KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in computer science and engineering

First cycle, 30 credits

Monolith to microservices using deep learning-based community detection

ANTON BOTHIN

Stockholm, Sweden, 2023

Monolith to microservices using deep learning-based community detection

ANTON BOTHIN

Master's Programme, Machine Learning, 120 credits

Date: June 12, 2023

Supervisors: Ragnar Thobaben, Johan Forsling

Examiner: Pawel Herman

School of Electrical Engineering and Computer Science

Host company: PE Accounting

Swedish title: Monolit till mikrotjänster med hjälp av djupinlärningsbaserad klusterdetektion

Abstract

The microservice architecture is widely considered to be best practice. Yet, there still exist many companies currently working in monolith systems. This can largely be attributed to the difficult process of updating a systems architecture. The first step in this process is to identify microservices within a monolith. Here, artificial intelligence could be a useful tool for automating the process of microservice identification.

The aim of this thesis was to propose a deep learning-based model for the task of microservice identification, and to compare this model to previously proposed approaches. With the goal of helping companies in their endeavour to move towards a microservice-based architecture. In particular, the thesis has evaluated whether the more complex nature of newer deep learning-based techniques can be utilized in order to identify better microservices.

The model proposed by this thesis is based on overlapping community detection, where each identified community is considered a microservice candidate. The model was evaluated by looking at cohesion, modularity, and size. Results indicate that the proposed deep learning-based model performs similarly to other state-of-the-art approaches for the task of microservice identification. The results suggest that deep learning indeed helps in finding nontrivial relations within communities, which overall increases the quality of identified microservices, From this it can be concluded that deep learning is a promising technique for the task of microservice identification, and that further research is warranted.

Keywords

Community detection, Deep learning, Graph neural network, Microservice, System architecture

Sammanfattning

Allmänt anses mikrotjänstarkitekturen vara bästa praxis. Trots det finns det många företag som fortfarande arbetar i monolitiska system. Detta då det finns många svårigheter runt processesen av att byta systemaritekture. Första steget i denna process är att identifiera mikrotjänster inom en monolit. Här kan artificiell intelligens vara ett användbart verktyg för att automatisera processen runt att identifiera mikrotjänster.

Denna avhandling syftar till att föreslå en djupinlärningsbaserad modell för att identifiera mikrotjänster och att jämföra denna modell med tidigare föreslagna modeller. Målet är att hjälpa företag att övergå till en mikrotjänstbaserad arkitektur. Avhandlingen kommer att utvärdera nyare djupinlärningsbaserade tekniker för att se ifall deras mer komplexa struktur kan användas för att identifiera bättre mikrotjänster.

Modellen som föreslås är baserad på överlappande klusterdetektion, där varje identifierad kluster betraktas som en mikrotjänstkandidat. Modellen utvärderades genom att titta på sammanhållning, modularitet och storlek. Resultaten indikerar att den föreslagna djupinlärningsbaserade modellen identifierar mikrotjänster av liknande kvalitet som andra state-of-the-art-metoder. Resultaten tyder på att djupinlärning bidrar till att hitta icke triviala relationer inom kluster, vilket ökar kvaliteten av de identifierade mikrotjänsterna. På grund av detta dras slutsatsen att djupinlärning är en lovande teknik för identifiering av mikrotjänster och att ytterligare forskning bör utföras.

Nyckelord

Klustringsdetektion, Djupinlärning, Graf-neuronnät, Mikrotjänst, Systemarkitektur

Acknowledgments

I extend a huge thanks to PE Accounting and its employees. I thank them both for introducing me to the problem of microservice identification, and for allowing me to evaluate the implemented deep learning model using their system. Further, a special thanks is extended to Johan Forsling, who acted as supervisor at the company. My gratitude is also extended to my supervisor at KTH Royal Institute of Technology, Ragnar Thobaben, whom provided a lot of useful feedback throughout the entire writing process. Finally, I thank Pawel Herman, who took his time to act as examiner for the thesis. Without these people, the thesis work could not have been finished.

Stockholm, June 2023
Anton Bothin

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

CHD	Cohesion at Domain Level
CHM	Cohesion at Message Level
CMQ	Conceptual Modularity Quality
GCN	Graph Convolutional Network
GNN	Graph Neural Network
IFN	Interface Number
NOCD	Neural Overlapping Community Detection
SMQ	Structural Modularity Quality

Chapter 1

Introduction

During the last decade, the microservice architecture has quickly become a predominant architecture for the development of software applications. Before this approach was developed, the common architectural style was to develop systems as monoliths; large single process systems responsible for all business logic [1]. In comparison, the microservice architecture consists of many small independent processes, each responsible for a single specific business function [2, 3]. In this way, the microservice architecture allows for greater scalability, flexibility, and maintainability compared to the monolithic architecture.

Because of the previous popularity of the monolithic approach, many companies still work in legacy monolith systems. These companies have great incentive to make the switch to a microservice architecture due to the significant benefits which this switch brings. For large systems this switch can be a very time consuming process. A common way to go about this is by iteratively extracting one microservice at a time from the monolith. This is beneficial since it allows the company to continue the implementation of new features, while in parallel moving to a microservice architecture. However, identifying good microservices within a monolith is not always trivial. Even though only a single microservice is extracted at a time, one still needs to consider all microservices as a whole. This is a difficult and time consuming process in itself, but it is one which a company needs to perform if they wish to reap the benefits which the microservice architecture has to offer.

1.1 Background

Historically, the above mentioned task of identifying microservices has been performed manually by the software engineers. In order to help with this manual task, three core principles of good microservices have been developed: (1) *Bounded context*, a microservice should only be responsible for a single business functionality, (2) *Size*, a microservice should not be too large, and (3) *Independency*, microservices should only communicate through their published interfaces [4].

Recently, researchers have looked into how to automate this task. In particular, clustering has been explored as one possible approach for the automatic detection of microservices [5, 6, 7, 8, 9]. These approaches have been shown to produce useful results. Nonetheless, they do not come without flaws. At most these approaches only uphold the bounded context and independency principles, allowing microservices of any size, which largely stems from the utilization of well-established, but simple, clustering techniques.

A lot of progress has been made in the filed of clustering and community detection, and recently a lot of focus have been placed on the use of deep learning for community detection [10, 11]. These approaches have been shown to be generally better suited for learning complex structural relationships. Because of this, it is very possible that utilizing these newer deep learning-based techniques would result in better microservice detection.

To summarize, companies currently working in legacy monolith systems have incentive to update their system architecture using the microservice style. To make this switch they first need to identify microservice candidates within the monolith. Many approaches have been suggested to automate this task. However, the current approaches do not utilize state-of-the-art techniques which are more suited to this type of task.

1.2 Problem statement

The problem which this thesis tackles is that of automatic microservice identification. In particular, the thesis examined whether newer deep learning-based community detection models can be utilized in order to improve the microservice identification process in comparison to previously suggested approaches. In order to tackle this problem, the following research questions have been posed:

How do deep learning-based community detection models hold up compared to simpler clustering based models when it comes to automatic microservice detection? Does the ability to learn complex structural relationships help in identifying relevant microservices while upholding the bounded context, size, and independency criteria?

These research questions serves as the basis for the thesis and for the overall research process. The purpose is to develop a model which can automatically identify microservice candidates within a monolithic system. The model can take in source code as input, and for each source code file in the monolith, assign it to one or more possible microservices. Further, the goal of this thesis is twofold. Firstly, the short-term goal is to help companies which are currently working in legacy monolith systems to make the switch to a microservice architecture. Secondly, the long-term goal is to help advance research within the field of community detection. This is done by exploring a non-trivial use case of deep learning-based community detection, and comparing the results with simpler clustering-based approaches.

1.3 Objective and scopes

In order to achieve the above mentioned goals, the following two research objectives are posed:

1. Implement a deep learning-based community detection model and qualitatively assess whether it can successfully identify relevant microservices.
2. Further, using quantitative measures, determine whether the model can uphold all three principles of the microservice architecture.

The focus of this thesis is on comparing the performance of deep learning-based community detection models to that of simpler clustering-based models when it comes to the identification of microservices. This is a broad topic covering many different areas, due to time constraints some limitations have been put in place:

- There exists a plethora of different deep learning-based models for community detection. Implementing and evaluating all models is outside the scope of this thesis. Therefore, a single deep learning model has been chosen, Neural Overlapping Community Detection (NOCD)

[12], which is described in Subsection 2.6.3. No other model besides this one has been implemented or evaluated.

- The model takes in source code as input. This source code can be preprocessed in many different ways, and the preprocessing steps undoubtedly have an effect on the final performance. The aim is however not to evaluate the impact of preprocessing, but to compare community detection models, and this is therefore outside the scope of this thesis. The same preprocessing steps as those used by previous microservice identification approaches are used.

1.4 Research methodology

This thesis focuses on how deep learning-based community detection models compare to the previously used clustering-based approaches. Some quantitative measures exist for evaluating microservices, however in-depth knowledge is also needed. Therefore a mixed approach was chosen, using both the quantitative and qualitative research method. In addition to the research methodology, this thesis followed design science as a research paradigm [13] when implementing the chosen deep learning model. In order to gather information about microservices, previous approaches to microservice identification, and deep learning-based community detection models, an extensive literature study has been conducted. Expert knowledge has been used to assess the usefulness of the identified microservices. And quantitative measures have been used to evaluate whether each microservice upholds the three criteria outlined in Section 1.1.

In summary, a literature study was conducted in order to gather information about current approaches to the problem which this thesis tackles. This information was then used to implement a model for the identification of microservices. These microservices were then evaluated using both quantitative and qualitative measures. Further details regarding the research method are presented in Chapter 3.

1.5 Commissioned work

PE Accounting, a financial service focusing on automation [14], is one of the many companies which currently work in a legacy monolith system. At the moment they are in the middle of a move towards adopting the microservice

architecture, and already have a couple of microservices in place. In order to speed up this process PE Accounting have commissioned the author of this thesis to develop an automated solution for the identification of new microservices within their monolith.

1.6 Structure of the thesis

Subsequent chapters of this thesis are outlined as follows:

- *Chapter 2: Background:* Theoretical and practical background is presented throughout this chapter. The information presented within is needed in order to understand the remaining chapters.
- *Chapter 3: Methods:* This chapter covers the research phases and outlines the developed model . This covers the preprocessing steps, model architecture, and hyperparameters.
- *Chapter 4: Results and analysis:* Results gathered from the evaluation are presented here, these results are also analyzed.
- *Chapter 5: Discussion:* The results presented are further discussed within this chapter.
- *Chapter 6: Conclusion and future work:* The discussions made in previous chapters are finally concluded here. This chapter aims to answer the research question, and proposes avenues for future work.

Chapter 2

Background

In this chapter, the background knowledge needed in order to understand the rest of the thesis is covered. The chapter starts of by introducing the concept of microservices and microservice architecture in Section 2.1. This section provides the definitions used throughout the thesis, and introduces some principles of microservices which is later used for evaluation. A definition of community detection is then provided in Section 2.2. Here, two popular community detection algorithms are also covered. Then, in Section 2.3, a framework is outlined, describing what is needed for the automatic detection of microservices. In this section, some core properties are identified which are required by a model in order for it to be applicable to microservice identification. Further, in Section 2.4, some metrics which can be used to evaluate these properties are described. This is then followed by an in-depth description of graph neural networks in Section 2.5. The graph neural network architecture is a natural choice for community detection, since such problems are often based on graphs that represent connections between entities. This also explains why all models covered in Subsection 2.6.3 are based on this architecture. Section 2.6 covers previous research related to the topic of this thesis. This covers both previous approaches to automatic microservice identification, and different deep learning-based methods for community detection. Finally, in Section 2.7, the chapter is summarized with some benefits and drawbacks of the previously proposed methods.

Table 2.1: Definitions of Microservice and Microservice architecture

Microservice: A microservice is a cohesive, independent process interacting via messages [3].
Microservice architecture: A microservice architecture is a distributed application where all its modules are microservices [3].

2.1 Microservice architecture

Since this project aims to automatically identify microservices, it can be useful to first introduce what is meant by microservices and microservice architecture. Most research papers generally agree on the definitions of these concepts. However, slight variations can be found. For the remainder of this thesis, the definitions provided in Table 2.1 are used. To get a clearer idea of what is meant by the first definition of a microservice, some further clarification is provided.

A microservice is cohesive, meaning that it should have a single well-defined responsibility, and all of its implemented functionality should be directly related to this responsibility. This can also be described as the microservice having bounded context [15, 4]. Further, a microservice needs to be independent, meaning that each microservice should be a separate process which can be independently deployed. These processes should then only communicate through their published interfaces, which can either be implemented with a RESTful API or message queue. Although not explicitly stated in the definition, a microservice should also be kept small in size [4]. This both provides better maintainability and helps in ensuring cohesion. From this, as was briefly mentioned in Section 1.1, it can be established that all microservices need to uphold three core principles: (1) *Cohesion / Bounded context*, (2) *Size*, and (3) *Independency* [4].

2.2 Community detection

In order to get a better understanding of the techniques discussed later in this chapter, this section aims to give a brief overview of what is meant by community detection. In literature, community detection and clustering are often used interchangeably [16], and for this project there is no need to distinguish between the two. However, to keep a consistent vocabulary

throughout this thesis, the term community detection is used. Broadly speaking, community detection is the process of identifying groups of related nodes within a network. These groups are generally found by looking for nodes that are more densely connected than the rest of the network [17].

One such approach to community detection is based on modularity optimization, which is one of the more popular community detection techniques. Many different algorithms for modularity optimization exists. However, among these the Louvain method is most commonly used. Modularity aims to measure the quality of communities by comparing the number of edges within a community to the expected number of edges in a random graph with the same degree distribution. By maximizing modularity, optimal communities are found [18, 19]. The modularity function Q can be formally written as:

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j) \tag{2.1}$$

In Eq. (2.1), A_{ij} are the elements of the adjacency matrix, m is the number of edges in the graph, k_i represent the degrees of node i, and c_i is the community to which node i belong. $\delta(c_i, c_j)$ is the Kronecker delta function which outputs 1 if nodes i and j belong to the same community, and 0 otherwise. $\frac{k_i k_j}{2m}$ is the expected number of edges between node i and j given a random graph with the same degree distribution [18].

Another model which can be used for community detection is the Bernoulli-Poisson model. Here, it is assumed that the number of edges between communities follows a Poisson distribution, and that the probability of there being an edge between node i and j is generated by a Bernoulli distribution.

$$A_{ij} \sim \text{Bernoulli}(1 - \exp\left(-F_i F_j^T\right)) \tag{2.2}$$

$$l(F) = \sum_{(i,j) \in E} \log\left(1 - \exp\left(-F_i F_j^T\right)\right) - \sum_{(i,j) \notin E} F_i F_j^T \tag{2.3}$$

Mathematically, this can be expressed as the adjacency matrix entries being sampled according to Eq. (2.2), where $F \in \mathbb{R}^{N \times C}$ denotes the affiliation matrix, N is the number of nodes, and C is the number of communities. Scalar

F_{ic} contains a non-negative number denoting the affiliation between node i and community c. Given the graph $G = (V, E)$, one can find the affiliation matrix F by maximizing the likelihood $l(F)$ calculated using Eq. (2.3) [20].

Many other community detection algorithms also exist. However, only modularity and Bernoulli-Poisson need to be mentioned since all models covered in Subsection 2.6.3 are based on these algorithms.

2.3 Framework for microservice identification

From the definitions established in Section 2.1, a connection to graphs can easily be made. Microservices are processes which communicate with each other. In the context of graphs, a microservice could be represented as a node, and its communication links could be represented as edges. Similarly, a monolithic system can also be represented as a graph, with source code files being nodes and connections between files being edges. The problem of microservice identification can therefore be simplified to extracting the graph representation of the microservice architecture from the monolithic graph representation. Although the architecture is dissimilar, the source code roughly remains the same. One can therefore simply cluster source code nodes according to their edge connections and business functionality in order to generate the microservice graph. Which makes the problem of microservice identification a community detection problem.

As was briefly discussed in Section 1.1, deep learning-based community detection has seen a recent influx of research, and these techniques seem particularly suited for the task of microservice identification. However, for these techniques to be applicable to microservice identification, a couple of criteria need to be upheld: (1) training is *unsupervised*, (2) the graph used as input is *attributed*, and (3) the detected communities can be *overlapping*. These three criteria are further expanded on in the listed order below:

- *Unsupervised*: When identifying microservices from a monolithic system, it can be assumed that no prior knowledge exists, and that the only available information is a graphical representation of the monolithic system. Because of this, the community detection method needs to be unsupervised. In some cases a semi-supervised approach could be applied, such as some microservices have already been

manually identified. However, unsupervised models are more general since they can cover all use cases.

- *Attributed*: An attributed graph is simply a graph in which each node is associated with a set of attributes. These attributes can contain information which is helpful for identifying similarities. Making use of this information is necessary in order to uphold the bounded context criteria discussed in Section 2.1.
- *Overlap*: When decoupling a monolithic system into microservices, it is often the case that some utility and entity classes need to be duplicated due to having strong coupling with several microservices. Further, microservices need to communicate by sending data between each other. This data is usually represented as a Data Transfer Object (DTO) [21]. These DTOs therefore need to be shared across microservices, which causes inherent overlap. This needs to be considered when identifying microservices, and the model therefore needs the ability to detect overlapping communities.

2.4 Evaluation metrics

Microservices need to uphold three criteria: (1) bounded context, (2) independence, and (3) size, as was mentioned in Section 1.1. The size criteria is trivial to measure, however the first two criteria are non-trivial. Previous research [22] has constructed metrics for evaluating just these two criteria. Bounded context can be evaluated using *Interface Number* (IFN), *Cohesion at Message Level* (CHM), and *Cohesion at Domain Level* (CHD). Independence can be meassured using *Structural Modularity Quality* (SMQ), and *Conceptual Modularity Quality* (CMQ). For all of the below described equations, N is the total number of microservices.

$$\text{IFN} = \frac{1}{N}\sum_{i=1}^{N} ifn_i \tag{2.4}$$

IFN, which equation is described by Eq. (2.4), simply measures the number of interfaces within a microservice. IFN is the average of all ifn_i, where ifn_i represents the number of interfaces within microservice i. A lower IFN means that it is more probable that each microservice has a single responsibility.

$$\text{CHM} = \frac{1}{N}\sum_{i=1}^{N} chm_i \tag{2.5}$$

CHM aims to measure cohesion by looking at the message level. Again, CHM is the average of all chm_i, where chm_i measures the cohesion of microservice i, as seen in Eq. (2.5). A higher CHM means better cohesion within the microservices.

$$chm_i = \begin{cases} \frac{\sum_{(k,m)} f_{msg}(opr_k, opr_m)}{\frac{1}{2}|O_i| \times (|O_i|-1)} & \text{if } |O_i| \neq 1 \\ 1 & \text{if } |O_i| = 1 \end{cases} \tag{2.6}$$

Each chm_i is calculated according to Eq. (2.6). opr_k and opr_m represent the set of all operations within interface k and m, both of which reside in microservice i.

$$f_{msg}(opr_k, opr_m) = \frac{\left|\frac{ret_k \cap ret_m}{ret_k \cup ret_m}\right| + \left|\frac{par_k \cap par_m}{par_k \cup par_m}\right|}{2} \tag{2.7}$$

The function $f_{msg}(opr_k, opr_m)$ calculates the Jaccard index as shown in Eq. (2.7). This function measures similarity between interfaces, which is done by comparing return values (ret_k and ret_m) and parameter values (par_k and par_m).

$$\text{CHD} = \frac{1}{N}\sum_{i=1}^{N} chd_i \tag{2.8}$$

CHD is very similar to CHM, as it also aims to measure cohesion, but at the domain level. The average is calulated according to Eq. (2.8). Just as with CHM, a higher CHD indicates that the microservices are more cohesive.

$$chd_i = \begin{cases} \frac{\sum_{(k,m)} f_{dom}(opr_k, opr_m)}{\frac{1}{2}|O_i| \times (|O_i|-1)} & \text{if } |O_i| \neq 1 \\ 1 & \text{if } |O_i| = 1 \end{cases} \tag{2.9}$$

Eq. (2.9) is identical to Eq. (2.6), with the only exception being that $f_{dom}(opr_k, opr_m)$ is used instead of $f_{msg}(opr_k, opr_m)$.

$$f_{dom}(opr_k, opr_m) = \frac{|f_{term}(opr_r) \cap f_{term}(opr_m)|}{|f_{term}(opr_r) \cup f_{term}(opr_m)|} \tag{2.10}$$

The function $f_{dom}(opr_k, opr_m)$ is calculated according to Eq. (2.10). It is very similar to $f_{msg}(opr_k, opr_m)$ with the difference being that all domain terms are used in order to calculate the similarity, not only message terms.

$$\text{SMQ} = \frac{1}{N} \sum_{i=1}^{N} scoh_i - \frac{1}{\frac{N(N-1)}{2}} \sum_{i \neq j}^{N} scop_{i,j} \tag{2.11}$$

SMQ measures modularity by looking at the structure of the microservice. This is done according to Eq. (2.11), where $scoh_i$ aims to measure cohesiveness within microservice i, and $scop_{i,j}$ aims to measure coupling between microservice i and j.

$$scoh_i = \frac{\mu_i}{N_i^2}, \quad scop_{i,j} = \frac{\sigma_{i,j}}{2(N_i \times N_j)} \tag{2.12}$$

Both of these values are in turn calculated according to Eq. (2.12). N_i and N_j are the number of classes within microservice i and j, μ_i is the total number of edges within microservice i, and $\sigma_{i,j}$ measures the total number of edges between microservice i and j. A higher SMQ indicate better modularization, which indicate that the microservices are more independent.

$$\text{CMQ} = \frac{1}{N}\sum_{i=1}^{N} ccoh_i - \frac{1}{\frac{N(N-1)}{2}}\sum_{i\neq j}^{N} ccop_{i,j} \tag{2.13}$$

$$ccoh_i = \frac{\mu_i}{N_i^2}, \quad ccop_{i,j} = \frac{\sigma_{i,j}}{2(N_i \times N_j)} \tag{2.14}$$

CMQ also measures modularity, but it is measured from a conceptual viewpoint. It is very similar to SMQ, with both Eq. (2.13) and Eq. (2.14) being identical to Eq. (2.11) and Eq. (2.12). The only difference is how μ_i and $\sigma_{i,j}$ are determined. They still represent the number of edges within and between microservices respectively. However, instead of looking at edges within the graph structure, an edge is identified between interface k and m if the intersection of domain terms within these interfaces is not empty. Just as with SMQ, a higher CMQ is better.

2.5 Graph neural networks

Graph Neural Networks (GNNs) [23, 24], as the name suggests, are a class of neural networks which has been designed to operate on graph-structured data. There exists several different types of GNNs, such as Graph Convolutional Networks (GCNs) [25], Graph Autoencoders (GAEs) [26], and Graph Attention Networks (GATs) [27]. These GNN models can largely be divided into two main categories: (1) spectral-based models [28, 25], and (2) spatial-based models [27, 29]. These two categories are further explained in Subsection 2.5.1, and 2.5.2, respectively.

2.5.1 Spectral-based methods

Spectral-based models operate on the Laplacian matrix, which is a matrix representation of the graph. This is useful since it allows for convolution-like operations on the graph. GCN is one example of a spectral-based model. Generally, spectral-based methods involve two steps: the graph convolution, followed by an activation function. The graph convolution operation functions by taking the linear combination of each node and its neighbours using a shared weight matrix, which results in a new feature representation. This feature representation is then passed through an activation functions which is required to be non-linear [25].

Formally, let $G = (V, E)$ be a graph with N nodes and F features for each node. Further, let $X \in \mathbb{R}^{N \times F}$ be a feature matrix for G, $A \in \mathbb{R}^{N \times N}$ be the adjacency matrix for G, and D be the diagonal node degree matrix, whose i-th diagonal element is the degree of the i-th node. The layer-wise propagation rule can then be defined as:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{2.15}$$

In Eq. (2.15), $W^{(l)}$ represents the weight matrix for the l-th layer. $H^{(l)} \in \mathbb{R}^{N \times F}$ is the feature representation for the l-th layer, with $H^{(0)} = X$. And $\sigma(\cdot)$ is the non-linear activation function. $\tilde{A} = A + I_N$ is the adjacency matrix with added self-connections, with I_N denoting the identity matrix. This is done in order to also take the feature vector of node i into consideration when performing the matrix multiplication. If no self-connections are added, then only the feature vectors of neighbours to node i are considered. Further, the adjacency matrix is symmetrically normalized using $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ where $\tilde{D}$ is the diagonal node degree matrix of $\tilde{A}$ [25].

2.5.2 Spatial-based methods

Spatial-based models operate directly on the graph structure, which is done by passing messages between adjacent nodes. GAT is an example of a spatial-based model. The two steps of spatial-based methods are: message passing and attention. In the message passing phase each node sends a message containing its own feature vector to all of its neighbours. Then, in the attention phase, the linear combination of each node and its neighbours is calculated using a shared weight matrix. Here, the weights are learned using a shared attentional mechanism [27].

Formally, using the same definitions as those described in Subsection 2.5.1, the output representation is calculated as follow:

$$\vec{h}_i' = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j \in N_i} \alpha_{ij}^k W^k \vec{h}_j\right) \tag{2.16}$$

$$\alpha_{ij} = \text{softmax}_j\left(a(W\vec{h}_i, W\vec{h}_j)\right) \tag{2.17}$$

In Eq. (2.16), K independent attention mechanisms are calculated, and their features are then concatenated, which is represented by $\|$. This is done in order to stabilize the learning process. α_{ij} are normalized attention coefficients, which in turn are calculated according to Eq. (2.17). Attention coefficients are only calculated for $j \in N_i$ where N_i is the set of first-order neighbours of i. $a : \mathbb{R}^F \times \mathbb{R}^F \rightarrow \mathbb{R}$ is the shared attentional mechanism. In both equations, W represents the weight matrix, and $\vec{h}_i$ represents the feature vector for node i. $\sigma(\cdot)$ is the activation function [27].

2.6 Related work

Much work has already been done both in the field of automatic microservice identification, and in the field of deep learning-based community detection. This section aims to review some of the related works in these areas. First of, in Subsection 2.6.1, the previously proposed methods of automatic microservice identification are covered. Most of these methods aim to generate a weighted graph representation of the code in order to perform clustering on the graph representation. Only the preprocessing steps discussed by these papers are relevant to this thesis, since a deep learning-based model is utilized instead of clustering. And not all preprocessing techniques discussed by the papers are utilized. They are however still presented as they are relevant to the general topic of automatic microservice identification. Subsequently, in Subsection 2.6.2, one preprocessing step in particular is covered in more detail due to its relevance to this thesis. This section then ends with Subsection 2.6.3, covering some deep learning-based community detection models. Only models which entirely fulfill (or almost entirely fulfill) the properties outlined in Section 2.3 are presented. Only one of the presented models is actually utilized. This choice, and the basis for this choice, is presented in Section 2.7.

2.6.1 Automatic microservice identification

Various methods of automatic microservice identification have been proposed throughout a handful of different research papers. However, to the knowledge of the author, none of these current approaches have explored the utilization of deep learning for microservice identification. Gysel et al. [5] uses a clustering based approach. By mapping software engineering artifacts on an undirected weighted graph, they can use weight-based clustering algorithms to decompose the graph into a set of microservices. Baresi et al. [6] expand on the previous research by instead performing clustering based

on the semantic similarity of OpenAPI interface specifications. Abdullah et al. [7] instead focus on the scalability and performance aspects of microservices. They use a scale weighted k-means clustering algorithm to find URL partitions with similar performance requirements, and these are then mapped to microservices. Brito et al. [8] utilize topic modelling to identify latent semantic structures from the source code. They use Latent Dirichlet Allocation (LDA) [30] to categorise the source code according to topics, which is then used together with structural dependencies to generate an undirected weighted graph. A clustering algorithm can then be used on this graph to generate microservices. At the time of writing this thesis, the most recent approach is by Trabelsi et al. [9]. Their approach consists of three phases: (1) class typing, (2) typed-service identification, and (3) services to microservices mapping. In the first phase (1) they label classes as either *Application*, *Entity*, or *Utility* according to whether the class pertains to business logic, database storage, or utility functionality respectively. This classification is done using a supervised model. During the second phase (2) they use the Louvain community detection algorithm [19] to cluster classes into services, this is done independently for each type of class. In the third phase (3), these services are then further clustered into microservices by using static and semantic weights together with the fuzzy C-means algorithm [31]. They preprocess the source code by generating a call graph [32] and a feature matrix which is exactly the type of data which is expected by a graph neural network. In particular, the feature matrix was generated using a pretrained CodeBERT model [33]. This model is further discussed in Subsection 2.6.2.

2.6.2 CodeBERT

CodeBERT is a transformer-based model which has been pretrained for downstream natural language and programming language applications. It has been shown to achieve state-of-the-art performance. The model has been trained partly on bimodal datapoints consisting of code functions with paired documentation, and partly on unimodal datapoints consisting of code functions with no linked documentation. For training, the dataset used included code from six different programming languages, importantly this set of programming languages included Java. The output of the model includes a feature embedding of the inputted source code, from which a feature matrix can be generated for downstream tasks [33]. In this thesis project, the downstream task is microservice identification. The models discussed in

Table 2.2: Related work in terms of desirable community detection properties for the task of microservice identification. If the property is present it is marked with a check mark (✓), if it is absent it is marked with a cross (✗).

Method	Unsupervised	Attributed	Overlap
NOCD [12]	✓	✓	✓
DMoN [34]	✓	✓	✗
UCoDe [35]	✓	✓	✓
SSGCAE [36]	✗	✓	✓

Subsection 2.6.3 all required an attribute matrix as input, and CodeBERT is an excellent model for generating such a matrix.

2.6.3 Deep learning-based community detection

The field of deep learning has seen a significant growth throughout recent years. This has in turn extended to community detection, where several deep learning-based techniques for community detection have been developed [10, 11]. Luckily, the three criteria discussed in Section 2.3 significantly narrow down the number of deep learning-based community detection methods which need to be considered.

Four methods, outlined in Table 2.2, have been identified as potential candidates for the task of automatic microservice detection. All of these models use some form of GNN architecture due to its suitability for attributed graph learning. Shchur & Günnemann [12] propose the Neural Overlapping Community Detection (NOCD) model, which uses a GCN architecture. This model finds overlapping communities by maximizing the negative log-likelihood of the graph reconstruction generated by a Bernoulli-Poisson (BP) model [20]. This structure is optimized for detecting overlapping communities, and therefore does not allow for the direct detection of disjoint (non-overlapping) communities. However, for microservice identification, this is a nonissue due to the inherent overlap discussed earlier. Even though the loss function has quadratic complexity, the model achieves excellent scalability by subsampling the graph, and it further shows that this converges to the same solution. Tsitsulin et al. [34] introduce Deep Modularity Networks (DMoN). Similarly to the previous method, DMoN is an unsupervised method utilizing a GCN architecture. Where the methods differ is that DMoN uses a modularity-based [18] loss function. This method outputs soft cluster

assignments which means that it can be used for overlapping community detection. However, even though the model can be used for overlapping community detection, it is not optimized for this task. The loss function which is used encourage strong connections between nodes within a community, and weak connection to any node outside of the community. This is not the case for overlapping communities, since there exists a non-weak connection between two communities which share overlapping nodes. This results in a loss function that penalizes overlapping communities. DMoN is therefore better suited for non-overlapping community detection. Moradan et al. [35] proposes Unified Community Detection (UCoDe). UCoDe is similar to DMoN in that it uses a GCN architecture with a modularity-based loss. Where the model differs is that it also introduces a contrastive loss which prevents it from penalizing overlap, allowing the model to detect non-overlapping and overlapping communities alike. Moradan et al. compares UCoDe to both NOCD and DMoN using two different quality measures: (1) Normalized Mutual Information (NMI) [37], and (2) Recall. From the evaluation it is shown that NOCD generally performs better according to the NMI metric, while UCode achieves better results on recall. He et al. [36] proposes a Semi-supervised Graph Convolutional Autoencoder (SSGCAE) for the task of overlapping community detection. The graph convolutional autoencoder is used to learn a latent representation. Then, similarly to previous methods, modularity maximization is used on the reconstruction in order to detect overlapping communities. Semi-supervision is used in order to boost performance by using prior knowledge of already existing communities. This semi-supervision step can however be removed with only slight modifications, resulting in an unsupervised method.

2.7 Summary

The topic of automatic microservice identification is not new, and much research has been done as there exists a plethora of previously proposed approaches. In a similar vein, many deep learning-based methods for community detection have been proposed. These two fields have however not seen overlap, which this thesis aims to rectify.

To utilize deep learning-based methods, an attributed graph is needed. Previous research of automatic microservice identification [9] has already discussed ways of generating attributed graphs from source code. A graph structure can simply be generated using a call graph [32] or class diagram,

Table 2.3: Benefits and drawbacks of the discussed deep learning-based community detection models, in relation to their ability to detect microservice-like communities. Beneficial features are marked with a plus sign (+), while drawback are marked with a minus sign (-).

Method	Benefits & Drawbacks
NOCD [12]	+ Optimized for overlapping community detection + Subquadratic sacalability - Subsamples graph during training, potential of information loss
DMoN [34]	+ Maintains scalability without subsampling - Optimized for non-overlapping community detection
UCoDe [35]	+ Handles both overlapping and non-overlapping community detection - Quadratic time complexity
SSGCAE [36]	+ Can utilize prior information to learn better community structures - Designed as a semi-supervised method - Quadratic time complexity

and an accompanying attribute matrix can be generated with models such as CodeBERT [33]. A deep learning-based model need to uphold three criteria in order to be useful for microservice identification, as discussed in Subsection 2.3 and 2.6.3. Four such models were identified. Of these models, the NOCD model is utilized for this thesis project. Some benefits and drawbacks of each model are presented in Table 2.3. The NOCD model was chosen since it upholds all necessary criteria while maintaining excellent scalability. One of its drawbacks is that subsampling is done in order to achieve this scalability. However, it is shown in the paper that the sacrifice of information from this subsampling is negligible, and that the model still converges to the same solution.

Chapter 3

Methods

This thesis has been split up into three separate research phases: (1) *Design*, (2) *Development*, and (3) *Evaluation*. These phases correspond to the activities outlined by the design science paradigm. Each phase has been further split into sub-phases, which is further elaborated upon throughout Section 3.1-3.3. An overview of the thesis' research phases is also presented in Figure 3.1.

3.1 Design

The design phase of the project was concerned with the overall architecture of the microservice identification model. It was during this phase that decisions about preprocessing steps and community detection models were taken. These decisions were largely based on the knowledge gained throughout the literature study. Overall, the final model can be seen as having two preprocessing steps followed by a training step. For preprocessing, first an adjacency matrix is generated from a class diagram of the source code. Then, an attribute matrix is generated by feeding the source code to a pretrained CodeBERT model. As the final step, the adjacency and attribute matrices are fed to a NOCD model in order to identify microservice candidates. These steps are explained in much more detail throughout Chapter 3.

3.2 Development

The development phase followed roughly the same sub-phases as the design phase, but with an added sub-phase for the implementation of evaluation

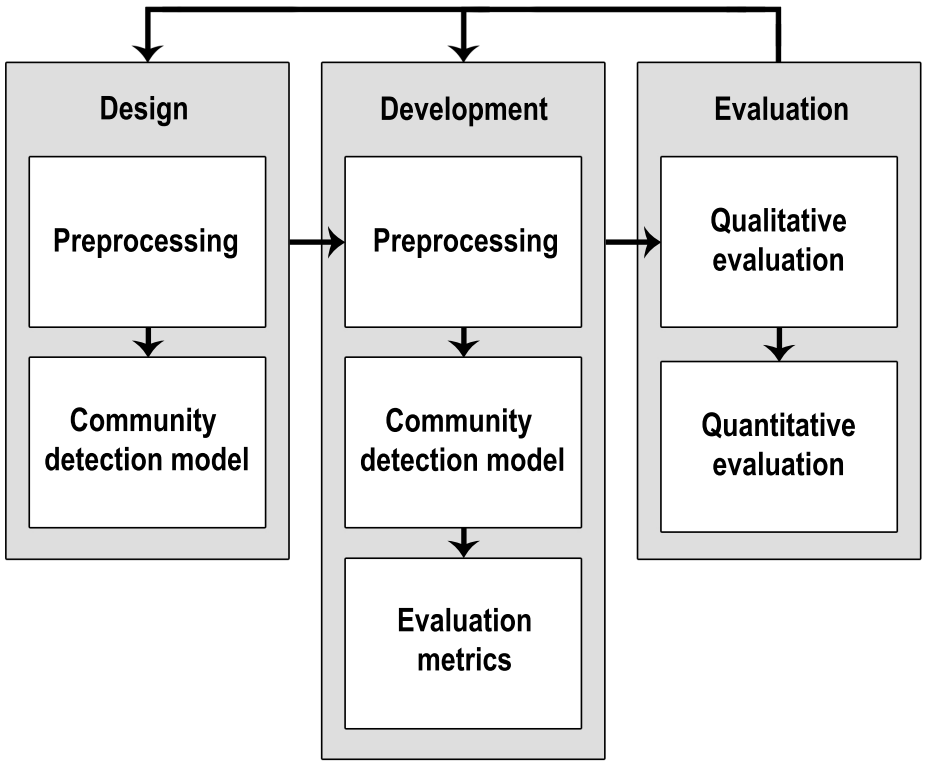


Figure 3.1: The thesis' research phases

metrics. This phase, as well as the design phase, were iterative processes in that sub-phases could be revisited in order to improve the model performance in accordance with the later evaluation phase. This section covers the proposed approach in more detail throughout Subsection 3.2.1-3.2.3. First, the hardware and software used is summarised in Section 3.2.1. Then, in Section 3.2.2, the preprocessing steps used are summarised. Finally, in Section 3.2.3, the architecture of the deep learning model is described. This includes the parameters used, the loss function, and the training loop.

3.2.1 Hardware and software

All experiments were performed on a computer running Windows 11 22H2 with an AMD Ryzen 5600X CPU, 16GB of RAM, and an RTX 3060 Ti GPU. All code was implemented using Python 3.10. Training was performed with PyTorch 1.13.0 using CUDA 11.7. The UML class diagrams were extracted using IntelliJ IDEA 2022.2.1 using the bundled Diagrams plugin.

3.2.2 Data preprocessing

The deep learning model used requires two inputs: (1) an adjacency matrix $A \in \{0,1\}^{N \times N}$, where N is the number of nodes, and (2) an attribute matrix $X \in \mathbb{R}^{N \times D}$, where D is the number of attributes. How these matrices are generated is further discussed in Subsection 3.2.2.1 and 3.2.2.2 respectively.

3.2.2.1 Adjacency matrix generation

In order to generate the adjacency matrix, a class diagram of the source code was first generated. This was done using the built-in Diagrams plugin for the IntelliJ IDEA *. The plugin generates a UML class diagram, which can then be exported to a GraphML file, an XML-based file format used for representing graphs. This GraphML file can then be read in Python using NetworkX †, a Python package for network and graph analysis. This package has a built-in function for generating an adjacency matrix from a graph, which was used.

3.2.2.2 Attribute matrix generation

As discussed in Subsection 2.6.2, a pretrained CodeBERT model was used in order to generate the attribute matrix. For each source code file, it was tokenized and then fed into the CodeBERT model. The output for each file is an attribute vector of size $D = 768$. These outputs are then aggregated in order to generate the attribute matrix X.

3.2.3 Model architecture

For model architecture, the NOCD model was chosen, as discussed in Section 2.7. The NOCD model utilizes a GCN architecture. It consists of two layers, the hidden layer has a size of 128, and the output layer has a size of C (the number of communities to detect). Batch normalization is performed after the first convolutional layer. ReLU is used as the activation function of each layer. Dropout rate is set to 0.5 for all layers. Weight decay is used with the regularization strength set to 10^{-2}. The Adam optimizer [38] is used with a learning rate set to 10^{-3}. A batch size of 2000 is used. The model trains for 500 epochs, early stopping is implemented with a patience of 10 [12].

*JetBrains, "UML class diagrams.". Available: https://www.jetbrains.com/help/idea/class-diagram.html

†NetworkX. Available: https://networkx.org/

The loss function is based on the Bernoulli–Poisson model, which was discussed in Section 2.2. In Section 2.2, Eq. (2.3), the negative log-likelihood of the Bernoulli–Poisson model was outlined. Real-world graphs have a tendency to be sparse, this is also true for UML class diagrams generated from source code. The second term of Eq. (2.3) therefore has a bigger effect on the loss. To counteract this both terms need to be balanced, resulting in the following loss function:

$$\mathbf{F} \coloneqq \text{GNN}_\theta(A, X) \tag{3.1}$$

$$\mathcal{L}(\mathbf{F}) = -\mathbb{E}_{(i,j)\sim P_E}\left[\log\left(1 - \exp\left(-\mathbf{F}_i\mathbf{F}_j^T\right)\right)\right] + \mathbb{E}_{(i,j)\sim P_N}\left[\mathbf{F}_i\mathbf{F}_j^T\right] \tag{3.2}$$

Eq. (3.2) uses the same definitions as those used by Eq. (2.3). That is to say, $\mathbf{F} \in \mathbb{R}^{N\times C}$ denotes the affiliation matrix, which is also the output of the GNN, as stated by Eq. (3.1). Here, N is the number of nodes, and C is the number of communities. Scalar $\mathbf{F}_{ic}$ contains a non-negative number denoting the affiliation between node i and community c, i.e. the probability that node i belongs to community c. P_E denotes a uniform distribution over edges, while P_N denotes a uniform distribution over non-edges. This is equivalent to adding weights N_E and N_N to the first and second term of Eq. (2.3), where N_E is the number of edges, and N_N is the number of nodes. The affiliation matrix $\mathbf{F}$ is then found by optimizing the parameters $\boldsymbol{\theta}^*$ of the GCN, which is done by minimizing the loss function according to Eq. (3.3).

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{F}) \tag{3.3}$$

The number of communities C was chosen in accordance to the number of published interfaces in each system. This was done with the assumption that each published interface roughly corresponds to its own business functionality. For PE Accounting, $C = 67$, and for PetClinic, $C = 4$. A threshold T is used in order to determine whether node i should be assigned to community c. This threshold was initially chosen by looking at the distribution of the non-zero entries of $\mathbf{F}$. This choice was later corroborated by performing hyperparamter tuning of T with increments of 0.1. For PE Accounting, $T = 0.5$, and for PetClinic, $T = 0.2$.

3.3 Evaluation

Evaluation of the developed model was performed both qualitatively and quantitatively. These two evaluation phases are discussed in Subsection 3.3.1 and 3.3.2 respectively.

3.3.1 Qualitative evaluation

The nature of what makes a good microservice is somewhat subjective. A qualitative evaluation of the identified microservices was therefore conducted. This evaluation was based on whether the microservice groupings made sense from a business functionality perspective. The model was implemented and evaluated for two different monolithic systems: (1) *PE Accounting*, and (2) *PetClinic*.

PE Accounting, the commissioning company, provided their own source code as a basis for this thesis project. In conjunction with the project, PE Accounting has simultaneously been working on manually extracting microservices from their monolithic system. A partial ground truth is therefore available, allowing for comparison between the microservices detected by the deep learning model to those identified by the company. Evaluation is done partly by the author using knowledge gained from the literature study, and partly using expert knowledge provided by developers at the company.

PetClinic * is a Java-based open-source project. The project has both a monolithic and a microservice-based implementation, which means that a complete ground truth is available. The evaluation of model performance on this system was done solely by the author, utilizing knowledge gained throughout the literature study, and insights gained from evaluation of the prior system.

3.3.2 Quantitative evaluation

Quantitative evaluation was done in order to measure how well each identified microservice upheld the three principles discussed in Section 1.1. As mentioned in Section 2.4, previous research [22] has constructed metrics for evaluating the bounded context and independence criteria of microservices. Bounded context can be evaluated using IFN, CHM, and CHD. Independence can be meassured using SMQ, and CMQ. Size can more easily be measured

*The Spring Petclinic Community. Available: https://spring-petclinic.github.io/

by taking the number of source code files assigned to each microservice. All of these metrics are reported in Chapter 4 for both the PE Accounting and PetClinic datasets. Trabelsi et al. [9] have also report these scores for their and two other state-of-the-art microervice identification solutions. The scores produced by the deep learning model are compared to these and evaluated using statistical hypothesis testing.

Chapter 4

Results and analysis

Throughout this chapter, results from both the qualitative and quantitative evaluations are presented. This is done in Section 4.1 and 4.2 respectively. In both sections, an analysis of the results is performed. The results are then further deliberated upon in Chapter 5.

4.1 Qualitative evaluation

The qualitative evaluation consisted of a manual inspection of the identified microservices. This was done for two different systems: (1) PE Accounting, and (2) PetClinic. Both systems are written in Java. However, they differ vastly in size. PE Accounting is a large-scale system consisting of thousands of classes. On the contrary, PetClinic is a very small system consisting of only 23 classes. For both systems, the manual inspection revealed that the model in fact could identify relevant microservices, and that the classes within any individual microservice were generally confined to a singly business functionality.

It was noted that a good value for the threshold was especially important for identifying good microservices. Since the NOCD model can assign one class to several microservices, having a low value for the threshold resulted in several utility classes appearing in every single microservice. This made the microservices bloated, and increasing the threshold to a reasonable value lowered the average size of a microservice from roughly 300 to around 40 in the PE Accounting system. This size corresponds well to the average size of the microservices which were manually identified by the company. A

good threshold value was found using hyperparameter tuning. This proved to be more difficult for the PetClinic system. This was due to the much lower number of microservices, which resulted in a much lower threshold value being required. Having too high a threshold value could result in some interface classes not being assigned to any microservice, causing high variance in the results. This made the tuning difficult Since a choice between consistency versus sometimes achieving better results had to be made. In the end, a lower threshold was deemed necessary. Some classes, such as `BaseEntity` and `NamedEntity` appeared in all identified microservices. This is reasonable since every single domain object in the PetClinic system inherited from these classes. Due to this, the average size of a microservice identified from the PetClinic system was 9, which is quite large when taking into consideration that four microservices were identified, and that the total system consists of only 23 classes.

A high percentage of microservices were correctly identified. However, some microservices clearly did not uphold the three core principles outlined in Section 1.1. For the incorrectly identified microservices, these spanned across several business functionalities, resulting in a lack of bounded context, and unproportionately large microservices. These incorrectly identified microservices were generally easy to spot since they often were double or quadruple the size of the correctly identified microservices. Trabelsi et al. [9] reported similar findings for their proposed approach, MicroMiner. In that most microservices were correctly identified, while some spanned several business functionalities. Analyzing only the qualitative results, the deep learning-based method performs similarly to previously proposed approaches such as MicroMiner.

4.2 Quantitative evaluation

For the quantitative evaluation, five metrics were used: IFN, CHM, CHD, SMQ, and CMQ. The details of these metrics were discussed in both Section 2.4 and Subsection 3.3.2. As a reminder, IFN, CHM, and CHD are used to evaluate bounded context, SMQ and CMQ are used to evaluate independence. These metrics are presented in Table 4.1 and 4.2 for both the PE Accounting and PetClinic systems. It is worth repeating that a lower IFN is better, while a higher CHM, CHD, SMQ, and CMQ is better.

Loss is also reported for both the PE Accounting and PetClinic systems, as can be seen in Figure 4.1. One can see a much clearer downward slope in the

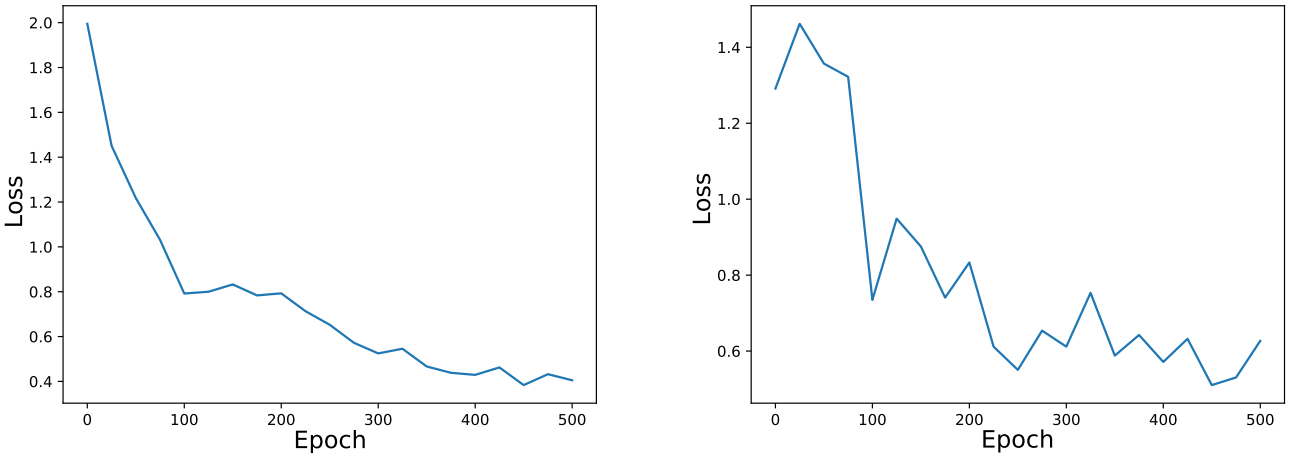
Table 4.1: Evaluation metrics IFN, CHM, and CHD. The NOCD model was also trained without an attribute matrix, this approach is marked with (†).

System	Approach	IFN	CHM	CHD
PE Accounting	NOCD	**2.1±0.54**	0.68±0.19	**0.69±0.21**
	NOCD†	8.6±3.43	0.48±0.22	0.48±0.21
	MicroMiner	2.6±0.80	**0.72±0.19**	0.66±0.17
PetClinic	NOCD	2.0±0.63	0.72±0.20	0.67±0.18
	NOCD†	**1.5±0.92**	**0.75±0.27**	0.75±0.20
	MicroMiner	1.8±0.70	0.74±0.19	**0.77±0.32**

Table 4.2: Evaluation metrics SMQ and CMQ. The NOCD model was also trained without an attribute matrix, this approach is marked with (†).

System	Approach	SMQ	CMQ
PE Accounting	NOCD	**0.03±0.009**	0.02±0.013
	NOCD†	-0.01±0.014	-0.01±0.014
	MicroMiner	0.02±0.019	**0.03±0.016**
PetClinic	NOCD	0.04±0.011	**0.03±0.011**
	NOCD†	0.03±0.022	0.01±0.014
	MicroMiner	**0.05±0.013**	**0.03±0.010**

loss when training on the PE Accounting system, compared to the volatility observed in the loss when training on the PetClinic system. This indicate that the model cannot learn as well when it comes to the PetClinic system. It is assumed that this stems from the small sample size and that the NOCD model can not learn relevant semantic relationships between classes due to the system being too small. Due to this observation, for the PetClinic system, the NOCD model was also trained without an accompanying attribute matrix. As can be seen in Table 4.1 and 4.2, this results in a slight increase of the bounded context and a slight decrease of independence for the identified microservices. The average size of the identified microservices remained the same when excluding the attribute matrix. No results were significantly different, which supports the idea that the attribute matrix did not have a significant effect on results for the PetClinic system. Running similar experiments on the PE Accounting system showed a much more obvious decrease in performance, as can clearly be seen from the IFN reported in Table 4.1.



(a) PE Accounting

(b) PetClinic

Figure 4.1: Training loss over number of epochs

Table 4.3: p-values calculated for each evaluation metric using a sample size of 10, a separate null hypothesis test was performed for each metric. For each test, $H_0 : \mu \leq \mu_0$, while $H_1 : \mu > \mu_0$ (except for IFN, where $H_0 : \mu \geq \mu_0$, while $H_1 : \mu < \mu_0$).

	IFN	**CHM**	**CHD**	**SMQ**	**CMQ**
p-value	0.14	0.48	0.15	0.07	0.07

In order to compare the results of the NOCD model to those reported for MicroMiner, null hypothesis testing was performed. This was done using the nonparametric Mann–Whitney U test. This test was chosen since the sample distribution is unknown.

The null hypothesis and the alternative hypothesis are outlined below:

Null Hypothesis: *The NOCD model does not identify better microservices than the existing MicroMiner model.*

Alternative Hypothesis: *The NOCD model identifies better microservices than the existing MicroMiner model.*

In order to test this overarching hypothesis, a separate null hypothesis test was performed for each evaluation metric. The p-values calculated from each of these tests are presented in Table 4.3. These tests used a somewhat

small sample size of 10 due to the MicroMiner approach taking a long time to train. Although different approaches exist for combining p-values from different hypothesis tests, in this case it is enough to see that for each test, the null hypothesis can not be rejected given a significance level of $\alpha = 0.05$. Therefore the overarching null hypothesis can not be rejected either. Thus, from analyzing the quantitative evaluation, it can not be stated that the deep learning-based method outperforms previously proposed approaches such as MicroMiner.

Chapter 5

Discussion

This chapter focuses on discussion around the results presented in Chapter 4. The main hypothesis which was outlined at the beginning of the thesis project was that a deep learning-based approach would perform better compared to previous approaches for the automatic identification of microservices. Contrary to this hypothesis, the results did not show improved performance. Instead, the results indicated that the deep learning-based method performed similarly to MicroMiner, a previously proposed approach which had been identified as current state-of-the-art. Although the results did not show direct improvement, it is still argued in Section 5.1 that the results are in support of using deep learning for the task of microservice identification. In Section 5.2, some reflections about strengths, weaknesses, and limitations of the thesis work are discussed. Then, in Section 5.3, the contributions and impact of the thesis work is stated. In Section 5.4, some deliberation of ethical requirements are made. Finally, in Section 5.5, the thesis work is discussed from a sustainability perspective.

5.1 Key findings

As mentioned, the results indicate that the deep learning model perform similarly to current state-of-the-art approaches such as MicroMiner. In this regard, it is worth mentioning that the deep learning based method proposed by this thesis does not need any prior knowledge of the monolithic system which it is being run on. The model can be run on several different systems with zero changes to its parameters. This means that microservices can be identified while having no prior information about the system. On the other

hand, MicroMiner requires manual adjustments of parameters in order to tell the model how it should balance the static relationships in comparison to the semantic analysis. This manual adjustment of parameters require knowledge about the system which the model is being run on. It is not a large manual step, however, it is a step which can be bypassed without affecting results by utilizing the deep learning-based approach.

One finding which was made during the qualitative evaluation of the model was that it performed worse for the PetClinic system, which was the smaller of the two systems. Although the model could still identify relevant microservices, it could not do this consistently. The results had high variance due to the incredibly small number of classes being trained on. This finding is further supported by the quantitative evaluation, which showed that removing the attribute matrix had close to zero effect on the results for the PetClinic system, indicating that the model could not properly learn semantic relationships between classes. The worse performance on the PetClinic system can also be further explained by the choice of loss function and the use of a GNN. Due to the inherent structure of GNNs, neighbouring nodes often output similar community affiliation vectors. In larger sparse systems, this leads to an improvement in performance. However, for the smaller more densely connected PetClinic system, this resulted in the model sometimes outputting four microservices with a lot of overlap. On top of this, the loss function used is based on the Bernoulli–Poisson model. This loss function encourages overlap between communities. This further encouraged the system to output very similar microservices for PetClinic. None of these issues affected the PE Accounting system due to it being much larger in size and more sparsely connected. Overall, these issues are most likely not a problem for practical applications. It is often easy to manually identify microservices for small systems. An automatic approach is therefore probably not relevant unless the system is truly monolithic. Thus, for the intended use case, the model should perform consistently well. A loss function based on modularity optimization may be able to produce better results for more densely connected systems, since this type of loss function actively discourages overlapping communities. However, as was outlined in Section 2.3, this discouragement is not optimal for systems with a similar architecture to PE Accounting. Nonetheless, it is still an avenue worth exploring as it was outside the scope of this thesis. This is further discussed in Section 6.2.

Another finding which was made during the qualitative evaluation was that some microservices spanned several business functionalities. These

incorrectly identified microservices were usually easy to spot due to being larger in size. However, it would of course be better if no such microservices were identified in the first place. Although the previous MicroMiner solution also reported similar issues, Trabelsi et al. [9] attempted to mitigate this problem by first identifying core interface classes which their microservices were then built around. A similar approach could perhaps have been taken by the deep learning model as well, this is further discussed in Section 6.2.

5.2 Reflections

Overall, the thesis work provides a strong foundation in support of using deep learning for the task of microservice identification. The proposed model matched state-of-the-art, which alone is an indication that deep learning is a promising technique. The model also does not require as much fine-tuning when being run on different monolithic systems, which increases its ease of use. That the thesis was able to test the model on two vastly different monolithic systems strengthens this argument. However, a weakness of the thesis is that it only compared the deep learning-based model with one other approach to microservice identification. There exists a couple of proposed approaches for this problem. Preferably the deep learning-based model would have been compared to all of them. However, not all models could be reimplemented, and there did not exist enough pre-existing data in order to perform meaningful statistical testing. Therefore the comparison had to be limited to the MicroMiner approach proposed by Trabelsi et al. [9]. Another point which can be seen as a weakness is the fact that both systems used for testing were built using Java. Although it is not believed that this should have any significant impact on performance, it is possible that differences in formatting and structuring standards between programming languages could have a small impact on the identified microservices. Since no system built using other programming languages was tested, only speculations can be made. This can be seen as a threat to external validity. It is also worth mentioning that testing was only performed by the author and employees at the commissioning company. These parties are not independent, which can introduce bias. This should be seen as a threat to the internal validity.

5.3 Contributions and societal impact

The results of this thesis contribute to both industry and academia. The results are of relevance to academia since a non-trivial use case of deep learning-based community detection models has been explored. This has provided insight regarding the performance of a state-of-the-art model on a real-world use case. Further, by comparing the results to simpler clustering-based approaches, it has helped in highlighting the benefits and disadvantages of using more complex models for community detection.

In regards to societal impact, the commissioning company PE Accounting has been directly impacted by easing the process of updating their system architecture using the microservice style. However, this impact should also extend to any other company which wish to undergo the same process, as the proposed model has been designed to work on any system.

5.4 Ethics

In the field of machine learning, the question of ethics often pertains to the type of data used. The model developed throughout thesis used the PE Accounting and PetClinic systems as its only source of input data. Although the PE Accounting system has databases containing personal information, this data was not used by the model during training. Only the `.java` source files were used, these contained no personal information. Similarly, the PetClinic system is an open-source project which contains no personal information. During preprocessing, a pretrained CodeBERT model was used. This model had been trained using open-source Github repositories, specifically using the CodeSearchNet Corpus Collection [39]. Since the data in this collection comes from publicly available repositories, it should not contain any ethically questionably information. With this knowledge, it has been determined that there are no ethical issues pertaining to this thesis.

5.5 Sustainability

When it comes to sustainability, previous research [40] indicate that the microservice architecture has a smaller energy footprint compared to the monolithic system. The model used is not computationally expensive, with training at most taking a couple of minutes. Use of the model should therefore

not require a high energy consumption. The developed model and its intended use case is therefore beneficial from a sustainability perspective.

Chapter 6

Conclusions and future work

This chapters aims to answer the research question stated in Section 1.2 and give suggestions for future works.

6.1 Conclusions

The research question was made in two parts. First, it asked how a deep learning-based model would hold up compared to the previous clustering-based approaches. Second, it asked whether the ability to learn complex relationships would help in upholding the bounded context, size, and independency criteria.

In Chapter 4 and 5 the deep learning model was compared mainly to MicroMiner, a previously proposed clustering-based method. It was shown that both models achieved a similar level of performance. Thus, it can be stated that while the deep learning model developed throughout this thesis did not outperform previous approaches, it does hold up compared to them. This suggests that deep learning indeed can be a viable approach for the task of automatic microservice identification.

When it comes to the second part of the research question, it was shown throughout both the qualitative and quantitative evaluation that the model indeed could identify relevant microservices which upheld bounded context, size, and independency. Since training the model without an attribute matrix produced noticeably worse results for the PE Accounting system, this suggests that learning complex semantic relationships indeed was a contributing factor to the quality of the identified microservices. Although a few identified

microservices did not uphold all criteria, these were easy to spot due to their size and unproportionately large ifn score. Thus it was easy to discard these microservices, leaving only microservices which upheld all criteria.

In conclusion, deep learning indeed is a viable method for the identification of microservices within a monolith system. Although current results do not show clear improvement, they highlight deep learning as a promising new approach. Further research is therefore highly encouraged, as advances within this field could results in significant performance increases when it comes to the task of automatic microservice identification.

6.2 Future work

The topics covered throughout this thesis span a wide research area, resulting in several possible directions for future work. This thesis did not produce results showing a clear advantage of using deep learning for the task of microservice identification. However, it is strongly believed that if the advantages of deep learning are better leveraged, then significant performance increases can be achieved. Future work could focus on either improving the model outlined throughout this thesis, or developing a model using other deep learning approaches. Due to time limitations, only the NOCD model was implemented for this thesis. However, all models outlined in Subsection 2.6.3 could be used for microservice identification. One could implement and compare these models. It would be especially interesting to compare a deep learning model which utilizes modularity optimization as a loss function with the Bernoulli-Poisson approached as was utilized in this thesis. As mentioned in Section 5.1, some microservices were incorrectly identified due to spanning several business functionalities, and for the smaller PetClinic system, results showed high variance. These problems might be solved by using a more sophisticated assignment strategy (instead of thresholding) and bagging respectively. Or they might also be solved by first identifying interface classes, and then building microservices around these, similarly to MicroMiner. Fixing these problems could be another point of future work. Again, in Section 5.1, it was mentioned that the statistical testing was not optimal. Future work could therefore also focus on more thorough testing. Re-implementing MicroMiner and other previous approaches would allow for better comparisons, and more robust tests such as the Wilcoxon matched-pairs signed ranks test [41] could be used. This would provide a better basis for future decision making.

References

[1] E. S. Raymond, *The art of Unix programming*. Addison-Wesley Professional, 2003. [Page 1.]

[2] J. Lewis and M. Fowler, "Microservices: a definition of this new architectural term," *MartinFowler.com*, vol. 25, no. 14-26, p. 12, 2014. [Page 1.]

[3] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," *Present and ulterior software engineering*, pp. 195–216, 2017. [Pages 1 and 8.]

[4] N. Dragoni, I. Lanese, S. T. Larsen, M. Mazzara, R. Mustafin, and L. Safina, "Microservices: How to make your application scale," in *Perspectives of System Informatics: 11th International Andrei P. Ershov Informatics Conference, PSI 2017, Moscow, Russia, June 27-29, 2017, Revised Selected Papers 11*. Springer, 2018, pp. 95–104. [Pages 2 and 8.]

[5] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, "Service cutter: A systematic approach to service decomposition," in *Service-Oriented and Cloud Computing: 5th IFIP WG 2.14 European Conference, ESOCC 2016, Vienna, Austria, September 5-7, 2016, Proceedings 5*. Springer, 2016, pp. 185–200. [Pages 2 and 16.]

[6] L. Baresi, M. Garriga, and A. De Renzis, "Microservices identification through interface analysis," in *Service-Oriented and Cloud Computing: 6th IFIP WG 2.14 European Conference, ESOCC 2017, Oslo, Norway, September 27-29, 2017, Proceedings 6*. Springer, 2017, pp. 19–33. [Pages 2 and 16.]

[7] M. Abdullah, W. Iqbal, and A. Erradi, "Unsupervised learning approach for web application auto-decomposition into microservices," *Journal of Systems and Software*, vol. 151, pp. 243–257, 2019. [Pages 2 and 17.]

[8] M. Brito, J. Cunha, and J. Saraiva, "Identification of microservices from monolithic applications through topic modelling," in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, 2021, pp. 1409–1418. [Pages 2 and 17.]

[9] I. Trabelsi, M. Abdellatif, A. Abubaker, N. Moha, S. Mosser, S. Ebrahimi-Kahou, and Y.-G. Guéhéneuc, "From legacy to microservices: A type-based approach for microservices identification using machine learning and semantic analysis," *Journal of Software: Evolution and Process*, p. e2503, 2022. [Pages 2, 17, 19, 26, 28, and 35.]

[10] D. Jin, Z. Yu, P. Jiao, S. Pan, D. He, J. Wu, P. Yu, and W. Zhang, "A survey of community detection approaches: From statistical modeling to deep learning," *IEEE Transactions on Knowledge and Data Engineering*, 2021. [Pages 2 and 18.]

[11] X. Su, S. Xue, F. Liu, J. Wu, J. Yang, C. Zhou, W. Hu, C. Paris, S. Nepal, D. Jin *et al.*, "A comprehensive survey on community detection with deep learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2022. [Pages 2 and 18.]

[12] O. Shchur and S. Günnemann, "Overlapping community detection with graph neural networks," *arXiv preprint arXiv:1909.12201*, 2019. [Pages 4, 18, 20, and 23.]

[13] P. Johannesson and E. Perjons, *An introduction to design science*. Springer, 2014, vol. 10. [Page 4.]

[14] PE Accounting, "Vår tjänst. (Swedish) [Our Service]." [Online]. Available: https://www.peaccounting.se/var-tjanst [Page 4.]

[15] E. Evans and E. J. Evans, *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004. [Page 8.]

[16] S. Fortunato, "Community detection in graphs," *Physics reports*, vol. 486, no. 3-5, pp. 75–174, 2010. [Page 8.]

[17] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002. [Page 9.]

[18] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006. [Pages 9 and 18.]

[19] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008. [Pages 9 and 17.]

[20] J. Yang and J. Leskovec, "Overlapping community detection at scale: a nonnegative matrix factorization approach," in *Proceedings of the sixth ACM international conference on Web search and data mining*, 2013, pp. 587–596. [Pages 10 and 18.]

[21] D. Alur, J. Crupi, and D. Malks, *Core J2EE patterns: best practices and design strategies*. Gulf Professional Publishing, 2003. [Page 11.]

[22] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service candidate identification from monolithic systems based on execution traces," *IEEE Transactions on Software Engineering*, vol. 47, no. 5, pp. 987–1007, 2019. [Pages 11 and 25.]

[23] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008. [Page 14.]

[24] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020. [Page 14.]

[25] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016. [Pages 14 and 15.]

[26] ——, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016. [Page 14.]

[27] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017. [Pages 14, 15, and 16.]

[28] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013. [Page 14.]

[29] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272. [Page 14.]

[30] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003. [Page 17.]

[31] J. C. Bezdek, R. Ehrlich, and W. Full, "Fcm: The fuzzy c-means clustering algorithm," *Computers & geosciences*, vol. 10, no. 2-3, pp. 191–203, 1984. [Page 17.]

[32] B. G. Ryder, "Constructing the call graph of a program," *IEEE Transactions on Software Engineering*, no. 3, pp. 216–226, 1979. [Pages 17 and 19.]

[33] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020. [Pages 17 and 20.]

[34] A. Tsitsulin, J. Palowitch, B. Perozzi, and E. Müller, "Graph clustering with graph neural networks," *arXiv preprint arXiv:2006.16904*, 2020. [Pages 18 and 20.]

[35] A. Moradan, A. Draganov, D. Mottin, and I. Assent, "Ucode: Unified community detection with graph convolutional networks," *arXiv preprint arXiv:2112.14822*, 2021. [Pages 18, 19, and 20.]

[36] C. He, Y. Zheng, J. Cheng, Y. Tang, G. Chen, and H. Liu, "Semi-supervised overlapping community detection in attributed graph with graph convolutional autoencoder," *Information Sciences*, vol. 608, pp. 1464–1479, 2022. [Pages 18, 19, and 20.]

[37] A. F. McDaid, D. Greene, and N. Hurley, "Normalized mutual information to evaluate overlapping community finding algorithms," *arXiv preprint arXiv:1110.2515*, 2011. [Page 19.]

[38] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014. [Page 23.]

[39] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, “Codesearchnet challenge: Evaluating the state of semantic code search,” *arXiv preprint arXiv:1909.09436*, 2019. [Page 36.]

[40] F. Khomh and S. A. Abtahizadeh, “Understanding the impact of cloud patterns on performance and energy consumption,” *Journal of Systems and Software*, vol. 141, pp. 151–170, 2018. [Page 36.]

[41] W. J. Conover, *Practical nonparametric statistics*. john wiley & sons, 1999, vol. 350. [Page 40.]

TRITA-EECS-EX-2023:254