Final Year Project Showcase 2026

Generating Distributed Code from Single Node Code

Mineth Weerasinghe, Himindu Kularathne, Methmini Madhushika, Dhanuka Lakshan
Department of Computer Science & Engineering, University of Moratuwa, Sri Lanka

Abstract

Migrating monolithic systems to microservices is a challenging task due to tightly coupled components and unclear service boundaries. Existing decomposition approaches often rely on either structural dependencies or semantic representations, limiting their ability to capture the full system context. This work proposes an automated framework that integrates structural and semantic signals using a graph-based learning approach. The system models the monolith as a heterogeneous graph and employs an edge-aware graph autoencoder to learn meaningful embeddings of program entities. Triplet loss is incorporated to preserve semantic similarity, while K-Means clustering is used to identify microservice boundaries. The framework is evaluated on benchmark datasets using standard metrics such as Structural Modularity (SM), Inter-Service Coupling (ICP), Normalized Entity Distribution (NED), and Interface Number (IFN). Results demonstrate improved decomposition quality and more balanced service partitions compared to existing approaches.

Project Objectives

- **Understand & Analyze:** Analyze existing microservice decomposition approaches.
- **Apply:** Evaluate decomposition quality using benchmark datasets and established metrics (SM, ICP, NED, IFN).
- **Create:** Design and develop a hybrid decomposition framework that integrates structural, semantic, and domain knowledge signals.
- **Optimize & Justify:** Optimize clustering and decomposition strategies, and justify improvements through comparative analysis against baseline methods.

Related Work

Existing approaches to monolith-to-microservices decomposition can be broadly categorized into structural, semantic, and hybrid methods [1].

Approach	Method	Structural	Semantic	Limitations
CHGNN [2]	Graph Neural Network	✓	✗	Does not capture code level semantic meaning
CoGCN [3]	Graph Neural Network	✓	✗	Relies primarily on structural dependencies without semantic understanding
Mono2Micro [4]	Dynamic Analysis + Clustering	✓	✓	Requires runtime traces and manual use-case definitions
MonoEmbed [5]	Contrastive Learning	✗	✓	Lacks structural awareness

Gap: Existing approaches fail to effectively integrate structural, semantic, and domain knowledge in a unified framework.

Methodology

The proposed approach models the monolithic system as a heterogeneous graph by representing program and resource entities as nodes, and CALLS and CRUD relations as typed edges. These features are projected into a shared latent space, where an edge-aware graph autoencoder learns embeddings that capture structural and semantic relationships. Triplet loss guides the embeddings to preserve service-level similarity, and K-Means clustering is applied to generate candidate microservices. Finaly, resulting decomposition is evaluated using microservice quality metrics.

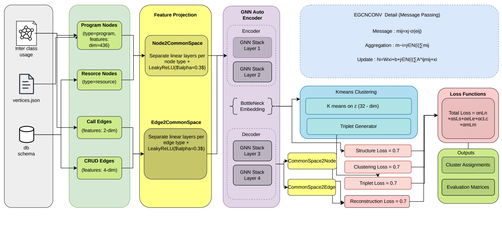

Figure 1. Microservice Decomposition Pipeline

- An edge-aware graph autoencoder learns compact embeddings capturing both structural and semantic relationships.
- The learned embeddings are clustered using K-Means to identify candidate microservices.
-

$$\mathcal{L}_{hybrid} = w_r\mathcal{L}_{rec} + w_t\mathcal{L}_{triplet}$$

$w_r = \max\left(\alpha, \frac{N-n}{N}\right)$, $w_t = \min\left(1-\alpha, \frac{n}{N}\right)$, $\mathcal{L}_{rec} = \frac{1}{3}\sum_{i\in\{p_1,p_2,n\}} \|x_{d,i} - x_{e,i}\|$, $\mathcal{L}_{triplet} = \max(0, \|h_{p_1} - h_{p_2}\| - \|h_{p_1} - h_n\| + \beta$

α: reconstruction–semantic balance, n: current epoch, N: total epochs, β: triplet margin.

- The optimization objective combines: Reconstruction loss, Structural loss, Clustering loss, Triplet loss

Experiments

Application	No of Classes	No of Methods
DayTrader	118	2577
Plants	40	1109
AcmeAir	86	1629

Table 1. Size statistics of benchmark applications

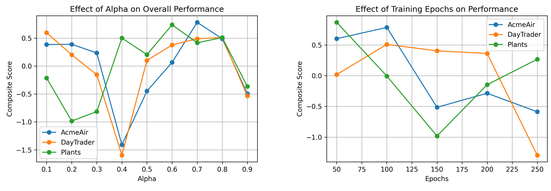

Figure 2. *Left:* Effect of α on decomposition (evaluated at 100 epochs). *Right:* Effect of training epochs on decomposition ($\alpha = 0.7$).

Results

Metrics: SM (cohesion vs coupling), ICP (inter-service calls), IFN (interfaces exposed), NED (size balance)

Goal: Maximize cohesion while minimizing inter-service dependencies and imbalanced cluster sizes.

Composite Score:

- Derive a single value across all metrics for each tool.
- Based on Sellami and Saied [5]; Weighted z-score normalization.
- SM is the dominant factor; thus weights, $W = 3, 1, 1, 1$

$$\text{Score}(T) = \frac{\sum w_m\left(\frac{x_{m,t}-\mu_m}{\sigma_m}\right)}{\sum|w_m|} \quad (1)$$

Dataset	Method	SM ↑	IFN ↓	ICP ↓	NED ↓	Composite ↑
DayTrader	CHGNN	0.13	5.70	0.55	**0.50**	-0.3043
	MonoEmbed	0.13	**1.35**	0.59	0.66	-0.4665
	OurTool	**0.14**	5.10	**0.48**	0.62	**0.7708**
Plants	CHGNN	**0.17**	3.60	0.51	**0.20**	**0.5188**
	MonoEmbed	0.11	**1.55**	**0.26**	0.82	-0.4449
	OurTool	0.15	4.31	0.58	0.21	-0.0738
AcmeAir	CHGNN	0.11	2.50	0.37	**0.00**	0.2782
	MonoEmbed	0.01	3.06	0.47	0.56	-1.1066
	OurTool	**0.23**	**2.36**	**0.28**	0.71	**0.8284**

Table 2. Comparison of decomposition quality across datasets

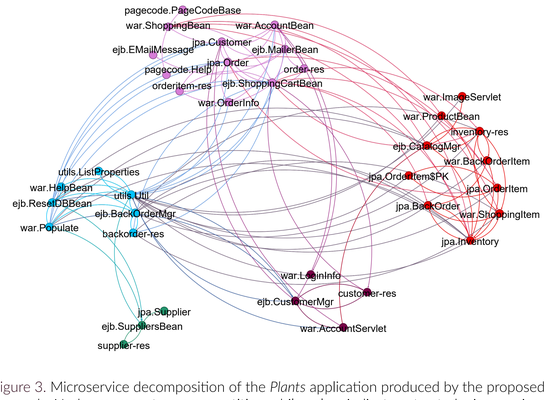

Figure 3. Microservice decomposition of the *Plants* application produced by the proposed approach. Nodes represent program entities, while colors indicate extracted microservices.

Conclusions

- Our hybrid approach improves decomposition with **SM and lower ICP** than CHGNN and MonoEmbed.
- Highest composite score in **2 out of 3 benchmark datasets**.
- Produces balanced partitions, **avoiding fragmentation**.

References

[1] G. Mazlami, J. Cito, and P. Leitner, "Extraction of microservices from monolithic software architectures," in *Proceedings of the IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 524–531.

[2] S. Ding, P. Liang, A. Tang, and P. Avgeriou, "Monolith to microservices: Representing application software through heterogeneous graph neural network," in *Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 2021.

[3] J. Liu, S. Wang, T. Chen, and Q. Zhang, "Cogcn: A graph convolutional network approach for microservice decomposition," *IEEE Access*, vol. 9, pp. 118 895–118 906, 2021.

[4] A. K. Kalia *et al.*, "Mono2micro: A practical and effective tool for decomposing monolithic java applications to microservices," in *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2021, pp. 1214–1224.

[5] K. Sellami and M. A. Saied, "Contrastive learning-enhanced large language models for monolith-to-microservice decomposition," *arXiv preprint arXiv:2502.04604*, 2025. [Online]. Available: https://arxiv.org/abs/2502.04604

{mineth.21, himindu.21, methmini.21, dhanuka.21}@cse.mrt.ac.lk

Supervised by: Dr. Srinath Perera, Dr. Nisansa de Silva, and Dr. Adeesha Wijayasiri

Department of Computer Science & Engineering
University of Moratuwa