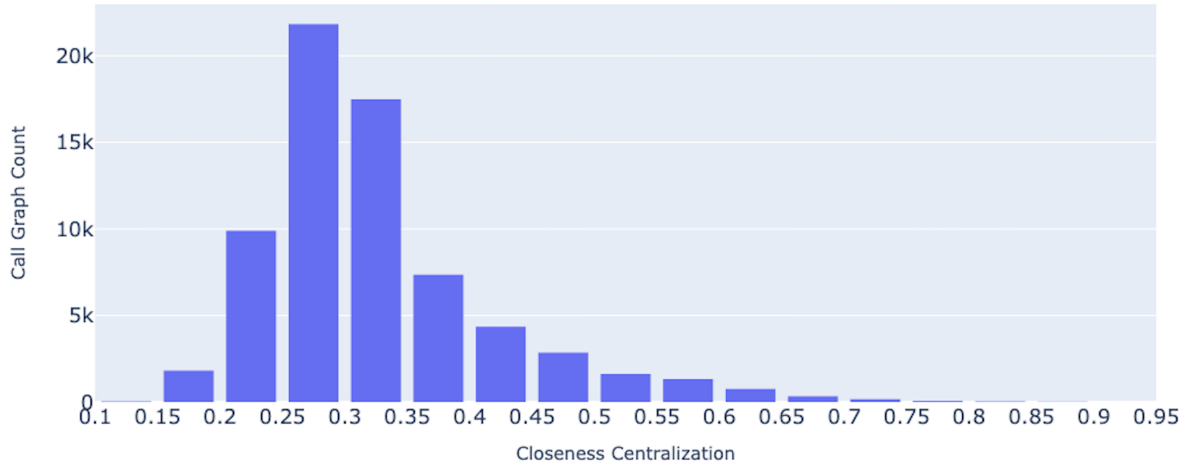

Figure 3.3: Call graph count histogram with closeness centralization bins

considered decentralized in terms of closeness centrality. The node counts are greater than 40 and the call graph topologies are very similar to trees. Tree network topology and high node count create a positive skewness in the closeness centralization histogram.

There are no highly centralized call graphs in the dataset, which can be a disadvantage while predicting response time variation of highly centralized graphs in terms of closeness centrality because they are not represented well in the dataset.

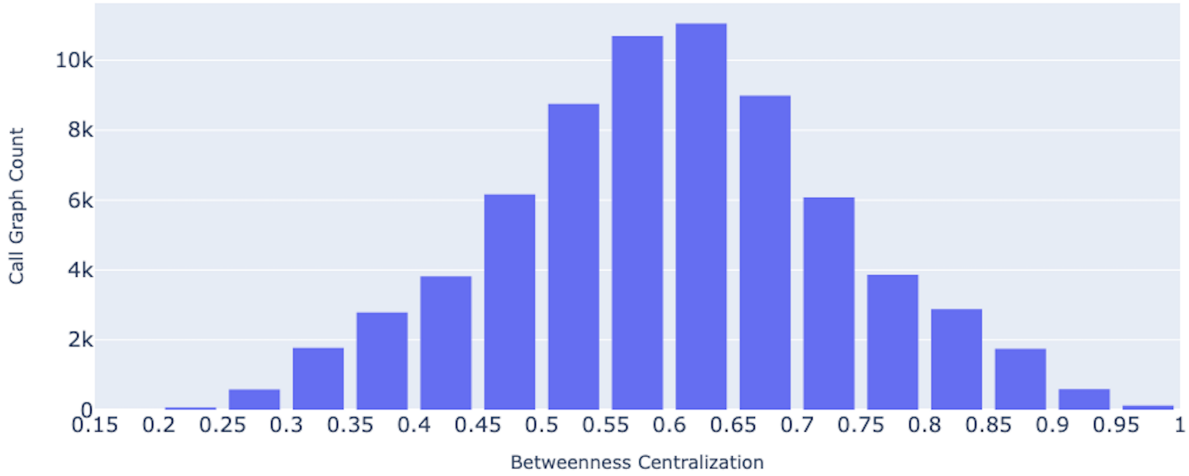

Figure 3.4: Call graph count histogram with betweenness centralization bins

In the betweenness centralization histogram in Figure 3.4, there is no skewness. The distribution is similar to a normal distribution. All centralization values are properly

represented in the dataset in terms of betweenness centrality except decentralized networks which have centralization between 0 and 0.25 in terms of betweenness centralization. Most of the graphs accumulated between 0.4 and 0.7. Database calls increase the betweenness centrality because these databases are more likely to be between two microservices. It is the reason for having no highly decentralized network in the dataset in terms of betweenness centralization.

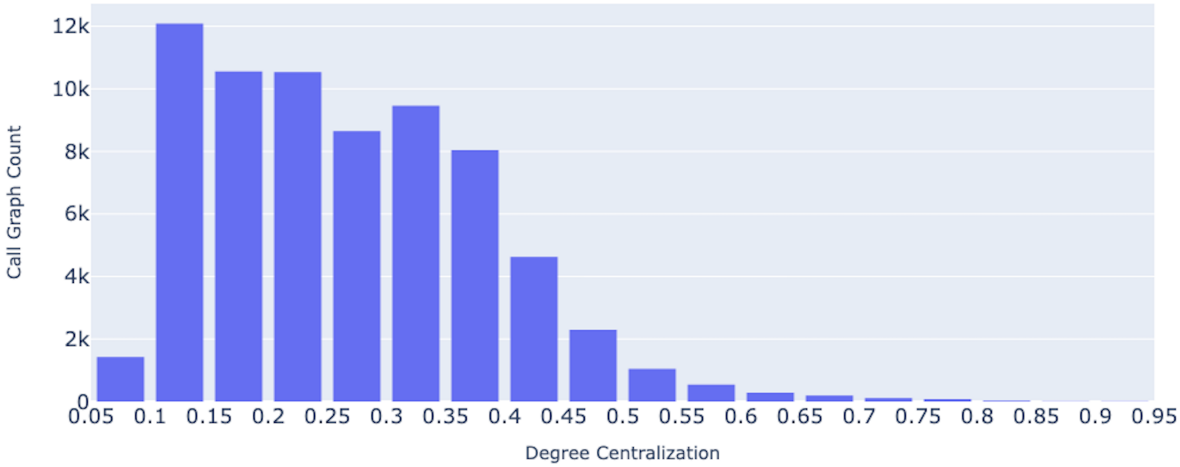

Figure 3.5: Call graph count histogram with degree centralization bins

In Figure 3.5, most of the degree centralizations are lower than 0.6. It is an expected result because there are a minimum 40 nodes in a call graph and most of the nodes have an indegree and an outdegree of one. Also, there are no nodes with a high degree that is close to node count. If the degree centralizations lower than 0.6 are considered, the distribution is similar to the normal distribution. It enables the prediction model to properly differentiate the impact of degree centralizations lower than 0.6 because they are represented equally in the dataset.

Figure 3.6 shows the harmonic centralization histogram. Harmonic centralization is similar to closeness centralization because they are both calculated based on distance. The harmonic centrality takes harmonic distance into account. On the other hand closeness centrality is based on the shortest path distance. The distributions of these two centralization measures based on distance are not the same.

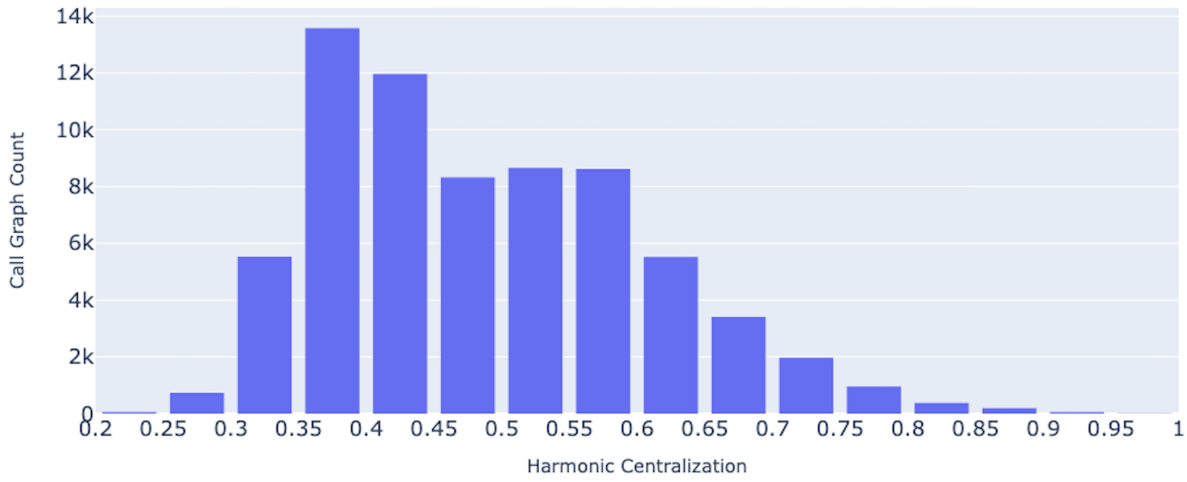

Figure 3.6: Call graph count histogram with harmonic centralization bins

3.3.1.2 Modularity

Modularity is a metric that shows the strength of the division of a graph into subgroups. High modularity means a dense connection between the nodes in the subgroups and a sparse connection between the subgroups. It is a popular measure for microservice call graphs. In literature, the microservices architecture is frequently mentioned with modularity. Although various definitions of modularity exist, modularity is defined from the topological perspective in this study because only topological features of call graphs are considered to evaluate microservices architectures. Therefore, modularity quantifies the proportion of edges within detected modules, subtracting the expected fraction of such edges if they were randomly distributed.

The modularity calculation is highly dependent on communities. There are various methods to assign communities to nodes. Three different community assignment methods are used while calculating modularity in this study. They are the Louvain algorithm [56], label propagation [57], and Clauset-Newman-Moore greedy algorithm [58] [59] which is called the greedy algorithm in this thesis.

The steps of community assignment with the label propagation algorithm are listed below.

- Each node is initially assigned a unique label.

- Nodes update their labels to the majority label among their neighbors. Ties are often broken randomly.
- This process continues until a stable state is reached.

Mathematically, the label update process involves selecting the majority label among neighbors to update a node's label. However, specific deterministic equations are often not used in label propagation methods, as they are often defined probabilistically and may not be easily reduced to a deterministic equation.

The steps of community assignment with the Louvain algorithm are listed below.

- Initially, nodes are assigned to communities.
- Louvain iteratively optimizes modularity by attempting to move nodes between communities. The objective is to increase modularity.

Mathematically, the process of moving nodes between communities to increase modularity is an attempt to make changes denoted by the following equation:

$$\Delta Q = \left[\frac{\Sigma_{in} + 2\Sigma_{tot}}{2m} - \left(\frac{k_{in} + \Sigma_{tot}}{2m}\right)^2\right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{k_{in}}{2m}\right)^2 - \left(\frac{\Sigma_{tot}}{2m}\right)^2\right] \quad (3.6)$$

- ΔQ**:** Modularity change when a node switches communities. If positive, the node changes its community; otherwise, it remains.
- Σ_{in}**:** Sum of weights of internal edges in the node's own community.
- Σ_{tot}**:** Sum of weights of all edges within the node's own community.
- k_{in}**:** Degree of the node within its own community.
- m**:** Total number of edges.

The steps of community assignment with the greedy algorithm are listed below.

- Initially, each node is considered as a separate community.
- Greedy modularity optimization involves iteratively merging communities that lead to the maximum increase in modularity.

Mathematically, the algorithm attempts to maximize modularity by considering changes to the assignment of nodes to communities. The change in modularity (ΔQ) for merging two communities is calculated using the equation:

$$\Delta Q = \frac{2}{2m}\left[\Sigma_{in} - \left(\frac{k_{in}}{2m}\right)^2\right] \tag{3.7}$$

- ΔQ**:** Modularity change when a node switches communities. If positive, the node changes its community; otherwise, it remains.
- Σ_{in}: Sum of weights of internal edges in the two communities being merged.
- k_{in}: Sum of degrees of nodes in the two communities being merged.
- m: Total number of edges.

After assigning the communities to the nodes in the microservice call graphs by three methods, the modularity values of microservices are computed by the following equation.

$$Q = \frac{1}{2m}\sum_{ij}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j) \tag{3.8}$$

In Equation 3.8 , A_{ij} is the element in the adjacency matrix, k_i and k_j are the degrees of nodes i and j respectively, m is the total number of edges, and $\delta(c_i, c_j)$ is the Kronecker delta function.

$$\delta(c_i, c_j) = \begin{cases} 1 & \text{if } c_i = c_j, \\ 0 & \text{if } c_i \neq c_j. \end{cases} \tag{3.9}$$

The Kronecker delta function is shown in Equation 3.9, It ensures that the terms in the summation only contribute when nodes i and j belong to the same community.

The three modularity features form three columns of the input in the tabular data format used during the training of the machine learning model. The distribution of modularity values obtained from modularity calculations directly impacts the performance of the experiments to be conducted. It is essential to verify whether the

distribution is skewed. When examining the impact of feature intervals on response time variation in experiment results, it is crucial to understand the skewness of the distribution, intervals with the representation ability, and intervals where values are not represented in the dataset. Therefore, distributions for the three modularity features are examined using histogram plots.

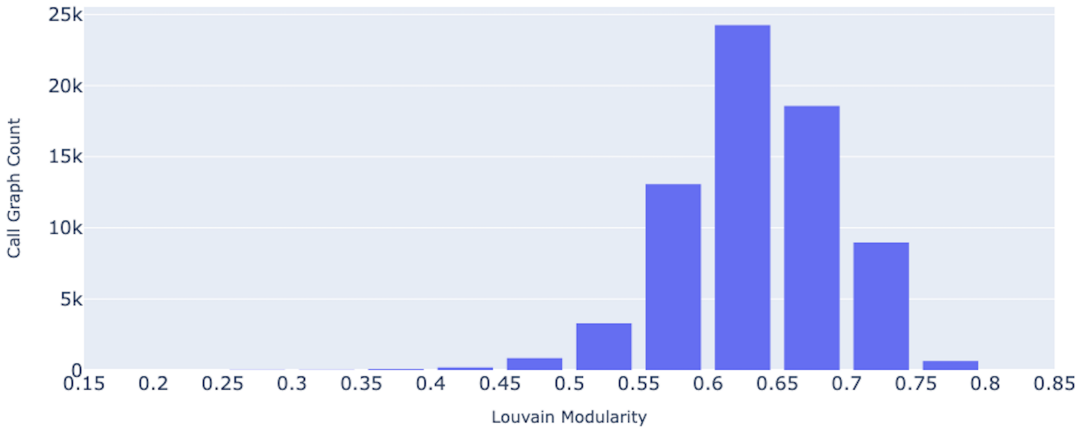

Figure 3.7: Call graph count histogram with Louvain modularity bins

Figure 3.7 shows the Louvain modularity and call graph count histogram. The modularity values are between 0.4 and 0.8. The call graphs in the dataset have no low modularity values which are less than 0.4. Most of the call graphs have modularity between 0.6 and 0.7.

The topology of call graphs is similar to a tree and modularity is calculated by using undirected graphs. Therefore, high modularity values are expected.

Figure 3.8 shows the label propagation modularity histogram. The distribution is very similar to the Louvain algorithm, but there is a small difference between Louvain and label propagation modularities. Louvain values are on average 0.1 higher than the label propagation algorithm. The reason for this difference is that the Louvain algorithm can find more dense subgroups than the label propagation algorithm.

Figure 3.9 shows the Greedy modularity distribution. The distribution is the same as Louvain modularity distribution. The greedy algorithm and Louvain algorithm find the same communities for this call graph dataset. The Pearson correlation coefficient between Louvain modularities and greedy modularities is 1. The distribution and the

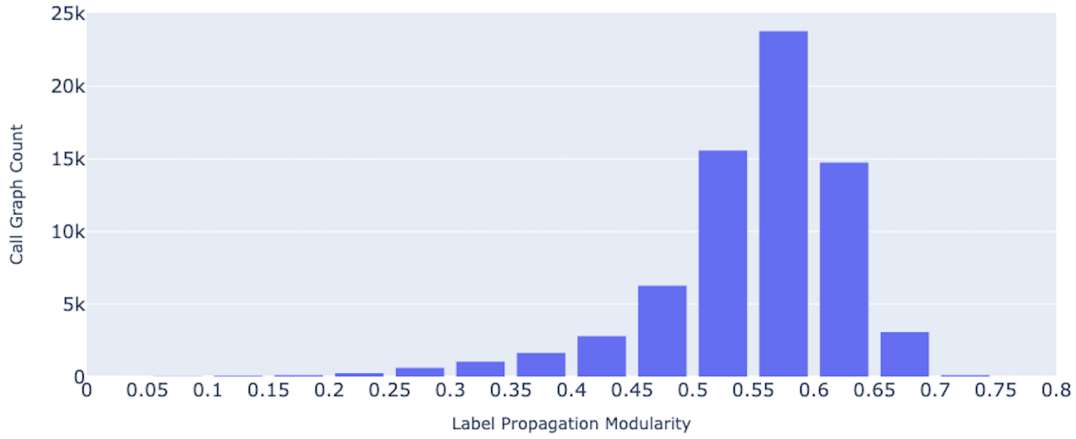

Figure 3.8: Call graph count histogram with label propagation modularity bins

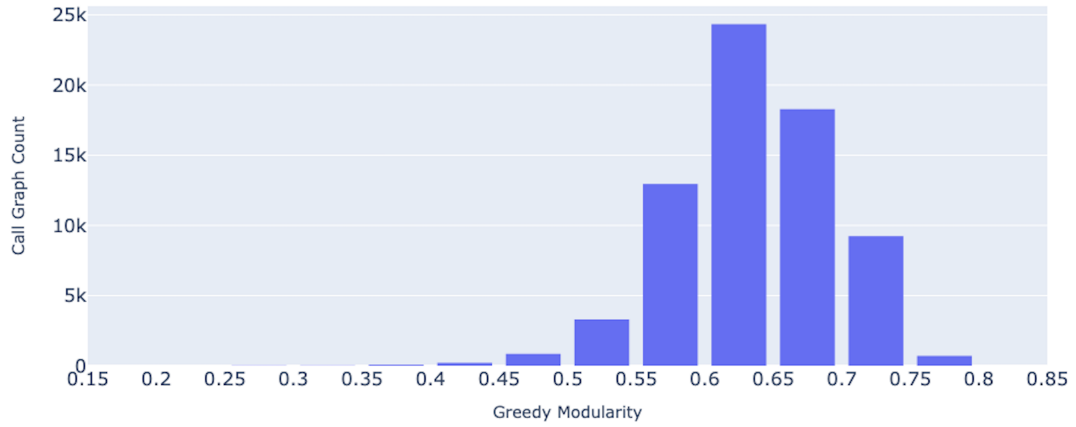

Figure 3.9: Call graph count histogram with greedy modularity bins

correlation coefficient show that these two modularity values are the same. It is not needed to use both of these features while predicting response time variation. The Louvain algorithm and label propagation algorithm are used in model training.

3.3.1.3 Other Topological Features

Although the impact of centralization and modularity on response time variation is one of the critical research questions in this study, achieving high model performance is also one of the primary goals. Therefore, it is necessary to represent the topology with other features. For this reason, node count, average degree, and loop count are added to the feature list.

Centralization and modularity metrics are more meaningful if the node count is in the same features list. The average degree serves a similar purpose. Although nodes generally have two degrees, one in-degree and one out-degree, representing how much the tree branches or deviates from a tree network structure, the average degree plays a significant role.

In the Alibaba dataset, there are very few networks with directed loops. Their papers indicate that out of approximately 20 million graphs, about 200 of them contain cyclic dependencies with a minimum of 3 nodes[15]. Therefore, loop count is calculated based on undirected topology like other features. While not as harmful as cyclic dependencies, situations with loops can lead to problematic circumstances if the maintenance strategy is not well designed.

The distribution of average degree, node count, and loop count values directly impacts the performance of the experiments to be conducted. It is essential to verify whether the distribution is skewed. When examining the impact of feature intervals on response time variation in experiment results, it is crucial to understand the skewness of the distribution, the representation in different intervals, and the intervals where values are not represented in the dataset. Therefore, distributions for the three features are examined using histogram plots.

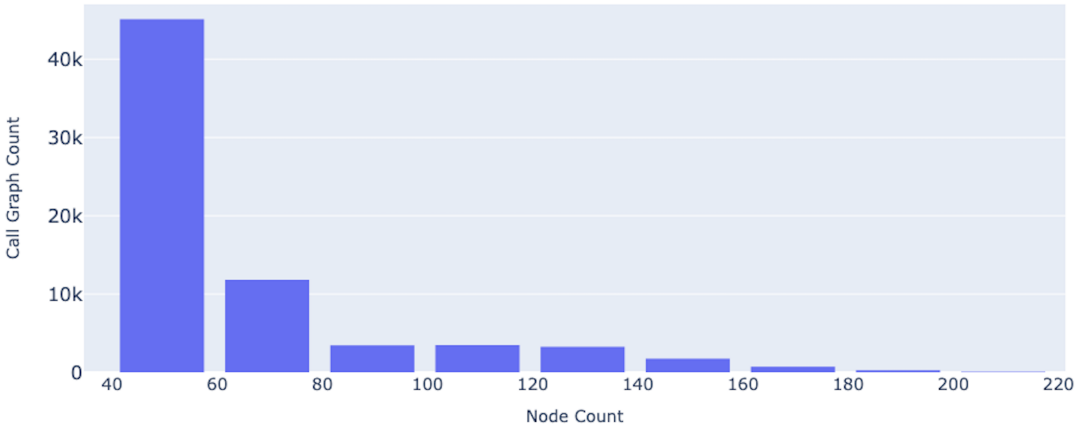

Figure 3.10: Call graph count histogram with node count bins

Node count distribution is shown in Figure 3.10. Most of the topologies have node counts between 40 and 50. The call graph count decreases with the increasing node count. It is valid for the whole Alibaba dataset, finding large graphs is harder than

finding small graphs in the Alibaba dataset.

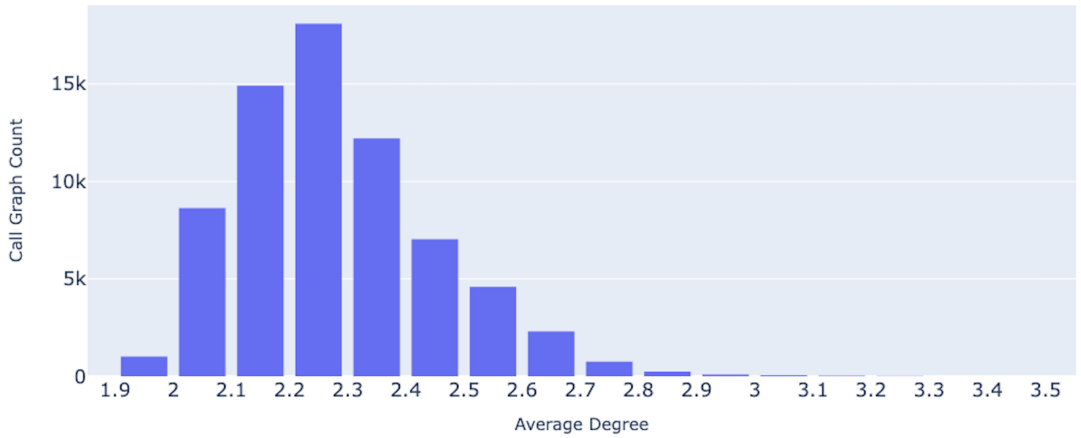

Figure 3.11: Call graph count histogram with average degree bins

Figure 3.11 shows the average degree distribution in the dataset. The expected average degree count is 2.25 for the dataset. It is an expected result because most of the nodes have one in-degree and one out-degree.

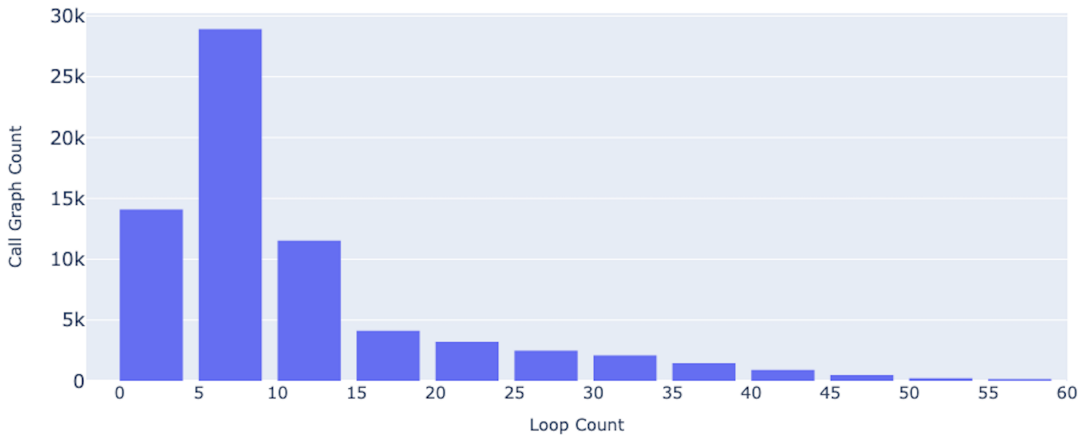

Figure 3.12: Call graph count histogram with loop count bins

Loop count distribution is shown in Figure 3.12. The loop count feature does not represent cyclic dependencies because undirected topology is used in this study. Database calls can create loops or some microservice call the same node. The loops may have a node count that is larger than 3. Although the loop count may appear to have a highly negative impact on response time variation, it is not easy to predict its effect without testing, as loops can occur for various reasons. Nevertheless, undirected loops that

are poorly designed or have not been properly tuned in production have the potential to create issues.

3.3.2 Correlation Analysis Between Topological Features

Since this thesis uses machine learning models to assess feature importance, the high correlation between features may adversely impact the interpretations. Therefore, an examination of the correlation between features is necessary. Correlated variables should either be removed from the feature list, or these correlations should be taken into account when interpreting feature importance. Pearson correlation coefficient is used to measure the correlation between topological features of microservice call graphs.

The Pearson correlation coefficient (r) measures the linear relationship between two variables, typically denoted as X and Y. The equation for calculating the Pearson correlation coefficient is given by:

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{3.10}$$

In equation 3.10:

- X_i and Y_i are the individual data points of variables X and Y respectively.
- $\bar{X}$ and $\bar{Y}$ are the means of variables X and Y respectively.
- n is the number of data points.

The value of r ranges from -1 to 1, where:

- $r = 1$: Perfect positive linear correlation
- $r = -1$: Perfect negative linear correlation
- $r = 0$: No linear correlation

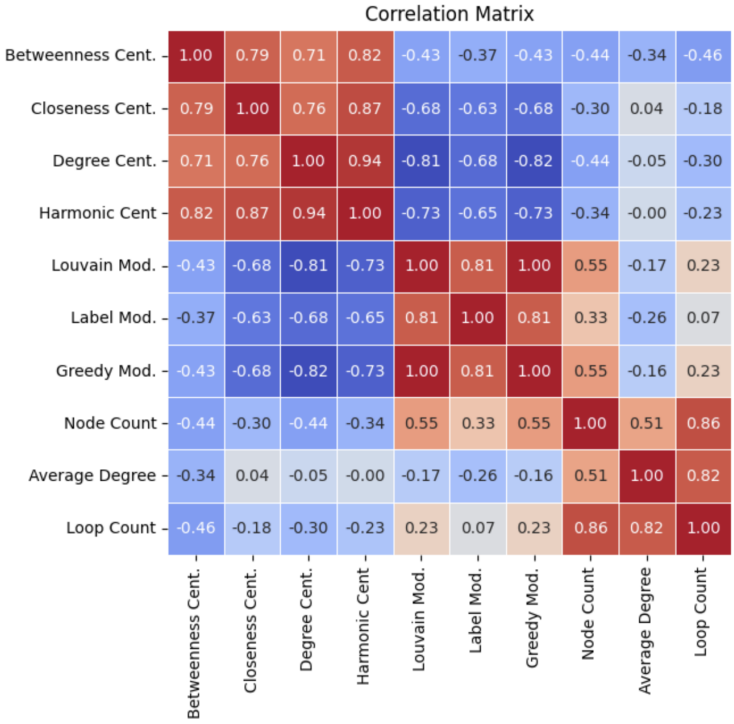

Figure 3.13: Correlation Matrix of Features

Figure 3.13 shows Pearson Correlation coefficients between topological features. Centralization metrics are positively correlated among themselves. Harmonic centralization and degree centralization are highly correlated.

Louvain modularity and greedy modularity have exactly the same values. Only Louvain modularity is used in model training. Greedy modularity is removed from the feature list. Label modularity and Louvain modularity are positively correlated.

Node count is positively correlated with loop count. It is expected because the high number of nodes increases the possibility of creating loops. Average degree and loop count are correlated for the same reason, but loop count is not removed from the feature list because it has the potential to explain situations with complex dependencies if the impact of loop count is high during the feature importance calculations.

There is a meaningful negative correlation between modularity and centralization because centralized networks have fewer communities. Since centralization features and modularity features are expected to be correlated within their own sets, feature elimination is not performed due to the observed correlation between these features.

However, because the greedy algorithm and the Louvain algorithm find the same communities, calculating the modularity feature with the greedy algorithm is unnecessary. While it may not influence the decisions made by the model, examining one of them is sufficient from the perspective of feature importance. Due to its high correlation between loop count - average degree and loop count - node count, loop count can be excluded from the feature list. However, it is not excluded from the feature list due to the potential to capture interesting situations in some microservice graphs, it is important to consider these correlations. If the impact is low, the impact of loop count can be ignored during the analysis.

3.3.3 Labels and Response Time Variation

Having smaller performance variations is a desired property for microservice architectures. Small response time variation means a more stable and easily testable system.

The main purpose of this research is to investigate the impact of topological features on response time variation (RTV) and to predict RTV using these features.

In this thesis, the prediction of response time variation is treated as a classification problem. Firstly, the response time variation of microservices in call graphs is calculated using the coefficient of variation.

$$CV = \frac{\text{Standard Deviation}}{\text{Mean}} \tag{3.11}$$

After calculating the response time variation for each call using the CV, the average response time variation of call graphs is computed. The average of the response time variations of calls is considered as the response time variation of the call graph.

Microservice call graphs are labeled by using different response time variation threshold (RTVT) values. Call graphs with RTV values above the RTVT are labeled as fluctuating, and those below are labeled as steady.

- When the RTVT value is set to 0.3, 51% of the 70k call graphs are labeled as

fluctuating, and 49% as steady.

- Setting the RTVT value to 0.4 resulted in 32% of the 70k call graphs labeled as fluctuating and 68% as steady.
- Setting the RTVT value to 0.5, 15% of the 70k call graphs are labeled as fluctuating, and 85% as steady.

The selection of the 0.3, 0.4, and 0.5 thresholds is primarily aimed at adjusting the sample sizes in the steady and fluctuating classes. When approximately 50%, 30%, and 15% of the data are labeled as fluctuating, the consistency and differences in the feature importance of models are analyzed. However, these thresholds also have real-world implications, affecting user experience. There is a study that considers a service having a coefficient of variation above 1.0 as an anomaly [60]. When a network with an RTV above 0.5 is considered to have a normal distribution in terms of response time variation based on service level, being fluctuating with a 0.5 threshold indicates a high probability of the service having anomalies, indicating the risk of improper functioning and a high rate of failures. In another study, it is mentioned that in a scenario where resources in the cloud are shared, 0.5 response time variation led to 1.7% of requests experiencing timeout conditions [61]. This underscores that the fluctuating class with a 0.5 threshold indicates a high likelihood of an average service experiencing timeouts. Considering service level agreements, being labeled as fluctuating with a 0.5 threshold is risky. In a network labeled as fluctuating with a 0.4 threshold, it is likely that there are services unable to meet service level agreements, potentially causing issues for some users in using the application properly. Being steady with a 0.3 threshold has not been considered a risky situation when looking at the literature. Networks labeled as fluctuating with this threshold may pose a problem for applications requiring stable response times and struggling to meet service level agreements. For example, in the case of a video and music streaming application providing a user experience at the edge in terms of response time, being steady with a 0.3 threshold is desirable.

When evaluating the Pearson Correlation Coefficient between RTV and other features, no correlation value above 0.3 is found. Node count and closeness centralization have the highest correlation values at -0.28.

Analyzing the correlation between the RTVT values and binary labels created with the model features provides valuable information for the model development stage. Therefore, the correlation between binary labels and continuous features is examined using the Point Biserial Correlation Coefficient.

The point-biserial correlation coefficient (r_{pb}) measures the linear relationship between a binary variable (coded as 0 or 1) and a continuous variable. It is calculated using the following equation:

$$r_{pb} = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{3.12}$$

In Equation 3.12:

- X_i is the continuous variable, and Y_i is the binary variable (coded as 0 or 1).
- $\bar{X}$ and $\bar{Y}$ are the means of variables X and Y, respectively.
- n is the number of data points.

The value of r_{pb} also ranges from -1 to 1, where:

- $r_{pb} > 0$: Positive correlation (as X increases, Y tends to be 1)
- $r_{pb} < 0$: Negative correlation (as X increases, Y tends to be 0)
- $r_{pb} = 0$: No linear correlation

- For the RTVT value of 0.3, except for node count and closeness centralization, there is no feature with a correlation coefficient above 0.2. Node count shows a slightly significant correlation with -0.34.
- For the RTVT value of 0.4, except for node count and average degree, there is no feature with a correlation coefficient above 0.2. Node count shows a slightly significant correlation with -0.31.
- For the RTVT value of 0.5, except for node count, there is no feature with a correlation coefficient above 0.2. Node count shows a slightly significant correlation with -0.21.

3.4 Machine Learning Models

The analyses conducted in this chapter reveal that the classification of RTV based on simple correlations is not possible. Therefore, using machine learning models for prediction provides a more meaningful classification model.

This section provides details on the various models used, model validation, hyperparameter optimization, and fine-tuning processes.

For the RTV classification problem, a baseline classifier generating random predictions is used. This baseline model is aware of the RTV label distribution and produces random predictions accordingly. The sklearn [62] Python library's Dummy Classifier is used for this purpose, with the strategy set to 'stratified' to ensure predictions are made with awareness of the distribution. The purpose of experiments is to produce statistically significant results by comparing the performance of this baseline model.

Various machine learning models are used to generate better predictions than the baseline classifier and statistically significant results. These models consist of a random forest classifier [33], a CatBoost Classifier [34], and a LightGBM [35] Classifier. The reason for using tree-based models is their success in determining feature importance which is one of the outputs of this thesis.

After creating the machine learning model, topological features along with the RTV values are given as features and labels to the model respectively. The model aims to achieve a high F1 score. To understand if there is an issue in the model-building procedure, the K-fold cross-validation technique is applied. Here, the fold number is set to 5. The 5-fold cross-validation reveals whether biases in the dataset will affect the model's performance, ensuring a more reliable training process. Once the model is validated, one can be ensured about the robustness of the model.

3.4.1 Training, Hyperparameter Optimization and Fine Tuning

Adjusting hyperparameters in ML models significantly changes the results. Therefore, to achieve the most successful results, it is important to find the most suitable parameters for both the dataset and the model. Once the model's parameters are de-

termined, the most successful model can be created. There are different methods and algorithms for hyperparameters. In this study, Grid Search [62] is used as the hyperparameter optimization method.

For the Grid Search method, the values that hyperparameters can take must be predefined. The algorithm tries all combinations, finding the combination that produces the best results. During the hyperparameter optimization and fine-tuning process, the hyperparameters and their corresponding values for the Random Forest Classifier, CatBoost, and LightGBM Classifier are determined.

For the LightGBM classifier, the hyperparameter combinations below are used.

- learning rate: [0.05, 0.1, 0.15],
- n estimators: [75,100,150],
- num leaves: [25,30,35],
- colsample bytree : [0.9, 1.0],
- subsample : [0.7, 0.8, 1.0],
- reg alpha : [0,0.5,1.0],
- reg lambda : [0,0.5,1.0],

For the Random forest classifier, the hyperparameter combinations below are used.

- n estimators: [200, 400, 600]
- max features: ['auto', 'sqrt'],
- bootstrap: [True, False],
- max depth : [5 10 20 30],
- min samples leaf: [1, 2, 4]

For the CatBoost classifier, the hyperparameter combinations below are used.

- learning rate: [0.03, 0,05, 0.08, 0.1],

- depth: [4, 6, 10, 15],
- l2 leaf reg: [1, 3, 5, 7, 9]

3.4.2 Performance Scores of ML Models and Statistical Tests

After the training processes of the machine learning models are completed, the analysis phase of the results begins. When reviewing the model's performance, F1 score, accuracy, and precision are considered. Analyzing the confusion matrix helps identify the strengths and weaknesses of the model. To check whether the obtained results are due to chance, the McNemar [36] statistical test is applied. When conducting the McNemar test, alongside the ML model's results, the results of a dummy baseline model generating stratified random predictions, taking into account the distribution, are used. After the results are examined and deemed statistically significant, the feature importance is analyzed.

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Total Examples}} \tag{3.13}$$

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \tag{3.14}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \tag{3.15}$$

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.16}$$

The equations of accuracy, precision, recall and F1 score metrics are shown in Equations 3.13, 3.14, 3.15, and 3.16 respectively.

The McNemar test is a statistical test used to determine if there are statistically significant differences between paired proportions from two related groups. The McNemar test statistic (χ^2) is calculated using Equation 3.17

$$\chi^2 = \frac{(|x - y| - 1)^2}{x + y} \tag{3.17}$$

where:

- x is the number of instances where the first condition is true and the second condition is false.
- y is the number of instances where the first condition is false and the second condition is true.

The p value for McNemar test can be computed using the chi-squared distribution. Let χ^2_{obs} be the observed test statistic, then the p value is given by:

$$p\text{-value} = P(\chi^2 > \chi^2_{\text{obs}}) = 1 - P(\chi^2 \leq \chi^2_{\text{obs}}) \tag{3.18}$$

After conducting the McNemar test, the p value is determined using Equation 3.18. It is anticipated that the obtained p value in the models under consideration for feature importance would be less than 0.001. Models with p values less than this threshold are then examined for their feature importance.

3.4.3 Feature Importance

The impact of topological features in ML models on microservice response time variation is calculated using SHAP (SHapley Additive exPlanations) [37]. SHAP values provide a theoretical framework for measuring the impact of individual features on the output of an ML model. The SHAP value for a specific feature quantifies its contribution to the prediction relative to a baseline prediction. The overall model prediction can be expressed as follows:

$$\text{Model Prediction} = \text{Baseline Prediction} + \sum_{i=1}^{N} \text{SHAP Value}_i \tag{3.19}$$

In Equation 3.19:

- N is the number of features.
- SHAP Value$_i$ represents the SHAP value for the i-th feature.

The SHAP values are computed based on Shapley values from game theory. For a specific example, the Shapley value of a feature is the mean marginal contribution of that feature to all possible feature subsets. The equation for computing the Shapley value (ϕ_i) is given by Equation 3.20.

$$\phi_i(f) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} [f(S \cup \{i\}) - f(S)] \tag{3.20}$$

In Equation 3.20:

- f is the model's prediction function.
- N is the set of all features.
- S is a subset of features.

The SHAP values are then calculated by averaging the Shapley values over all possible orderings of the features. In practice, higher absolute SHAP values indicate a greater impact of a feature on the model's predictions. By analyzing the SHAP values, one can gain insights into which features are driving specific predictions and understand the importance of each feature in the overall model behavior.

CHAPTER 4

EXPERIMENTS

4.1 Introduction

In this chapter, there are experiments conducted to answer the research questions of the thesis. The research questions to be answered are listed below.

- Can the response time variation of call graphs be predicted using the topology of the microservice call graph? How successful can machine learning algorithms be in this prediction task?
- In microservice architectures, which are mentioned frequently with decentralization and modularity, are the centralization and modularity features obtained from the topology of call graphs effective on response time variation? If so, how are they effective?
- What is the impact of features such as node count, average degree, and loop count obtained from microservice call graphs on response time variation?

Experiments where machine learning models predict microservice call graph response time variation using topological features of microservice call graphs demonstrate both the predictability of response time variation with topological features and the level of success achieved. For this purpose, the accuracy and F1 scores of machine learning models, as well as confusion matrices, are shared. These performance scores and confusion matrices include the answer to the first research question.

Feature importance and dependence plots drawn using SHAP reveal the impact of each topological feature on the machine learning model's predictions of response time

variation. Additionally, they provide the ranking of feature effects. Centralization and modularity dependence plots are examined to understand the impact of centralization and modularity features on response time variation, addressing the second research question. The third research question aims to understand the impact of node count, average degree, and loop count on response time variation. To achieve this, feature importance plots and dependence plots containing node count and average degree are used.

This chapter, apart from experiments seeking answers to research questions, explains the limitations of the experiments and situations that threaten the validity. Also, it discusses the results of the experiments.

4.2 Experiment Setup

The classification experiments are conducted with three different label types. These label types are calculated with different RTVT values, which are 0.3, 0.4, and 0.5. To achieve the best accuracy and F1 scores, three different ML models are tried that are random forest classifier, LightGBM classifier, and CatBoost classifier. The objective of striving to achieve the best F1 score and accuracy in these experiments is twofold: to demonstrate the predictive ability of response time variation with topological features and that the models with high scores provide more accurate results in terms of feature importance. The optimal parameters for each labeling type are calculated for these models by using the Grid search algorithm.

In Table 4.1, the hyperparameters resulting from the grid search algorithm for Random Forest Classifier, LightGBM classifier, and CatBoost classifier are presented. Since there are different binary classification problems created with three different thresholds for response time variation, the grid search algorithm yields different results for each threshold.

Table 4.1: Hyperparameters

	RTV Thresholds		
	0.3	**0.4**	**0.5**
Random Forest N Estimator	100	100	75
Random Forest Max Features	sqrt	sqrt	sqrt
Random Forest Criterion	entropy	auto	auto
Random Forest Max Depth	10	5	30
Random Forest Min Samples Leaf	2	2	2
LightGBM Learning Rate	0.05	0.05	0.05
LightGBM N Estimator	100	100	75
LightGBM Num Leaves	30	25	35
LightGBM Subsample	0.8	1	1
LightGBM Reg Alpha	0.5	0.5	0.1
LightGBM Reg Lambda	0.01	0.01	0.01
CatBoost Learning Rate	0.03	0.03	0.03
Catboost Depth	10	10	15
Catboost L2 Leaf Reg	7	5	5

4.3 Results

Table 4.2, Table 4.3, and Table 4.4 show the accuracies and F1 scores of the Dummy Baseline Model, Random Forest Classifier, LightGBM classifier, and CatBoost classifier for RTVT 0.3, RTVT 0.4, and RTVT 0.5 labels.

All performance scores in all tables are statistically significant. The decision on this comment is made using the McNemar test, comparing the results of algorithms with a dummy baseline classifier.

When the RTVT value is configured as 0.3, 51% of the 70,000 call graphs are categorized as fluctuating, while 49% are classified as steady. The reason for choosing the value 0.3 is its ability to evenly split the dataset in half. Consequently, the baseline classifier achieves accuracy and F1 score values close to 0.5. During model training, the sizes of the training and test sets were equally utilized for this label value. Their identical distribution is the main factor contributing to the high performance of the model compared to other labels across all models.

Configuring the RTVT value to 0.4 resulted in 32% of the 70,000 call graphs being

Table 4.2: Performance Scores for RTVT 0.3 Label

ML Model	Accuracy	F1 Score
Dummy Baseline Model	0.499	0.516
Random Forest	0.811	0.816
LightGBM	0.822	0.8
CatBoost	0.826	0.827

Table 4.3: Performance Scores for RTVT 0.4 Label

ML Model	Accuracy	F1 Score
Dummy Baseline Model	0.50	0.38
Random Forest	0.77	0.68
LightGBM	0.79	0.71
CatBoost	0.80	0.72

classified as fluctuating, while 68% were categorized as steady. In this example, the test set has the same size as the 0.3 label, but their distributions are different. However, to address the imbalance in the training set, the number of call graphs labeled as fluctuating is equal to the number labeled as steady. The performance decrease compared to the RTVT 0.3 label is due to the imbalance in the distribution of the training set and test set. However, if this imbalance issue is not addressed, much lower F1 scores are obtained.

Configuring the RTVT value to 0.5 results in 15% of the 70,000 call graphs being labeled as fluctuating, while 85% are categorized as steady. For the RTVT 0.5 label, a training set is used with an equal number of steady and fluctuating graphs, similar to the RTVT 0.4 and 0.3 labels. However, no filtering is applied to the test set, resulting

Table 4.4: Performance Scores for RTVT 0.5 Label

ML Model	Accuracy	F1 Score
Dummy Baseline Model	0.52	0.22
Random Forest	0.72	0.47
LightGBM	0.75	0.49
CatBoost	0.76	0.49

in fewer fluctuating graphs compared to steady graphs, maintaining the same distribution as the original dataset. While resolving the class imbalance issue significantly increases the F1 score, the main cause of performance decline is the distribution difference between the training and test sets.

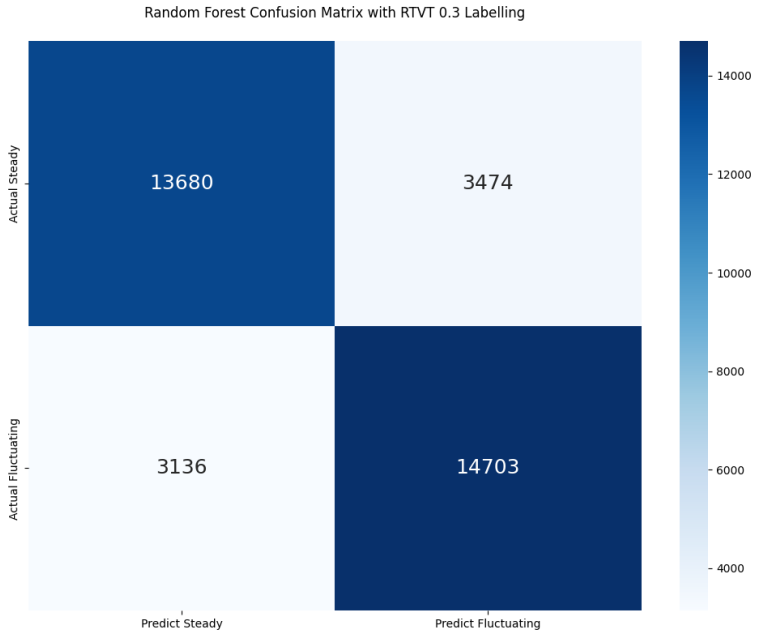

Figure 4.1: Random Forest Model Confusion Matrix with RTVT 0.3 Labelling

The CatBoost classifier has better results than the other algorithms in terms of both accuracy and F1 score for all RTVT labels. The ranking for both performance scores is the same for all three labels, from best to worst: CatBoost, followed by LightGBM,

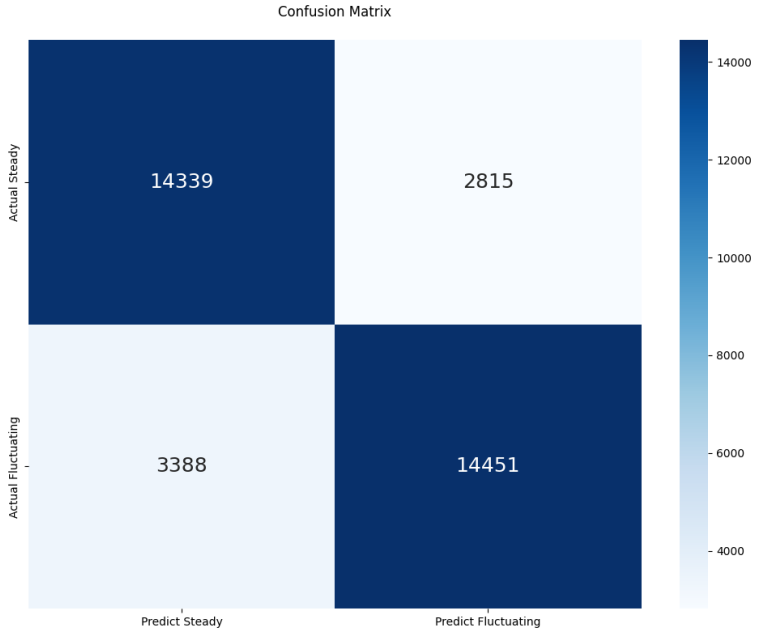

Figure 4.2: LGBM Classifier Confusion Matrix with RTVT 0.3 Labelling

and finally, Random Forest. Therefore, when investigating the importance of topological features, CatBoost is considered the most reliable algorithm. Additionally, the correlation with LightGBM's feature importance is examined. For Random Forest, evaluating feature importance is deemed unnecessary.

The results of the confusion matrices in Figures 4.1, 4.2, and 4.3 indicate that the true positive and true negative rates of the three models are similar. There is no model biased towards either true positive or true negative for this label. If there are specific areas where one model outperformed the others, it would be necessary to consider this when evaluating the feature importance of the models. However, in terms of true positive and true negative results, CatBoost performs better than the other models. CatBoost is slightly better than LightGBM for this RTVT 0.3 label. It manages to transfer a small number of examples where LightGBM makes false negative and false positive predictions to true negative and true positive.

Figure 4.4 shows that LightGBM primarily considers closeness centralization and degree centralization in its predictions. Following these, it gives importance to node count and the modularity value calculated with the label propagation algorithm. In contrast, betweenness centralization and harmonic centralization are evaluated as less

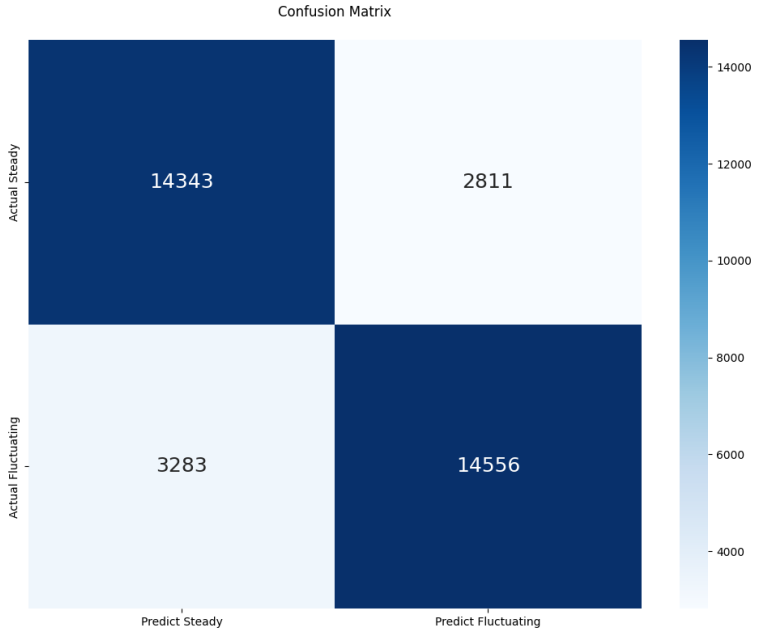

Figure 4.3: Catboost Classifier Confusion Matrix with RTVT 0.3 Labelling

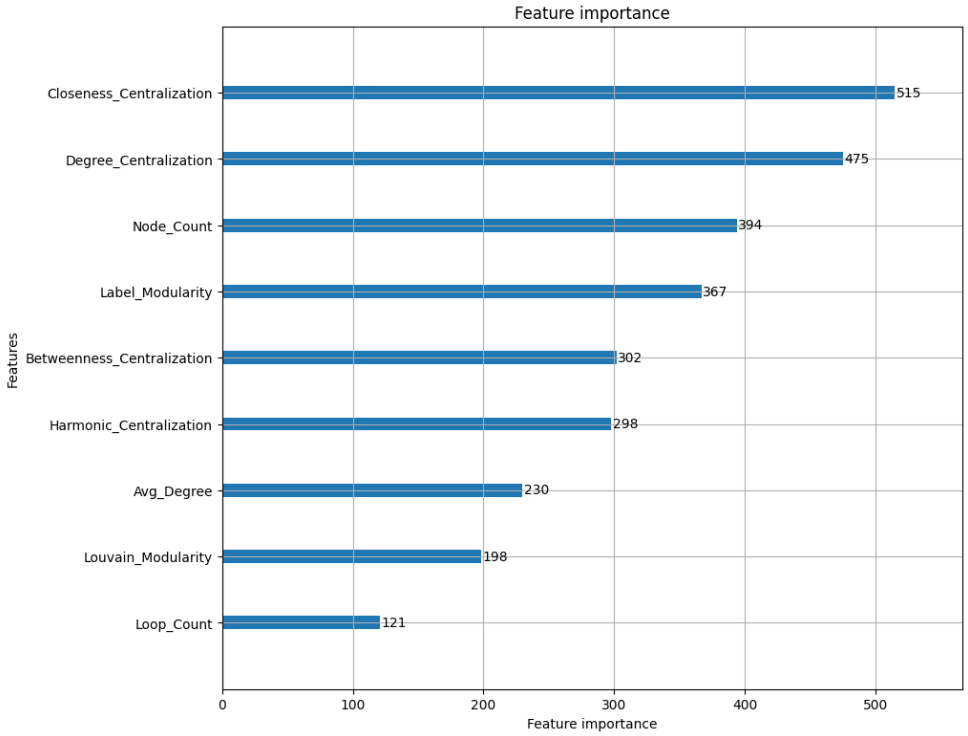

Figure 4.4: LGBM Classifier Feature Importance Plot with RTVT 0.3 Labelling

important compared to other centralization metrics. These two centralization metrics, which have a high correlation, are similarly important to the model. Interestingly,

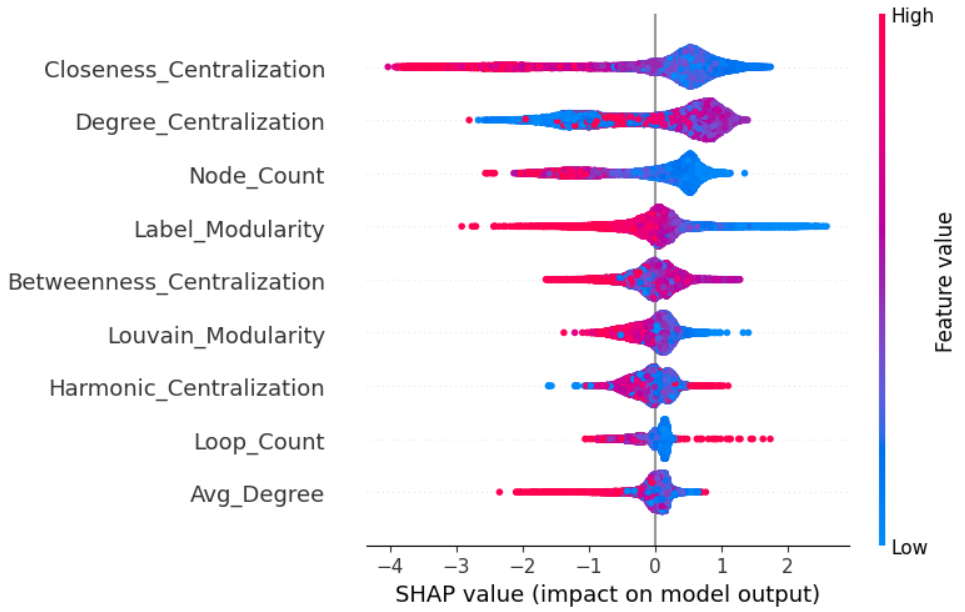

Figure 4.5: Catboost Classifier Feature Importance Plot with RTVT 0.3 Labelling

while attaching significance to label modularity, LightGBM does not use Louvain Modularity with the same degree of impact as label modularity. The impact of loop count, examined in an undirected manner, is negligible in the model's predictions. After Louvain modularity and loop count, the average degree is identified as the least important feature.

The importance of the features for the CatBoost classifier in Figure 4.5 is nearly identical to those of LightGBM. However, CatBoost has evaluated Louvain modularity as a more important feature compared to LightGBM. The slight score difference between these two models arises from the importance of Louvain modularity in the feature importance perspective.

In Figure 4.5, where the feature importances of the CatBoost classifier are evaluated, there is not only a ranking of feature importance, but also an exploration of the relationship between SHAP values and feature values. According to these evaluations, low closeness centralization values contribute to predicting a label of 1. In other words, there is a positive relationship between having low closeness centralization values and being in the fluctuating class of the microservice call graph. High closeness centralization values are more effective in the model's decision-making com-

pared to low closeness centralization values. High closeness centralization values significantly contribute to the model evaluating the microservice call graph as steady. Contrary to expectations, having a decentralized microservices architecture in terms of closeness centralization does not lead to stable response time.

For degree centralization, the situation is the opposite of closeness centralization. From the perspective of degree centralization, graphs that are more centralized have higher response time variation according to the model, while decentralized graphs are evaluated as more steady. Low centralization values are more effective in decision-making compared to high centralization values.

In terms of label modularity and Louvain modularity, the likelihood of the model evaluating more modular microservices architecture as steady is higher. Low modularity values are in a more influential position in the model's decision-making compared to high modularity values. For node count, microservices architectures with low node count are more likely to be evaluated as having higher fluctuation. The impact of high or low values of betweenness and harmonic centralization on the model's decisions cannot be clearly interpreted. These centralization values gain meaning for the model when interpreted in conjunction with other features.

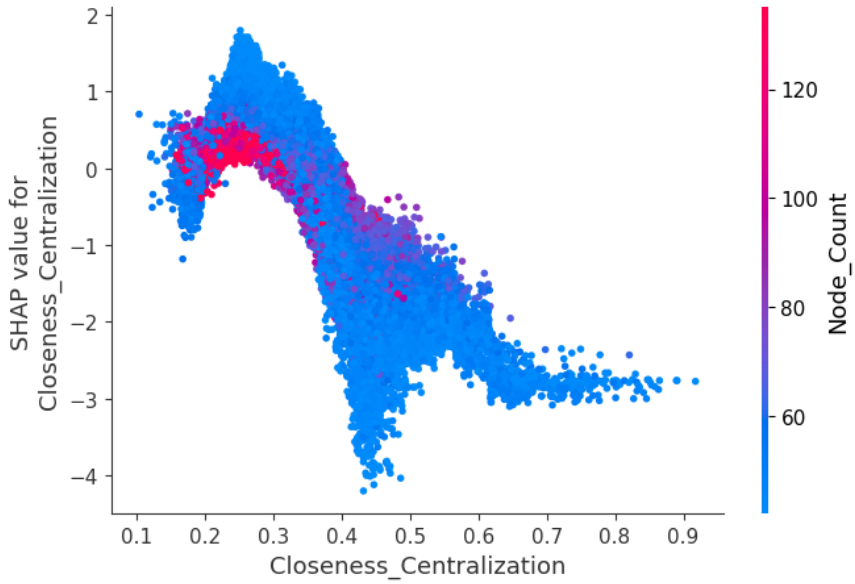

Figure 4.6: Dependence Plot of Closeness Centralization and Response Time Variation Class with RTVT 0.3 Labelling

In Figure 4.6, there is a dependence plot for closeness centralization calculated with SHAP values. According to this dependence plot, microservice call graphs with closeness centralization values below 0.3 are more likely to be evaluated as fluctuating. Microservice call graphs with closeness centralization between 0.4 and 0.5 are evaluated as steady. The dependence plot shows that the model gives the highest importance to the range of closeness centralization values between 0.4 and 0.5. Graphs with centralization above 0.5 are more likely to be evaluated as steady.

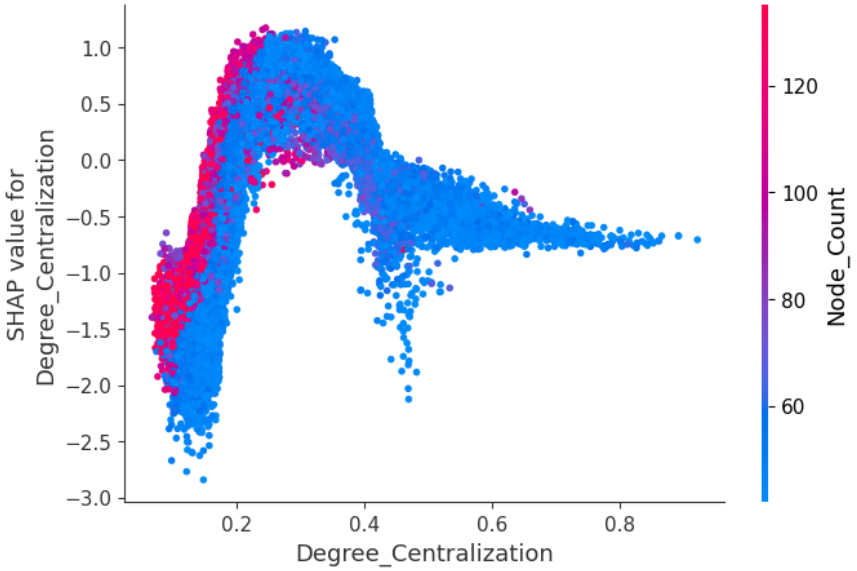

Figure 4.7: Dependence Plot of Degree Centralization and Response Time Variation Class with RTVT 0.3 Labelling

In Figure 4.7, there is a dependence plot for degree centralization calculated with SHAP values. According to this dependence plot, microservice call graphs with degree centralization values below 0.25 are more likely to be evaluated as steady. As the centralization value falls below 0.25, the probability of being evaluated as steady increases. microservice call graphs with degree centralization between 0.25 and 0.45 are evaluated as fluctuating. Call graphs with centralization above 0.45 are more likely to be evaluated as steady, but the impact of this range is not as high as the rest of the values. The degree centralization values that are below 0.25 have the highest absolute SHAP values among all topological features. It means that creating designs with degree centralization below 0.25 is the most effective way to achieve a design with response time variation below the average from a topological perspective.

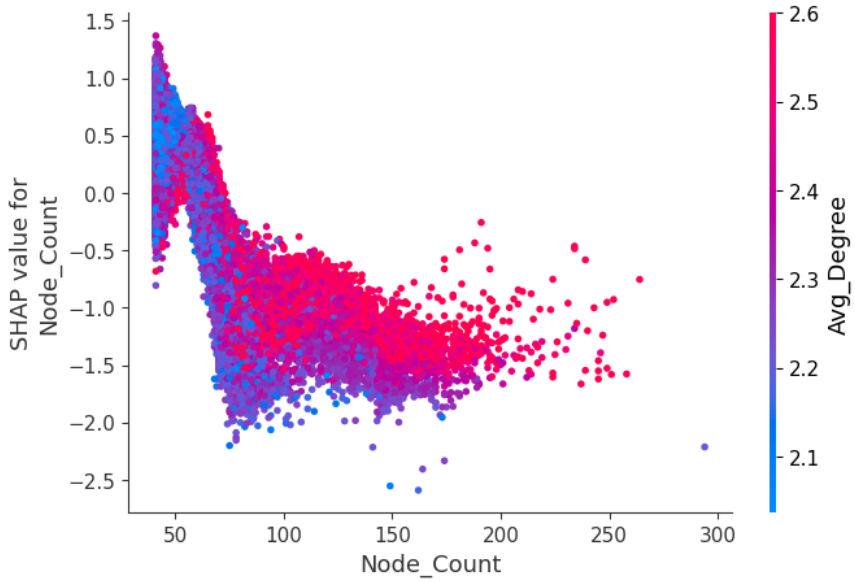

Figure 4.8: Dependence Plot of Node Count and Response Time Variation Class with RTVT 0.3 Labelling

In Figure 4.8, there is a dependence plot for node count calculated with SHAP values. As node count values increase, the model's evaluation of the microservice call graph as steady also increases. The dependence plot indicates that microservice call graphs with a node count greater than 60 are more likely to be evaluated as steady. In graphs with the same node count, it is observed that the effect of the node count feature is variable. This implies that the node count feature is more meaningful when evaluated in conjunction with other topological features.

In Figure 4.9, there is a dependence plot for label modularity calculated with SHAP values. The label modularity dependence plot indicates that as modularity increases, the probability of the model evaluating microservice call graphs as steady also increases. Graphs with modularity values above 0.6 are considered steady by the model. For values below 0.6, there is a high probability of being evaluated as fluctuating. As modularity decreases, the impact of the model becomes more uncertain.

In Figure 4.10, there is a dependence plot for betweenness centralization calculated with SHAP values. The dependence plot for betweenness centralization shows that it does not exhibit a clear positive or negative impact. It gains meaningful interpretation when evaluated in conjunction with other features, as evidenced by a dependence plot.

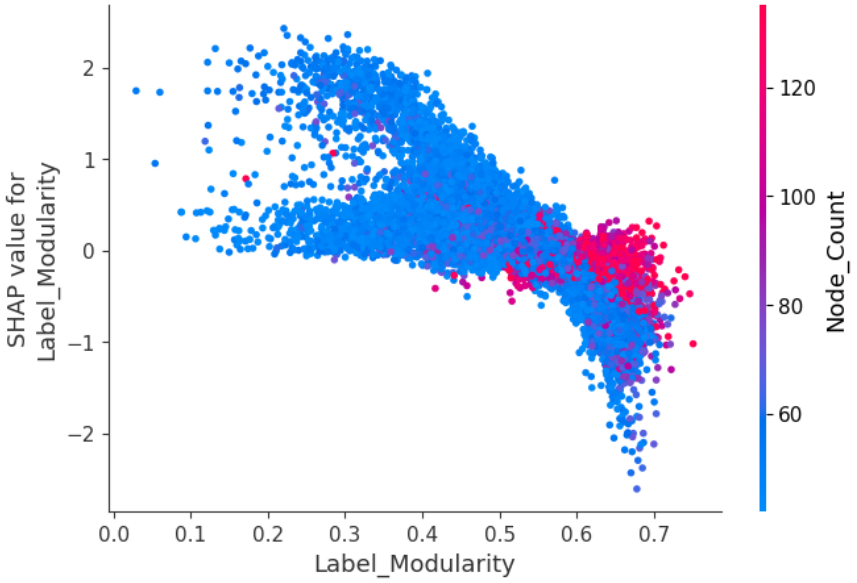

Figure 4.9: Dependence Plot of Label Modularity and Response Time Variation Class with RTVT 0.3 Labelling

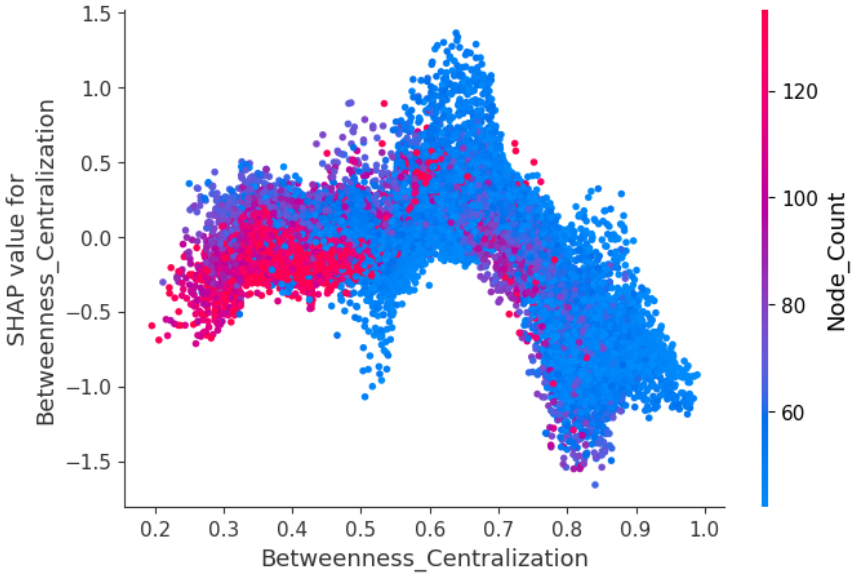

Figure 4.10: Dependence Plot of Betweenness Centralization and Response Time Variation Class with RTVT 0.3 Labelling

Between 0.55 and 0.7, there appears to be a high probability of the model evaluating microservice call graphs as fluctuating.

In Figure 4.11, there is a dependence plot for Louvain modularity calculated with

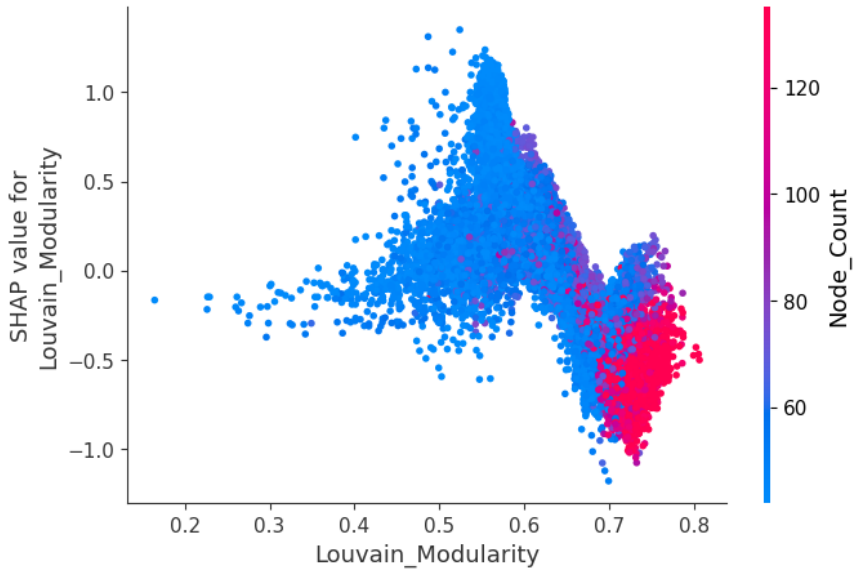

Figure 4.11: Dependence Plot of Louvain Modularity and Response Time Variation Class with RTVT 0.3 Labelling

SHAP values. Louvain modularity has a similar dependence plot to label modularity. However, unlike label modularity, there is a region where Louvain modularity has a peak, particularly for modularity values below 0.6 and above 0.5, where it tends to evaluate microservice call graphs as fluctuating.

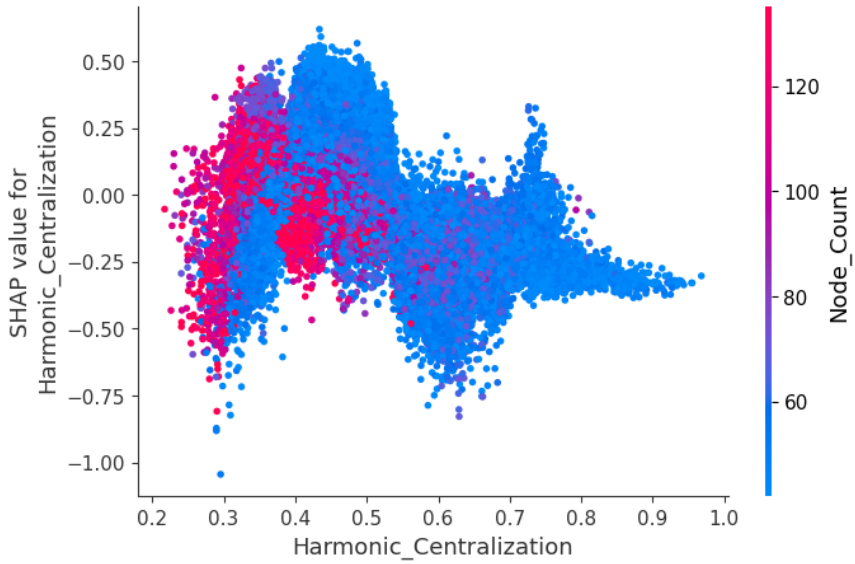

Figure 4.12: Dependence Plot of Harmonic Centralization and Response Time Variation Class with RTVT 0.3 Labelling

Figure 4.12 shows that using the harmonic centralization feature, the likelihood of evaluating as fluctuating or steady is more uncertain compared to other features. In the dependence plot, the highest impacts are observed around values of 0.3, 0.45, and 0.6, but the prediction direction is steady, fluctuating, and steady, respectively. Beyond 0.6, the model is more likely to evaluate microservice call graphs as steady. However, due to both the low impact and the continuous change in the prediction direction of this feature, making a meaningful interpretation becomes challenging.

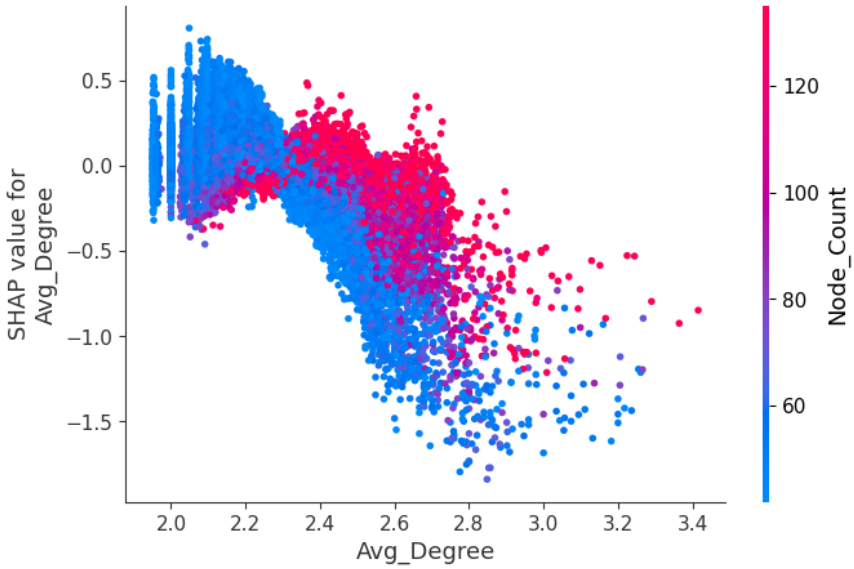

Figure 4.13: Dependence Plot of Average Degree and Response Time Variation Class with RTVT 0.3 Labelling

Figure 4.13 indicates that the effect of the average degree feature on the CatBoost classifier is uncertain below 2.3. Beyond this value, the likelihood of being evaluated as steady appears to be higher. Additionally, microservice graphs with low node count and high average degree show a higher probability of being evaluated as steady.

In Figure 4.14, there is a confusion matrix for the Catboost classifier model trained using RTVT 0.4 labels. Unlike the RTVT 0.3 label, an excess of false positives is noticeable here. The probability of the examples that the model identifies as fluctuating to actually be fluctuating is lower compared to the RTVT 0.3 label. Due to the success of the model in evaluating examples as steady, a relatively high accuracy value is achieved similar to the RTVT 0.3 label. However, when evaluated in terms of F1 score, the same interpretation is not possible because the false positive count is high.

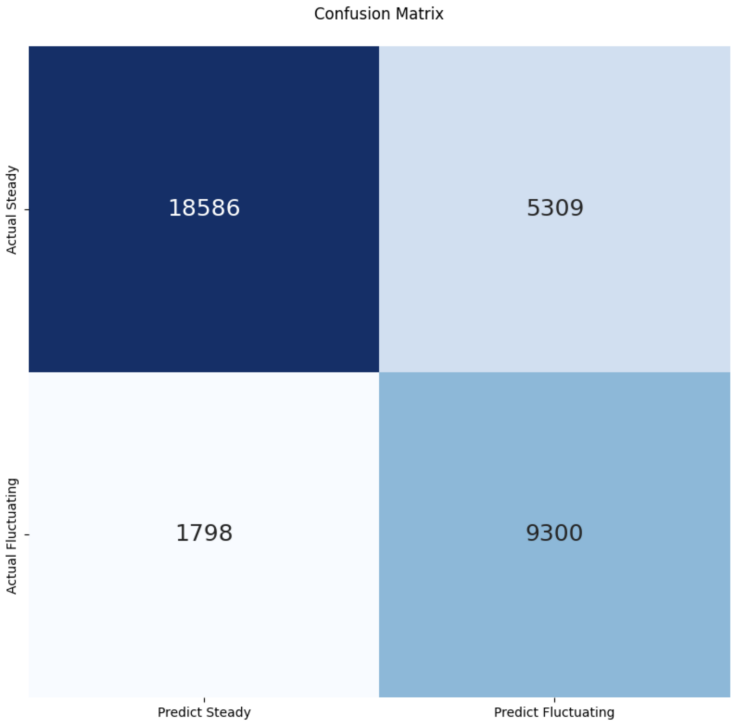

Figure 4.14: Catboost Classifier Confusion Matrix with RTVT 0.4 Labelling

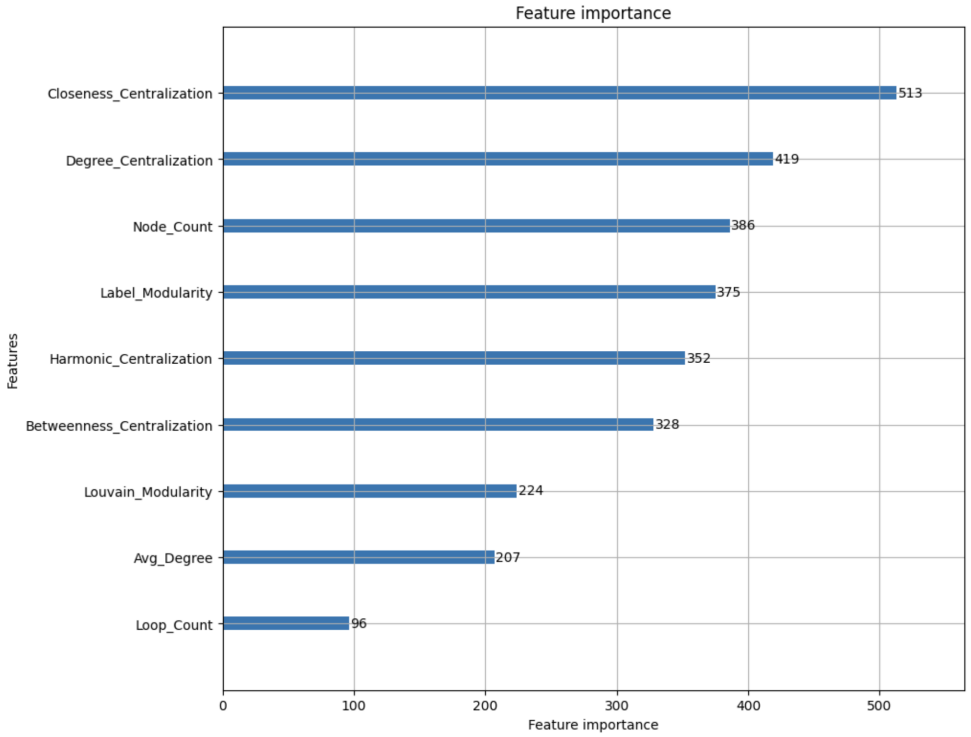

Figure 4.15: LGBM Classifier Feature Importance Plot with RTVT 0.4 Labelling

Figure 4.15 shows that the LightGBM classifier that is trained with RTVT 0.4 labels mainly considers closeness centralization and degree centralization in its predictions. Following these, it gives importance to node count and the modularity value calculated with the label propagation algorithm. On the other hand, betweenness centralization and harmonic centralization are evaluated as less important compared to other centralization metrics. These two centralization metrics, which are correlated, are similarly important to the model. Interestingly, while attaching significance to label modularity, LightGBM does not rely on Louvain Modularity. The impact of loop count, examined in an undirected manner, is negligible in the model's predictions. After loop count, the average degree is identified as the least important feature. The feature importance difference between the LightGBM classifier with RTVT 0.3 labels and RTVT 0.4 labels is negligible, they are almost identical.

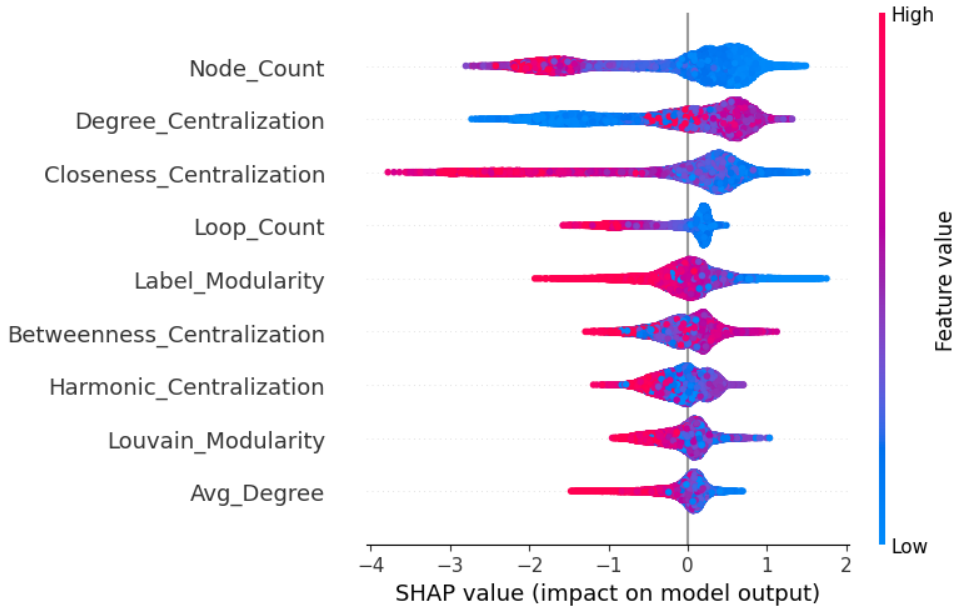

Figure 4.16: Catboost Classifier Feature Importance Plot with RTVT 0.4 Labelling

Figure 4.16 shows that when the CatBoost classifier is trained with RTVT 0.4 labels, it draws a feature importance graph that is different from that of RTVT 0.3 labels. Although the way features affect the outcome is similar, the ranking of importance changes. The model considers node count as the most important feature. Loop count also gains importance and achieves a high rank in terms of feature importance.

Figure 4.17 is a dependence plot for closeness centralization calculated with SHAP

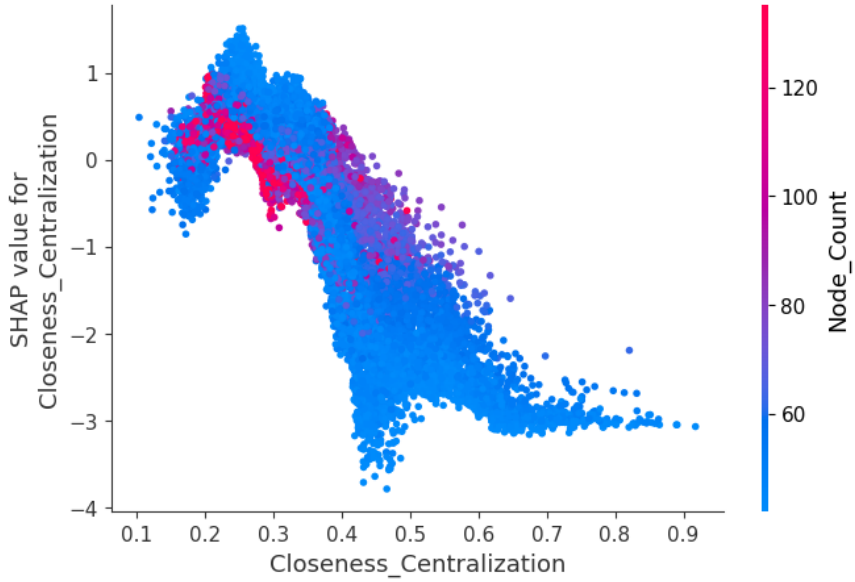

Figure 4.17: Dependence Plot of Closeness Centralization and Response Time Variation Class with RTVT 0.4 Labelling

values. According to this dependence plot, microservice call graphs with closeness centralization values below 0.35 are more likely to be evaluated as fluctuating, but the effect is uncertain in this range. Microservice call graphs with closeness centralization above 0.35 are evaluated as steady. The dependence plot shows that as closeness centralization increases, the probability of microservice call graphs being evaluated as fluctuating also increases.

In Figure 4.18, there is a dependence plot for degree centralization calculated with SHAP values. According to this dependence plot, microservice call graphs with degree centralization values below 0.2 are more likely to be evaluated as steady. As the centralization value falls below 0.2, the probability of being evaluated as steady increases. microservice call graphs with degree centralization between 0.2 and 0.4 are evaluated as fluctuating. Graphs with centralization above 0.4 are more likely to be evaluated as steady, but the impact of this range is lower than the rest of the microservice call graphs. The SHAP values observed at low degree centralization values are remarkably high, unlike in any other topological feature. This indicates the preference for graphs with low degree centralization for performance stability and low response time variation.

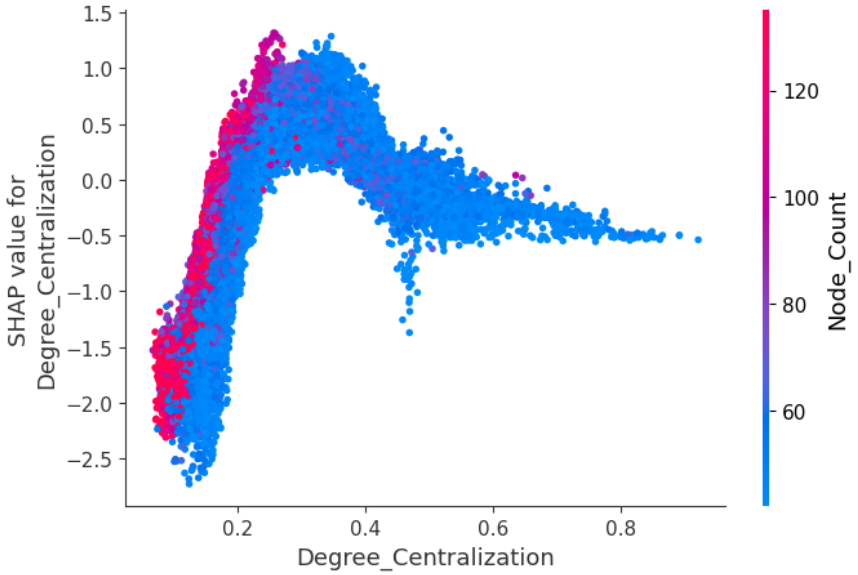

Figure 4.18: Dependence Plot of Degree Centralization and Response Time Variation Class with RTVT 0.4 Labelling

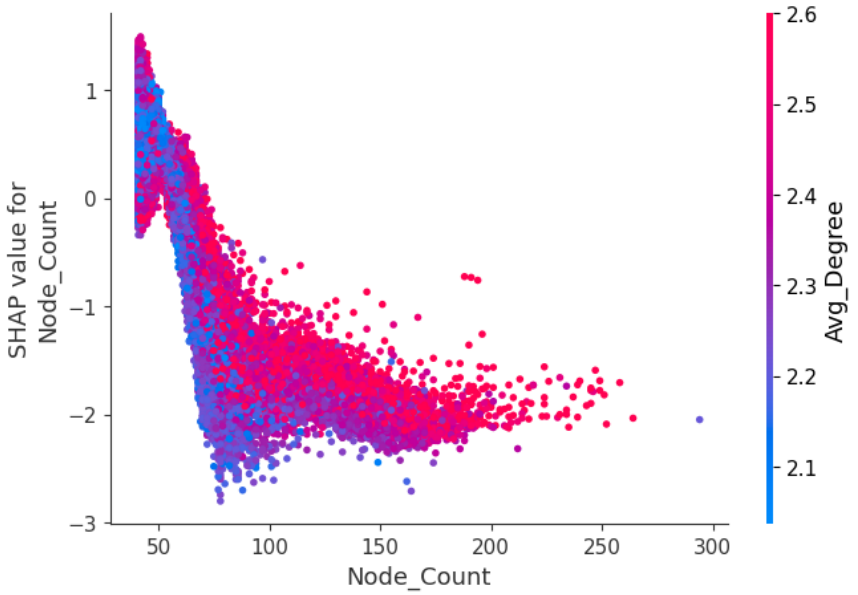

Figure 4.19: Dependence Plot of Node Count and Response Time Variation Class with RTVT 0.4 Labelling

In Figure 4.19, there is a dependence plot for node count calculated with SHAP values. As node count values increase, the model's evaluation of the microservice call graph as steady also increases. The dependence plot indicates that microservice call

graphs with a node count greater than 60 are more likely to be evaluated as steady.

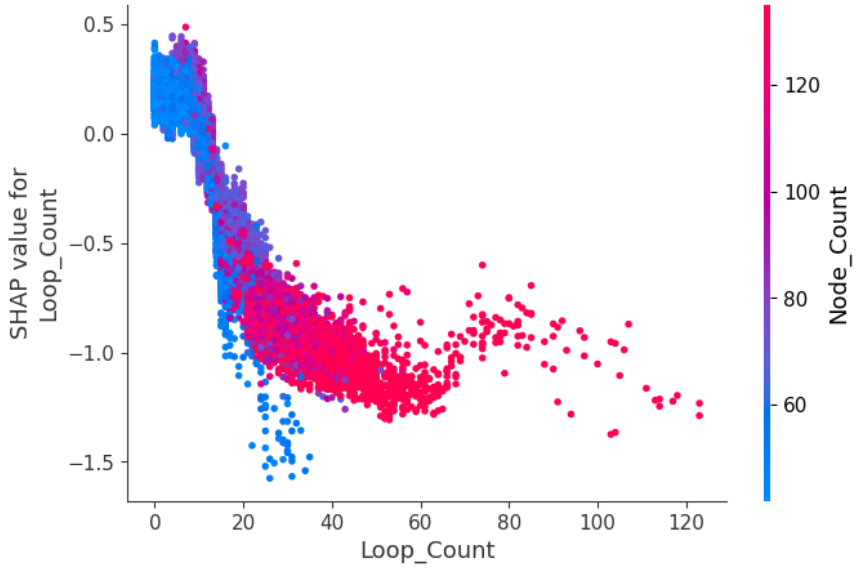

Figure 4.20: Dependence Plot of Loop Count and Response Time Variation Class with RTVT 0.4 Labelling

In Figure 4.20, there is a dependence plot for loop count calculated with SHAP values. As node count values increase, the model's evaluation of the microservice call graph as steady also increases. The dependence plot indicates that microservice call graphs with a node count greater than 16 are more likely to be evaluated as steady. The call graphs with loop count lower than 15 are evaluated as fluctuating by the CatBoost classifier. We cannot say that the higher undirected loop count is better for microservice call graphs, it is not a feasible interpretation. The database and cache calls create loops and it decreases the response time variation.

In Figure 4.21, there is a dependence plot for label modularity calculated with SHAP values. The label modularity dependence plot indicates that as modularity increases, the probability of the model evaluating microservice call graphs as steady also increases. Graphs with modularity values above 0.6 are considered steady by the model. For values below 0.6, there is a high probability of being evaluated as fluctuating. As modularity decreases, the impact of the model becomes more uncertain. Even when modularity is above 0.6, certain examples with a high node count are evaluated as fluctuating, though with a low impact.

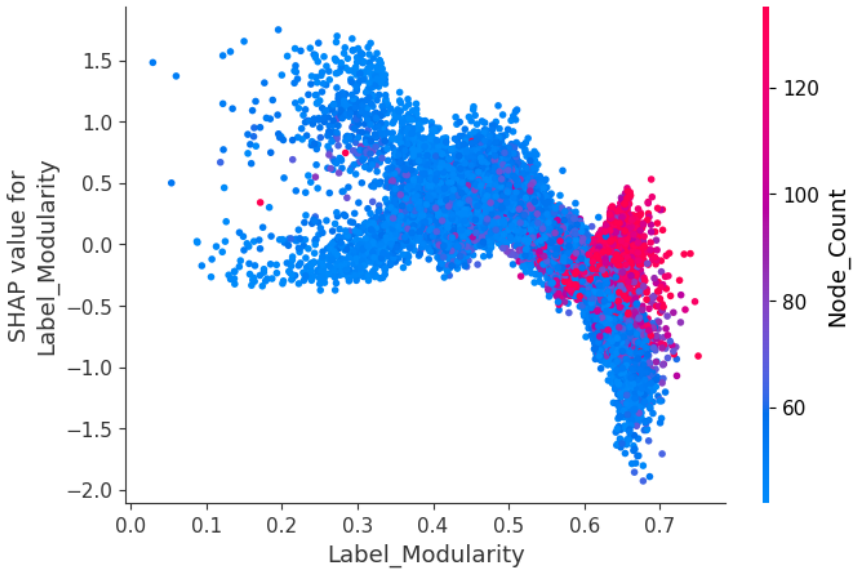

Figure 4.21: Dependence Plot of Label Modularity and Response Time Variation Class with RTVT 0.4 Labelling

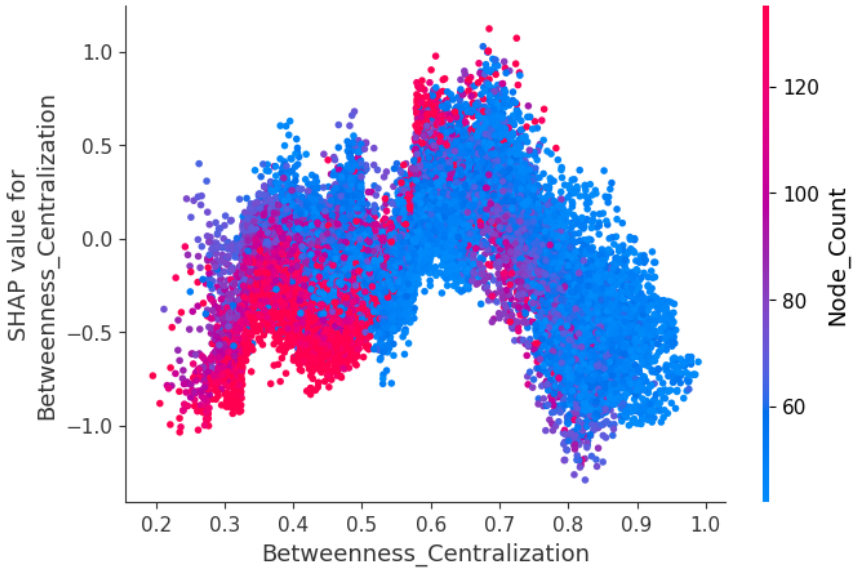

Figure 4.22: Dependence Plot of Betweenness Centralization and Response Time Variation Class with RTVT 0.4 Labelling

In Figure 4.22, there is a dependence plot for betweenness centralization calculated with SHAP values. The dependence plot for betweenness centralization shows that it does not exhibit a clear positive or negative impact. It gains meaningful interpretation when evaluated with other features, as evidenced by a dependence plot. Between 0.6

and 0.75, there appears to be a high probability of the model evaluating microservice call graphs as fluctuating.

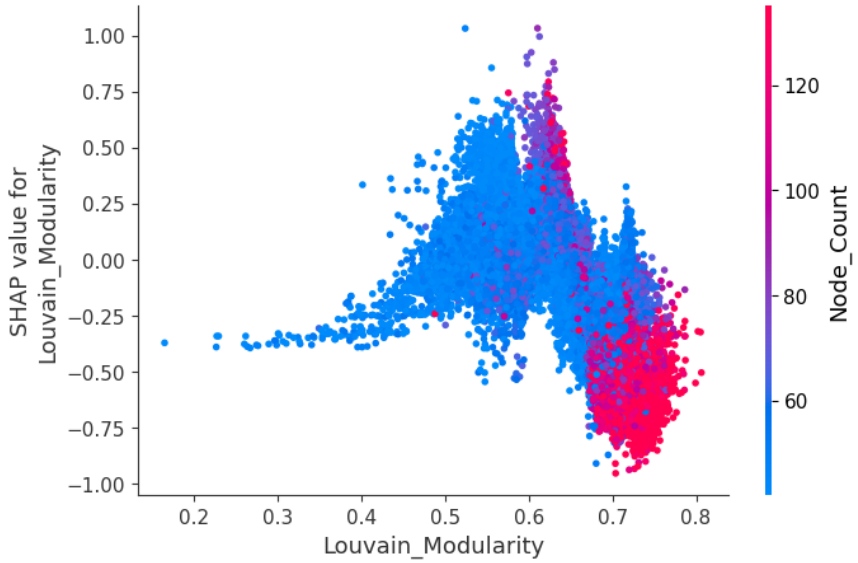

Figure 4.23: Dependence Plot of Louvain Modularity and Response Time Variation Class with RTVT 0.4 Labelling

In Figure 4.23, there is a dependence plot for Louvain modularity calculated with SHAP values. Louvain modularity has a similar trend in the dependence plot to label modularity. However, unlike label modularity, there is a region where Louvain modularity has a high impact, particularly for modularity values below 0.65 and above 0.55, where it tends to evaluate microservice call graphs as fluctuating.

In Figure 4.24, interpreting the impact of the harmonic centralization feature is not entirely feasible, much like it is for RTVT 0.3 labels. Moreover, it is not a feature with a particularly high impact. Unlike other features, no discernible trend is observed between the SHAP value and the feature value. The interpretations made for RTVT 0.3 labels are applicable to this label as well, with only changes in peak values.

The dependence plot for the average degree in Figure 4.25 for RTVT 0.4 labels shows that, before the value of 2.4, the ability of the average degree to evaluate a microservice call graph as fluctuating or steady is weak. In this region, the model makes decisions based on other topological features. From the perspective of average degree, the model evaluates call graphs as steady after the threshold of 2.4. In this re-

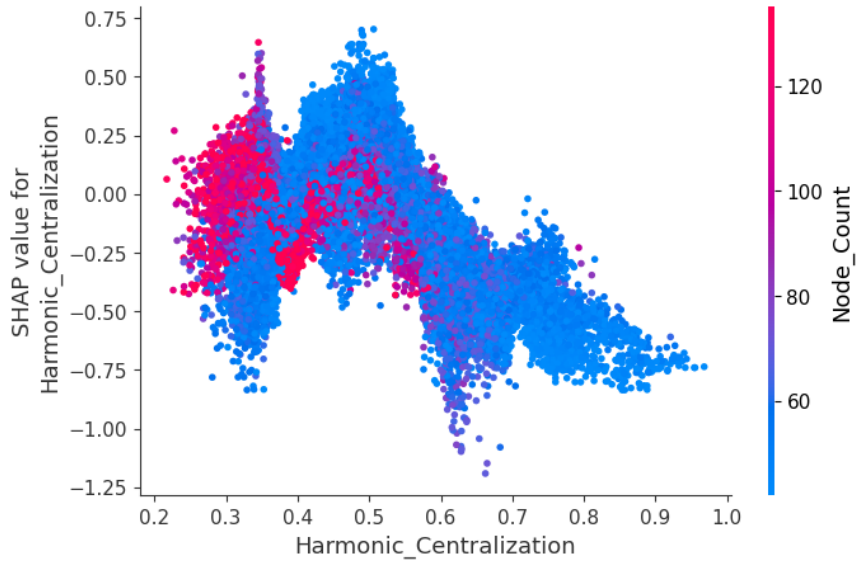

Figure 4.24: Dependence Plot of Harmonic Centralization and Response Time Variation Class with RTVT 0.4 Labelling

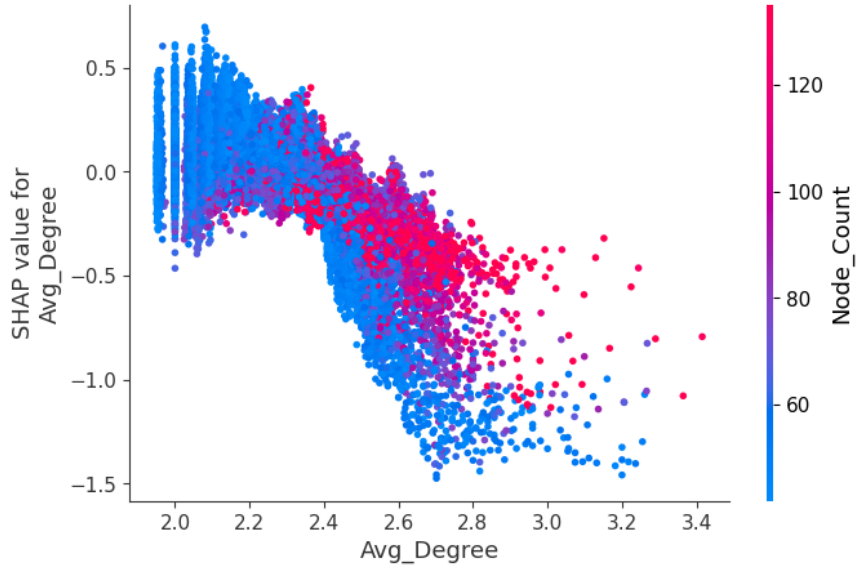

Figure 4.25: Dependence Plot of Average Degree and Response Time Variation Class with RTVT 0.4 Labelling

gion, as the average degree increases, the likelihood of the model evaluating the graph as steady also increases. Among graphs with the same average degree, those with low node count have an increasing impact of average degree on the model's decisions.

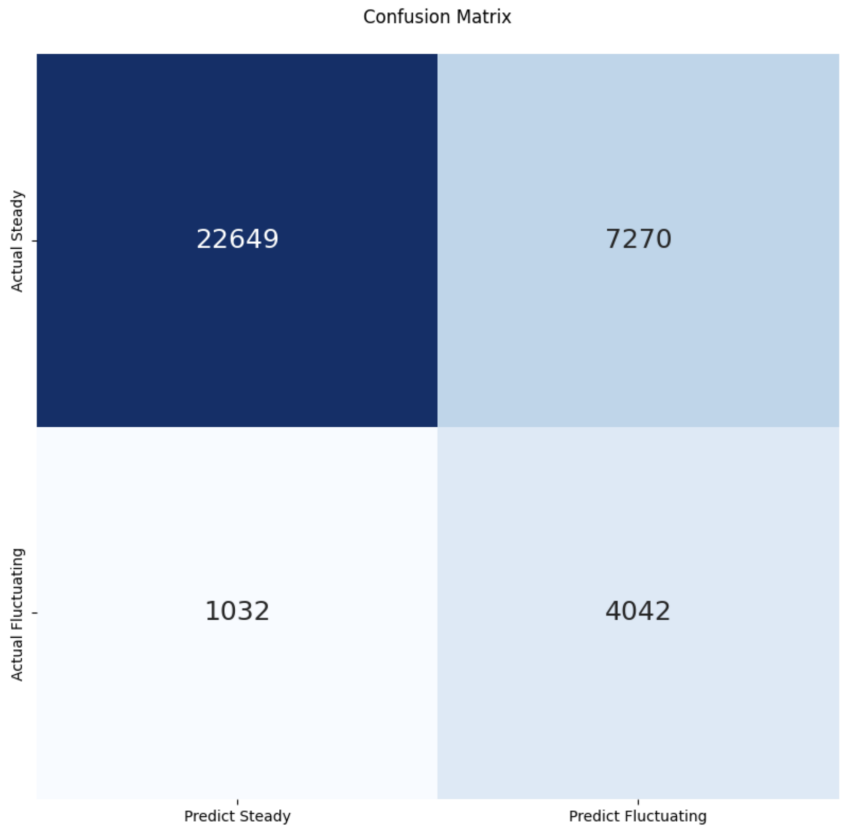

Figure 4.26: Catboost Classifier Confusion Matrix with RTVT 0.5 Labelling

The confusion matrix in Figure 4.26 for the CatBoost classifier trained with data labeled as RTVT 0.5 reveals that the model has a significant number of false negatives, indicating that a considerable portion of examples identified as fluctuating are incorrectly classified. Despite achieving satisfactory accuracy values in predictions labeled as steady, the low scores obtained in precision result in much lower F1 scores compared to other RTVT labels. While the model's F1 score is significantly better than the baseline, it can be considered an unsuccessful model in terms of F1 score. Therefore, when evaluating feature importance, one should approach the model's fluctuating decisions at the feature level with skepticism.

In terms of performance scores, the LightGBM classifier is in a worse condition than the CatBoost classifier for this label too. Therefore, the results of the LightGBM classifier for this label should also be approached with skepticism. When examining feature importance values, apart from an increased impact of the modularity value calculated with the label propagation algorithm, there is no significant difference in

the model compared to other labels.

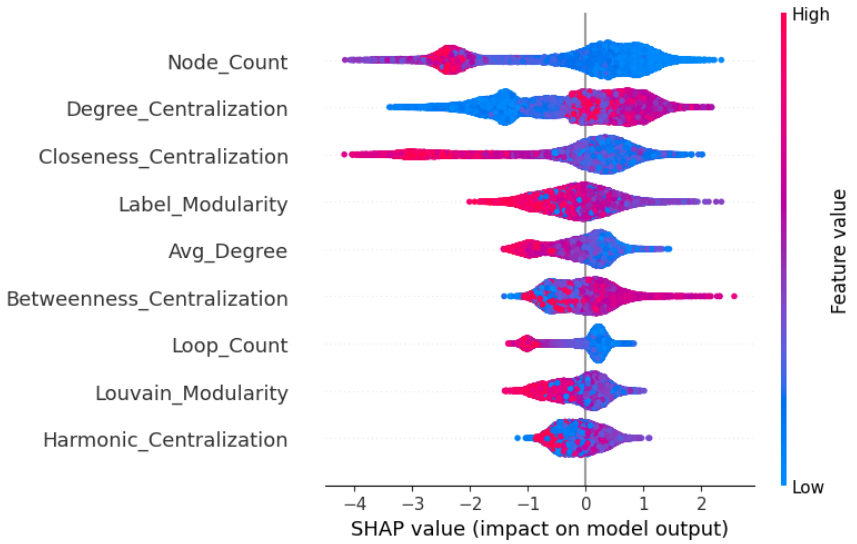

Figure 4.27: Catboost Classifier Feature Importance Plot with RTVT 0.5 Labelling

The feature importance plot of the CatBoost Classifier trained with RTVT 0.5 values in Figure 4.27 indicates that the impact of features is similar to that of RTVT 0.4. The effect of loop count has decreased compared to its impact in the RTVT 0.4 label, while the impact of average degree has increased. The dependence plots in Appendix A show that the pattern between the feature value and SHAP value is very similar, but the threshold values where the model starts to evaluate the call graph as steady or fluctuating changes. Similar to how low values of degree centralization increase the likelihood of being evaluated as a steady graph for RTVT 0.3 and RTVT 0.4 labels, a similar situation applies to RTVT 0.5. This pattern holds not only for degree centralization, but also for all features.

4.4 Discussion on Results

The experiment results demonstrate that the design of microservice architectures has significant effects on response time variation. ML models show scores above 0.8 in terms of accuracy, precision, and F1 score, and these scores are statistically significant according to the McNemar test. Performance scores and statistical test results suggest that the feature importance information of these models is reliable. Particularly,

information obtained from the dependence plots of features with high SHAP values provides valuable insights on how to design the microservice architecture topology to achieve low response time variation.

The first research question of the thesis is whether the topological features of microservice call graphs could predict response time variation. The thesis claims that, within the scope of this question, topological features are effective in predicting response time variation. Even without allocating resources and memory information to microservices, models can achieve performance scores above 0.8 which indicates the importance of topology. Furthermore, the thesis claims that ML models are successful in this prediction task.

Due to the frequently associated terms of decentralized and modular with microservice architectures, the second research question of the thesis focused on the impact of centralization and modularity features. Upon examining the experiment results, it is observed that centralization has a significant effect on model outputs. However, stating that this effect implies decentralized architectures are better in terms of response time variation is not sufficient because decentralized call graphs do not necessarily have low response time variation for all centralization metrics. In the case of closeness centralization, the likelihood of the model predicting low response time variation is higher for call graphs with high centralization. On the other hand, for degree centralization, which is another highly impactful centralization metric, the probability of the model predicting low response time variation is higher for decentralized call graphs. While it may not be entirely accurate to label graphs with high closeness centralization as decentralized since closeness centralization is calculated with undirected graphs, also examining betweenness and harmonic centralization dependence plots reveals that it is not entirely possible to say that decentralized graphs have low response time variation. Therefore, instead of establishing a relationship solely between centralization and response time variation, it is more accurate to evaluate each centralization metric separately. From a designer's perspective, maintaining a low degree centralization is the most easily achievable feature related to centralization because it is a centralization metric that is more easily noticeable. In this regard, if some advice is to be given to a microservice architecture designer through this thesis, keeping degree centralization very low would be preferable for response time

variation advantages. Additionally, as shown in the dependence plots, after a certain centralization value, the impact of features remains constant. Although it is not entirely possible to explain this phenomenon, it is probable that after reaching a certain centralization threshold, maximum attention is given to microservices in the central position, which may lead to stability remaining constant beyond a certain point.

The impact of modularity features, also examined within the scope of the second research question, can be understood from the dependence plots. Although different algorithms are used to calculate modularity, modularity metrics do not focus on features as diverse as centralization metrics. The community assignment algorithms in this thesis, capable of working with graphs containing millions of nodes, works for call graphs with a majority of microservices ranging between 40 and 200, provide close results. This explains the similar trends observed in dependence plots for the two modularity features. SHAP values for modularity metrics are meaningful and the modularity features are influential on model decisions. Models are more likely to predict low response time variation for call graphs with high modularity. Therefore, this thesis advises a designer who is designing a microservice architecture to create call graphs with high modularity to achieve architectures with low response time variation.

Meaningful interpretations can also be made regarding the impact of the node count and average degree features in the last research question. It does not seem very feasible to provide an interpretation on loop count through this thesis because of its high correlation with node count and its low impact according to SHAP values. For node count feature, it is not entirely clear why models predict low response time variation for call graphs with a high node count, but the impact of node count on the model is high. The presence of more stable calls may come from cache and database calls, in call graphs with a high node count. Additionally, the use of decomposed call graphs may contribute to achieving more stable results. Similarly, evaluating the model's call graph as having low response time variation in cases of high average degree can also be explained through decomposition. Since the source code within microservices has not been evaluated within the scope of this thesis, comments made on decomposition cannot be exact. Nevertheless, meaningful dependence plots have been presented that can provide designers with valuable information and insights into these features.

4.5 Discussion on Threats to Validity

In this study, which investigates the impact of topological features of microservice call graphs on response time variation, there are some threats to the validity of the experiments conducted. The results of this thesis can be affected by several factors such as the feature selection, the data sources, response time variation measurement, and the statistical methods used. in this section, the thesis discusses threats to the validity of this thesis and the methods that are used to solve the validity issues.

4.5.1 External Validity

The dataset forming the basis of the study is the Alibaba Cluster Trace data. Although this dataset contains billions of calls and 20 million microservice call graphs, it is a sampled dataset within a specific time range, which is 7 days. In the paper[15] where Luo et al. (2021) published their belief that this 7-day data is a good representation of Alibaba cloud infrastructure, there may be various anomalies that are overlooked during this time period and pose a threat to the validation of this study.

The structure of microservice call graphs in the Alibaba dataset closely resembles a tree structure. This suggests that the topologies in the dataset may actually have a bias. When examining microservices architectures where tree-like structures are rare, it should be considered that the results of this study may be biased.

4.5.2 Internal Validity

The entire Alibaba dataset is not used in this study due to its size. The dataset used in this study, consisting of randomly drawn 70k microservice call graphs, may not accurately reflect the Alibaba dataset. To address this issue, three different random samples were taken. Kolmogorov-Smirnov tests are applied to response time variation distributions of these three random samples until three random samples with similar distributions in terms of response time variation. The final dataset size obtained is 70k. Although these tests are applied, details that are overlooked in the sampling phase pose a risk to the validation of the study and limit its generalizability. In

addition, this study carries the possibility of being specific to this dataset. Common characteristics of large microservice call graphs in the Alibaba cloud infrastructure may exist, leading to bias when interpreting other architectures.

The response time of microservice call graphs is directly related to the processing power and memory resources allocated to microservices. Also, the runtime characteristics affect the response time variation. In this study, which only examines the impact of topology, information about these resources and characteristics is not used. This should be taken into account when evaluating the accuracy of the results of the study. Especially, the management of the resources of the sample taken for this study is important because it is conducted through sampling.

4.5.3 Construct Validity

In this study, features are pre-selected. While the feature list includes popular metrics that come to mind when talking about topological features, there are many ways to express topology differently. This factor limits the study that investigates the impact of topological features on response time variation. Additionally, the features in this thesis are based on undirected graphs. The use of directed features could provide more insight into the impact of topology. Besides the contribution of directed features to model performance, there are situations where not using these features could be risky. Cyclic dependency negatively affects the variation in response time in a call graph, and the absence of this information could be a threat to experiments. In the dataset used in the thesis, out of 20 million call graphs, there are not even 500 call graphs with cyclic dependency. Additionally, in the sample taken, call graphs without cyclic dependency are selected. Therefore, this threat has been mitigated. Since the aim of the thesis is to examine the effect of topology on response time variation, the most crucial information is found in undirected features. The model results confirm this claim.

When calculating response time variation, having more than 3 calls in the dataset ensures more accurate variation values. Calls between microservices vary in number. Some of these calls occur in large numbers and provide a better representation of response time variation. However, some calls are seen 4 times. The differences in

call numbers are also a factor that may affect the results.

Selecting call graphs with more than 40 microservices is a crucial decision that will impact the results of the research in terms of microservice count. Call graphs with fewer than 40 microservices are also worth investigating. However, call graphs with more than 40 microservices have a memcached service ratio close to 50%, and the stability of the memcached service ratio compared to call graphs with fewer than 40 nodes is an important factor in selecting the number 40. This is because the memcached service ratio significantly affects response time variation. When the ratios of memcached microservices in call graphs are not similar, serious biases occur in the results. Therefore, selecting call graphs with more than 40 nodes helps to mitigate this bias. Another advantage of selecting the number 40 is a more uniform distribution in terms of topology, as the occurrence of similar topologies increases as the number of microservices decreases. The reason for precisely selecting the number 40 is the provision of the memcached service ratio in the paper where the dataset is shared because not all datasets have been examined in this thesis [15]. Additionally, despite a heavy-tail distribution in terms of node count, 10% of call graphs are larger than 40, and this information is also available in the paper where the dataset is shared. A dataset consisting of 2 million call graphs has been sufficient to obtain significant results from this thesis.

4.5.4 Conclusion Validity

In the examination of the relationship between the topological features of microservice call graphs and response time variation, models with F1 scores, precision, and accuracy exceeding 0.8 were used. The McNemar statistical test was applied to verify the statistical significance of the results. The relationship between features and response time variation (RTV) was explored by evaluating SHAP values. The aim was to ensure the validity of the conclusions drawn in the thesis by using high-performance models, statistically significant results, and SHAP values.

CHAPTER 5

CONCLUSION

In this thesis, various research questions regarding the relationship between topological features and response time variation of microservice call graphs have been investigated.

Only by utilizing the topological features of call graphs, it is possible to predict whether microservice call graphs are steady or fluctuating in terms of response time variation. Machine learning algorithms have demonstrated successful results in these predictions. When the first half of microservice call graphs, with low response time variation labeled as steady, and the other half with high response time variation labeled as fluctuating, precision, accuracy, and F1 score values all exceeded 0.8. These results are statistically significant upon testing.

Examining centralization metrics, it is concluded that graphs with high closeness centralization and low degree centralization are evaluated as steady by the most successful model, the CatBoost classifier. These two centralization metrics are identified as the most influential features on response time variation. The results do not demonstrate a completely linear relationship between these two metrics. The impact of features on model decisions varies within certain intervals.

When modularity values obtained using the Louvain algorithm and label propagation algorithm are examined, it is observed that modular call graphs are considered steady by the model for both of these metrics. According to the model, call graphs with higher node counts are more likely to be steady. Intervals in which the model evaluates the call graph as steady are identified for betweenness centralization, harmonic centralization, average degree, and loop count features, and the feature importance

plots in this thesis show these intervals.

The relationship between the used topological features and call graphs in the Alibaba dataset has been analyzed in a detailed way. Considering the results of this thesis, more effective designs for response time variation can be achieved while designing microservice architectures.

5.1 Future Work

This thesis explores the relationship between the topological features of microservice call graphs and response time variation, interpreting the impact of these features. There are alternative methods that could be applied to examine the relationship between microservice call graph topology and response time variation in more detail and with greater accuracy. Although these methods are not implemented in this thesis, they are planned for future work. They will be used to provide a more detailed explanation of the relationship between microservice call graph topology and response time variation.

The dataset used in this thesis has been prepared with a sample taken from the Alibaba cluster trace dataset. Although methods are used to maximize the sample's representational capabilities, future studies will aim to enhance resource and memory management to utilize the entire dataset. This approach will eliminate the threats caused by sampling. Examining response time variation along with runtime characteristics can produce valuable insights for microservice architecture design, and adding runtime characteristics to the scope of this research is planned as future work. Different types of calls, such as RPC and DB calls in the Alibaba dataset, are evaluated in a similar manner, considering their differences to provide diverse interpretations. The rationale behind the selection of features used in the thesis is explained in Chapter 3, but additional features that can be derived from topology should be explored. The selected features are obtained by converting call graphs into undirected graphs, but the directed form of call graphs, which contains important features representing topology, is also worthy for further investigation. Therefore, using topological features of directed call graph features is planned for future work.

While the machine learning models obtained in the thesis provide sufficient performance and significance to explain the impact of topological features on response time variation, better-performing models can be achieved, especially with the preference for deep learning models. The Alibaba dataset has enough data to train a deep learning model, and maximum performance can be obtained by using the entire dataset. Although deep learning models may not be as successful as machine learning models in terms of feature explanation, they are a valuable method for illustrating how effective topology is on response time variation and achieving the maximum performance.

All the call graphs used in the thesis are from the same source, limiting the generalizability of the results. To increase the diversity of microservice call graph topologies, different datasets will be explored. Although there may not be datasets as comprehensive as the Alibaba dataset yet, upcoming datasets in this field will be followed, and similar methodologies will be applied to enrich the data from these different sources.

REFERENCES

[1] O. Zimmermann, "Microservices tenets: Agile approach to service development and deployment," *Computer Science - Research and Development*, vol. 32, p. 301–310, Nov. 2016.

[2] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Software*, vol. 35, no. 3, pp. 24–35, 2018.

[3] M. Fowler and J. Lewis, "Microservices: a definition of this new architectural term," 2014.

[4] D. Taibi, V. Lenarduzzi, C. Pahl, and A. Janes, "Microservices in agile software development: A workshop-based study into issues, advantages, and disadvantages," in *Proceedings of the XP2017 Scientific Workshops*, XP '17, (New York, NY, USA), Association for Computing Machinery, 2017.

[5] C. Pahl and P. Jamshidi, "Microservices: A systematic mapping study.," *CLOSER (1)*, pp. 137–146, 2016.

[6] S.-P. Ma, C.-Y. Fan, Y. Chuang, W.-T. Lee, S.-J. Lee, and N.-L. Hsueh, "Using service dependency graph to analyze and test microservices," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 02, pp. 81–86, 2018.

[7] S.-P. Ma, C.-Y. Fan, Y. Chuang, I.-H. Liu, and C.-W. Lan, "Graph-based and scenario-driven microservice analysis, retrieval, and testing," *Future Generation Computer Systems*, vol. 100, pp. 724–735, 2019.

[8] Álvaro Brandón, M. Solé, A. Huélamo, D. Solans, M. S. Pérez, and V. Muntés-Mulero, "Graph-based root cause analysis for service-oriented and microservice architectures," *Journal of Systems and Software*, vol. 159, p. 110432, 2020.

[9] L. Meng, F. Ji, Y. Sun, and T. Wang, "Detecting anomalies in microservices with execution trace comparison," *Future Generation Computer Systems*, vol. 116, pp. 291–301, 2021.

[10] H. Farsi, D. Allaki, A. En-Nouaary, and M. Dahchour, "A graph-based solution to deal with cyclic dependencies in microservices architecture," in *2022 9th International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 254–259, 2022.

[11] M. Cojocaru, A. Uta, and A.-M. Oprescu, "Microvalid: A validation framework for automatically decomposed microservices," in *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 78–86, 2019.

[12] A. Santos and H. Paula, "Microservice decomposition and evaluation using dependency graph and silhouette coefficient," in *Proceedings of the 15th Brazilian Symposium on Software Components, Architectures, and Reuse*, SBCARS '21, (New York, NY, USA), p. 51–60, Association for Computing Machinery, 2021.

[13] C. Zhong, H. Zhang, C. Li, H. Huang, and D. Feitosa, "On measuring coupling between microservices," *Journal of Systems and Software*, vol. 200, p. 111670, 2023.

[14] H. Tian, Y. Zheng, and W. Wang, "Characterizing and synthesizing task dependencies of data-parallel jobs in alibaba cloud," in *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '19, (New York, NY, USA), p. 139–151, Association for Computing Machinery, 2019.

[15] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, "Characterizing microservice dependency and performance: Alibaba trace analysis," in *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '21, (New York, NY, USA), p. 412–426, Association for Computing Machinery, 2021.

[16] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, J. He, and C. Xu, "An in-depth study of microservice call graph and runtime performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 3901–3914, 2022.

[17] S. Eismann, C.-P. Bezemer, W. Shang, D. Okanović, and A. van Hoorn, "Microservices: A performance tester's dream or nightmare?," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ICPE '20, (New York, NY, USA), p. 138–149, Association for Computing Machinery, 2020.

[18] P. Leitner and J. Cito, "Patterns in the chaos—a study of performance variation and predictability in public iaas clouds," *ACM Trans. Internet Technol.*, vol. 16, apr 2016.

[19] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production cloud services," in *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 104–113, 2011.

[20] S. Lehrig, H. Eikerling, and S. Becker, "Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics," in *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*, QoSA '15, (New York, NY, USA), p. 83–92, Association for Computing Machinery, 2015.

[21] A. S. Abdelfattah and T. Cerny, "Roadmap to reasoning in microservice systems: A rapid review," *Applied Sciences*, vol. 13, no. 3, p. 1838, 2023.

[22] A. S. Abdelfattah and T. Cerny, "Roadmap to reasoning in microservice systems: A rapid review," *Applied Sciences*, vol. 13, no. 3, p. 1838, 2023.

[23] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, and Y. Al-Hammadi, "The evolution of distributed systems towards microservices architecture," in *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, pp. 318–325, IEEE, 2016.

[24] C. E. Cuesta, E. Navarro, and U. Zdun, "Synergies of system-of-systems and microservices architectures," in *Proceedings of the International Colloquium on Software-Intensive Systems-of-Systems at 10th European Conference on Software Architecture*, pp. 1–7, 2016.

[25] S. Hassan and R. Bahsoon, "Microservices and their design trade-offs: A self-

adaptive roadmap," in *2016 IEEE International Conference on Services Computing (SCC)*, pp. 813–818, IEEE, 2016.

[26] R. V. O'Connor, P. Elger, and P. M. Clarke, "Continuous software engineering—a microservices architecture perspective," *Journal of Software: Evolution and Process*, vol. 29, no. 11, p. e1866, 2017.

[27] V. W. Berger and Y. Zhou, "Kolmogorov–smirnov test: Overview," *Wiley statsref: Statistics reference online*, 2014.

[28] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[29] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference* (Stéfan van der Walt and Jarrod Millman, eds.), pp. 56 – 61, 2010.

[30] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[31] D. Freedman, R. Pisani, and R. Purves, "Statistics (international student edition)," *Pisani, R. Purves, 4th edn. WW Norton & Company, New York*, 2007.

[32] M. Jenders, G. Kasneci, and F. Naumann, "Analyzing and predicting viral tweets," in *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13 Companion, (New York, NY, USA), p. 657–664, Association for Computing Machinery, 2013.

[33] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1, pp. 278–282, IEEE, 1995.

[34] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," *Advances in neural information processing systems*, vol. 31, 2018.

[35] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, pp. 3146–3154, 2017.

[36] P. A. Lachenbruch, "Mcnemar test," *Wiley StatsRef: Statistics Reference Online*, 2014.

[37] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 4765–4774, Curran Associates, Inc., 2017.

[38] M. I. Rahman, S. Panichella, and D. Taibi, "A curated dataset of microservices-based systems," *SSSME-2019*, 2019.

[39] Alibaba, "Alibaba/clusterdata: Cluster data collected from production clusters in alibaba for cluster management research." Available at https://github.com/alibaba/clusterdata.

[40] D. S. H. Tam, Y. Liu, H. Xu, S. Xie, and W. C. Lau, "Pert-gnn: Latency prediction for microservice-based cloud-native applications via graph neural networks," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, (New York, NY, USA), p. 2155–2165, Association for Computing Machinery, 2023.

[41] X. Li, J. Zhou, X. Wei, D. Li, Z. Qian, J. Wu, X. Qin, and S. Lu, "Topology-aware scheduling framework for microservice applications in cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 5, pp. 1635–1649, 2023.

[42] S. Luo, H. Xu, K. Ye, G. Xu, L. Zhang, G. Yang, and C. Xu, "The power of prediction: Microservice auto scaling via workload learning," in *Proceedings of the 13th Symposium on Cloud Computing*, pp. 355–369, 2022.

[43] CloudStatus, "Report on cloud performance and availability status," 2010. Available at www.cloudstatus.com.

[44] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[45] A. Palczewska, J. Palczewski, R. M. Robinson, and D. Neagu, "Interpreting random forest models using a feature contribution method," in *2013 IEEE 14th International Conference on Information Reuse & Integration (IRI)*, pp. 112–119, IEEE, 2013.

[46] A. Orlenko and J. H. Moore, "A comparison of methods for interpreting random forest models of genetic association in the presence of non-additive interactions," *BioData mining*, vol. 14, no. 1, pp. 1–17, 2021.

[47] A. Orlenko and J. H. Moore, "A comparison of methods for interpreting random forest models of genetic association in the presence of non-additive interactions," *BioData mining*, vol. 14, no. 1, pp. 1–17, 2021.

[48] A. Orlenko and J. H. Moore, "A comparison of methods for interpreting random forest models of genetic association in the presence of non-additive interactions," *BioData mining*, vol. 14, no. 1, pp. 1–17, 2021.

[49] A. A. Alabdullah, M. Iqbal, M. Zahid, K. Khan, M. N. Amin, and F. E. Jalal, "Prediction of rapid chloride penetration resistance of metakaolin based high strength concrete using light gbm and xgboost models by incorporating shap analysis," *Construction and Building Materials*, vol. 345, p. 128296, 2022.

[50] CatBoost, "Catboost reference papers," 2018. Available at https://catboost.ai/en/docs/concepts/educational-materials-papers.

[51] N. H. M. Khalid, A. R. Ismail, and N. A. Aziz, "Interpretation of machine learning model using medical record visual analytics," in *Proceedings of the 8th International Conference on Computational Science and Technology: ICCST 2021, Labuan, Malaysia, 28–29 August*, pp. 633–645, Springer, 2022.

[52] S. B. Jabeur, C. Gharib, S. Mefteh-Wali, and W. B. Arfi, "Catboost model and artificial intelligence techniques for corporate failure prediction," *Technological Forecasting and Social Change*, vol. 166, p. 120658, 2021.

[53] M. Luo, Y. Wang, Y. Xie, L. Zhou, J. Qiao, S. Qiu, and Y. Sun, "Combination of feature selection and catboost for prediction: The first application to the estimation of aboveground biomass," *Forests*, vol. 12, no. 2, p. 216, 2021.

[54] S. Hussain, M. W. Mustafa, T. A. Jumani, S. K. Baloch, H. Alotaibi, I. Khan, and A. Khan, "A novel feature engineered-catboost-based supervised machine learning framework for electricity theft detection," *Energy Reports*, vol. 7, pp. 4425–4436, 2021.

[55] A. A. Ibrahim, R. L. Ridwan, M. M. Muhammed, R. O. Abdulaziz, and G. A. Saheed, "Comparison of the catboost classifier with other machine learning methods," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 11, 2020.

[56] V. A. Traag, L. Waltman, and N. J. Van Eck, "From louvain to leiden: guaranteeing well-connected communities," *Scientific reports*, vol. 9, no. 1, p. 5233, 2019.

[57] G. Cordasco and L. Gargano, "Community detection via semi-synchronous label propagation algorithms," in *2010 IEEE international workshop on: business applications of social network analysis (BASNA)*, pp. 1–8, IEEE, 2010.

[58] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Physical review E*, vol. 70, no. 6, p. 066111, 2004.

[59] S. YANG, "Networks: An introduction by m. e. j. newman," *The Journal of Mathematical Sociology*, vol. 37, no. 4, pp. 250–251, 2013.

[60] S. Roy and N. Feamster, "Characterizing correlated latency anomalies in broadband access networks," *SIGCOMM Comput. Commun. Rev.*, vol. 43, p. 525–526, aug 2013.

[61] S. K. Barker and P. Shenoy, "Empirical evaluation of latency-sensitive application performance in the cloud," in *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*, MMSys '10, (New York, NY, USA), p. 35–46, Association for Computing Machinery, 2010.

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

APPENDIX A

THE DEPENDENCE PLOTS OF TOPOLOGICAL FEATURES WITH RTVT 0.5 LABELS

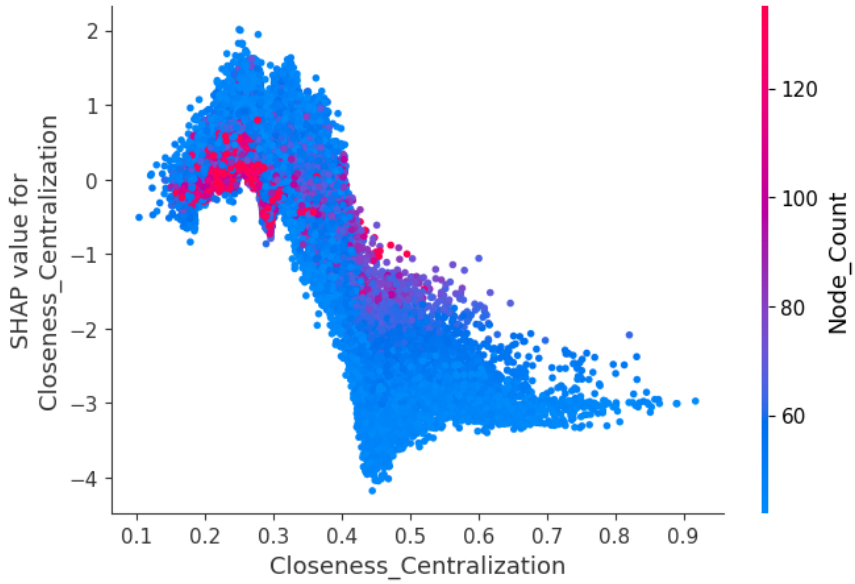

Figure A.1: Dependence Plot of Closeness Centralization and Response Time Variation Class with RTVT 0.5 Labelling

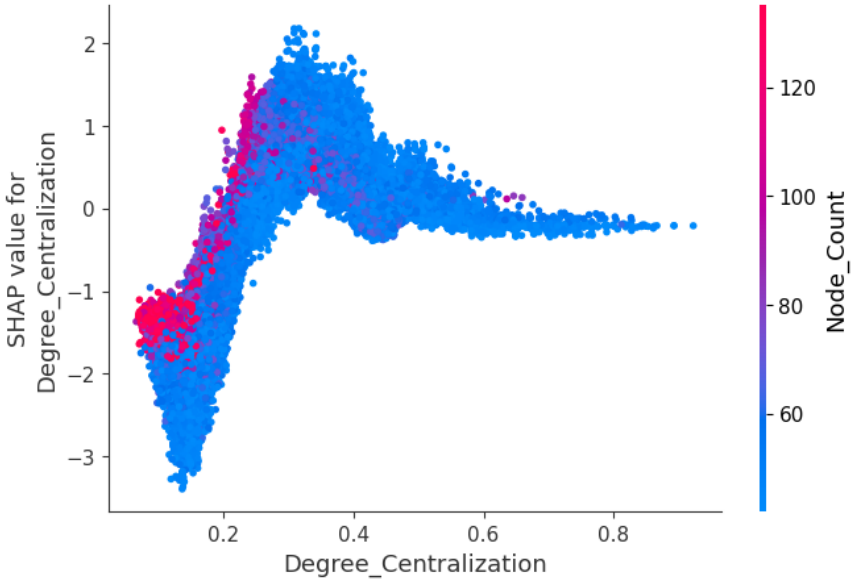

Figure A.2: Dependence Plot of Degree Centralization and Response Time Variation Class with RTVT 0.5 Labelling

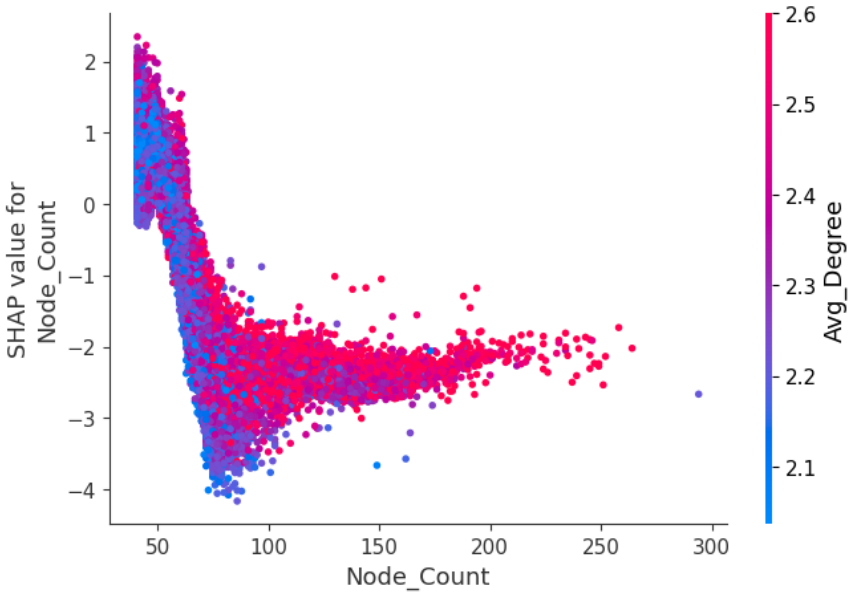

Figure A.3: Dependence Plot of Node Count and Response Time Variation Class with RTVT 0.5 Labelling

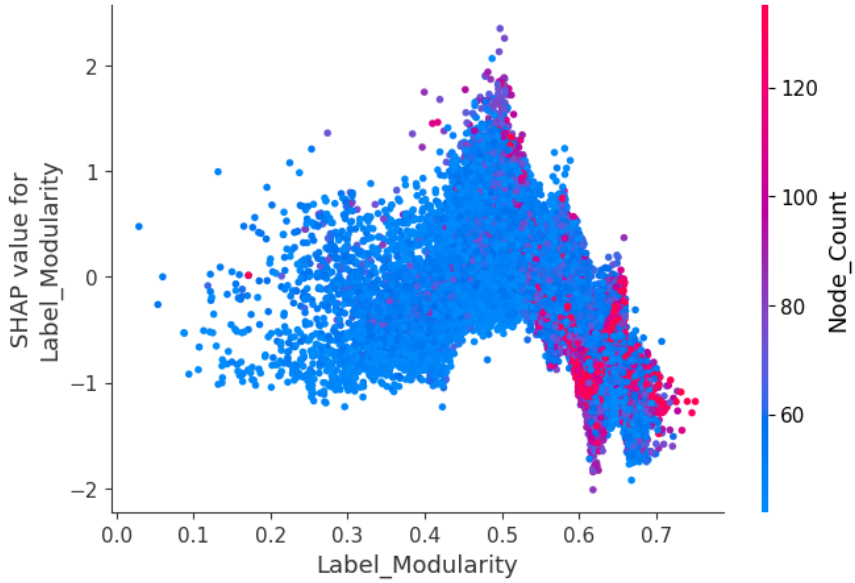

Figure A.4: Dependence Plot of Label Modularity and Response Time Variation Class with RTVT 0.5 Labelling

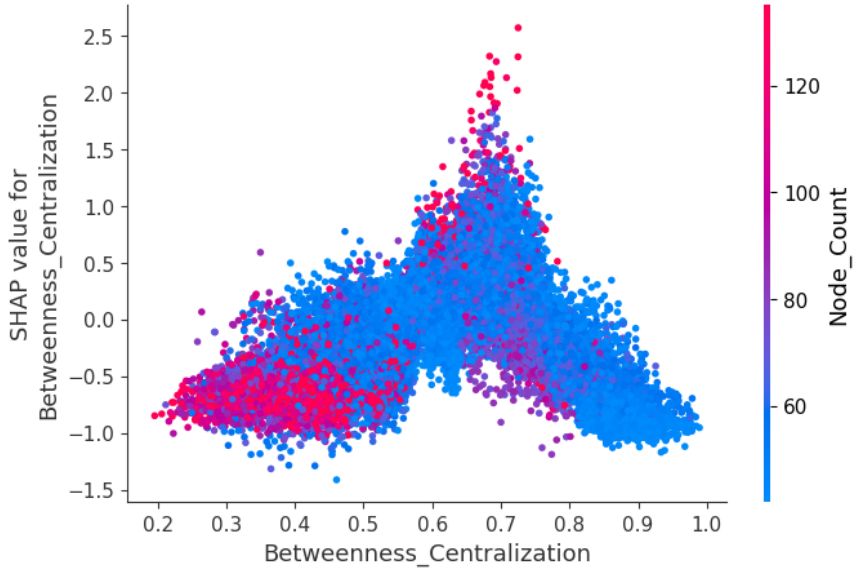

Figure A.5: Dependence Plot of Betweenness Centralization and Response Time Variation Class with RTVT 0.5 Labelling

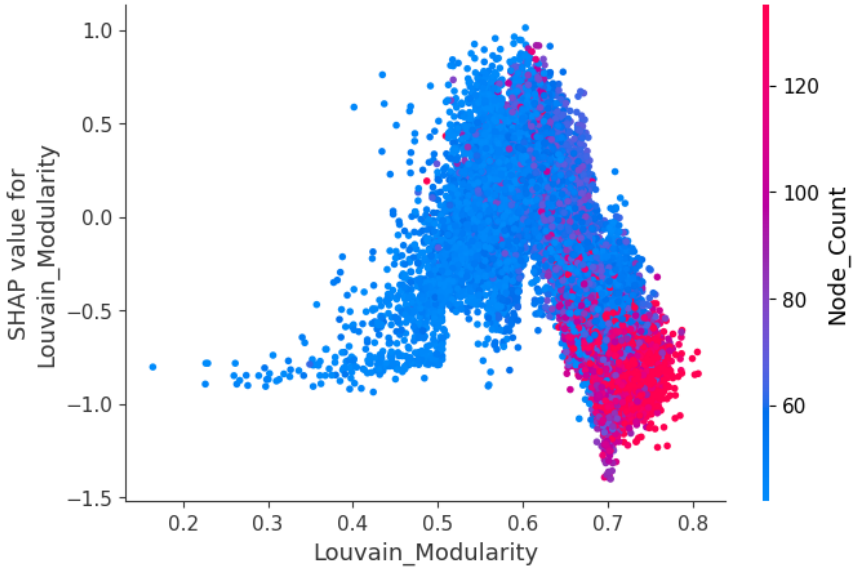

Figure A.6: Dependence Plot of Louvain Modularity and Response Time Variation Class with RTVT 0.5 Labelling

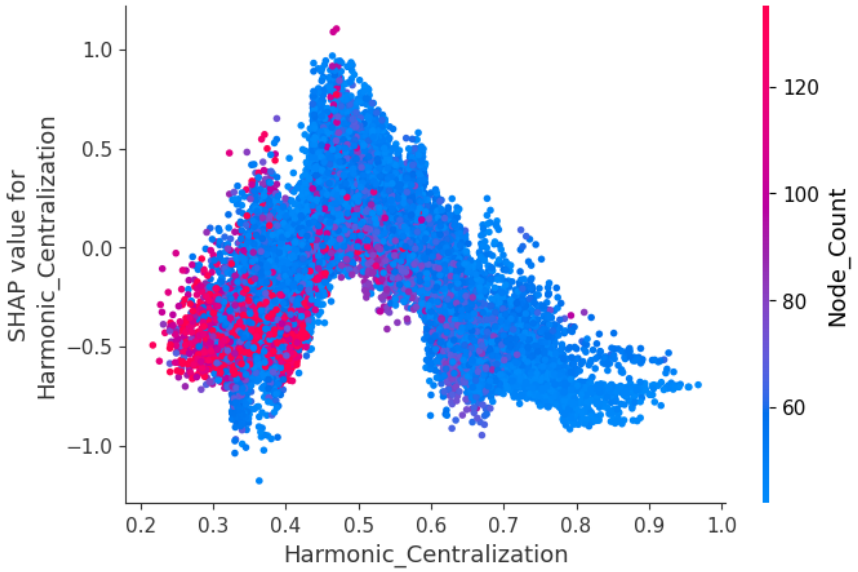

Figure A.7: Dependence Plot of Harmonic Centralization and Response Time Variation Class with RTVT 0.5 Labelling

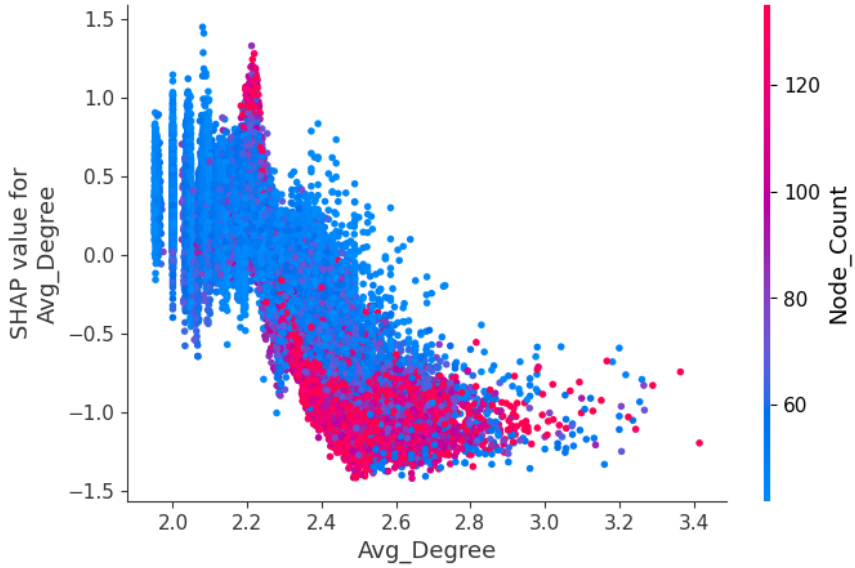

Figure A.8: Dependence Plot of Average Degree and Response Time Variation Class with RTVT 0.5 Labelling