Monolith to Microservices: Representing Application Software through Heterogeneous GNN

Alex Mathai¹, Sambaran Bandyopadhyay¹*, Utkarsh Desai¹† and Srikanth Tamilselvam¹
¹IBM Research

Abstract

Monolith software applications encapsulate all functional capabilities into a single deployable unit. While there is an intention to maintain clean separation of functionalities even within the monolith, they tend to get compromised with the growing demand for new functionalities, changing team members, tough timelines, non-availability of skill sets, etc. As such applications age, they become hard to understand and maintain. Therefore, microservice architectures are increasingly used as they advocate building an application through multiple smaller sized, loosely coupled functional services, wherein each service owns a single functional responsibility. This approach has made microservices architecture as the natural choice for cloud based applications. But the challenges in the automated separation of functional modules for the already written monolith code slows down their migration task.

Graphs are a natural choice to represent software applications. Various software artifacts like programs, tables and files become nodes in the graph and the different relationships they share, such as function calls, inheritance, resource (tables, files) access types (create, read, update, delete) can be represented as links in the graph. We therefore deduce this traditional application decomposition problem to a heterogeneous graph based clustering task. Our solution is the first of its kind to leverage heterogeneous graph neural network to learn representations of such diverse software entities and their relationships for the clustering task. We study the effectiveness by comparing with works from various software engineering and existing graph representation based techniques. We experiment with applications written in an object oriented language like Java and a procedural language like COBOL and show that our work is applicable across different programming paradigms.

1 Introduction

Monolith architecture is the traditional unified model for designing software applications. Since it encapsulates multiple business functions or components into a single deployable unit, it is easy for developers to violate the modularity principle and still deliver fully functional applications. But such applications become difficult to understand and hard to maintain as they age because it increases complexity in understanding application code and predicting change impact [Kuryazov et al., 2020].

Microservices [Lewis and Fowler, 2014; Thönes, 2015] are seen as an alternative. It aims to realize the software application as a package of small services where each service is responsible for a single functionality. It brings multiple benefits like efficient team structuring, independence in development & deployment, enables flexible scaling and less restriction on technology or programming language preference. Figure 1 shows the significant differences between the two architectural styles for the same e-commerce software. But monoliths are prevalent in the market and are actively used by many customers. Migrating from monolith to microservices is a labour intensive task. It often involves domain experts, microservices architects and monolith developers working in tandem to analyze the application from multiple views and identify the components of monolith applications that can be turned into cohesive, granular service. Even then, these migration projects can take years for completion and such skills are not easily available.

The software engineering community refers to this migration process as a software decomposition task which has been an active area of research for more than twenty years. There are works [Mancoridis et al., 1998; Mancoridis et al., 1999; Tzerpos and Holt, 2000; Harman et al., 2002; Praditwong et al., 2010; Ouni et al., 2012; Alizadeh et al., 2018; Mahouachi, 2018; Mazlami et al., 2017] that tried to leverage the syntactical relationships between the programs and treated this decomposition as an optimization problem to improve different quality metrics like cohesion, coupling, number of modules, amount of changes etc. Fritzsch et al. [Fritzsch et al., 2018] presented a detailed survey on the different approaches for this problem. While the accuracy of these approaches has been evolving over time, they have their drawbacks such as 1) reliance on external artifacts like logs, commit history, etc. 2) focus on only a subset of the programs 3)

*Sambaran is currently at Amazon Science.
†Utkarsh is currently at Google. All the authors were affiliated to IBM Research when the work was conducted.
less attention to non-program artifacts like the tables, files 4) minimal consideration for transactional data. Recently [Jin et al., 2019] executed test cases to extract runtime traces and consider each execution as a business function. But this work relies on access to runtime traces and complete coverage of test cases which cannot be always guaranteed. Also, this work doesn’t view the application from a data and transaction perspective for decomposition.

Graphs are the natural structure to represent the application’s structural and behavioral information [Mancoridis et al., 1998; Desai et al., 2021]. The structural information of the application identified through different application entities such as programs, files, database tables, queue can be represented as nodes and its different program to program dependencies like call, extends, implements or the different CRUD operations (create, read, update, delete operations) that happen from programs to data resources (table, file, queues) can be represented as edges in the graph. The behavioral information of the application identified through the sequence of programs and resources that come together to support a business function can be captured as node/edge attributes. Figure 2 captures the construction of heterogeneous graph from a sample java code. The monolith to microservices task can now be viewed as generate based clustering task which involves 1) Learning a good representation of the graph structure 2) Using the learnt representation for the clustering task. Graph neural networks have achieved state of the art results for multiple graph-based downstream tasks such as node classification and graph classification [Kipf and Welling, 2017; Velickovic et al., 2018; Xu et al., 2019]. Most graph neural networks follow message passing mechanisms where the vector representation of a node is updated by combining its own features and aggregated features from its neighborhood [Gilmer et al., 2017]. Recently [Desai et al., 2021] showed how the programs and its relationships in the application can be represented as a graph and proposed a multi-objective graph convolution network that combined node representation & node clustering clustering task by diluting outliers [Bandyopadhyay et al., 2019b]. But since the framework did not consider application’s data resources like database tables, files and the different relationships that exists between programs & resources, the functional independence property of microservices is not completely satisfied. It can only be complete when we factor dependencies between the resources and find right microservice ownership. Graph neural networks have also been used recently for heterogeneous graphs [Zhang et al., 2019; Wang et al., 2019]. However, existing heterogeneous graph neural networks are not used directly to cluster the nodes of different types in a unified framework. For the purpose of converting a monolith application to microservices, we aim to design a heterogeneous graph neural network that can obtain both node and edge representations and generate a clustering of nodes.

In this work, we propose a novel graph neural network based solution to refactor monolith applications into a desired number of microservices. The main contributions of our paper are listed below.

1. We translate the application software’s structural and behavior properties into heterogeneous graph through nodes, edges and node/edge attributes.
2. We introduce a novel heterogeneous graph neural network, referred to as CHGNN, that enables the representation of data resources and programs and clusters artifacts into micro-services in a unified framework.
3. We show that inclusion of heterogeneous information generates better quality microservices recommendations through four publicly available monolith applications.
4. We demonstrate that our approach is applicable to different programming paradigms by showing improvements on Java and COBOL based applications.

2 Methodology

Given a monolith application, we want to partition the monolith into $K$ clusters, with $K$ provided by a subject matter expert (SME), where each cluster is a group of programs and resources that perform a well-defined functionality. An ideal
cluster should exhibit high cohesion, i.e., have strong interaction within the cluster and low coupling i.e., less interaction between clusters.

2.1 Converting Applications to Graph

We now describe our approach to represent an application as a graph, given it’s source code. The primary programming constructs in different languages are different - a class in Java and a program in COBOL. Hence, in the rest of this work, we refer to classes or programs as simply programs for consistency, unless otherwise specified. Consider a simple Java application as shown in Figure 2. Each class or program in the application can be represented as a Program node in the graph. Certain programs might also access resources such as database tables, files or other data structures. Such artifacts can be represented as Resource nodes in the graph. We denote the combined set of Program and Resource nodes as \( V \), the set of all nodes. We also allow for multiple edge types. We establish an undirected CALLS edge from Program node A to Program node B if there is a method in the program A that calls a method from program B. We also identify resource usage and create a CRUD edge from Program node X to Resource node R, if the program X accesses resource R. Static analysis tools\(^1\) make the call chains and resource usage within the application available by analyzing the source code. \( E \) denotes the combined set of all edges between the various nodes in the graph. The edges are unweighted and multiple method calls or resource usages between two nodes are still represented by a single edge.

We now generate the node attribute matrix, corresponding to the Program and Resource nodes of the graph. APIs exposed by most web-based applications (UI elements in the case of a non web-based application) are also referred to as EntryPoint Specifications [Dietrich et al., 2018], or simply, Entrypoints (EPs). The methods invoked through these APIs are annotated with tags such as @Api as shown in Figure 2. We refer to such methods as entrypoint methods and the corresponding classes or programs as entrypoint programs. Each entrypoint program can thus be associated with multiple entrypoints through different entrypoint methods. From an entrypoint method, we can obtain a sequence of invoked methods and their corresponding programs during the execution trace of that Entrypoint. If \( EP \) is the set of Entrypoints in an application and \( V_P \), is the set of Program nodes, we can define a matrix \( A[V_P \times |EP|] \), such that \( A(i,p) = 1 \) if program \( i \) is present in the execution trace of entrypoint \( p \), else 0. Additionally, we define another matrix \( C[V_P \times V_P] \) such that \( C(i,j) \) is the number of Entrypoint execution traces that contain both programs \( i \) and \( j \). If a program is not invoked in an execution trace for any Entrypoint, we remove the corresponding Program node from the graph. Finally, classes or programs may also inherit from other classes or programs or implement Interfaces. In Figure 2, class \( A \) inherits from class \( S \). Although this establishes a dependency between the programs, it is not a direct method invocation. Hence, this dependency is not included as an edge in the graph, but as a Program node attribute. Therefore, we define a matrix \( I[V_P \times |EP|] \) and set \( I(i,j) = I(j,i) = 1 \) if programs \( i \) and \( j \) are related via an inheritance relationship and 0 otherwise. The attribute matrix for Program nodes is the concatenation of \( A, C \) and \( I \), and denoted as \( X_P \).

For Resource nodes, the Inheritance features are not applicable. The \( A \) and \( P \) matrices are obtained by summing up the corresponding rows from the respective \( X_P \) matrices. The relationship between Program nodes and Resource nodes is many-to-many and this formulation simply aggregates features from all related programs into the resource. This gives us the resource attribute matrix \( X_R \). Each constituent matrix of \( X_P \) and \( X_R \) is row-normalized individually. The final set of node attributes are denoted as \( X_V = \{X_P, X_R\} \). The Edge attributes for CALLS edges is simply the vector \([1, 0]\) and there are no additional features. For CRUD edges, the attribute vector represents the type of resource access performed. Since a program can access a resource in more than one fashion, this is a \( 1 \times 4 \) vector, where each attribute represents the associated access type - [Create, Read, Update, Delete]. Hence a program that reads and updates a Resource node will have \([0, 1, 1, 0]\) as the edge feature. The edge attribute matrix is represented as \( X_E \).

Thus, the entire application can be represented by a heterogeneous graph as \( G = (V, E, X_V, X_E) \). Let us assume that \( \phi(v) \) and \( \psi(e) \) denote the node-type of \( v \) and edge-type of \( e \) respectively. Let us use \( x_v \in \mathbb{R}^{D_v(\phi)} \) to denote the attribute vector for the node \( v \) which belongs to \( D_v(\phi) \) dimensional space. Similarly, \( x_e \in \mathbb{R}^{D_e(\psi)} \) is the edge attribute of the edge \( e \).

2.2 Proposed Heterogeneous Graph Neural Network

In this subsection, we aim to propose a graph neural network (GNN) which can (i) handle different node and edge types in the graph, (ii) obtain vector representation of both nodes and edges by jointly capturing both the structure and attribute information, (iii) output community membership of all the nodes in the graph in a unified framework. We refer the proposed architecture as CHGNN (Community aware Heterogeneous Graph Neural Network). There are different steps in the design of CHGNN as described below.

Mapping Entities to Common Vector Space

Due to heterogeneity from different software artifacts, attributes associated with nodes and edges of the input graph are not of same types and they can have different dimensions. Such heterogeneity can be addressed in the framework of message passing heterogeneous graph neural networks in two ways: (i) Map the initial attributes to a common vector space using trainable parameter matrices at the beginning [Wang et al., 2019]; (ii) Use different dimensional parameter matrices while aggregating and combine information at each step of message passing [Vashishth et al., 2020]. We choose the first strategy since that makes the subsequent design of the GNN simpler and helps to add more layers in the GNN. So, we introduce type specific trainable matrices \( W_{\phi(v)} \in \mathbb{R}^{F(0) \times D_v(\phi)} \) for nodes and \( W_{\psi(e)} \in \mathbb{R}^{F(0) \times D_e(\psi)} \) for edges \( \forall v \in V \) and

\(^1\)https://github.com/soot-oss/soot
∀e ∈ E.
\[ h^{(0)}_v = σ(W_{φ(ν)}x_ν) \in \mathbb{R}^{F(0)}; \quad h^{(0)}_e = σ(W_ψ(ε)x_ε) \in \mathbb{R}^{F(0)} \]

where σ is a nonlinear activation function. \(x_ν\) and \(x_ε\) are the initial attribute vectors of node \(ν\) and edge \(ε\) respectively. \(h^{(0)}_ν\) and \(h^{(0)}_e\) are considered as 0th layer embeddings for \(ν\) and \(ε\) respectively. They are fed to the message passing framework as discussed below.

**Message Passing Layers for Nodes and Edges**

Message passing graph neural networks have achieved significant success for multiple node and graph level downstream tasks. In this framework, we obtain vector representation for both nodes and edges of the heterogeneous graph. There are \(L ≥ 1\) message passing layers. We define the \(l\)th layer (\(1 ≤ l ≤ L\)) of this network as follows.

As the first step of a message passing layer, features from neighborhood are aggregated for each node. In recent literature, it has been shown that both node and edge representation improves the downstream performance for multiple applications [Jiang et al., 2019; You et al., 2020; Bandyopadhyay et al., 2019a]. Following that, we also design the GNN to exchange features between nodes and edges, and update the vector representation for both. In each layer, we use two parameter matrices \(W_1^{(l)} \in \mathbb{R}^{F(l) \times F(l-1)}\) and \(W_2^{(l)} \in \mathbb{R}^{F(l) \times F(l-1)}\) to handle node and edge embeddings respectively. For a node \(ν \in V\), its neighborhood information is aggregated as:

\[
z^{(l)}_ν = \sum_{u ∈ N(ν)} \frac{1}{√d_u √d_v} W_1^{(l)}(u^{-1}) \ast σ(W_2^{(l)}(u^{-1})) \quad (2)
\]

where \(d_u\) and \(d_v\) are the degrees of the nodes \(u\) and \(v\) respectively. \(1/√d_u √d_v\) is used to symmetrically normalize the degrees of the nodes in the graph [Kipf and Welling, 2017]. \(σ()\) is a nonlinear activation function and ∗ is Hadamard (element-wise) product. Next, the aggregated information \(z^{(l)}_ν\) is combined with the embedding of node \(ν\) to update it as follows:

\[
h^{(l)}_ν = σ(z^{(l)}_ν + \frac{1}{d_ν} h^{(l-1)}_ν) \quad (3)
\]

\(h^{(l)}_ν\) is considered as the node embedding of node \(ν\) at \(l\)th layer. To update the embedding of an edge \((u, v)\), we use the updated embeddings of two end point nodes and the existing embedding of the edge as follows (||: concatenation of vectors):

\[
h^{(l)}_{uv} = σ\left(W_3^{(l)} \left(\frac{h^{(l)}_u + h^{(l)}_v}{2} \parallel h^{(l-1)}_{uv}\right)\right) \quad (4)
\]

where \(W_3^{(l)} \in \mathbb{R}^{F(l) \times (F(l) + F(l-1))}\) is a parameter matrix. This completes the definition of the layer \(l\) of the message passing network. Please note that the dimensions of the trainable matrices \(W_1^{(l)}, W_2^{(l)}\) and \(W_3^{(l)}\) determine the dimension of the embedding space of the nodes and edges.

To build the complete network, we first map the heterogeneous nodes and edges to a common space using Equation 1. Subsequently, we use 2 message passing layers \((l = 1, 2)\) as encoders (compressing the feature space) and next 2 message passing layers \((l = 3, 4; L = 4)\) as decoders (decompressing the feature space), with \(F(0) > F(1) > F(2) < F(3) = F(1) < F(4) = F(0)\). To map the node and edge features to their respective input attribute space, we again use linear transformations followed by activation functions as shown below.

\[
x_ν = σ(\hat{W}_{φ(ν)}h^{(L)}_ν) \in \mathbb{R}^{Dφ(ν)}; \quad x_ε = σ(\hat{W}_{ψ(ε)}h^{(L)}_ε) \in \mathbb{R}^{Dφ(ε)} \quad (5)
\]

These reconstructed node and edge attributes are used to design the loss functions as discussed next.

**Design of the Loss Functions and Joint Clustering of Heterogeneous Nodes**

As we do not have any prior knowledge about which software artifact should belong to which cluster, the problem is inherently unsupervised in nature. We use three types of reconstruction errors as follows.

**Node Attribute Reconstruction:** We try to bring the initial node features \(x_ν\) and the reconstructed node features \(\hat{x}_ν\) close to each other by minimizing \(∑_{ν ∈ V} ||x_ν - \hat{x}_ν||_2^2\).

**Edge Attribute Reconstruction:** With similar motivation as above, we minimize \(∑_{ε ∈ E} ||x_ε - \hat{x}_ε||_2^2\).

**Link Reconstruction:** Above two loss components do not capture anything about the link structure of the heterogeneous graph. Let us introduce the binary variables \(a_{u,v} \forall u, v ∈ V\) such that \(a_{u,v} = 1\) if \((u,v) ∈ E\) and \(a_{u,v} = 0\) otherwise. We want to ensure that embeddings of two nodes are close to each other if there is an edge between them by minimizing \(∑_{u,v ∈ V} (a_{u,v} - x_ν \cdot x_ε)^2\).

**Unifying Node Clustering:** The primary goal of our work is to refactor a monolith application to microservices for an application software. After we map the monolith to a heterogeneous graph, the goal can be achieved by clustering the nodes of the graph. As the nodes are represented in the form of vectors through the heterogeneous GNN encoder as discussed in Section 2.2, one intuitive approach is to pass these trained embeddings to a clustering algorithm as a post-processing step. But such a decoupled approach is always suboptimal since the GNN is unaware of the end clustering objective. So in this paper, we unify the clustering objective with the heterogeneous GNN as follows.

The node embeddings at the end of encoding layers (i.e., \(L/2\) layers) are \(h^{(L/2)}_v, ∀v ∈ V\). We design a k-means++ objective [Arthur and Vassilvitskii, 2006] by introducing two parameter matrices \(M ∈ \{0, 1\}^{V \times K}\) and \(C ∈ \mathbb{R}^{K \times F(k/2)}\). \(M\) is the binary cluster assignment matrix where each row sums up to 1. We assume to know the number of clusters \(K\), \(M_{vk} = 1\) if node \(v\) belongs to \(k\)th cluster and \(M_{vk} = 0\) otherwise. \(^2\) \(k\)th row of \(C\), denoted as \(C_k\), is the center of \(k\)th cluster in the embedding space. Node clusters and the

\(^2\)To avoid cluttering of notations, we use \(M_{vk}\) instead of \(M_{\text{index}(v), k}\), where \(1 ≤ \text{index}(v) ≤ |V|\)
corresponding cluster centers can be obtained by minimizing
\[ \sum_{v \in V} \sum_{k=1}^{K} M_{vk} |\| h_v^{L/2} - C_k |\|^2. \]

Hence, the total loss to be minimized by the proposed heterogeneous graph neural network is:
\[ \min_{\mathcal{W}, M, C} \mathcal{L} = \alpha_1 \sum_{v \in V} |\| x_v^e - \hat{x}_v |\|^2 + \alpha_2 \frac{1}{2} \sum_{e \in E} |\| x_e - \hat{x}_e |\|^2 \]
\[ + \alpha_3 \sum_{u,v \in V} \left( a_{uv} - x_u \cdot x_v \right)^2 + \alpha_4 \sum_{v \in V} \sum_{k=1}^{K} M_{vk} |\| h_v^{L/2} - C_k |\|^2 \]

where \( \mathcal{W} \) contains the trainable parameters of the GNN described in Section 2.2. \( \alpha_1, \alpha_2, \alpha_3 \) and \( \alpha_4 \) are non-negative weight factors. We set them in such a way that individual loss components contribute equally in the first iteration of the algorithm.

2.3 Training and Analysis

First, we pre-train the parameters of the GNN without including the clustering loss component, i.e., setting \( \alpha_4 = 0 \) in Equation 6. We use ADAM optimization technique [Kingma and Ba, 2014] to update the parameters of GNN. Once the pre-training is completed, we use alternating optimization techniques to update each of clustering parameters \( M \) and \( C \), and parameters of GNN \( \mathcal{W} \), while keeping others fixed. Using Lloyd’s update rule for k-means [Arthur and Vassilvitskii, 2006], we update \( M \) and \( C \) as:
\[ M(v, k) = \begin{cases} 1, & \text{if } k = \arg \min_{k' \in \{1, \ldots, K\}} |\| h_v^{L/2} - C_{k'} |\|^2 \\ 0, & \text{Otherwise} \end{cases} \]
\[ C_k = \frac{1}{N_k} \sum_{v \in C_k} h_v^{L/2} \]

where \( N_k = \sum_{v \in V} M_{vk} \). Due to the presence of clustering loss component in Equation 6, updating the parameters \( \mathcal{W} \) of GNN can pull the node embeddings close to their respective cluster centers further, along with reconstructing initial node and edge attributes and the link structure.

The forward pass of heterogeneous GNN takes \( O(|E|) \) time since messages are computed and passed over the edges of the graph. Link reconstruction component in Section 2.2 takes \( O(|V|^2) \) time. This can easily be relaxed by reconstructing only the existing nodes (i.e., when \( a_{uv} = 1 \)) with some negative samples for non-existing edges [Grover and Leskovec, 2016]. Since, number of nodes in the heterogeneous graph constructed to represent a monolith application is typically not very large for most of the real world applications, we reconstruct the full link structure.

3 Experimental Evaluation

3.1 Datasets (Monolith Applications) Used

To study the effectiveness of our approach, we chose four publicly-available web-based monolith applications namely

| Dataset          | Description         | Lang  | #Class | #Resources | #Clusters |
|------------------|---------------------|-------|-------|------------|-----------|
| Daytrader        | Trading App         | Java  | 111   | 11         | 8         |
| PBIW             | Plant Store         | Java  | 36    | 6          | 6         |
| Acme-Air         | Airline App         | Java  | 38    | 6          | 4         |
| GenApp           | Insurance App       | Cobol | 30    | 10         | 9         |

Table 1: Details about the monolith applications studied

Daytrader, Plantsbywebsphere, Acme-Air and GenApp. We chose these applications, as they are publicly available and show a good diversity in terms of the programming paradigm followed, languages & technologies used, user objectives and code complexity with respect to lines of code and the number of functions. Details of each monolith are provided in Table 1. We did not include DietApp in the baseline [Desai et al., 2021] since the public repository did not contain the stored procedure scripts that explains the relationship between programs and the database tables. Though our approach does not necessarily depend on the presence of resources, we chose to study only on applications that have resources, to show its contributions to the microservices generation process.

In what follows, we perform a quantitative and qualitative analysis of each predicted cluster in all the four applications. The quantitative metrics are a guide towards creating better clusters. However, given the heterogeneous formulation of our task, there are some aspects of evaluation that are not fully covered by these metrics. One such aspect is the ownership of data resources by each micro-service. Since the end task is to decompose the monolith to micro-services, each micro-service must ideally take complete responsibility of a particular data resource. Hence, grouping a data resource with the programs that update it, is a higher priority than maximizing metrics. We throw light on this aspect through a qualitative analysis of our algorithm’s recommended clusters.

3.2 Quantitative Metrics

To perform a quantitative evaluation of the clusters, we use 4 established graph metrics. They are 1) Modularity [Newman and Girvan, 2004] (Mod), 2) Structural Modularity [Jin et al., 2019] (S-Mod), 3) Non-Extreme Distribution [Wu et al., 2005] (NED) and 4) Interface Number [Jin et al., 2019] (IFN). Mod and S-Mod calculate the structural quality of the clusters - higher values indicate higher cluster modularity. NED and IFN are two desirable cluster properties but are not as important as the modularity metrics. The NED metric checks if the cluster sizes are evenly distributed and IFN measures the cross-connections between clusters.

3.3 Baseline Algorithms and Experimental Setup

As converting a monolith application to a heterogeneous graph and applying a heterogeneous graph neural network is a novel direction, we have designed most of the baselines

https://github.com/WASdev/sample.daytrader7
https://github.com/WASdev/sample.plantsbywebsphere
https://github.com/acmeir/acmeir
https://www.ibm.com/support/pages/cb12-general-insurance-application-genapp-ibm-cics-ts
https://github.com/SebastianBienert/DietApp/
with motivations from existing works on graph representation learning. They are discussed below.

**COGCN** [Desai et al., 2021]: In this baseline, the monolithic application with both program and resource nodes are converted to a homogeneous graph. Subsequently, COGCN, which has GCN layers trained on reconstruction [Kipf and Welling, 2016] and clustering loss [Arthur and Vassilvitskii, 2006], is applied on the homogeneous graph to obtain the micro-services. Note that COGCN does not consider edge attributes and only updates node embeddings in each layer.

**HetCOGCN**: Here, we create the heterogeneous graph as discussed in Section 2.1, but consider all the edges as of similar types. We map the heterogeneous nodes to a common vector space as done in Section 2.2 by using node-type specific parameter matrices \( W_{\phi (v)} \)'s. Then, COGCN is applied on that graph. So, this baseline has additional trainable parameters compared to the previous baseline.

**CHGNN-EL**: This is a variant of our proposed architecture CHGNN. We drop the edge feature re-construction loss from the optimization procedure by setting \( \alpha_2 = 0 \) in Equation 6.

**CHGNN**: This is the complete model of CHGNN as proposed by us in this work.

Please note that COGCN is a homogeneous graph based approach. HetCOGCN only uses heterogeneous nodes but homogeneous edges. CHGNN-EL uses both heterogeneous nodes and edges, but do not consider heterogeneous edge reconstruction error to train. Finally, CHGNN is a completely heterogeneous model.

**Architecture Specifications**: In each of these experiments, we use a two-layer encoder and a two-layer decoder. The two encoder layers reduce the dimensions from \( F^{(0)} \) to 64 and 32 respectively. Similarly, the two decoders increase the dimensions from 32 to 64 and then from 64 to \( F^{(0)} \).

**Training Procedure**: Before adding the clustering loss component, we pre-train the model for 150 epochs (\( lr = 0.01 \)), allowing the model to better understand the application’s graph structure. For this first round of training, we set \( \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\} = \{0.4, 0.2, 0.4, 0\} \) for CHGNN. Similarly, we set \( \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\} = \{0.5, 0, 0.5, 0\} \) for HetCOGCN, COGCN and CHGNN-EL. We then add the clustering loss and train the model for another 150 epochs (\( lr = 0.005 \)). During this second round of training, we set \( \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\} = \{0.1, 0.1, 0.1, 0.7\} \) for CHGNN. Similarly, we set \( \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\} = \{0.1, 0.0, 0.1, 0.8\} \) for HetCOGCN, COGCN and CHGNN-EL.

To generate the final cluster assignments for each application, we use the latest values of \( M(i, k) \) - i.e. the value obtained at the end of the 300\(^{th}\) epoch.

### 3.4 Quantitative and Qualitative Results on Separating Micro Services

As can be seen from Table 2, there is no single model that outperforms every metric on each application. However, there are some general trends that are visible. Upon analysing Mod and S-Mod for all four applications, we notice that both CHGNN-EL and CHGNN fare better than COGCN. We also notice a far superior performance in Genapp using our model when compared to the two baselines. This is highly promising as COBOL based applications are highly non modular and spread out and consequently more difficult to refactor. Another interesting observation is that COGCN does not have a best performing metric in any of the four applications. We believe that this underscores the importance of having a heterogeneous representation of the different node types in the graph. The average std deviation for Mod and S-Mod numbers in Table 2 is \( \pm 0.016 \) and \( \pm 0.015 \) respectively.

As a qualitative analysis, we study our predictions for the Acme Air application. As seen in Figure 3, we observe that the algorithm has identified four distinct functional clusters. These are 1) Customer, 2) Booking, 3) Flight and 4) Session Manager. The programs in each of the cluster clearly have very close dependencies within the cluster and seem to be contributing majorly to the common identified business function. This can be observed by both the edges among programs within the clusters and also with the program names. Associated with each cluster is a group of data nodes (dashed circle) that interact closely with the programs in their cluster. As seen, the Customer cluster has a `customer` table and the Flight/Airport cluster has three tables - `flightsegment`, `flight` and `airportcodemapping` etc. We clearly see that CHGNN is able to correctly group programs and their associated data resources. By creating such clusters, we move towards better quality microservices that own their resources. More details on the qualitative study is presented in the supplementary material.
Table 2: Performance of partitioning Monolith to Microservices (results averaged over 3 runs)

|                  | PBW (K=6) |             |             |             | Genapp (K=9) |             |             |             |
|------------------|-----------|-------------|-------------|-------------|--------------|-------------|-------------|-------------|
|                  | Mod (↑)   | S-Mod (↑)   | IFN (↑)     | NED (↑)     | Mod (↑)      | S-Mod (↑)   | IFN (↑)     | NED (↑)     |
| COGCN            | 0.219     | 0.034       | 3.722       | 0.76        | 0.225        | 0.08        | 2.7         | 0.55        |
| HetCOGCN         | 0.254     | 0.037       | 3.611       | 0.87        | 0.336        | 0.141       | 2.33        | 0.6         |
| CHGNN-EL         | 0.253     | 0.05        | 3.66        | 0.62        | 0.438        | 0.172       | 2.48        | 0.608       |
| CHGNN            | 0.258     | 0.07        | 3.66        | 0.77        | 0.494        | 0.164       | 2.11        | 0.6         |
| ACME (K=4)       |           |             |             |             | 0.85         | 0.08        | 4.125       | 0.873       |
|                  |           |             |             |             | 0.28         | 0.08        | 4.041       | 0.821       |
|                  |           |             |             |             | 0.26         | 0.075       | 3.95        | 0.817       |

4 Conclusion

We motivated the need for migrating software applications from monolith to microservices. We highlighted the key properties of microservices: i) loosely coupled smaller services ii) independence in functionality and data ownership. We discussed the importance of looking at the application not just from programs, but also from the resources and data access perspective for the decomposition task. We showed how a heterogeneous graph is a natural choice to capture the different entities and relationships that exist in an application. We proposed a novel heterogeneous graph neural network that enables the representation of data resources and programs, and jointly cluster artifacts into micro-services. We showed that inclusion of heterogeneous information generates better quality microservices recommendations through four publicly available monolith applications. We believe our work can reduce developers’ efforts significantly. In the future, we want to consider prior knowledge and give more importance to certain relationships like update over read and extends over implements. We also aim to study the decomposition task at a more granular level from programs to functions and tables to columns.

References

[Alizadeh et al., 2018] Vahid Alizadeh, Marouane Kessen-tini, Mohamed Wiem Mkaouer, Mel Ocinneide, Ali Ouni, and Yuanfang Cai. An interactive and dynamic search-based approach to software refactoring recommendations. *IEEE Transactions on Software Engineering*, 46(9):932–961, 2018.

[Arthur and Vassilvitskii, 2006] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[Bandyopadhyay et al., 2019a] Sambaran Bandyopadhyay, Anirban Biswas, M Narasimha Murty, and Rama Suri Narayanan. Beyond node embedding: a direct unsupervised edge representation framework for homogeneous networks. *arXiv preprint arXiv:1912.05140*, 2019.

[Bandyopadhyay et al., 2019b] Sambaran Bandyopadhyay, N Lokesh, and M Narasimha Murty. Outlier aware network embedding for attributed networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 12–19, 2019.

[Desai et al., 2021] Utkarsh Desai, Sambaran Bandyopadhyay, and Srikanth Tamilselvam. Graph neural network to dilute outliers for refactoring monolith application. In *Proceedings of 35th AAAI Conference on Artificial Intelligence (AAAI’21)*, 2021.

[Dietrich et al., 2018] Jens Dietrich, François Gauthier, and Padmanabhan Krishnan. Driver generation for java ee web applications. In *2018 25th Australasian Software Engineering Conference (ASWEC)*, pages 121–125. IEEE, 2018.

[Fritzsch et al., 2018] Jonas Fritzsch, Justus Bogner, Alfred Zimmermann, and Stefan Wagner. From monolith to microservices: A classification of refactoring approaches. *CoRR*, abs/1807.10059, 2018.

[Gilmer et al., 2017] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272, 2017.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[Harman et al., 2002] Mark Harman, Robert M Hierons, and Mark Proctor. A new representation and crossover operator for search-based optimization of software modularization. In *GECCO*, volume 2, pages 1351–1358, 2002.

[Jiang et al., 2019] Xiaodong Jiang, Pengsheng Ji, and Sheng Li. Censnet: Convolution with edge-node switching in graph neural networks. In *IJCAI*, pages 2656–2662, 2019.

[Jin et al., 2019] Wuxia Jin, Ting Liu, Yuanfang Cai, Rick Kazman, Ran Mo, and Qinghua Zheng. Service candidate identification from monolithic systems based on execution traces. *IEEE Transactions on Software Engineering*, 2019.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.
We summarize the notations in Table 3.

5 Supplementary Material

5.1 Hardware Requirements

All experiments were run on a system with 32 GB RAM, a 6-Core Intel i7 processor and a 4 GB AMD Graphics Card.

5.2 Training Details

All training details have been specified in Section 3.3. This includes the values of different hyper-parameters like \(\alpha_i\) and learning rate (lr). We also detail the training strategies used for successful convergence. All the experiments on the considered applications take around 6 minutes for completion.

5.3 Notations Used

We summarize the notations in Table 3.
5.4 Qualitative Study

In this section, we cover the job profile of the participants, details on how the study was conducted and the feedback on the microservices recommendations. We also provide a detailed comparative analysis on two of the applications covering two programming paradigms.

5.5 Participants Profile

To study the efficacy of the microservices recommendations, we requested participation from four software engineers to analyze four applications. On an average, the participants had industrial experience of 13 years in different software engineering roles. All the four participants had prior working experience on Java programming language. Two of the participants also had experience with working on COBOL applications. Two annotators (R1 & R2) who had an understanding of COBOL took an average of 4 weeks to understand the GenApp application before they participated in this study. Two other annotators (R3 & R4) spent an average of 2 weeks to understand the three Java based applications before they participated in this study.

5.6 Study Instructions

For each application, we provided the instructions shown in Figure 4 to the respective participants. The application specific details like the application code reference and clustering outputs (the microservices recommendations - available as jsons and sunburst chart images) are mentioned in each instruction. As an example, Figure 5 captures the sunburst chart for the CHGNN and COGCN approaches that were presented to R1 & R2.

5.7 Assets

The following public Applications - Daytrader ⁸, Plantsby-websphere (PBW) ⁹, Acme-Air ¹⁰ and GenApp ¹¹ are used for this study. All of them are Apache Licensed assets. We used the PyTorch Geometric ¹² framework for model implementation which is released as MIT License.

5.8 Qualitative studies

For the study, we requested each participant to compare the results from COGCN to our CHGNN model through the json and sunburst chart images provided. We anonymized the model details in the inputs. Overall, we found that the participants had a greater agreement with the microservice recommendations produced by CHGNN than those produced by COGCN. Table 4 captures the participants’ comments for their selection for each of the applications they evaluated. The participants pointed out scope for improvements in the CHGNN recommended clusters and they had their own suggestions about how certain programs or tables could be moved to a different service. However, compared to the COGCN model, they recognized that CHGNN produces more acceptable and accurate clusters. Additionally, we provide our own detailed qualitative analysis of the results for one application from each of the programming paradigms - Java (OOP) and COBOL (procedural). From the feedback received, we are positive that our work helps the developers to get closer to the ideal microservices design and can substantially reduce their migration effort. This highlights the importance of factoring in resources in addition to programs for clustering and the efficacy of our heterogeneous network.

5.9 Authors’ Comparative analysis

Authors’ Comparative analysis on GenApp

Figure 5 showcases the resulting clusters for GenApp by CHGNN and COGCN models. Functionally, GenApp is an application that creates insurance policies and processes
| Application | Language | Reviewer Selection | Reviewer Reasons |
|-------------|----------|--------------------|------------------|
| GenApp      | COBOL    | CHGNN              | R1. I have analysed both sets of outputs on 3 aspects. A) Number of datasets or tables correctly grouped in the cluster (more the better), B) Number of programs in the cluster that are incorrectly placed (less the better) and C) Number of clusters that do not have a well defined function (less the better). On all 3 aspects, I find that the blind1.json is a better result. |
|             |          |                    | R2. In Blind-1, my biggest problem is that TESTC1 comes with a Policy cluster. In Blind-2, my main problems are 1) LGAPVS01 is grouped wrong. 2) LGACDB02 and Customer-Secure table are grouped wrong 3) All tables are grouped together and LGAPVS01 is thrown in as a bonus. Relatively speaking, Blind-1 is far better. |
| Daytrader   | Java     | CHGNN              | R3. Reason : Keysequence, KeysequenceDirect, keygen table are all together. TradeSLSBean and TradeDirect which are common in functionality are put together. Blind2 has a very large clustering with tables like keygen packed together which looks wrong. But still Blind2 also should be improved to group TradeConfig and TradeAction |
|             |          |                    | R4. Overall I think it is a good separation of functionality, I could associate a name with each cluster to some extent. Only a few things bothering me: most db tables end up in the same cluster as TradeDirect, but since TradeSLSBean is also present here, it probably makes sense. Also the ping classes assigned to the cluster with Order and Holding data seem a bit random. |
| PBW         | Java     | CHGNN              | R3. Reason : supplier service came out well with its db contained within it. Inventory, Shopping and backorder which are closely connected are clubbed together with inventory and backorder table together. Also in Blind2, there is no good explanation for catalogmanager and emailmessage |
|             |          |                    | R4. In Blind 1, there are clear Supplier, Inventory/Backorder and Customer clusters with the only issue being customer-db moved to a different cluster. Blind 2 on the other hand has no clear separation of clusters. Inventory/Backorder and Shopping seem mixed up. |
| Acme-Air    | Java     | CHGNN              | R3. Reason : Customer service came out as a well separated cluster. I find that unlike Blind2 it didn’t mix the booking with authentication service. At a first glance, I was confused why FlightService abstract class is separated from FlightServiceImpl and kept with BookingService. But BookingService seem to have dependency only to the only FlightService implemented method getFlightByFlightId. |
|             |          |                    | R4. Flight and Customer clusters look more complete in Blind 1 and other clusters look better overall. In Blind 2, the customer table seems misplaced and the bottom left and bottom right clusters seem to be a bit overloaded. |

Table 4: For all the four applications, the reviewers chose the microservices recommendations by CHGNN over COGCN.

customer-claims. Hence it has 2 types of databases - the customer databases and a group of policy databases (house, motor, vehicle etc.). This entire codebase is broken into 2 main logical groups (containing code and data) - the 1st deals with customer acquisition and the 2nd deals with policies bought and claimed by customers. It is implemented as a monolithic COBOL codebase that contains 30 program nodes and 10 resource nodes. Hence, on the whole, resource nodes contribute to 25% of all available nodes in the software. This makes GenApp an excellent candidate for analysing the recommendations by our heterogeneous GNN. Both R1 and R2 agreed that the recommendations made by CHGNN are better suited for microservices than COGCN. Their comments are depicted in Table 4 for your perusal. Upon analysis of both outputs, we observe the following differences between the 2 results.

- COGCN groups many tables together into 1 cluster. By doing so it mixes the tables from the policy group along with the tables from the customer group (Policy and Customer-2 grouped into 1 cluster).
- COGCN also separates the programs that update the tables into another cluster - this causes undesirable cross CUD (create, update and delete) connections.

ex. LGAPDB01, LGDPDB01 and LGUPDB01 are not grouped with the policy tables.

- CHGNN on the other hand groups almost all the tables and the associated programs correctly. This reduces cross CUD connections. At the same time, it keeps the customer tables and policy tables in three separate clusters - successfully partitioning the functionality (Policy, Customer-1 and Customer-2 are separate clusters).

- One anomaly for CHGNN is the placement of LGTESTC1 which is a customer related business-script with other policy related business-scripts.

Authors’ Comparative analysis on acme-air

Acme-Air is an application that captures key functionalities in managing an airline called "Acme Air". It contains overall 38 nodes which includes 32 java programs and 6 db tables. Figure 6 shows the output from the two models for acme-air application. We list below the differences between the two recommendations made by CHGNN and COGCN and provide reasons why CHGNN’s output is a closer recommendation to ideal microservices.

- CHGNN extracted a well separated Customer microservice compared to COGCN
• COGCN has customersession table mixed with booking table in the booking microservice
• COGCN mixed the bookings service with authentication service
• COGCN separated AuthService and AuthServiceImpl
• COGCN separated AcmeAirConfiguration and ServiceLocator which its most dependent on
• One drawback in CHGNN’s output is that it clusters FlightService in the booking microservice. Although FlightService has connections with BookingServiceImpl, it should have been aligned with FlightServiceImpl and Flightloader in the Flight/Airport microservice.

5.10 Limitations
Like we discussed in the conclusion section, the current method works on the program and resource level but the decomposition task can be studied at a more granular level from programs to functions and tables to columns. Also, we had to limit our study to only four medium sized applications as it takes 3-4 weeks for a software developer to understand the implementation structure of the application and provide feedback for the generated recommendations. However, based on the positive feedback from the qualitative analysis, we believe that our approach can significantly reduce developers’ effort in finalizing the ideal microservices design for the migration activity.
Figure 5: GenApp application Sunburst charts (a) and (b) representing the microservice recommendations using CHGNN and COGCN models respectively.
Figure 6: Acme-Air application Sunburst charts (a) and (b) representing the microservice recommendations using CHGNN and COGCN models respectively.