How Powerful Are Classic Graph Neural Networks? A Survey

Spiros Maggioros*†
Spiros.Maggioros@pennmedicine.upenn.edu

Reinti Pasai*
el21801@mail.ntua.gr

Eleni Nasopoulou*
el21087@mail.ntua.gr

*National Technical University of Athens, Athens, Greece
†AI²D Center, University of Pennsylvania, Philadelphia, PA, USA

***Abstract*—Graph-structured data is ubiquitous across scientific domains, yet effectively processing its complex topology remains a fundamental challenge. Graph representation learning addresses this by mapping variable-sized networks into fixed vector spaces, enabling standard analytical workflows. In this survey, we perform a comprehensive empirical analysis of diverse embedding techniques, ranging form classic unsupervised models like Graph2Vec, NetLSD, and DeepWalk to Graph Neural Networks, including GIN, GCN, GAT, and PNA. We benchmark these architectures on real-world datasets spanning social media and bioinformatics to assess their adaptability to different graph characteristics. Beyond standard classification and clustering tasks, our methodology rigorously evaluates computational efficiency, embedding dimensionality trade-offs, and resilience to structural perturbations. Through this multi-faceted investigation, we aim to derive comparative insights into the operational characteristics and stability of these distinct representational paradigms. Our code to replicate all of our experiments is open sourced here: https://github.com/spirosmaggioros/information_systems**

***Index Terms*—Graphs, Graph Neural Networks, Kernels, Deep Learning, Representation Learning**

I. Introduction

Graph-structured data appear across diverse domains from molecular chemistry and protein networks to social media and cybersecurity systems. The fundamental challenge lies in transforming graph structures into fixed-dimensional vectors amenable to machine learning algorithms. Graph embedding methods address this by mapping graphs into continuous vector spaces that preserve structural and semantic properties while enabling efficient computation. Traditional graph analysis relies on graph kernels that use handcrafted features such as shortest paths, graphlets, and random walks [1]. While theoretically sound, these approaches suffer from poor generalization across domains, require domain expertise for feature engineering, and fail to capture complex patterns in large-scale networks. Moreover, kernel methods typically lack explicit embeddings, restricting compatibility with general-purpose learning algorithms. Recent advances in deep learning have enabled a shift toward data-driven graph embeddings. Inspired by natural language processing techniques that learn distributed representations of words and documents [2], [3] modern approaches treat graphs as documents and subgraphs as words. This paradigm encompasses unsupervised methods like Graph2Vec [4] and DeepWalk [5] that learn task-agnostic representations.

Graph2Vec, DeepWalk, and NetLSD [6] are unsupervised methods that learn exclusively from graph topology without node features or labels. This structural focus enables rapid training and high computational efficiency, making them suitable for large-scale applications where labeled data is unavailable. However, ignoring node attributes typically yields lower classification accuracy than supervised methods. These embeddings often serve as auxiliary features for node-level prediction or initialization for supervised models. This work evaluates these trade-offs through empirical benchmarking across classification, clustering, efficiency, and robustness.

First introduced by Micheli, 2009 [7] and Scarselli et al., 2008 [8] as a form of recurrent neural network [9], Graph Neural Networks (GNNs) have recently become a widely used method to tackle problems like classification and link prediction tasks in domains like social networks, telecommunication networks, biological networks or brain connectomes [10] where other learning techniques fail. Almost all modern message-passing GNN models use convolutional or attentional layers and an aggregation process to compute latent embeddings. Thus, GNNs excel when applied to networks where nodes include features, in contrast with graph representation learning models that only capture the network structure. Despite their current appeal, GNNs are criticized by many for their limited expressiveness and generalization. Xu et al. [11] proved that GNNs are as powerful as the 1-WL algorithm and highlighted the minimal discriminative power of commonly used aggregators. Currently, GNN models occupy top positions on the Open Graph Benchmark (OGB) [12] and Long-Range Graph Benchmark (LRGB) [13] leaderboards. Nevertheless, classic GNNs still lead the leaderboard in some of these benchmarks [14]. In this work, we are going to evaluate a number of these models on social, bioinformatics and small molecules networks evaluating classification performance, clustering quality, computational efficiency, and robustness to perturbations. Our contributions include comprehensive empirical comparison under controlled

conditions, investigation of embedding dimensionality trade-offs, and assessment of robustness characteristics for real-world deployment.

II. Related Work

A. Graph2Vec

Graph2Vec [4] addresses fundamental limitations of node level embedding aggregation by extending document embedding approaches, particularly doc2vec [2], [15], to whole-graph representation learning. The method treats entire graphs as documents and rooted subgraphs extracted through Weisfeiler Lehman relabeling [1] as vocabulary words, enabling unsupervised embedding via skip-gram training [2]. This approach represents a significant departure from traditional graph kernels that rely on handcrafted structural features, instead learning data-driven representations adapted to specific graph corpora. On benchmark graph classification datasets, Graph2Vec achieves 83.15% accuracy on MUTAG, 60.17% on PTC, and 73.30% on PROTEINS, substantially outperforming node-level aggregation approaches by over 10 percentage points [4]. The method demonstrates particular strength on large-scale datasets where data-driven learning shows clear advantages. On Android malware detection involving 10,560 API dependency graphs with average size of 2,637 nodes, Graph2Vec achieves 99.03% accuracy, surpassing Deep WL kernels (98.16%) [16], WL kernels (97.12%) [1], and substantially outperforming node2vec (81.25%) and sub2vec (76.83%) [4], [17]. This 18 percentage point improvement over node2vec aggregation suggests that global structural properties cannot be adequately recovered through local node representations alone.

However, Graph2Vec exhibits mixed performance across different dataset characteristics. On NCI1 and NCI109 molecular datasets, it achieves 73.22% and 74.26% accuracy respectively, compared to approximately 80% for WL kernels [1]. This performance gap suggests that Graph2Vec's data-driven approach requires sufficient training examples to surpass carefully designed handcrafted features, particularly on smaller datasets with limited structural diversity. Computational efficiency analysis reveals Graph2Vec occupies a middle ground: slower than node2vec due to subgraph extraction overhead, yet significantly faster than sub2vec which requires extensive random walk sampling. The method generalizes well to unseen graphs when trained on large, diverse datasets but struggles with limited training data or low structural variation.

B. DeepWalk

DeepWalk [5] pioneered the application of neural language modeling to network representation learning by establishing a formal connection between graph topology and natural language structure. The method treats truncated random walks on graphs as sentences in a corpus, recognizing that vertex sequences exhibit power-law frequency distributions similar to word distributions in natural language. This insight enables direct application of skip-gram models [2] to learn vertex embeddings that capture neighborhood relationships and community structure.

Originally designed for node-level tasks, DeepWalk demonstrates substantial improvements over spectral clustering methods on social network datasets. DeepWalk achieves up to 10% higher F1-scores on social networks such as BlogCatalog, Flickr, and YouTube, and in some configurations (e.g., on Flickr) it matches or surpasses baselines while using 60% less labeled training data [5]. DeepWalk's online learning algorithm and trivial parallelization enable exceptional scalability, processing networks with millions of nodes efficiently. This scalability makes DeepWalk applicable to web-scale graphs where batch spectral methods become computationally prohibitive.

DeepWalk established the foundation for subsequent node embedding methods, most notably node2vec [18], which extends DeepWalk with biased random walk strategies controlled by return parameter p and in-out parameter q. Node2vec achieves 22.3% relative improvement over DeepWalk in Macro-F1 scores on BlogCatalog and 21.8% on Wikipedia by learning appropriate exploration strategies that balance breadth-first and depth-first search [18]. For link prediction tasks, node2vec outperforms DeepWalk by up to 3.8% in AUC scores across multiple network types. However, DeepWalk's uniform random walk strategy provides limited control over neighborhood exploration, potentially missing important structural patterns that biased strategies can capture.

A critical limitation emerges when applying DeepWalk to whole-graph tasks. Since DeepWalk produces node-level embeddings, adapting it for graph-level representation requires aggregation strategies such as averaging or pooling. Simply averaging DeepWalk or node2vec embeddings yields only 81.25% accuracy on Android malware detection [4], an 18 percentage point gap compared to methods designed explicitly for graph-level representation. This substantial performance difference demonstrates that global structural properties are not adequately captured through local node aggregation, motivating the development of whole-graph embedding approaches like Graph2Vec that directly learn graph-level representations.

C. Graph Neural Networks

Due to deep learning being a defacto standard for various tasks, a lot of effort has been put to generalize neural networks to graph structured data. A pioneer modern work was proposed by Kipf and Welling [9], GCNs are still used to this day due to their scalability. Another big step was made by veličković et al. with Graph Attentional Networks(GAT) [10] that leveraged masked self-attentional layers enabling nodes to attend over their neighborhood's features. This was later improved by Brody et al. [19] proposing GATv2, improving the average error by 11.5% in some tasks. In 2019, Xu et al. [11] proposed Graph Isomorphism Networks(GIN), proving that GNNs are at most as powerful as the 1-WL test and identifying graph structures that aforementioned GNNs can't distinguish. Corso et al. [20], taking advantage of the analysis presented at [11], introduced Principal Neighborhood Aggregation(PNA), a

architecture that combines multiple aggregators with degree-scalers, proving that using a single aggregator fails to distinguish simple graph structures. All Aforementioned GNNs make up the "classic" models, modern GNN models leverage SSL [21], [22], which is particularly useful in graph datasets where we might have text or images as input and labels are not always known. The GIANT framework proposed at [21] uses XR-Transformers, a model that is considered state-of-the-art, the resulting transformer is then used as ans encoder to generate numerical node features for each node, the encoder can change to address other data forms like images, where a CNN can be used. Though the GIANT framework dominates node property prediction leaderboards, it's not widely used for graph-level tasks, and interestingly so, work proposed by [14] showed that classic GNN models can still produce strong results at graph-level tasks when used with edge features, normalization, dropout, residual connections, feed-forward networks and positional encoding.

D. NetLSD

To address the ubiquitous yet computationally challenging task of graph comparison, Tsitsulin et al. [6] introduced the Network Laplacian Spectral Descriptor (NetLSD). While ideal comparison measures must be permutation-invariant, size-invariant, and scale-adaptive, prior methods have struggled to satisfy these desiderata simultaneously. For instance, Graph Edit Distance (GED) [23] is NP-hard [24] and computationally prohibitive for large collections, while alternative graph kernels [25]–[30] and statistical representations like FGSD [31] often lack scalability. Grounded in spectral graph theory and drawing on the physical analogy by Kac [32] regarding whether one can "hear the shape" of an object, NetLSD extracts a compact, continuous graph signature derived from the solution of the heat [33], [34] or wave [35] kernel involving the normalized Laplacian matrix. This approach ensures the method is not only expressive, inheriting the formal properties of the Laplacian spectrum commonly utilized in graph mining [36], [37], but also computationally efficient, enabling constant-time similarity computations that outperform previous works in both expressiveness and efficiency.

NetLSD [6] demonstrates scalability and robustness across diverse graph analysis tasks, consistently outperforming representation-based baselines such as NetSimile [38] and FGSD [31]. The method proves particularly effective in distinguishing global structural patterns, such as community organization within Stochastic Block Models (SBM). In scenarios where graph size increases, NetLSD's accuracy improves, rising from 57.40% to 84.63%, whereas competitors like FGSD see performance degrade from 58.00% to 51.57%, indicating that prior methods struggle to capture global structure beyond local variations. This structural expressiveness is matched by superior computational efficiency. On the massive REDDIT-L dataset containing graphs with over one million nodes, NetLSD computes spectral signatures in approximately 16 minutes using an eigenvalue approximation strategy. In contrast, both NetSimile [38] and FGSD [31] fail to complete processing within a 24-hour cutoff, establishing NetLSD as a uniquely viable solution for comparing web-scale graphs where traditional kernel methods and statistical aggregations become computationally prohibitive.

However, NetLSD requires precise tuning. For instance, the wave kernel favors complete-graph normalization while the heat kernel requires empty-graph normalization [6]. The method also inherits spectral limitations like cospectrality and "abnormal sensitivity," where minor perturbations cause significant spectral shifts [39]. Additionally, eigenvalue approximation faces diminishing returns—doubling steps from 200 to 400 yields negligible accuracy gains (rising from 0.980 to 0.985) [40], and newer architectures like FeatherGraph demonstrate superior runtime and performance [41]. Thus, while NetLSD resolves the scalability bottlenecks of Graph Edit Distance [23] and FGSD [31], successful deployment necessitates careful calibration to mitigate these intrinsic constraints.

III. General Notation

We define a Graph as $G(V, E)$ Where $V = \{1, 2, ..., n\}$ represents the nodes and $E = \{(u_i, v_i), ..., (u_j, v_j)\}$ represents the edges. Each graph can appear as an adjacency list where we map nodes with lists of other nodes or with an adjacency matrix $A \in \{0, 1\}^{n \times n}$ where $A_{ij} = 1 : (u_i, u_j) \in E$. We also define a node labeling function λ_i that maps nodes to a class l and a graph labeling function μ_i that maps graphs to a class c. We will have two types of graphs in total, graphs that have node attributes, thus, for each node $u_i \in V \longrightarrow h_i \in R^d$ and graphs that don't have node attributes, thus, we have to learn from structure only. Our objective for graph classification tasks is to learn a function that produces optimal embeddings for each graph $f(G, \mu) : R^{n \times n} \longrightarrow R^d$ and for node classification we have to learn a function f that produces optimal embeddings for each node $f(G, \lambda) : R^{n \times n} \longrightarrow R^{n \times d}$. For short, we define these embeddings as Φ.

IV. Models

A. Graph Representation Learning Models

Graph2Vec: Graph2Vec [4] produces embeddings $\Phi \in \mathbb{R}^{G \times d}$ for a dataset of graphs $\mathcal{G} = \{G_1, G_2, ..., G_n\}$. Rooted subgraph extraction follows the Weisfeiler–Lehman procedure [1]. For a root node n and height h, the procedure generates identifier $sg_n^{(h)}$ through iterative relabeling based on sorted multisets of neighbor labels, where height corresponds to the maximum distance from the root node. The training objective for each subgraph $sg_n^{(h)}$ extracted from graph G_i is:

$$J(\Phi) = -\log \Pr(sg_n^{(h)} | \Phi(G_i)) \tag{1}$$

where $\Phi(G_i) \in \mathbb{R}^{\delta}$ represents the graph embedding vector.

Computing $\Pr(sg_n^{(h)} | \Phi(G_i))$ with softmax normalization over the full subgraph vocabulary requires $O(|V_{sg}|)$ complexity. Graph2Vec instead applies negative sampling from the doc2vec PV-DBOW framework, approximating the objective by sampling k negative subgraphs not present in the target

graph. This reduces the per-update cost to $O(k)$ where $k \ll |V_{sg}|$. The gradient update follows:

$$\Phi \leftarrow \Phi - \alpha \frac{\partial J}{\partial \Phi} \tag{2}$$

where α denotes the learning rate. Training iterates over all graphs for e epochs, shuffling graphs at each epoch. For each graph, rooted subgraphs up to height h are extracted around every node before applying stochastic gradient descent updates.

DeepWalk: DeepWalk [5] generates d-dimensional node embeddings $\Phi \in \mathbb{R}^{|V| \times d}$ for graph $G = (V, E)$. The algorithm samples γ random walks of length t from each vertex $v_i \in V$, with each walk W_{v_i} uniformly sampling neighbors at each step.

The skip-gram training objective [2] processes each vertex v_j in walk W_{v_i} by iterating over a context window of size w. For each context vertex $u_k \in W_{v_i}[j-w : j+w]$, the algorithm minimizes:

$$J(\Phi) = -\log \Pr(u_k | \Phi(v_j)) \tag{3}$$

Direct computation of $\Pr(u_k|\Phi(v_j))$ via softmax over all vertices requires $O(|V|)$ operations per sample. Hierarchical softmax reduces this to $O(\log |V|)$ by arranging vertices as leaves of a binary tree. For vertex u_k with tree path $(b_0, b_1, ..., b_{\lceil \log |V| \rceil})$ from root to leaf:

$$\Pr(u_k | \Phi(v_j)) = \prod_{l=1}^{\lceil \log |V| \rceil} \Pr(b_l | \Phi(v_j)) \tag{4}$$

where each factor represents a binary classifier at tree node b_l. Huffman encoding assigns shorter paths to frequently sampled vertices, further improving efficiency.

The algorithm executes γ passes over vertices. The learning rate α begins at 2.5% and decreases linearly with the number of vertices processed [5]. The power-law distribution of vertex frequencies in random walks creates sparse gradient updates, enabling lock-free asynchronous parallel training across multiple workers.

For graph-level tasks, node embeddings aggregate via mean pooling $\Phi(G) = \frac{1}{|V|} \sum_{v \in V} \Phi(v)$ or max pooling, though such simple aggregation cannot recover global structural properties effectively [4].

NetLSD: NetLSD [6] constructs a permutation- and size-invariant graph representation by modeling global structural properties through spectral physics. Given a graph G with normalized Laplacian $L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ and eigenvalues $0 \leq \lambda_1 \leq \cdots \leq \lambda_n \leq 2$, the method extracts a signature based on the solution to either the heat or wave equation.

For heat diffusion, governed by $\frac{\partial u_t}{\partial t} = -L u_t$, the solution is the heat kernel $H_t = e^{-tL}$. This acts as a low-pass filter, emphasizing global community structure while smoothing out local noise. The heat trace is defined as:

$$h_t = \text{tr}(H_t) = \sum_{j=1}^{n} e^{-t\lambda_j} \tag{5}$$

Alternatively, to capture resonance patterns and high-frequency details that heat diffusion dampens, NetLSD employs the wave equation $\frac{\partial^2 u_t}{\partial t^2} = -L u_t$. This describes a wave propagating through the graph medium, with the solution given by the complex-valued wave kernel $W_t = e^{-itL}$. Unlike the heat kernel, the wave kernel does not decay high eigenvalues, allowing it to "hear" finer local shapes. The wave trace at time $t \in [0, 2\pi)$ is:

$$w_t = \text{tr}(W_t) = \sum_{j=1}^{n} e^{-it\lambda_j} \tag{6}$$

Computing the full spectrum requires $O(n^3)$ time, which is prohibitive for large graphs. To address this, the authors propose two approximation strategies. For very large graphs or small time scales t, the method utilizes a Taylor expansion of the matrix exponential, enabling a linear-time approximation using the first two terms:

$$h_t \approx n - t\text{tr}(L) + \frac{t^2}{2}\text{tr}(L^2) \tag{7}$$

Since $\text{tr}(L) = n$ and $\text{tr}(L^2) = \sum_{ij} L_{ij}^2$, this is computable in $O(m)$ time. For manageable graph sizes, the method adopts a more accurate strategy based on Weyl's law [42] to capture medium-scale properties. Instead of the full spectrum, it computes only k eigenvalues at both ends of the spectrum (smallest and largest) using the block Krylov-Schur algorithm [43] and approximates the interloping eigenvalues via linear interpolation.

B. Graph Neural Network Models

GCN: Graph Convolutional Network (GCN) [9] introduced a spectral-based approach to graph convolutions that operates directly on graphs through localized first-order approximations of spectral graph convolutions. Given a graph with node feature matrix $X \in \mathbb{R}^{N \times D}$ where N is the number of nodes and D is the feature dimension, GCN propagates information through the graph structure using a layer-wise propagation rule:

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

where $\tilde{A} = A + I_N$ is the adjacency matrix with added self-connections, $\tilde{D}$ is the degree matrix of $\tilde{A}$, $W^{(l)}$ is a trainable weight matrix for layer l, $H^{(l)}$ represents node features at layer l (with $H^{(0)} = X$), and σ is an activation function. The symmetric normalization $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ ensures that the scale of feature representations remains consistent across layers. Each layer effectively aggregates features from a node's immediate neighbors, and stacking multiple layers allows the model to capture information from increasingly larger neighborhoods. The model is computationally efficient with complexity $O(|E|DF')$ where F' is the number of output features, making it linear in the number of edges and suitable for large-scale graphs.

***GAT**:* Graph Attention Network(GAT) [10] was the first model that introduced masked self-attention to GNN models, enabling each node to assign different importances to other nodes of the same neighborhood. This is also beneficial for interpretability. The model is applied to graphs that contain node attributes $h = \{h_1, h_2, ..., h_n\}$, $h_i \in R^d$, each layer produce new embeddings $h' = \{h'_1, h'_2, ..., h'_n\}$, $h'_i \in R^{d'}$. Self-attention is applied to all the nodes with a shared attention mechanism $\alpha : R^d \times R^d \longrightarrow R$ that computes attention coefficients

$$e_{ij} = \alpha(Wh_i, Wh_j)$$

that indicates importance of node's j features to node i. $W \in R^{d' \times d}$ is a weight matrix. Authors used a single-layer feedforward neural network as the attention mechanism that computes α_{ij}. Also, they found multi-head attention beneficial, thus, computing new embeddings h'_i requires concatenating or averaging all the aggregated features from each head

$$h'_{ij} = \oplus\sigma(\sum_{j \in N_i} a^k_{ij} W^k h_j)$$

Where σ is the sigmoid function and k indicates the k-th attention mechanism. In the last layer, concatenation is not possible, using averaging instead.

The model is computationally efficient as it needs $O(|V|dd' + |E|d')$ and individual head's computations can be parallelized.

***GIN**:* Graph Isomorphism Network(GIN) [11] is a simple architecture that satisfies the following theorem. Let A be a GNN with a sufficient number of GNN layers, A maps any graphs G_1 and G_2 that the Weisfeiler-Lehman test of isomorphism decides as non-isomorphic, to different embeddings if 1) A aggregates and updates node features iteratively with

$$h^{(k)}_v = \phi(h^{(k-1)}_v, f(\{h^{(k-1)}_v : u \in N(v)\}))$$

where f and ϕ are injective functions, with f operating on multisets, and 2) A's graph-level readout, which operates on the multiset of node features $\{h^{(k)}_v\}$, is injective. As Xu et al. showed that GNN's are at most as powerful as the WL test, this model generalizes the WL test and achieves the best possible results. GIN leverages sum aggregators, as they proved that sum aggregators are injective when used on multisets. Functions f and ϕ are learned with a multi-layer perceptron(MLP). For graph classification tasks, GIN's readout function concatenates all graph representation across all iterations/layers

$$h_G = CONCAT(READOUT(\{h^{(k)}_v | v \in G\}) | k = 0, 1, \ldots, K)$$

READOUT can be replaced with the sum function that most probably generalizes the WL test.

***PNA**:* Principal Neighborhood Aggregation [20] proposed using multiple aggregators concurrently as they proved that in order to discriminate between multisets of size n, at least n aggregators are needed. PNA combines mean aggregation $\mu(X^l)$, maximum and minimum aggregation $max(X^l), min(X^l)$ and standard deviation aggregation $\sigma(X^l)$, resulting in the following GNN layer

$$X^{(t+1)}_i = U(X^{(t)}_i, PNA_{(j,i) \in E} M(X^{(t)}_i, E_{j \longrightarrow i}, X^{(t)}_j))$$

Where PNA is the PNA operator that consists of the tensor product of all the aforementioned aggregators with the logarithmic scalers, as explained in Theorem 2 [20], $E_{j \longrightarrow i}$ is the feature of the edge (j, i), M and U are neural networks.

V. Datasets & Evaluation

A. Training & Results

We selected 4 datasets from TUDataset [44] list in total: ENZYMES, IMDB-MULTI, MUTAG and PROTEINS(full). Out of them, ENZYMES and PROTEINS have node attributes, thus, can be used for GNN training. For every training iteration we used 75% of the data for training and 25% for testing. We present testing results over 3 training iterations and the best checkpoint was selected using F1 score. All the GNN models were trained on one A100 for 500 total epochs with a patience of 100 epochs to ensure convergence, we chose Adadelta [45] as the optimizer with an initial learning rate of 0.1 and a batch size of 2 graphs. For all the other models, we utilized Optuna [46] to maximize the F1 score over 200 trials(on cpu), searching though specified parameters of each model. More specifically, for Graph2Vec we search *wl_iterations*: [2, 3, 4, 5], for DeepWalk we search *walk_number*: [2, 5, 7], *walk_length*: [5, 7, 10] and *window_size*: [3, 5, 10], for NetLSD we have a more extensive hyperparameter space including kernel (Heat vs Wave), timescale vectors, eigenvalues choice and normalization. We also search the optimal learning rate between 10^{-4} and 10^{-1} on a logarithmic scale. For every trial, the model generated embeddings are evaluated using two distinct downstream classifiers: SVM and MLP. All of our models yield 128-dimensional embeddings for each graph. (Here we need to specify what less or more dimensions are worse, don't mention it each time after)

TABLE I
DATASETS USED FROM TUDATASET

Dataset	*Graphs*	*Classes*	*Avg. Nodes*	*Avg. Edges*
ENZYMES	600	6	32.63	62.14
MUTAG	188	2	30.32	30.77
IMDB-MULTI	1500	3	13.00	65.94
PROTEINS	1113	2	39.06	72.82

***ENZYMES**:* The ENZYMES dataset contains 600 proteins from each of the 6 enzyme commission top-level enzyme classes and the goal was to correctly predict enzyme class membership for these proteins. So each protein is represented with a graph and we have a graph classification task.

For the GCN model we used 4 layers all with 256 hidden channels and a final layer of 128 output channels, dropout at 0.5. For the GAT and GIN model we used 2 layers all with 64 hidden channels and a final layer of 128 output channels, dropout at 0.5. For the PNA model we used 4 layers all with 64 hidden channels and a final layer of 128 output channels,

TABLE II
TESTING RESULTS FOR ENZYMES

Model	*ACC*	*AUC*	*F1*
GCN	0.61 ± 0.02	0.84 ± 0.02	0.61 ± 0.02
GAT	$\mathbf{0.66 \pm 0.01}$	$\mathbf{0.85 \pm 0.004}$	$\mathbf{0.67 \pm 0.01}$
PNA	0.63 ± 0.05	0.85 ± 0.03	0.63 ± 0.05
GIN	0.61 ± 0.02	0.84 ± 0.02	0.61 ± 0.02
NetLSD-SVM	0.36 ± 0.02	0.65 ± 0.03	0.35 ± 0.02
NetLSD-MLP	0.32 ± 0.01	0.63 ± 0.01	0.31 ± 0.01
DeepWalk-SVM	0.31 ± 0.03	0.60 ± 0.01	0.30 ± 0.03
DeepWalk-MLP	0.30 ± 0.04	0.61 ± 0.01	0.30 ± 0.04
Graph2Vec-SVM	0.33 ± 0.02	0.66 ± 0.02	0.32 ± 0.03
Graph2Vec-MLP	0.32 ± 0.01	0.65 ± 0.03	0.31 ± 0.02

dropout at 0.5. We should note that when we used 1) a smaller dropout value or no dropout at all, 2) more layers or 3) more hidden channels, overfitting increased. Testing results presented on Table II. After 3 runs, GAT had the best results by far, with 3% higher Accuracy and F1 score than PNA, which came second. Not only GAT produce better results, but it has a much faster inference time than PNA with 1.44ms on average, in contrast with $2ms$ on average for PNA.

Among the unsupervised graph representation learning models, NetLSD with the SVM classifier achieved the highest accuracy of 36%, benefiting from the Wave kernel and unnormalized Laplacian which successfully captured the dataset's discriminative spectral resonance patterns. It slightly outperformed Graph2Vec (33%) and DeepWalk (31%), suggesting that global spectral signatures were more effective than subgraph or random-walk based features for this specific task. In terms of efficiency, the NetLSD + SVM pipeline proved faster, recording an average inference time of 5.35ms per graph compared to 6.53ms for the MLP variant. Both configurations demonstrated efficient resource utilization, with peak memory usage averaging approximately 5.3MB, confirming the lightweight nature of the spectral embedding generation.

All three topology-based methods (NetLSD, Graph2Vec, DeepWalk) significantly underperformed relative to GNN approaches, achieving less than half the accuracy of GAT. This consistent performance gap across all unsupervised embedding methods highlights that node attribute information is critical for the ENZYMES dataset, where purely topological features lack sufficient discriminative power to distinguish between the six structurally similar enzyme classes.

PROTEINS*:* The PROTEINS dataset is a medium sized molecular property prediction dataset. This dataset is used for molecular property prediction aiming to predict whether molecules are enzymes or not, thus, a binary classification task. We used the full version of the dataset, resulting in node attributes with a dimensionality of 29, which is the biggest out of all the datasets we are performing analysis on.

For the GNN models, we used the exact same model architectures as with the ENZYMES dataset, epochs, patience and batch size except for the PNA model, where we used 2 total layers instead of 4. For Graph2Vec, NetLSD and DeepWalk, we used 128-dimensional embeddings. Testing results are presented in Table III. We observed faster convergence as training results achieved good scores very early, this is logical as we have more features to train on and a much larger dataset than ENZYMES. After 3 runs, GIN and PNA had the best results with 3% higher accuracy and F1 score than GAT. Because we only used 2 layers for the PNA model, the average inference time was only 0.7ms in contrast with 1.9ms for GAT. GIN had the best inference time with 0.3ms on average.

TABLE III
TESTING RESULTS FOR PROTEINS

Model	*ACC*	*AUC*	*F1*
GCN	0.72 ± 0.002	0.76 ± 0.007	0.67 ± 0.004
GAT	0.76 ± 0.01	0.75 ± 0.004	0.70 ± 0.01
PNA	$\mathbf{0.79 \pm 0.003}$	$\mathbf{0.85 \pm 0.003}$	0.72 ± 0.003
GIN	$\mathbf{0.79 \pm 0.01}$	0.76 ± 0.02	$\mathbf{0.73 \pm 0.001}$
DeepWalk-SVM	0.70 ± 0.01	0.75 ± 0.01	0.65 ± 0.01
DeepWalk-MLP	0.72 ± 0.01	0.76 ± 0.00	0.65 ± 0.02
Graph2Vec-SVM	0.68 ± 0.01	0.72 ± 0.00	0.63 ± 0.01
Graph2Vec-MLP	0.68 ± 0.01	0.72 ± 0.00	0.63 ± 0.02
NetLSD-SVM	0.70 ± 0.01	0.75 ± 0.01	0.65 ± 0.02
NetLSD-MLP	0.73 ± 0.03	0.75 ± 0.01	0.66 ± 0.01

Among graph representation learning models, NetLSD with the MLP classifier achieved the highest performance with 73% accuracy, marginally outperforming DeepWalk (72%) and surpassing Graph2Vec (68%). This performance shift suggests that for protein structures, both the local sequential patterns encoded by DeepWalk and the global spectral signatures captured by NetLSD (specifically using the Wave kernel) provide highly discriminative power. Notably, both NetLSD and DeepWalk benefited from the non-linear decision boundaries of the MLP classifier, showing a $2-3\%$ improvement over their SVM counterparts. In comparison, NetLSD was significantly faster, with the SVM configuration averaging 6.65ms per graph and the MLP variant 7.47ms, while DeepWalk avarege inference time was 129ms. Additionally, NetLSD maintained a compact memory footprint, with peak usage averaging approximately 8.7MB to 9.7MB.

All unsupervised methods performed achieved lower performance compared to supervised GNN approaches on this dataset, which is expected given they operate purely on graph structure without node attributes.

IMDB-MULTI*:* The IMDB-MULTI dataset contains 1,500 movie collaboration graphs from three genres, where each graph represents ego-networks of actors and actresses who appeared in movies from the same genre. This presents a graph classification task with three classes. Testing results are presented in Table IV.

Graph2Vec substantially outperformed DeepWalk by 5% in accuracy and F1 score. Graph2Vec's data-driven subgraph embedding approach proved more effective at capturing the structural patterns distinguishing different movie genres. Both SVM and MLP classifiers achieved similar performance with Graph2Vec embeddings, suggesting the learned representations are robust across different downstream classifiers. DeepWalk's uniform random walk strategy appears insufficient for

TABLE IV
TESTING RESULTS FOR IMDB-MULTI

Model	*ACC*	*AUC*	*F1*
DeepWalk-SVM	0.44 ± 0.00	0.61 ± 0.01	0.43 ± 0.01
DeepWalk-MLP	0.44 ± 0.03	0.59 ± 0.02	0.43 ± 0.03
Graph2Vec-SVM	0.49 ± 0.01	0.66 ± 0.01	0.48 ± 0.01
Graph2Vec-MLP	0.49 ± 0.01	0.66 ± 0.01	0.48 ± 0.01
NetLSD-SVM	0.51 ± 0.03	0.67 ± 0.03	0.50 ± 0.03
NetLSD-MLP	$\mathbf{0.52 \pm 0.01}$	$\mathbf{0.69 \pm 0.01}$	$\mathbf{0.51 \pm 0.01}$

this task, likely due to the relatively small graph sizes (average 13 nodes) where global structural properties matter more than local neighborhood patterns. Graph2Vec's average inference time was 0.28 ms per graph on IMDB-MULTI.

We also evaluated NetLSD, the optimization process consistently identified the *Wave* kernel as superior to the Heat kernel, typically preferring the unnormalized Laplacian ('normalized_laplacian=False') and 'empty' or 'no' normalization strategies. As shown in Table IV, NetLSD performed slightly better than Graph2Vec and DeepWalk, achieving the highest overall accuracy of 52% with the MLP classifier. This performance indicates that the global spectral signatures captured by the Wave kernel are particularly effective for distinguishing the dense, ego-centric structures of movie collaboration networks. Unlike the bio-informatics datasets where local features were dominant, the superior performance of NetLSD here reinforces the importance of global topological descriptors for social network analysis.

In terms of computational efficiency, the NetLSD + MLP pipeline recorded an average inference time of 5.13ms per graph, slightly faster than the SVM variant at 5.47ms. While slower than Graph2Vec's 0.28ms, NetLSD maintains a reasonable runtime profile given its better accuracy. Both NetLSD configurations exhibited consistent resource usage, with peak memory consumption averaging approximately 12.7MB, confirming the method's scalability for processing collaboration networks.

MUTAG*:* The MUTAG dataset comprises 188 chemical compounds labeled by their mutagenic effect on a specific bacterium, representing a binary classification task. MUTAG graphs are relatively small (average 30.32 nodes, 30.77 edges) but structurally complex. Testing results are presented in Table V.

TABLE V
TESTING RESULTS FOR MUTAG

Model	*ACC*	*AUC*	*F1*
DeepWalk-SVM	0.87 ± 0.01	0.86 ± 0.01	0.90 ± 0.01
DeepWalk-MLP	0.87 ± 0.02	0.86 ± 0.01	0.90 ± 0.01
Graph2Vec-SVM	0.85 ± 0.02	0.84 ± 0.00	0.89 ± 0.01
Graph2Vec-MLP	0.84 ± 0.03	0.85 ± 0.03	0.89 ± 0.01
NetLSD-SVM	$\mathbf{0.92 \pm 0.02}$	$\mathbf{0.94 \pm 0.02}$	$\mathbf{0.94 \pm 0.01}$
NetLSD-MLP	0.91 ± 0.01	0.93 ± 0.01	0.93 ± 0.01

DeepWalk achieved strong performance on MUTAG with 87% accuracy and F1 score of 0.90, slightly outperforming Graph2Vec by 2-3$\%$. Both methods achieved consistent results with F1 scores around 0.89-0.90, demonstrating that our hyperparameter optimization was effective. The choice between SVM and MLP had minimal impact for both models. DeepWalk's average inference time was 43.19 ms per graph on MUTAG.

For NetLSD, similar to our other experiments, the optimization consistently selected the *Wave* kernel, suggesting that the resonance patterns and high-frequency spectral details are critical for capturing the subtle structural variations in chemical compounds that define mutagenicity. NetLSD achieved the best performance on MUTAG with 92% accuracy, outperforming both DeepWalk and Graph2Vec by 5%. This substantial performance gap demonstrates that for molecular graphs like MUTAG, global spectral properties captured by NetLSD provide a richer representation than the local neighborhood sampling of DeepWalk or the subgraph distributions of Graph2Vec.

Furthermore, NetLSD maintained exceptional computational efficiency, with the MLP variant averaging just 4.43ms per graph compared to DeepWalk's 43.19ms, offering a solution that is both more accurate and nearly an order of magnitude faster. Resource utilization was minimal, with peak memory usage averaging approximately 1.8MB for both configurations, confirming the method's lightweight footprint for molecular graph analysis.

B. Sensitivity Analysis: Embedding Dimensionality

In this section, we are going to evaluate the trade-off between accuracy and computational cost, introducing a series of hidden(h)-output(o) channels for GNN and out dimensions (d) for GRL models. We are evaluating both Peak GPU/CPU memory usage for training and average inference time with both CPU and GPU(when available).

GNN

As noted in subsection A, overfitting occurred when more hidden channels were introduced, so we tried to minimize hidden channels as out channels increased. We only present testing results, thus, a higher accuracy or f1-score doesn't necessarily means that we have a better model. For example, GIN-h32-o32 has a train f1-score=0.65, GIN-h64-o256 has a train f1-score=0.69 and GIN-h64-o128 has a train f1-score=0.71. In general, all of the GNN models showed similar testing results, so we used the combination that produced results with less over/under-fitting. Peak GPU memory is also an important metric as with large graph datasets we have to introduce a large batch size to reduce training time and a deeper network to improve accuracy, both require more GPU memory. All of our training iterations used one A100 with 16 CPUs per GPU, note that torch.autocast() was used to reduce floating point operations from torch.float32 to torch.bfloat16, we didn't use torch.compile() or any other caching technique. GAT and PNA yielded logical results, as GPU memory linearly increased when hidden or output channels increased, in contrast with GIN, that showed impressive resilience, only differing a max-

imum of 5MB for all combinations. Results are presented in Tables VI, VII and VIII

TABLE VI
GAT: IMPACT OF EMBEDDING DIMENSION ON PERFORMANCE (ENZYMES)

Comb.	ACC	AUC	F1	GPU Mem.	Avg. time(ms)
h32-o32	0.67	0.86	0.67	42.8MB	1.29
h32-o64	0.66	0.84	0.66	45.9MB	1.48
h64-o128	0.66	0.85	0.67	59MB	2.54
h64-o256	**0.68**	**0.87**	**0.68**	76.8MB	3.63
h128-o256	0.67	0.85	0.67	94.2MB	4.99
h128-o512	**0.68**	0.84	**0.68**	143.8MB	5.38

TABLE VII
PNA: IMPACT OF EMBEDDING DIMENSION ON PERFORMANCE (PROTEINS)

Comb.	ACC	AUC	F1	GPU Mem.	Avg. time(ms)
h32-o32	0.77	0.81	0.72	60MB	7.04
h32-o64	0.77	0.80	0.72	60.3MB	5.26
h64-o128	**0.79**	**0.85**	0.72	65MB	4.63
h64-o256	0.77	0.81	0.72	68.3MB	4.19
h128-o256	0.78	0.81	**0.73**	80MB	15.00
h128-o512	0.78	0.80	0.72	85.2MB	10.90

TABLE VIII
GIN: IMPACT OF EMBEDDING DIMENSION ON PERFORMANCE (PROTEINS)

Comb.	ACC	AUC	F1	GPU Mem.	Avg. time(ms)
h32-o32	**0.80**	0.78	**0.74**	50MB	9.28
h32-o64	0.79	0.75	0.72	48.1MB	6.31
h64-o128	0.79	0.76	0.73	51.1MB	9.61
h64-o256	**0.80**	**0.81**	0.73	52.3MB	7.48
h128-o256	0.79	0.73	0.72	50MB	7.26
h128-o512	0.78	0.80	**0.74**	55.7MB	6.17

GRL

To evaluate the trade-off between representation resolution and computational cost, we performed a sensitivity analysis on the MUTAG dataset using the NetLSD-SVM model. For each target embedding dimension $d \in \{32, 64, 128, 256, 512\}$, we conducted a dedicated hyperparameter optimization search (50 trials) with Optuna [46] to identify the optimal spectral configuration specifically suited for that resolution.

TABLE IX
NETLSD: IMPACT OF EMBEDDING DIMENSION ON PERFORMANCE (MUTAG)

Dim (d)	ACC	AUC	F1	CPU Mem.	Avg. time(ms)
32	0.86	0.91	0.90	3.16	6.74
64	**0.91**	0.93	**0.93**	3.61	7.54
128	0.87	0.92	0.90	5.66	6.12
256	0.88	0.91	0.91	7.56	8.94
512	0.89	**0.94**	0.92	5.62	11.69

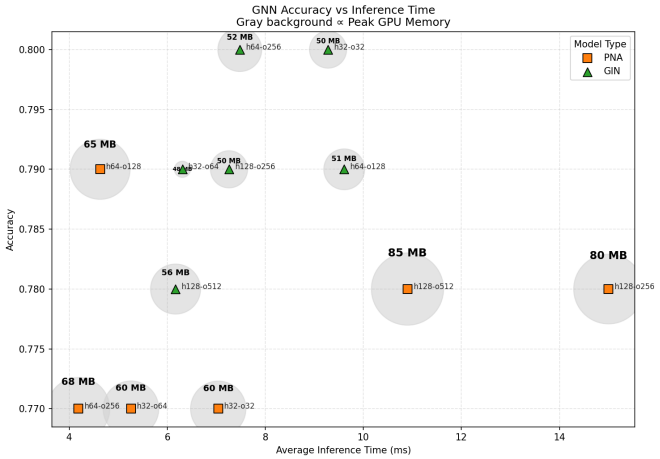

Fig. 1. Performance of GIN vs PNA on PROTEINS(Accuracy, Inference Time and Peak GPU Memory)

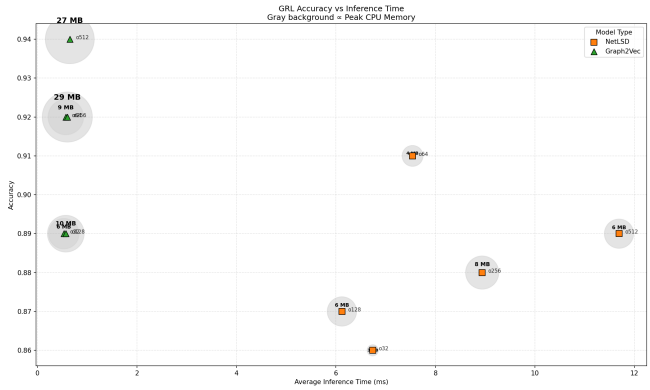

Fig. 2. Performance of NetLSD vs Graph2Vec on MUTAG(Accuracy, Inference Time and Peak CPU Memory)

The results indicate that NetLSD is highly capable of compressing structural information into compact representations. We observe a performance jump when increasing dimensionality from $d = 32$ to $d = 64$, where accuracy improves by over 5%. Interestingly, $d = 64$ represents a performance "sweet spot," achieving the highest observed accuracy (91%) and F1-score (93%). Further increasing the dimensionality to 128 or 256 yielded slightly lower performance, likely due to the inclusion of noisy high-frequency spectral components that do not contribute to the classification task. While $d = 512$ recovered slightly worst performance of $d = 64$, it required nearly $2\times$ the time.

Regarding computational efficiency, both inference time and memory consumption remained low across all configurations. Inference time averaged under 12 ms per graph, though the scaling behavior was notably non-linear. Surprisingly, $d = 128$ achieved the fastest throughput (6.12 ms), outperforming even the lower-dimensional settings, while the highest dimension

($d = 512$) incurred a significant latency cost, nearly doubling the time required by the optimal configuration. Memory usage followed a similar pattern of efficiency, consistently staying below 8 MB, with the optimal accuracy configuration ($d = 64$) requiring only 3.61 MB. The observed fluctuations in both time (e.g., $d = 64$ taking longer than $d = 128$) and memory (e.g., the drop at $d = 512$) are attributed to the optimizer selecting different eigenvalue computation strategies, such as partial computation versus full spectrum decomposition, specifically tailored to each target dimension.

For Graph2Vec, we conducted hyperparameter optimization across 200 trials per dimension using Optuna, searching over WL iterations and learning rate. The results presented in Table X reveal markedly different behavior compared to NetLSD.

TABLE X
GRAPH2VEC: IMPACT OF EMBEDDING DIMENSION ON PERFORMANCE (MUTAG)

Dim (d)	ACC	AUC	F1	CPU Mem.	Avg. time(ms)
32	0.89	0.88	0.88	6.15	0.54
64	0.92	0.92	0.90	9.09	0.57
128	0.89	0.90	0.88	10.07	0.57
256	0.92	0.92	0.90	29.41	0.60
512	**0.94**	**0.96**	**0.94**	26.89	0.65

Graph2Vec achieves peak performance at $d = 512$ (94% accuracy, 96% AUC). Unlike NetLSD's optimal performance at $d = 64$, Graph2Vec exhibits non-monotonic scaling with a dip at $d = 128$ (89%) despite strong results at $d = 64$ and $d = 256$ (both 92%). This suggests the data-driven subgraph vocabulary benefits from higher dimensions, though optimal selection requires careful tuning.

Computationally, Graph2Vec maintains sub-millisecond inference across all dimensions—substantially faster than NetLSD's 6-12 ms range. Inference scales minimally from 0.54 ms ($d = 32$) to 0.65 ms ($d = 512$), a mere 20% increase for $16\times$ dimensional expansion. Memory consumption remains under 11 MB through $d = 128$, but jumps to 27-29 MB at $d = 256$ and $d = 512$ due to expanded vocabulary and embedding matrix storage during training.

In conclusion, high-dimensional embeddings are not strictly necessary for molecular graphs like MUTAG. A compact vector of size 64 is sufficient to capture the necessary global topological signatures, offering an optimal balance of maximizing classification accuracy while minimizing time.

C. Clustering & Manifolds

The comparative analysis across all three datasets reveals that the relative performance of Graph2Vec and DeepWalk depends strongly on dataset characteristics and structural properties. Graph2Vec excels on social network datasets such as IMDB-MULTI, where global structural patterns and subgraph distributions effectively distinguish between different graph classes. DeepWalk achieves competitive performance on biological graphs like PROTEINS, where local neighborhood patterns captured through random walks provide useful features for classification tasks.

Unsupervised clustering analysis using both k-means and spectral clustering methods reveals varying degrees of natural grouping in the learned embeddings. The results are presented in Table XI.

TABLE XI
CLUSTERING PERFORMANCE COMPARISON

Dataset	Model	K-means ARI	Spectral ARI
PROTEINS	PNA	0.001	**0.266**
	NetLSD-MLP	0.059	0.104
	DeepWalk-MLP	**0.183**	0.114
ENZYMES	GAT	**0.090**	**0.151**
	NetLSD-SVM	0.026	0.026
IMDB-MULTI	NetLSD-SVM	**0.007**	0.014
	Graph2Vec-MLP	0.006	**0.021**
MUTAG	NetLSD-SVM	**0.323**	**0.345**
	DeepWalk-SVM	0.268	0.319

We expected spectral clustering to yield better results when the output embeddings are non convex or non linear. As shown from testing results and in 11, we can identify that ENZYMES and IMDB-MULTI are the two most difficult embeddings to work with and indeed, when spectral clustering was used, we got significantly better results than kmeans, improving the ARI score from 0.09 to 0.151 for the ENZYMES and from 0.006 to 0.021 for IMDB-MULTI.

D. Stability Analysis

To evaluate the robustness of trained models, we conducted a stability analysis by introducing random structural perturbations to the graphs. Specifically, we randomly removed or added edges and shuffled node attributes. For each perturbation, we recomputed the embeddings and compared them to the original vectors using two distinct metrics: Cosine Similarity and Euclidean Distance. We also compared downstream classification performance via changes in classification Accuracy and F1-score. Inference is performed on 100% and not only on 25% of the set like before.

NetLSD

TABLE XII
EMBEDDING STABILITY ANALYSIS FOR **NETLSD** ON MUTAG. HIGHER COSINE SIMILARITY INDICATES BETTER STRUCTURAL PRESERVATION. LOWER EUCLIDEAN DISTANCE INDICATES LESS POSITIONAL SHIFT.

Dataset	Perturbation	*Cosine Similarity*	*Euclidean Distance*
MUTAG	Original	1.0000	0.00
	Remove 2.5%	0.9961	6.84
	Remove 5%	0.9948	10.84
	Remove 7.5%	0.9882	19.87
	Remove 10%	0.9771	24.84
	Add 5%	0.9940	10.97
	Add 10%	0.9962	13.81
	Add 15%	0.9954	18.60
	Add 20%	0.9947	20.80
	Add 25%	0.9890	24.91
	Shuffle attributes	1.0000	0.00

1) Analysis on MUTAG (SVM): The results demonstrate a clear dichotomy in the stability of NetLSD. First, the model is

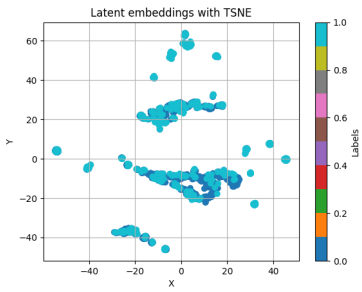

Fig. 3. PROTEINS – NetLSD

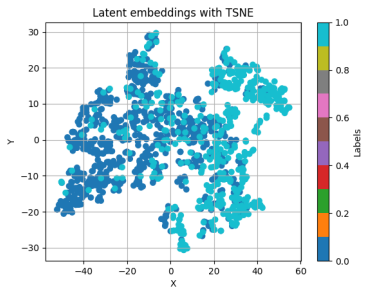

Fig. 4. PROTEINS – PNA

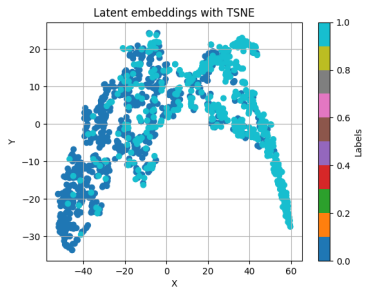

Fig. 5. PROTEINS – GIN

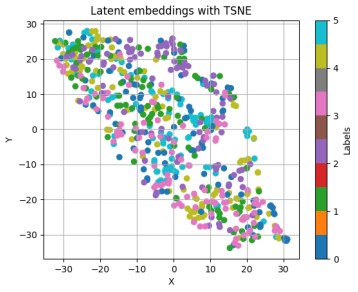

Fig. 6. ENZYMES – NetLSD

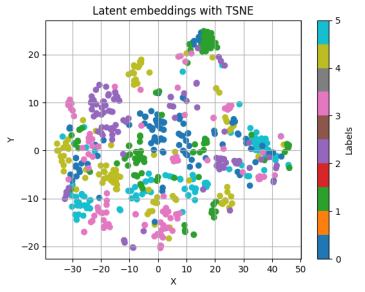

Fig. 7. ENZYMES – GAT

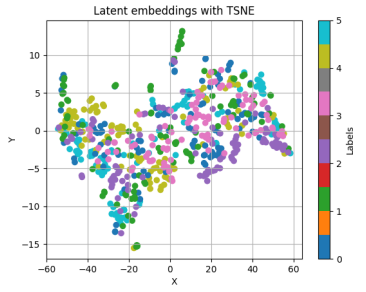

Fig. 8. ENZYMES – GIN

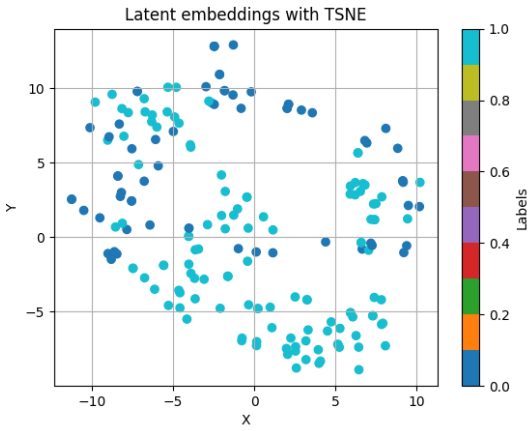

Fig. 9. MUTAG – NetLSD

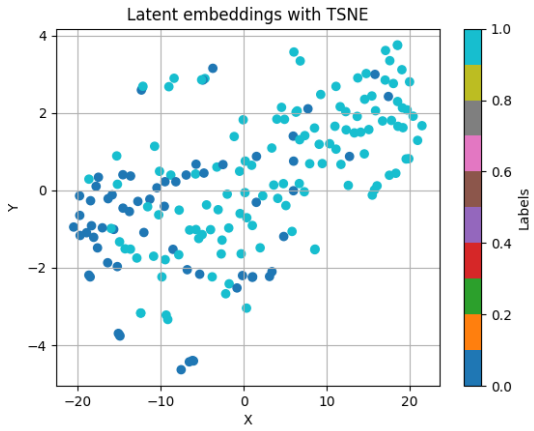

Fig. 10. MUTAG – Graph2Vec

Fig. 11. t-SNE visualization of latent embeddings across different datasets and models. The first row shows PROTEINS embeddings, the second row shows ENZYMES embeddings, and the third row shows MUTAG embeddings.

completely invariant to node attribute shuffling (Accuracy and Stability scores remain unchanged), confirming that NetLSD relies exclusively on spectral structural features (heat kernel traces) and ignores node labels. Second, we observe a significant disconnect between angular stability and classification performance under topological perturbations. While the Cosine Similarity remains remarkably high (> 0.97) across all edge additions and removals, the classification accuracy drops precipitously, losing up to 40% performance (Table XIII). This can be explained by the Euclidean Distance metric in Table XII, which increases drastically (up to 24.91). This indicates that while the "direction" of the embedding vectors remains stable, the perturbations cause significant translational shifts in the embedding space. Since the SVM classifier relies on spatial margins, these positional shifts push graphs across the decision boundary, leading to misclassification despite the high cosine stability. We conclude that NetLSD is highly sensitive to graph topology, where even minor edge removals (2.5%) can render

TABLE XIII
IMPACT OF PERTURBATIONS ON **NETLSD** CLASSIFICATION PERFORMANCE. VALUES IN PARENTHESES SHOW THE DIFFERENCE FROM THE ORIGINAL BASELINE.

Dataset	Perturbation	*Accuracy (Diff)*	*AUC (Diff)*	*F1 (Diff)*
MUTAG	Original	0.87	0.85	0.87
	Remove 2.5%	0.57 (−0.30)	0.58 (−0.27)	0.58 (−0.29)
	Remove 5%	0.64 (−0.23)	0.65 (−0.20)	0.65 (−0.22)
	Remove 7.5%	0.58 (−0.29)	0.55 (−0.30)	0.59 (−0.28)
	Remove 10%	0.47 (−0.40)	0.46 (−0.39)	0.48 (−0.39)
	Add 5%	0.64 (−0.23)	0.63 (−0.22)	0.65 (−0.22)
	Add 10%	0.59 (−0.28)	0.57 (−0.28)	0.60 (−0.27)
	Add 15%	0.52 (−0.35)	0.51 (−0.34)	0.53 (−0.34)
	Add 20%	0.51 (−0.36)	0.48 (−0.37)	0.52 (−0.35)
	Add 25%	0.56 (−0.31)	0.54 (−0.31)	0.57 (−0.30)
	Shuffle attributes	0.87 (0.00)	0.85 (0.00)	0.87 (0.00)

the learned embeddings ineffective for classification.

2) Analysis on PROTEINS (MLP): In the case of the PROTEINS dataset classified with an MLP, we observe a distinct behavior compared to MUTAG. While the embedding space undergoes severe distortion, evidenced by massive Euclidean distances reaching 105.67 at 30% edge removal, far exceeding those observed in MUTAG, the downstream classification remains surprisingly resilient. Unlike the SVM on MUTAG, which collapsed under smaller shifts, the MLP maintains performance within a 2% margin even under extreme structural degradation. This suggests that the non-linear decision boundaries of the MLP can effectively accommodate the translational shifts inherent to NetLSD's perturbations. Consequently, although the embedding vectors suffer from significant positional instability, the semantic separability of the classes is preserved, allowing the MLP to maintain high predictive power despite the aggressive topological noise.

TABLE XIV
EMBEDDING STABILITY ANALYSIS FOR **NETLSD** ON PROTEINS. HIGHER COSINE SIMILARITY INDICATES BETTER STRUCTURAL PRESERVATION. LOWER EUCLIDEAN DISTANCE INDICATES LESS POSITIONAL SHIFT.

Dataset	Perturbation	*Cosine Similarity*	*Euclidean Distance*
PROTEINS	Original	1.0000	0.00
	Remove 2.5%	0.9954	24.11
	Remove 5%	0.9931	24.81
	Remove 7.5%	0.9909	34.40
	Remove 10%	0.9874	39.26
	Remove 15%	0.9770	49.93
	Remove 20%	0.9580	68.06
	Remove 30%	0.9009	105.67
	Add 5%	0.9979	16.15
	Add 10%	0.9950	27.55
	Add 15%	0.9920	36.77
	Add 20%	0.9883	43.96
	Add 30%	0.9775	63.52
	Shuffle attributes	1.0000	0.00
	Shuffle & Rem. 30%	0.9009	105.67

Graph2Vec

For Graph2Vec, we performed stability analysis on the PROTEINS dataset. Graph2Vec demonstrates moderate sensitivity to topological perturbations, with edge removal causing accuracy drops of up to 23% at 30% edge removal (Table

TABLE XV
IMPACT OF PERTURBATIONS ON **NETLSD** CLASSIFICATION PERFORMANCE (PROTEINS). VALUES IN PARENTHESES SHOW THE DIFFERENCE FROM THE ORIGINAL BASELINE.

Dataset	Perturbation	*Accuracy (Diff)*	*AUC (Diff)*	*F1 (Diff)*
PROTEINS	Original	0.75	0.72	0.74
	Remove 2.5%	0.75 (+0.00)	0.73 (+0.01)	0.75 (+0.01)
	Remove 5%	0.75 (+0.00)	0.73 (+0.01)	0.75 (+0.01)
	Remove 7.5%	0.74 (−0.01)	0.72 (−0.00)	0.74 (−0.00)
	Remove 10%	0.73 (−0.02)	0.71 (−0.01)	0.73 (−0.02)
	Remove 15%	0.75 (−0.00)	0.72 (+0.00)	0.74 (+0.00)
	Remove 20%	0.74 (−0.01)	0.71 (−0.01)	0.73 (−0.01)
	Remove 30%	0.72 (−0.02)	0.70 (−0.02)	0.72 (−0.02)
	Add 5%	0.74 (−0.01)	0.72 (−0.01)	0.73 (−0.01)
	Add 10%	0.73 (−0.02)	0.70 (−0.02)	0.72 (−0.02)
	Add 15%	0.75 (+0.00)	0.73 (+0.01)	0.75 (+0.00)
	Add 20%	0.74 (−0.00)	0.72 (+0.00)	0.74 (−0.00)
	Add 30%	0.73 (−0.02)	0.71 (−0.01)	0.72 (−0.02)
	Shuffle attributes	0.75 (+0.00)	0.72 (+0.00)	0.74 (+0.00)
	Shuffle & Rem. 30%	0.73 (−0.01)	0.71 (−0.01)	0.73 (−0.01)

XVII). Adding edges has a more dramatic effect on F1 scores, with losses of up to 30% at 20% edge addition. The model shows minimal sensitivity to attribute shuffling, with only a $2-3\%$ performance decrease, suggesting that Graph2Vec relies primarily on structural features. The embedding stability analysis (Table XVI) reveals that cosine similarity degrades more rapidly than NetLSD, dropping to 0.45 at 30% edge addition, indicating substantial directional changes in the embedding space. Euclidean distances remain relatively small, reflecting the lower dimensionality and different embedding characteristics of unsupervised graph representations. Compared to NetLSD's extreme sensitivity where even 2.5% edge removal causes 30% accuracy drops, Graph2Vec shows more gradual degradation, though its reliance on precise Weisfeiler-Lehman subgraph vocabularies makes it vulnerable to substantial directional shifts under moderate perturbations.

TABLE XVI
EMBEDDING STABILITY ANALYSIS FOR **GRAPH2VEC** ON PROTEINS. HIGHER COSINE SIMILARITY INDICATES BETTER STRUCTURAL PRESERVATION. LOWER EUCLIDEAN DISTANCE INDICATES LESS POSITIONAL SHIFT.

Dataset	Perturbation	*Cosine Similarity*	*Euclidean Distance*
PROTEINS	Original	1.0000	0.00
	Remove 2.5%	0.8784	1.06
	Remove 5%	0.8538	1.18
	Remove 7.5%	0.8378	1.25
	Remove 10%	0.8191	1.34
	Remove 15%	0.7889	1.46
	Remove 20%	0.7632	1.56
	Remove 30%	0.6995	1.78
	Add 5%	0.7769	1.58
	Add 10%	0.6556	1.98
	Add 15%	0.5666	2.25
	Add 20%	0.5071	2.44
	Add 30%	0.4501	2.67
	Shuffle attributes	0.9999	0.03
	Shuffle & Rem. 30%	0.6994	1.78

GIN

For GIN, we performed stability analysis on the PROTEINS dataset. PROTEINS is considered a well structured dataset consisting of dense graphs, thus, making it more robust to

TABLE XVII
IMPACT OF PERTURBATIONS ON **GRAPH2VEC** CLASSIFICATION PERFORMANCE. VALUES IN PARENTHESES SHOW THE DIFFERENCE FROM THE ORIGINAL BASELINE.

Dataset	Perturbation	*Accuracy (Diff)*	*AUC (Diff)*	*F1 (Diff)*
PROTEINS	Original	0.74	0.74	0.69
	Remove 2.5%	0.65 (−0.09)	0.68 (−0.06)	0.66 (−0.04)
	Remove 5%	0.63 (−0.11)	0.67 (−0.07)	0.65 (−0.04)
	Remove 7.5%	0.62 (−0.12)	0.66 (−0.08)	0.64 (−0.05)
	Remove 10%	0.61 (−0.13)	0.65 (−0.09)	0.64 (−0.05)
	Remove 15%	0.58 (−0.16)	0.63 (−0.11)	0.63 (−0.06)
	Remove 20%	0.55 (−0.19)	0.61 (−0.13)	0.62 (−0.07)
	Remove 30%	0.51 (−0.23)	0.58 (−0.16)	0.61 (−0.08)
	Add 5%	0.66 (−0.08)	0.65 (−0.09)	0.58 (−0.11)
	Add 10%	0.68 (−0.06)	0.66 (−0.08)	0.58 (−0.12)
	Add 15%	0.67 (−0.07)	0.63 (−0.11)	0.52 (−0.17)
	Add 20%	0.61 (−0.13)	0.56 (−0.17)	0.39 (−0.30)
	Add 30%	0.59 (−0.15)	0.55 (−0.19)	0.40 (−0.29)
	Shuffle attributes	0.72 (−0.02)	0.72 (−0.02)	0.67 (−0.03)
	Remove 30% & Shuffle attributes	0.51 (−0.23)	0.58 (−0.16)	0.61 (−0.08)

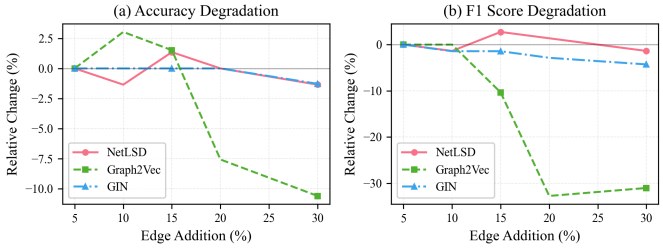

Fig. 12. Degradation of Accuracy and F1-score for NetLSD, Graph2Vec and GIN when edge addition is applied.

perturbations. We tested removing and adding up to 30% of total percentage of edges resulting in a maximum of 2% reduction in Accuracy and AUC and 1% for F1-score (Table XIX). We also observed that shuffling node attributes does not affect classification performance. As for the embeddings, we note that adding edges result in more different embeddings than removing edges, particularly for a small percentage of total edges (Table XVIII). Also shuffling node attributes yields the same exact embeddings, likely a result of the nature of PROTEINS.

GAT

For GAT, we performed stability analysis on the ENZYMES dataset. ENZYMES is a similar dataset to PROTEINS, some differences are ENZYMES include multiple classes, is smaller and the dimensionality of node attributes is smaller. We applied the same perturbations as with PROTEINS. We observed larger differences in classification performance, with addition of edges having a more significant effect (Table XXI). As for the embedding stability, we notice a smaller euclidean distance than with PROTEINS, though this is somewhat expected as we provide the average difference of all the embeddings and ENZYMES is almost half the size of PROTEINS, also cosine similarity is almost perfect for all the perturbations (Table XX). The difference between classification performance and embedding stability is a result of ENZYMES consisting of 6 total classes, thus, a more difficult task.

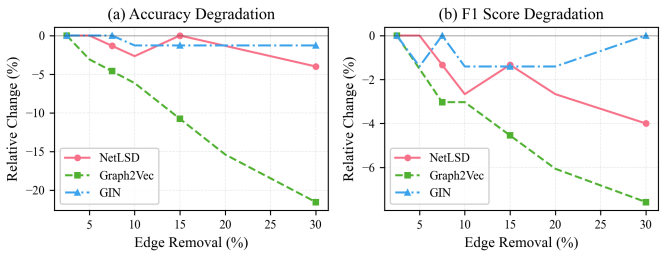

Fig. 13. Degradation of Accuracy and F1-score for NetLSD, Graph2Vec and GIN when edge removal is applied.

TABLE XVIII
EMBEDDING STABILITY ANALYSIS FOR **GIN** ON PROTEINS. HIGHER COSINE SIMILARITY INDICATES BETTER STRUCTURAL PRESERVATION. LOWER EUCLIDEAN DISTANCE INDICATES LESS POSITIONAL SHIFT.

Dataset	Perturbation	*Cosine Similarity*	*Euclidean Distance*
PROTEINS	Original	1.0000	0.00
	Remove 2.5%	0.9971	28.4128
	Remove 5%	0.9944	33.1448
	Remove 7.5%	0.9882	19.87
	Remove 10%	0.9908	40.066
	Remove 15%	0.9766	64.1023
	Remove 20%	0.9646	80.6428
	Remove 30%	0.9300	133.4811
	Add 5%	0.9932	32.8347
	Add 10%	0.9858	54.1876
	Add 15%	0.9776	74.6001
	Add 20%	0.9739	97.2779
	Add 30%	0.9605	138.9212
	Shuffle attributes	1.0000	0.00
	Remove 30% & Shuffle attributes	0.9463	108.5605

TABLE XIX
IMPACT OF PERTURBATIONS ON **GIN** CLASSIFICATION PERFORMANCE. VALUES IN PARENTHESES SHOW THE DIFFERENCE FROM THE ORIGINAL BASELINE.

Dataset	Perturbation	*Accuracy (Diff)*	*AUC (Diff)*	*F1 (Diff)*
PROTEINS	Original	0.79	0.78	0.71
	Remove 2.5%	0.78 (−0.01)	0.78 (0.00)	0.71 (0.00)
	Remove 5%	0.78 (−0.01)	0.77 (−0.01)	0.70 (−0.01)
	Remove 7.5%	0.78 (−0.01)	0.78 (0.00)	0.71 (0.00)
	Remove 10%	0.78 (−0.01)	0.77 (−0.01)	0.70 (−0.01)
	Remove 15%	0.77 (−0.02)	0.77 (−0.01)	0.70 (−0.01)
	Remove 20%	0.77 (−0.02)	0.77 (−0.01)	0.70 (−0.01)
	Remove 30%	0.77 (−0.02)	0.76 (−0.02)	0.71 (0.00)
	Add 5%	0.78 (−0.01)	0.78 (0.00)	0.70 (−0.01)
	Add 10%	0.78 (−0.01)	0.78 (0.00)	0.69 (−0.02)
	Add 15%	0.78 (−0.01)	0.79 (+0.01)	0.69 (−0.02)
	Add 20%	0.78 (−0.01)	0.78 (0.00)	0.68 (−0.03)
	Add 30%	0.77 (−0.02)	0.78 (0.00)	0.67 (−0.04)
	Shuffle attributes	0.79 (0.00)	0.78 (0.00)	0.71 (0.00)
	Remove 30% & Shuffle attributes	0.77 (−0.01)	0.77 (0.00)	0.72 (+0.01)

TABLE XX
EMBEDDING STABILITY ANALYSIS FOR **GAT** ON ENZYMES. HIGHER COSINE SIMILARITY INDICATES BETTER STRUCTURAL PRESERVATION. LOWER EUCLIDEAN DISTANCE INDICATES LESS POSITIONAL SHIFT.

Dataset	Perturbation	*Cosine Similarity*	*Euclidean Distance*
ENZYMES	Original	1.0000	0.00
	Remove 2.5%	0.9998	0.5223
	Remove 5%	0.9992	1.4974
	Remove 7.5%	0.9982	2.7408
	Remove 10%	0.9969	4.0100
	Remove 15%	0.9944	6.2450
	Remove 20%	0.9921	8.2057
	Remove 30%	0.9872	11.9780
	Add 5%	0.9990	1.4018
	Add 10%	0.9982	2.0989
	Add 15%	0.9971	2.6283
	Add 20%	0.9959	3.0830
	Add 30%	0.9939	3.9622
	Shuffle attributes	1.0000	0.00
	Add 30% & Shuffle attributes	0.9938	3.8288

TABLE XXI
IMPACT OF PERTURBATIONS ON **GAT** CLASSIFICATION PERFORMANCE. VALUES IN PARENTHESES SHOW THE DIFFERENCE FROM THE ORIGINAL BASELINE.

Dataset	Perturbation	*Accuracy (Diff)*	*F1 (Diff)*
ENZYMES	Original	0.83	0.84
	Remove 2.5%	0.83 (0.00)	0.84 (0.00)
	Remove 5%	0.84 (+0.01)	0.84 (0.00)
	Remove 7.5%	0.83 (0.00)	0.83 (−0.01)
	Remove 10%	0.82 (−0.01)	0.82 (−0.02)
	Remove 15%	0.80 (−0.03)	0.80 (−0.04)
	Remove 20%	0.79 (−0.04)	0.79 (−0.05)
	Remove 30%	0.76 (−0.07)	0.74 (−0.10)
	Add 5%	0.81 (−0.02)	0.81 (−0.03)
	Add 10%	0.76 (−0.07)	0.77 (−0.07)
	Add 15%	0.75 (−0.08)	0.75 (−0.09)
	Add 20%	0.71 (−0.12)	0.72 (−0.12)
	Add 30%	0.68 (−0.15)	0.68 (−0.16)
	Shuffle attributes	0.84 (+0.01)	0.84 (0.00)
	Add 30% & Shuffle attributes	0.66 (−0.13)	0.67 (−0.17)

VI. DISCUSSION

Our comprehensive empirical analysis reveals a distinct operational dichotomy between supervised Graph Neural Networks and unsupervised representation learning methods, governed primarily by the information density of node attributes versus global topology. On bioinformatics datasets rich in node features, such as ENZYMES and PROTEINS, GNN architectures (specifically GAT and PNA) dominate, as unsupervised methods like NetLSD and Graph2Vec fail to capitalize on the discriminative power of biochemical attributes. However, on structure-only datasets like MUTAG, NetLSD achieves remarkable performance (92% accuracy), demonstrating that spectral methods can offer an optimal balance of high accuracy and low computational cost even without node attributes. Ultimately, our results suggest that no single paradigm is universally superior. Instead, the optimal choice depends on the specific characteristics of the dataset, particularly the trade-off between the discriminative power of node features and the richness of the underlying topology.

From a deployment perspective, the trade-off between computational resources and representation quality is critical. Our sensitivity analysis shows that while increasing dimensionality strictly raises memory usage and inference time, it does not consistently improve accuracy beyond a certain point. We found that higher-dimensional spaces often introduce overfitting and can hinder classification. Regarding structure discovery, the consistent superiority of spectral clustering over K-means suggests that the learned embedding manifolds are inherently non-convex. Finally, our stability analysis indicates that models like NetLSD and GNNs are more stable to perturbations, though degradation also depends on the dataset structure.

REFERENCES

[1] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, pp. 2539–2561, 2011.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[3] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," 2013. [Online]. Available: https://arxiv.org/abs/1310.4546

[4] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning distributed representations of graphs," 2017. [Online]. Available: https://arxiv.org/abs/1707.05005

[5] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '14. ACM, Aug. 2014, p. 701–710. [Online]. Available: http://dx.doi.org/10.1145/2623330.2623732

[6] A. Tsitsulin, D. Mottin, P. Karras, A. Bronstein, and E. Müller, "Netlsd: Hearing the shape of a graph," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, ser. KDD '18. ACM, Jul. 2018, p. 2347–2356. [Online]. Available: http://dx.doi.org/10.1145/3219819.3219991

[7] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.

[8] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017. [Online]. Available: https://arxiv.org/abs/1609.02907

[10] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018. [Online]. Available: https://arxiv.org/abs/1710.10903

[11] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" 2019. [Online]. Available: https://arxiv.org/abs/1810.00826

[12] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," 2021. [Online]. Available: https://arxiv.org/abs/2005.00687

[13] V. P. Dwivedi, L. Rampášek, M. Galkin, A. Parviz, G. Wolf, A. T. Luu, and D. Beaini, "Long range graph benchmark," 2023. [Online]. Available: https://arxiv.org/abs/2206.08164

[14] Y. Luo, L. Shi, and X.-M. Wu, "Can classic gnns be strong baselines for graph-level tasks? simple architectures meet excellence," 2025. [Online]. Available: https://arxiv.org/abs/2502.09263

[15] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," 2014. [Online]. Available: https://arxiv.org/abs/1405.4053

[16] P. Yanardag and S. V. N. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015.

[17] B. Adhikari, Y. Zhang, N. Ramakrishnan, and B. A. Prakash, "Distributed representation of subgraphs," 2017. [Online]. Available: https://arxiv.org/abs/1702.06921

[18] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," 2016. [Online]. Available: https://arxiv.org/abs/1607.00653

[19] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" 2022. [Online]. Available: https://arxiv.org/abs/2105.14491

[20] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, "Principal neighbourhood aggregation for graph nets," 2020. [Online]. Available: https://arxiv.org/abs/2004.05718

[21] E. Chien, W.-C. Chang, C.-J. Hsieh, H.-F. Yu, J. Zhang, O. Milenkovic, and I. S. Dhillon, "Node feature extraction by self-supervised multi-scale neighborhood prediction," 2022. [Online]. Available: https://arxiv.org/abs/2111.00064

[22] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," 2018. [Online]. Available: https://arxiv.org/abs/1809.10341

[23] K. Riesen and H. Bunke, "Approximate graph edit distance computation by means of bipartite graph matching," *Image and Vision Computing*, vol. 27, no. 7, pp. 950–959, 2009, 7th IAPR-TC15 Workshop on Graph-based Representations (GbR 2007). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S026288560800084X

[24] C.-L. Lin, "Hardness of approximating graph transformation problem," in *Proceedings of the 5th International Symposium on Algorithms and Computation*, ser. ISAAC '94. Berlin, Heidelberg: Springer-Verlag, 1994, p. 74–82.

[25] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Proceedings of the Fifth IEEE International Conference on Data Mining*, ser. ICDM '05. USA: IEEE Computer Society, 2005, p. 74–81. [Online]. Available: https://doi.org/10.1109/ICDM.2005.132

[26] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Learning Theory and Kernel Machines*, B. Schölkopf and M. K. Warmuth, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 129–143.

[27] R. Kondor and H. Pan, "The multiscale laplacian graph kernel," 2016. [Online]. Available: https://arxiv.org/abs/1603.06186

[28] G. Nikolentzos, P. Meladianos, and M. Vazirgiannis, "Matching node embeddings for graph similarity," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, Feb. 2017. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10839

[29] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. 77, pp. 2539–2561, 2011. [Online]. Available: http://jmlr.org/papers/v12/shervashidze11a.html

[30] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1365–1374. [Online]. Available: https://doi.org/10.1145/2783258.2783417

[31] S. Verma and Z.-L. Zhang, "Hunt for the unique, stable, sparse and fast feature learning on graphs," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/d2ddea18f00665ce8623e36bd4e3c7c5-Paper.pdf

[32] M. Kac, "Can one hear the shape of a drum?" *The American Mathematical Monthly*, vol. 73, no. 4, pp. 1–23, 1966. [Online]. Available: http://www.jstor.org/stable/2313748

[33] M. M. Bronstein and A. M. Bronstein, "Shape recognition with spectral distances," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 5, pp. 1065–1071, 2011.

[34] J. Sun, M. Ovsjanikov, and L. Guibas, "A concise and provably informative multi-scale signature based on heat diffusion," in *Proceedings of the Symposium on Geometry Processing*, ser. SGP '09. Goslar, DEU: Eurographics Association, 2009, p. 1383–1392.

[35] M. Aubry, U. Schlickewei, and D. Cremers, "The wave kernel signature: A quantum mechanical approach to shape analysis," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 2011, pp. 1626–1633.

[36] L. Akoglu, M. McGlohon, and C. Faloutsos, "oddball: Spotting anomalies in weighted graphs," in *Advances in Knowledge Discovery and Data Mining*, M. J. Zaki, J. X. Yu, B. Ravindran, and V. Pudi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 410–421.

[37] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[38] M. Berlingerio, D. Koutra, T. Eliassi-Rad, and C. Faloutsos, "Network similarity via multiple social theories," in *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ser. ASONAM '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 1439–1440. [Online]. Available: https://doi.org/10.1145/2492517.2492582

[39] M. Tantardini, F. Ieva, L. Tajoli, and C. Piccardi, "Comparing methods for comparing networks," *Scientific Reports*, vol. 9, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:208278792

[40] L. K. Szakács, A. Benczúr, and F. Béres, "Whole graph embedding methods and their performances," 2023.

[41] B. Rozemberczki and R. Sarkar, "Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models," 2020. [Online]. Available: https://arxiv.org/abs/2005.07959

[42] H. Weyl, "Ueber die asymptotische verteilung der eigenwerte," *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, vol. 1911, pp. 110–117, 1911. [Online]. Available: http://eudml.org/doc/58792

[43] V. Hernandez, J. E. Roman, and V. Vidal, "Slepc: A scalable and flexible toolkit for the solution of eigenvalue problems," *ACM Trans. Math. Softw.*, vol. 31, no. 3, p. 351–362, Sep. 2005. [Online]. Available: https://doi.org/10.1145/1089014.1089019

[44] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs," 2020. [Online]. Available: https://arxiv.org/abs/2007.08663

[45] M. D. Zeiler, "Adadelta: An adaptive learning rate method," 2012. [Online]. Available: https://arxiv.org/abs/1212.5701

[46] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019. [Online]. Available: https://arxiv.org/abs/1907.10902