ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.

Graph Representation Learning for Software Architecture Recovery

Rakhshanda Jabeen[1], Morgan Ericsson[1], Jonas Nordqvist[2] and Anna Wingkvist[1]

[1]Linnaeus University, Computer Science and Media Technology, Växjö, Sweden
[2]Linnaeus University, Mathematics, Växjö, Sweden

Abstract.
Software architecture recovery aims to infer a system's modular organization from source code, bridging the gap between design intent and implementation structure. Traditional techniques rely on handcrafted heuristics and often fail to capture deeper architectural relationships. We investigate whether GNNs can recover these relationships by framing the task as unsupervised representation learning over a multi-relational software graph. Our approach learns node embeddings that reflect architectural boundaries, offering a promising alternative to existing recovery methods.

1 Introduction

Software architecture defines a system's key components, their relationships, and the rules governing their interactions. Maintaining accurate architectural documentation is challenging, as modern systems evolve rapidly and often diverge from their intended designs [1]. Software architecture recovery (SAR) aims to reconstruct a system's high-level design from its implementation artifacts.

Recovering architectural modules is challenging because the source code expresses many overlapping design signals. However, dependency relationships (e.g, *Call*, *Import*, or *inheritance*), folder hierarchies, and code semantics all reflect how responsibilities are organized in practice. Dependencies capture how entities interact, folder locations mirror modular boundaries, and code identifiers encode developer intent and concepts [2, 3]. Together, these signals provide a foundation for inferring architectural structure from the implementation.

Traditional SAR approaches rely on hand-crafted heuristics or manually weighted combinations of structural, textual, and directory signals [3]. While effective in some settings, such methods struggle to generalize across systems with diverse styles and dependency patterns, motivating the use of learning-based techniques that can integrate multiple cues directly from the source code. Graph neural networks (GNNs) naturally operate on relational graphs and update node representations by aggregating information from neighbors via message passing [4]. They combine structural and semantic signals into a unified representation suitable for architecture recovery.

Building on GNNs, we introduce ARL (Architecture Representation Learning), an unsupervised method that learns architecture-aware embeddings of source code entities. In its base form, ARL treats each dependency type as a separate message-passing channel, enabling the encoder to distinguish among

ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.

different interaction patterns. We also propose ARL-E, which augments this representation with edge attributes capturing the member-level interactions (e.g., method calls, field accesses, import-driven references) that give rise to each dependency. These attributes summarize (i) which program elements triggered the dependency and (ii) how frequently the interaction occurs. The motivation is that architectural cohesion is shaped not only by the existence of a dependency, but also by its semantic origin and interaction frequency.

To obtain node embeddings, we use a heterogeneous Graph Attention Network (GAT) [5] trained with contrastive learning [6]. The resulting representations are then clustered to recover an architectural decomposition of the system.

2 Background and Related Work

Knowledge graphs (KGs) represent entities and their typed relationships as directed, labeled multi-graphs, or equivalently as sets of triples (h, r, t), where h and t are entities and r is a relation [7]. This abstraction naturally aligns with multi-relational heterogeneous graphs used in machine learning. GNNs operate directly on such relational structures via iterative message passing, shaping node embeddings based on neighborhood semantics and topology. They have shown strong performance across relational reasoning tasks, including computer vision, software engineering, and link prediction [4].

Software artifacts fit directly into this view: code files or classes serve as primary nodes, while internal members, such as functions and variables, generate typed static dependencies (e.g., calls, inheritance, imports) that capture rich relational structure. Modeling software systems as multi-relational graphs, therefore frames SAR as a representation-learning problem in which GNNs can derive embeddings that reveal the system's cohesive modular components.

Recent SAR methods have evolved from using structural dependencies or text similarity alone to combining multiple signals. SARIF [3] represents a recent state-of-the-art fusion approach that integrates structural, semantic, and directory features through hand-crafted weighting schemes. NEGAR [8] adopts node2vec-style representation learning for dependency graphs but disregards the heterogeneous relations or semantic cues from source code. LLM-based approaches have also been explored, using chain-of-thought prompting to map entities to predefined architectures [9]. While GNNs have been applied to several software engineering tasks, their use for unsupervised architecture recovery remains largely unexplored. To the best of our knowledge, no prior work applies GNN-based relational representation learning to SAR.

3 Methodology

The overall pipeline is shown in Figure 1. We now describe each component in detail, beginning with the construction of the multi-relational software graph.

ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.

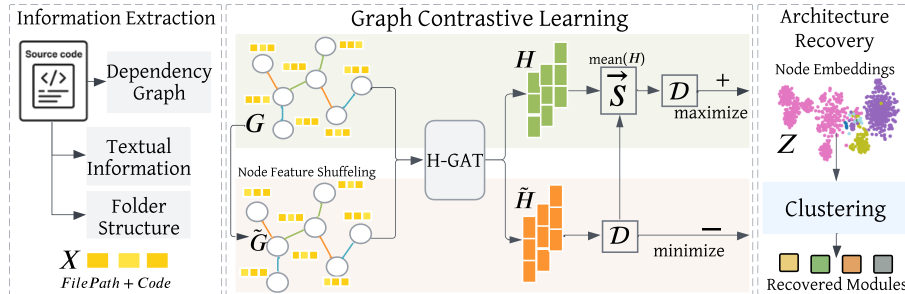

Fig. 1: Overview of ARL: (1) multi-relational software graphs are constructed with node and edge features; (2) a heterogeneous GAT encodes the graph and is optimized using contrastive learning; (3) the learned node embeddings are clustered to recover architectural components.

Graph Construction. We construct a multi-relational graph $G = (\mathcal{V}, X, A_{\mathcal{R}})$, where the nodes $\mathcal{V}$ represent source code entities, such as files or classes. Each relation type $\{A^{(r)} \mid r \in \mathcal{R}\}$ defines a separate directed adjacency matrix based on dependency information extracted using Depends [10]. Node features (X) are generated by combining tokens from the folder hierarchy and code semantics derived from file names and symbol identifiers that are not involved in inter-dependencies. Identifiers are split according to standard naming conventions, filtered for low-information tokens, and encoded as binary vectors, followed by l_2 normalization.

To extend the relational structure, we incorporate edge attributes that describe how each dependency arises. A dependency between two files may originate from interactions between different kinds of program elements (e.g., function-function, function-var, file-file). For each source–target member pair, we create a token capturing this interaction type and encode it as a binary vector followed by l_2 normalization. We additionally record the frequency of the dependency using a log-normalized count of how often the interaction occurs between the two files. The final edge attribute vector is the concatenation of the member-level interaction encoding and the dependency count.

Heterogeneous Graph Encoder and Contrastive Learning. Software dependencies vary in how strongly they reflect architectural cohesion, and heterogeneous GNNs outperform homogeneous ones on multi-relational software graphs [11]. We therefore use a heterogeneous GAT [5], which performs relation-specific message passing and applies attention to weight each connection. Each relation type has its own transformation and attention heads, and the outputs from all transformations are aggregated to produce the node embeddings.

We train the encoder using a contrastive objective inspired by Deep Graph Infomax (DGI) [6]. A corrupted view, $\tilde{G}$, is created by shuffling node features row-wise. The encoder generates embeddings H and $\tilde{H}$ for G and $\tilde{G}$, which

ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.

are contrasted using a discriminator $\mathcal{D}(\vec{h}, \vec{s}) = \vec{h}^\top W \vec{s}$, where $\vec{s}$ is a summary vector obtained by averaging the node embeddings of original graph, and W is a learnable weight matrix. The model is trained to score original node–summary pairs higher than corrupted ones using a binary cross-entropy objective.

Clustering. The resulting node embeddings (Z) are clustered using agglomerative clustering with Ward linkage. The number of clusters k is chosen from a search range by maximizing the silhouette score to obtain the final modules.

3.1 Experiments and Evaluation

Datasets. We evaluate CLAR on four open-source systems with available ground-truth (GT) architectures. Table 1 reports their basic properties and homophily ratio ($\mathcal{H}$)[1]. For JabRef and TeamMates, GT labels are produced by system experts at the SAEroCon workshops [11], while Bash and HDF follow the mappings provided in prior SAR studies [3].

Table 1: Dataset statistics and homophily.

System	Language	LoC	Files	Dependencies	Modules	$\mathcal{H}$
JabRef	Java	59K	1 131	18 426	6	0.54
TeamMates	Java	55K	778	17 818	15	0.29
Bash	C	115K	292	1 222	14	0.61
HDF	C	153K	153	1 239	15	0.45

Evaluation Metrics. We assess the alignment between the recovered and ground truth (GT) architectures using several standard architecture similarity measures: MoJoFM, A2A, and $\mathrm{C2C_{cvg}}$. We also report the Adjusted Rand Index (ARI), a clustering agreement measure that ranges from -0.5 to 1.

MoJoFM measures the normalized number of move and join operations required to transform the recovered architecture into the GT. A2A builds on this by considering a broader range of edit operations for entities and modules. $\mathrm{C2C_{cvg}}$ quantifies the proportion of GT clusters that are covered by clusters in the recovered architecture, based on a specified coverage threshold [3]. Higher values for these metrics indicate a better alignment between the architectures.

Experimental Configuration. We use a single heterogeneous GAT layer with one attention head, trained for 50 epochs using Adam with a learning rate of 10^{-3}. After training, we cluster the node embeddings, selecting k from [5, 20]. Each experiment is repeated 100 times with different random seeds. $\mathrm{C2C_{cvg}}$ is computed using a coverage threshold of 0.5, counting a module as matched only when more than half its entities appear in one cluster.

For comparison, we use SARIF [3] as a baseline. SARIF integrates dependency types, textual similarity, and folder co-location into a weighted graph and

[1] $\mathcal{H}$ measures the fraction of dependencies whose endpoints lie in the same cluster.

ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.

applies modularity-based community detection. We run the official implementation[2] with default parameters.

4 Results and Analysis

Table 2 reports the median scores of 100 runs across all datasets. On average across all systems and metrics, ARL improves over SARIF by 13.4 points, while ARL-E (with edge attributes) yields a slightly higher gain of 13.9 points.

Table 2: Similarity of recovered architectures to GT. Metrics: MoJoFM (M), A2A (A), $C2C_{cvg}$ (C), and ARI (I) are scaled by 100. Δ = ARL − SARIF and Δ-E = ARL-E − SARIF. The top value per metric is highlighted.

Tech.	Bash				HDF				Jabref				TeamMates			
	M	A	C	I	M	A	C	I	M	A	C	I	M	A	C	I
SARIF	76	84	13	39	62	87	27	37	65	80	0	7	76	87	33	39
ARL	92	96	69	76	77	88	32	47	91	85	12	24	83	87	33	33
ARL-E	91	96	69	76	78	88	32	48	91	86	15	29	84	87	33	31
Δ	+16	+12	+56	+37	+15	+1	+5	+10	+26	+5	+12	+18	+7	0	0	-6
Δ-E	+15	+12	+56	+37	+16	+1	+6	+11	+26	+5	+15	+22	+8	0	0	-8

Dataset-Specific Behavior. Bash and JabRef show the most considerable improvements, consistent with their moderate homophily ratios of 0.54 and 0.61, respectively. Most dependencies occur within the same module, providing strong signals for GAT. Consequently, MoJoFM increases from 76 to 92 in Bash and from 65 to 91 in JabRef, with corresponding gains in ARI. HDF shows a similar pattern, but with smaller margins, exhibiting an increase of 15-16 in MoJoFM. This suggests that dependency information is moderately informative here; edge attributes add some benefit but do not dominate performance.

TeamMates exhibits limited improvements. Its low homophily (0.29) indicates strong inter-module coupling, meaning dependency-based features provide weak architectural cues. Consequently, ARL yields modest improvements in MoJoFM, and other metrics remain close to SARIF. ARL-E does not help further, as edge attributes amplify noisy cross-module interactions.

Metrics Behavior. The largest gains appear in MoJoFM and ARI, especially in more homophilic systems. $C2C_{cvg}$ is inherently unstable because it counts a ground truth module as matched only when a single recovered cluster contains at least half of its entities at a coverage threshold of 0.5. Even a small split can cause the cluster to fall below this threshold, leading to a sharp drop in the score—particularly in systems with a few large modules. A2A remains uniformly high across methods and therefore provides limited discriminative value.

[2] https://github.com/anonymous2f4a9d/SARIF_FSE23

ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.

5 Conclusion

We presented ARL, a software architecture recovery approach that learns node embeddings using a heterogeneous GAT encoder over a multi-relational dependency graph. Clustering these embeddings yields coherent modular structures that align well with the ground truth. We also evaluated ARL-E, which extends ARL with edge attributes capturing member-level interaction types and interaction strength. While ARL-E provides small additional gains in some systems, its benefits remain limited in more heterotrophic systems where cross-module interactions dominate.

Future work will focus on lifting these learned representations into an explainable, symbolic layer to support the derivation of architectural constraints and provide justifiable reasoning. This direction aims to integrate neural and symbolic techniques for neuro-symbolic architectural analysis.

References

[1] Steffen Herbold, Christoph Knieke, Andreas Rausch, and Christian Schindler. Neurosymbolic architectural reasoning: Towards formal analysis through neural software architecture inference. *arXiv preprint arXiv:2503.16262*, 2025.

[2] Joshua Garcia, Ivo Krka, Chris Mattmann, and Nenad Medvidovic. Obtaining ground-truth software architectures. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 901–910. IEEE, 2013.

[3] Yiran Zhang, Zhengzi Xu, Chengwei Liu, Hongxu Chen, Jianwen Sun, Dong Qiu, and Yang Liu. Software architecture recovery with information fusion. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1535–1547, 2023.

[4] Luís C Lamb, Artur Garcez, Marco Gori, Marcelo Prates, Pedro Avelar, and Moshe Vardi. Graph neural networks meet neural-symbolic computing: A survey and perspective. *arXiv preprint arXiv:2003.00330*, 2020.

[5] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[6] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

[7] Federico Bianchi, Gaetano Rossiello, Luca Costabello, Matteo Palmonari, and Pasquale Minervini. Knowledge graph embeddings and explainable ai. *arXiv preprint arXiv:2004.14843*, 2020.

[8] Jiayi Chen, Zhixing Wang, Yuchen Jiang, Jun Pang, Tian Zhang, Minxue Pan, and Jianwen Sun. Negar: network embedding guided architecture recovery for software systems. In *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*, pages 367–376. IEEE, 2022.

[9] Satrio Adi Rukmono, Lina Ochoa, and Michel Chaudron. Deductive software architecture recovery via chain-of-thought prompting. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, pages 92–96, 2024.

[10] Hung Viet Nguyen. Depends: Multilingual dependency analysis tool. `https://github.com/multilang-depends/depends`, 2023. Accessed: 2025-08-17.

[11] Rakhshanda Jabeen, Morgan Ericsson, Jonas Nordqvist, and Anna Wingkvist. Graph convolution networks for mapping source code entities to architectural modules. In *22nd IEEE International Conference on Software Architecture (ICSA 2025), Odense, Denmark, March 31-April 4, 2025*. IEEE, 2025.