

EgoNEExT: Capturing Structural Node Roles through Ego-Network Optimal Transport

Ashkan Dehghan¹, Paweł Pralat¹, François Th  berge², and Mariam Walaa¹

¹ Toronto Metropolitan University, Toronto, ON, M5B 2K3 Canada
{ashkan.dehghan, pralat, mariam.walaa}@torontomu.ca

² Tutte Institute for Mathematics and Computing, Ottawa, ON, K1J 8K6 Canada
theberge@ieee.org

Abstract. Network embeddings map complex graph topologies into continuous vector spaces for downstream machine learning tasks. While classical node embeddings preserve spatial proximity, structural embeddings capture local topological roles regardless of network position. We adapt **NEExT**, a whole-graph embedding framework built on Wasserstein optimal transport, to construct structural node representations. By decomposing a network into a collection of local ego-nets, our method treats each ego-net as an individual graph instance and produces node-level structural vectors. On a real-world node classification task the ego-net representation outperforms the node’s own features and matches the best purpose-built structural embedding, laying the groundwork for a flexible, interpretable structural embedding pipeline, **EgoNEExT**.

Keywords: structural node embedding, NEExT, EgoNEExT

1 Introduction

In network science, the primary goal of many machine learning applications is to uncover hidden patterns, predict relationships, or analyse the dynamics of complex, interconnected systems. However, native graph-structured data is non-Euclidean, making it difficult to process with standard analytical algorithms. Network embeddings bridge this gap by mapping the rich topological information of a graph into a lower-dimensional, continuous geometric space. By assigning dense vectors to the nodes of a graph, embeddings translate complex relational dynamics into computationally tractable features. This transformation is fundamental for downstream predictive tasks, regardless of whether the ultimate goal is to understand a person’s role within a collaboration network, measure similarity between users on Amazon or Yelp, predict a protein’s behaviour in a biological network, or generate targeted recommendations on social media platforms.

While all node embeddings aim to extract useful features from graphs, they differ significantly in the specific topological properties they preserve. *Classical node embeddings* are designed around spatial proximity: they assign coordinates such that nearby nodes in geometric space are highly likely to share an edge or

exist in the same local community. Because they inherently capture the likelihood of direct connections, classical embeddings excel at link prediction, where missing edges can be inferred directly from spatial closeness. On the other hand, *structural node embeddings* focus on a node’s topological function rather than its specific location or neighbourhood identity. These embeddings capture information about a node’s global role or local structural topology (its ego-net), mapping nodes with similar structural roles (such as central hubs or connecting bridges) close together, even if they reside in completely disconnected parts of the network. Because different applications (e.g., node classification versus link prediction) require distinct properties to be preserved, practitioners often have to switch between the two families or rely on feature selection to capture the right structure. To learn more about embeddings, see, for example, [1] or Chapter 6 in [9]; for a specific application of structural graph embeddings in detecting bots in social networks, see [6].

In addition to representing individual nodes, many network science applications require analysing entire networks as distinct data instances. Complex real-world systems often manifest not as a single global graph, but as dynamic collections of networks, such as datasets of protein structures, financial transaction networks across different institutions, temporal snapshots of evolving social systems, or sets of subgraphs representing distinct communities. Performing predictive tasks on these systems, such as classifying chemical properties or detecting anomalous financial activity, calls for graph-level embeddings. By mapping an entire network into a single fixed-dimensional vector, whole-graph embeddings capture global structure, enabling machine learning pipelines to evaluate and compare entire networks as standard feature vectors. For more details on graph-level embeddings, see, for example, [13] or Chapter 9 in [9].

Historically, graph kernels served as the standard approach for graph-level classification and clustering. In this paradigm, similarity between pairs of networks is computed by recursively decomposing them into simpler substructures (such as random walks, shortest paths, or graphlets) and applying kernel functions for use in algorithms like Support Vector Machines (SVMs), as exemplified by the Weisfeiler–Lehman graph kernel framework [18]. While effective for similarity-based evaluation, traditional graph kernels have distinct limitations. First, they compute pairwise similarity scores rather than explicitly generating continuous vector representations for each network, making them difficult to integrate into broader vector-space machine learning models. Furthermore, they often suffer from high computational complexity and rely on fixed, hand-crafted substructures, which paved the way for modern whole-graph embedding approaches that learn representations automatically.

To address the limitations of traditional graph kernels, our framework, **NE-ExT** (Network Embedding Exploration Tool) [4, 5], uses optimal transport theory to construct explicit whole-graph vector representations. The pipeline begins by computing a base set of node features for a graph G (interpretable structural measurements in this paper). The resulting collection of n feature vectors for the n nodes of G is normalized to form a probability distribution over a metric space

equipped with a distance function, such as Euclidean distance. **NEExT** then employs the Wasserstein distance, also known in the literature as the Monge–Kantorovich–Rubinstein distance, earth mover’s distance, or optimal transport distance [19], to evaluate dissimilarity between graphs by calculating the distance between their corresponding probability distributions. Because the Wasserstein distance is a proper metric connected to the optimal transport problem [22], it measures the minimal cost of transforming one graph’s node distribution into another [20, 11]. Finally, **NEExT** projects these graphs into a D -dimensional vector space such that pairwise Wasserstein distances are preserved as faithfully as possible in the geometric distance between resulting vectors, yielding compact, graph-level embeddings ready for downstream machine learning tasks.

Building upon the flexibility of the **NEExT** framework, we adapt its whole-graph embedding methodology to construct structural node embeddings. The core intuition relies on local graph decomposition: for each node v in a network G with n nodes, we extract its k -hop ego-net, the sub-network induced by v and every node within k hops of it. This procedure transforms a single network G into a collection of n distinct subgraphs. By treating each ego-net as an individual graph instance, **NEExT** computes whole-graph representations for every local sub-network, effectively mapping each node v to a continuous vector that encodes its local structural context. As a result, nodes with topologically similar ego-nets are embedded close to one another in vector space, regardless of their physical distance or component separation within the global network, producing a structural node embedding.

In this short conference paper, we propose a simple, prototype strategy for generating structural node embeddings. We demonstrate its potential through empirical experiments on a real-world network, specifically focusing on a node classification task using ground-truth labels. The complete framework will add **NEExT**’s feature selection, in both supervised and unsupervised settings, to give a flexible, interpretable structural embedding model.

The paper is structured as follows. In Section 2 we introduce our prototype pipeline, **EgoNEExT**. Section 2.1 explains how **NEExT** turns a collection of m graphs into an embedding. Section 2.2 explains how structural node embedding reduces to embedding a collection of graphs, which **NEExT** already handles, and introduces the two bag constructions used here (k -hop censuses and random-walk surveys). The two feature scopes (local and global) are explained in Section 2.3, and the assembled pipeline is presented in Section 2.4. Section 3 evaluates **EgoNEExT** on a real-world node classification task: Section 3.1 tests whether ego-net representations carry more label signal than the node’s own features, and Section 3.2 places them among six node embedding methods from the literature (node2vec at two settings) evaluated under the same protocol. Section 4 concludes the paper and outlines future directions.

2 The NEE $\mathbf{xT}$ Framework and Ego-Net Representations

This section introduces the method that every experiment in this paper runs. The base layer is **NEE $\mathbf{xT}$** , an open-source Python framework for embedding *collections of graphs* via user-defined, interpretable structural node features [4, 5]. On top of it, this work introduces **EgoNEE $\mathbf{xT}$** , a structural node embedding built on that framework: for the purpose of embedding the nodes of a single graph, one labelled subgraph is carved out of the graph around every node, and the resulting collection of ego-nets is embedded exactly as **NEE $\mathbf{xT}$** embeds any graph collection. Nodes with similar ego-nets thereby receive similar vectors, wherever they sit in the graph.

2.1 Graph Collections, Structural Features, and Embeddings

NEE $\mathbf{xT}$ operates on a collection $\mathcal{G} = \{G_1, \dots, G_m\}$ of m graphs $G_j = (V_j, E_j)$, each optionally carrying a graph-level label, and follows a fixed pipeline: compute structural features for every node, summarize each graph as a distribution of its members’ feature vectors, embed those distributions in a common vector space, and hand the embeddings to any downstream model, here a classifier. The design premise of **NEE $\mathbf{xT}$** [4] is that features are *chosen and interpretable* rather than learned: every quantity entering the representation is a named graph measurement. The embedding used in this study is a linear map of those measurements, so every coordinate of a node’s representation remains traceable to the features that produced it. **NEE $\mathbf{xT}$** also provides exact and entropic (Sinkhorn) optimal-transport vectorizations and unsupervised graph-neural-network encoders (GCN, GraphSAGE, GIN), so richer embeddings are a configuration choice within the same pipeline.

Structural node features. For a node v in a graph G , **NEE $\mathbf{xT}$** computes a vector of p structural features. The feature set is user-defined: a library of standard measures is provided and continues to grow, and users can register their own. The measures available at the time of writing are PageRank; degree, closeness, betweenness, eigenvector, and load centrality; the clustering coefficient; local efficiency; LSME, a neighbourhood degree-profile feature; basic expansion, the growth profile of neighbourhood shells; and the community-aware feature β^* [10]. Each feature can additionally be aggregated over hop shells: with vector length L , feature f contributes the components

$$\phi_f^{(0)}(v) = f(v), \quad \phi_f^{(i)}(v) = \frac{1}{|N_i(v)|} \sum_{u \in N_i(v)} f(u) \quad (1 \leq i \leq L-1),$$

where $N_i(v)$ is the set of nodes at hop distance exactly i from v (when the shell $N_i(v)$ is empty, we simply set $\phi_f^{(i)}(v) = 0$); the one exception is basic expansion, whose L components are the shell-growth ratios themselves rather than shell means. The vector length therefore sets how much of the neighbourhood enters through the features themselves, as opposed to through the ego-net construction

of Section 2.2. At $L = 1$ each node is described by its own p values, $\phi(v) \in \mathbb{R}^p$; note that at this length LSME, degree centrality, and basic expansion all reduce to re-scalings of the node’s degree, and coincide after standardization whenever the features are computed on a single graph. Feature columns are standardized (z -scored) before embedding; the frame in which the standardization is fit depends on the feature scope (see Section 2.3).

Distribution embeddings. Each graph in the collection is treated as a weighted empirical distribution over feature space, $\mu_j = \sum_{u \in V_j} w_{ju} \delta_{\phi(u)}$, with uniform mass $w_{ju} = 1/|V_j|$ unless the graph carries explicit member weights (see Section 2.2). **NEExT** embeds these distributions jointly into $\mathbb{R}^D$ so that distances between embedding rows track the Wasserstein distances between the distributions (see Section 1). The embedding used throughout this paper is the approximate Wasserstein vectorization of the **vectorizers** library [21], a linear method that collapses each distribution to its mass-weighted mean feature vector, so that distances between embedding rows reflect distances between the distributions’ mean feature vectors and higher-moment shape is not represented; the exact and Sinkhorn variants recover it at greater cost. The embedding dimension D is a free parameter.

2.2 Node Classification as Graph-collection Classification

EgoNEExT represents a node by the structure of its neighbourhood rather than by its own feature vector alone, and the question this paper studies is when that neighbourhood carries more label signal than the node itself. The construction is as follows. Let $G = (V, E)$ carry a node labelling $y : V \rightarrow \mathcal{Y}$. For each node v we extract a subgraph around it (its *bag*), attach v ’s label to the bag as its graph label, and collect the bags into a graph collection:

$$\mathcal{G}(G, k) = \{ (S_k(v), y(v)) : v \in V \}, \quad S_k(v) = G[N_{\leq k}(v)],$$

where $N_{\leq k}(v)$ is the set of nodes within k hops of v (including v) and $G[U]$ denotes the subgraph induced by the node set U . Node classification thereby *becomes* graph-collection classification: the pipeline of Section 2.1 runs on the bag collection as on any other, and each bag’s predicted graph label is the prediction for its centre. At $k = 0$ the bag is the singleton graph on v , $S_0(v) = G[\{v\}] = (\{v\}, \emptyset)$, whose feature content under the global scope of Section 2.3 is exactly the node’s own full-graph profile. The node-features baseline of Section 3 feeds precisely this representation to the classifier, so in feature content the baseline is the $k = 0$ limit of the ego-net family rather than an unrelated method.

Census and survey bags. Two bag constructions are used in this paper. The k -hop ego-net is a *census*: every node within k hops is a member, with equal weight. Its boundary is sharp and its size is dictated by degree: on a graph with hubs, a single high-degree neighbour can pull hundreds of nodes into a bag at $k = 2$, while a leaf’s bag stays tiny. A complementary *survey* construction uses

random walks to define a smoother, degree-adaptive notion of locality. From the centre v we run n_w walks of length ℓ ; at each step the walker returns to v with restart probability α and otherwise moves to a uniformly random neighbour. Let $c_v(u)$ be the number of visits to u across all of v ’s walks. The restart is what shapes the bag: the visit frequencies estimate the personalized PageRank vector of v , so mass concentrates near the centre and decays along the graph’s own geometry, with α setting the reach (a small α lets walks travel further before returning); the number of walks n_w sets the sampling budget of that estimate, and $\alpha = 0$ recovers restart-free random walks. The bag members are the nodes with $c_v(u) \geq \tau$, with the centre always retained, and each member carries a membership weight

$$w_{vu} = \frac{c_v(u)}{\sum_{u'} c_v(u')}$$

with the sum running over the retained members, which the embedding consumes as the bag’s distribution mass in place of uniform weights. The visit floor τ removes nodes visited fewer than τ times: without it, the one-visit fringe inflates the bag while carrying almost no mass, so it adds cost without adding signal. A walk bag therefore trades the census’s crisp hop boundary for smooth, community-adaptive locality, at the price that member count no longer equals effective sample size. Because τ is an absolute count rather than a fraction of visits, its effect depends on the total visit budget $n_w \ell$; with n_w fixed, the three parameters (ℓ, α, τ) set how far a bag reaches, and Section 3 evaluates three settings, from tight to wide, alongside the hop censuses.

2.3 Feature Scopes: What the Members are Measured Against

A bag member’s structural features can be computed against two different frames of reference, and the choice changes what the bag’s distribution means. Under the **local** scope, features are computed *inside* the extracted bag: $\phi^{\text{loc}}(u \mid S_k(v))$ measures u ’s role within the bag (a member’s degree is its number of neighbours within the ego-net, its closeness is measured over paths inside the ego-net, and so on), so the distribution describes the bag’s internal shape, and the same node contributes different values to different bags. Under the **global** scope, features are computed *once* on the source graph G and projected onto bag members: every member contributes its true full-graph values $\phi^{\text{glob}}(u) = \phi(u \mid G)$, identical in every bag that contains it, and the distribution becomes a poll of the members’ global roles. Standardization respects the frame: local-scope columns are z -scored across all bag rows (so nodes appearing in many bags contribute one row per bag), while global-scope columns are z -scored on the source graph’s rows *before* projection; under the global scope, bag overlap therefore cannot skew the scaling.

The two scopes answer different questions. The local scope describes the internal shape of the neighbourhood, answering “what does the neighbourhood look like?” The global scope samples the graph’s nodes with their global roles, answering “who is in the neighbourhood?” Which scope predicts a label better depends on the dataset. The experiments of Section 3 run both. The scopes also

differ sharply in cost: the global scope pays one feature pass per source graph regardless of bag count, while the local scope pays one pass per bag.

2.4 The Assembled Pipeline

Figure 1 assembles the pieces. From a labelled graph, **EgoNEExT** builds one bag per node (a k -hop census or a random-walk survey); members are described by p structural features under a chosen scope; each bag becomes a weighted distribution in feature space; the Wasserstein vectorization embeds all bags jointly in $\mathbb{R}^D$; and a classifier maps embeddings to centre labels.

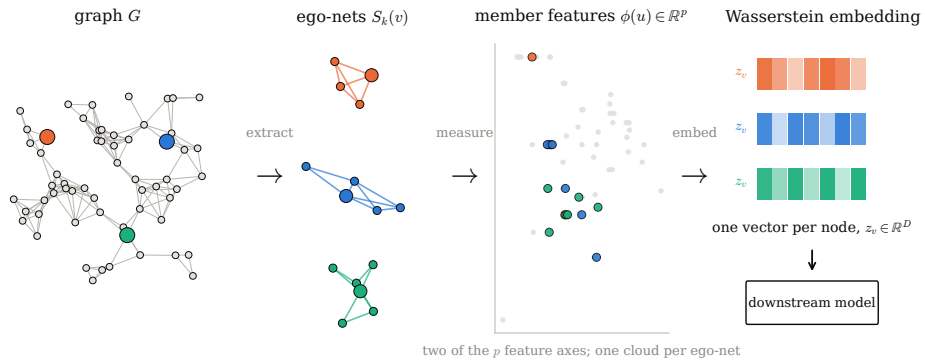


Fig. 1. The **EgoNEExT** pipeline. A graph is decomposed into one ego-net per node, each ego-net is summarized as a distribution of its members’ structural feature vectors in $\mathbb{R}^p$, and the Wasserstein vectorization maps every distribution to one vector in a shared D -dimensional space. The node’s label travels with its ego-net, so any downstream model can be trained on the resulting node embeddings.

3 Experiments

Dataset and protocol. We evaluate **EgoNEExT** on Roman Empire [15], a word graph built from the Roman Empire Wikipedia article: nodes are words, edges connect consecutive or dependency-linked words, and each word’s label is its syntactic role (22,662 nodes, 32,927 edges, mean degree 2.9, 18 imbalanced classes with majority share 0.14). The graph is sparse and near-chain-like, so neighbourhoods stay small at every radius ($k = 5$ covers roughly 0.1% of the graph), and its labels are *heterophilous*: a word’s role is written in its context rather than in its own connectivity, which makes it a natural probe for whether the neighbourhood can provide additional signal than the node itself. Only the graph structure and the class label are used; the dataset’s native 300-dimensional fastText node attributes are excluded, so every result measures purely structural signal.

Every method is evaluated on the same stratified sample of 3,000 centres under 10 stratified 70/30 splits shared across all methods, so comparisons are paired. We report macro-F1, which weighs every class equally (mean $\pm$ std over splits; accuracy in small parentheses), and two-sided paired t -tests over the splits for the small-margin comparisons. Two references bracket every method: a *permutation floor* (the identical pipeline trained on shuffled labels; majority guessing scores macro-F1 0.014) and a *node-features baseline* (the centre’s own full-graph feature values fed directly to the classifier, the $k = 0$ limit of Section 2.2). The **EgoNEExT** configuration is fixed throughout: all 11 features of the library at vector length $L = 1$ under both feature scopes, the approximate Wasserstein embedding at $D = 11$, and an XGBoost classifier [3].

3.1 Ego-Net Representations

We build one bag per centre under both constructions of Section 2.2: k -hop ego-nets for $k \in \{1, \dots, 5\}$ (median bag sizes 4, 7, 13, 19, 25) and random-walk bags with $n_w = 100$ walks at three locality settings, tight ($\ell = 5$, $\alpha = 0.30$, $\tau = 3$), default ($\ell = 10$, $\alpha = 0.15$, $\tau = 3$) and wide ($\ell = 20$, $\alpha = 0.05$, $\tau = 2$), with median bag sizes 9, 18 and 34. Table 1 and Figure 2 report every construction and scope.

Construction	Bag	Size	Local	Global
Permutation floor	—	—	0.052 $\pm$ 0.005 (0.100)	
Node features	—	—	0.137 $\pm$ 0.012 (0.217)	
k -hop ego-net	$k=1$	4	0.150 $\pm$ 0.007 (0.306)	0.128 $\pm$ 0.009 (0.228)
	$k=2$	7	0.158 $\pm$ 0.012 (0.262)	0.080 $\pm$ 0.011 (0.149)
	$k=3$	13	0.097 $\pm$ 0.006 (0.180)	0.062 $\pm$ 0.006 (0.125)
	$k=4$	19	0.068 $\pm$ 0.007 (0.137)	0.058 $\pm$ 0.005 (0.118)
	$k=5$	25	0.066 $\pm$ 0.008 (0.133)	0.058 $\pm$ 0.005 (0.116)
Random-walk bag	tight	9	0.157 $\pm$ 0.016 (0.266)	0.134 $\pm$ 0.011 (0.243)
	default	18	0.135 $\pm$ 0.014 (0.235)	0.120 $\pm$ 0.009 (0.217)
	wide	34	0.091 $\pm$ 0.007 (0.172)	0.097 $\pm$ 0.008 (0.176)

Table 1. Roman Empire: macro-F1 (mean $\pm$ std over 10 stratified 70/30 splits; accuracy in parentheses) per bag construction and feature scope, with the median bag size, bracketed by the permutation floor and the node-features baseline (scope-independent, hence one value). Best overall in bold.

Here are some conclusions based on the above experiment. (i) *The neighbourhood beats the node.* Local $k = 2$ (0.158, $p = 0.014$ against the baseline), local tight walk bags (0.157, $p = 0.003$) and local $k = 1$ (0.150) all exceed the node-features baseline (0.137). The margin is modest in macro-F1 (+0.013 to +0.021) and much larger in accuracy (0.306 against 0.217 at $k = 1$), so part of the neighbourhood’s advantage is better handling of the majority classes. (ii) *Performance decays with radius by dilution, not saturation:* macro-F1 falls monotonically beyond $k = 2$ (0.158 $\rightarrow$ 0.066 at $k = 5$) even though the widest bags cover only $\sim 0.1\%$ of the graph, so the bags are not exhausting the graph; rather, members three hops away belong to other clauses and dilute the signal.

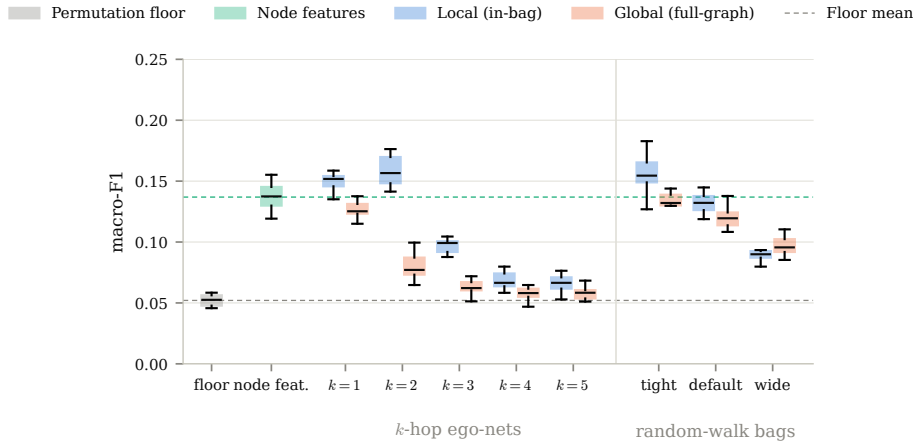


Fig. 2. Roman Empire: macro-F1 across the 10 shared splits for both bag constructions under the local (in-bag) and global (full-graph) feature scopes, bracketed by the permutation floor (grey) and the node-features baseline (green). The dashed lines mark the floor mean (grey) and the node-features mean (green).

(iii) *The local scope wins at every construction but the wide walk bag* (0.091 against 0.097), and the global scope collapses toward the floor beyond $k = 1$: on a sparse, nearly regular graph most nodes’ full-graph feature vectors are almost identical, so a bag of global values is nearly constant, whereas the local scope fingerprints the shape of the small fragment around the word. (iv) *Walk bags obey the locality ordering* (tight > default > wide in both scopes), but no walk setting exceeds the best hop census.

3.2 Comparison with Node Embedding Methods

We place **EgoNEE_{XT}** among six node embedding methods from the literature (node2vec at two settings), evaluated under the identical protocol: the same centres, splits and classifier, with a permutation floor computed per method. *Proximity* methods place nodes that co-occur on short random walks close together, so the embedding records where a node sits in the graph: DeepWalk [14] and node2vec [8], the latter at a breadth-first (0.25, 4) and a depth-first (4, 0.25) setting of its return and in-out parameters. *Structural* methods place nodes with similar surroundings close together regardless of their distance in the graph: struc2vec [17], Role2Vec [2] and GraphWave [7]. The walk methods use our own walk sampler and a gensim skip-gram [12, 16]; the three structural methods are our own implementations of the original papers. Every embedding is fit unsupervised on the full graph and, like the bags, sees no label. The five embeddings learned from walks or roles (all but GraphWave) have width $D = 16$, wider than the bags’ $D = 11$; GraphWave keeps its native 100 features. Table 2 and Figure 3 report the comparison.

Family	Method	macro-F1
References	Permutation floor	0.052 ± 0.005
	Node features	0.137 ± 0.012
	Best ego-net bag (local $k=2$)	0.158 ± 0.012
Proximity	DeepWalk	$0.048 \pm 0.005^{\ddagger}$
	node2vec (BFS)	$0.053 \pm 0.008^{\ddagger}$
	node2vec (DFS)	$0.058 \pm 0.004^{\ddagger}$
Structural	struc2vec	$0.081 \pm 0.009^{\ddagger}$
	Role2Vec	$0.076 \pm 0.006^{\ddagger}$
	GraphWave	0.148 ± 0.009

Table 2. Roman Empire: node embedding methods against the references of Table 1, macro-F1 (mean $\pm$ std over the same 10 splits), $D = 16$ for the five learned embeddings and the native 100 features for GraphWave, embedding seed 42. The floor row is the bag pipeline’s; every embedding’s own paired floor lies within 0.026 of it. ‡ marks a method below the best bag at $p < 0.05$ (two-sided paired t -test over the 10 splits); GraphWave, the one unmarked method, is statistically indistinguishable from it. Best in bold.

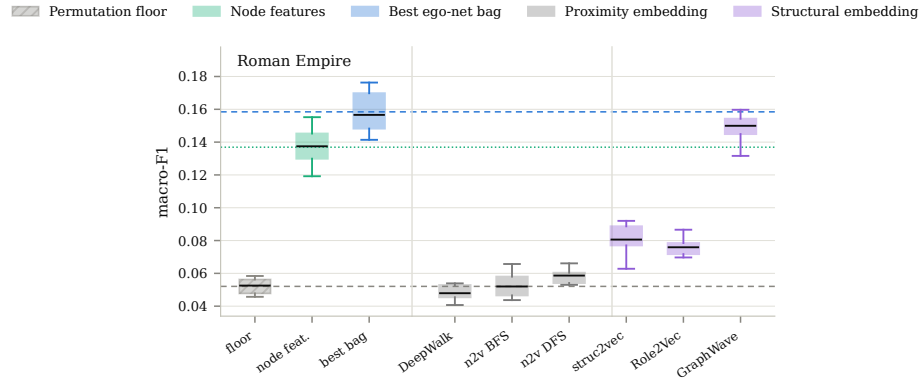


Fig. 3. Roman Empire: macro-F1 over the 10 shared splits for the six node embedding methods (grey: proximity family, violet: structural family) next to the three references drawn the same way: the permutation floor (hatched), the node features (green) and the best ego-net bag (blue). Horizontal lines mark the mean of each reference.

DeepWalk and both node2vec settings sit on the permutation floor (0.048 to 0.058 against 0.052): on heterophilous labels, where a node sits in the graph carries no information about its role. GraphWave (0.148) is the only embedding above the node features (0.137; $p = 0.019$) and is statistically indistinguishable from the best bag (0.158; $p = 0.054$), while struc2vec (0.081) and Role2Vec (0.076) fall between the floor and the node features. The ego-net representation therefore behaves as a member of the structural family: on a label written in the shape of a small fragment, it matches the best purpose-built role embedding

and beats the rest. Further datasets, a mechanism analysis and a leakage-free region-holdout evaluation are deferred to the journal version.

4 Conclusions

In this short conference paper, we introduced **EgoNEExT**, a prototype pipeline that reframes structural node embedding as a whole-graph embedding task, which the **NEExT** framework already handles. On Roman Empire the ego-net representation beats the node’s own features and matches the best purpose-built role embedding under a paired protocol. We outline below the extensions under development for the journal version of this work.

While the current prototype fixes the bag construction by hand (a hop radius or one of three walk settings), future iterations will let the construction policy and its parameters be chosen for the task: set by experienced users, or optimized by the framework in both supervised (when ground-truth labels are available) and unsupervised settings.

Similarly, while the current prototype retains all extracted node features without filtering, the **NEExT** framework natively supports feature selection strategies across both supervised and unsupervised paradigms. Integrating these capabilities into **EgoNEExT** will enable the generation of structural node embeddings that are individually tailored to specific graph topologies or targeted downstream machine learning tasks.

Finally, we have conducted illustrative experiments on synthetic benchmark networks and three additional real-world datasets to evaluate key properties of **EgoNEExT**. Due to space constraints, these results are omitted from this short paper. Moving forward, we plan to expand this experimental suite to systematically validate the framework’s extraction policies and hyperparameter choices. These comprehensive findings, along with fully reproducible code, will be featured in the expanded journal extension and made publicly available in our GitHub repository.

References

1. Aggarwal, M., Murty, M.N.: Machine learning in social networks: embedding nodes, edges, communities, and graphs. Springer Nature (2020)
2. Ahmed, N.K., Rossi, R.A., Lee, J.B., Willke, T.L., Zhou, R., Kong, X., Eldardiry, H.: Learning role-based graph embeddings. arXiv preprint arXiv:1802.02896 (2018)
3. Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794 (2016)
4. Dehghan, A., Pralat, P., Théberge, F.: Network embedding exploration tool (NE-ExT). In: Modelling and Mining Networks (WAW 2024), Lecture Notes in Computer Science, pp. 65–79. Springer (2024)
5. Dehghan, A., Pralat, P., Théberge, F.: Network embedding exploration tool (NE-ExT). arXiv preprint arXiv:2503.15853 (2025)

6. Dehghan, A., Siuta, K., Skorupka, A., Dubey, A., Betlen, A., Miller, D., Xu, W., Kamiński, B., Prałat, P.: Detecting bots in social-networks using node and structural embeddings. *Journal of Big Data* **10**(1), 119 (2023)
7. Donnat, C., Zitnik, M., Hallac, D., Leskovec, J.: Learning structural node embeddings via diffusion wavelets. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1320–1329 (2018)
8. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 855–864 (2016)
9. Kaminski, B., Prałat, P., Théberge, F.: *Mining complex networks*. Chapman & Hall/CRC (2021)
10. Kamiński, B., Prałat, P., Théberge, F., Zając, S.: Predicting properties of nodes via community-aware features. *Social Network Analysis and Mining* **14**(1) (2024)
11. Kolouri, S., Naderializadeh, N., Rohde, G.K., Hoffmann, H.: Wasserstein embedding for graph learning. In: *International Conference on Learning Representations* (2021)
12. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: *Advances in Neural Information Processing Systems* 26, pp. 3111–3119 (2013)
13. Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., Jaiswal, S.: graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005* (2017)
14. Perozzi, B., Al-Rfou, R., Skiena, S.: DeepWalk: Online learning of social representations. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 701–710 (2014)
15. Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A., Prokhorenkova, L.: A critical look at the evaluation of GNNs under heterophily: Are we really making progress? In: *The Eleventh International Conference on Learning Representations (ICLR)* (2023)
16. Řehůřek, R., Sojka, P.: Software framework for topic modelling with large corpora. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pp. 45–50 (2010)
17. Ribeiro, L.F.R., Saverese, P.H.P., Figueiredo, D.R.: struc2vec: Learning node representations from structural identity. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 385–394 (2017)
18. Shervashidze, N., Schweitzer, P., Van Leeuwen, E.J., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* **12**(9) (2011)
19. Sommerfeld, M., Munk, A.: Inference for empirical wasserstein distances on finite spaces. *Journal of the Royal Statistical Society Series B: Statistical Methodology* **80**(1), 219–238 (2018)
20. Togninalli, M., Ghisu, E., Llinares-López, F., Rieck, B., Borgwardt, K.: Wasserstein weisfeiler-lehman graph kernels. *Advances in neural information processing systems* **32** (2019)
21. Tutte Institute for Mathematics and Computing: Vectorizers: vectorization of assorted kinds of data. <https://github.com/TutteInstitute/vectorizers> (2021). Python library
22. Villani, C., et al.: *Optimal transport: old and new*, vol. 338. Springer (2009)