Hierarchical Shortest-Path Graph Kernel Network

Jiaxin Wang[1] **Wenxuan Tu**[2*] **Jieren Cheng**[1,2*]
[1]School of Cyberspace Security, Hainan University
[2]School of Computer Science and Technology, Hainan University
`{twx,992730}@hainanu.edu.cn`

Abstract

Graph kernels have emerged as a fundamental and widely adopted technique in graph machine learning. However, most existing graph kernel methods rely on fixed graph similarity estimation that cannot be directly optimized for task-specific objectives, leading to sub-optimal performance. To address this limitation, we propose a kernel-based learning framework called Hierarchical Shortest-Path Graph Kernel Network (**HSP-GKN**), which seamlessly integrates graph similarity estimation with downstream tasks within a unified optimization framework. Specifically, we design a hierarchical shortest-path graph kernel that efficiently preserves both the semantic and structural information of a given graph by transforming it into hierarchical features used for subsequent neural network learning. Building upon this kernel, we develop a novel end-to-end learning framework that matches hierarchical graph features with learnable *hidden* graph features to produce a similarity vector. This similarity vector subsequently serves as the graph embedding for end-to-end training, enabling the neural network to learn task-specific representations. Extensive experimental results demonstrate the effectiveness and superiority of the designed kernel and its corresponding learning framework compared to current competitors. Code is available at `https://github.com/JXWANG-GRAPH/HSP-GKN`.

1 Introduction

Graph neural networks (GNNs) have become a prominent approach in graph machine learning [1, 2, 3, 4, 5, 6, 7, 8, 9]. Most GNN models belong to the family of message-passing neural networks, where node representations are updated by passing messages and aggregating information from neighboring nodes [10, 11, 12, 13, 14, 15]. However, existing GNNs typically use permutation-invariant readout functions to aggregate the node representations into a graph embedding. Recent studies have shown that their expressive power is constrained by the 1-Weisfeiler-Lehman (1-WL) test [16, 17], which limits their ability to capture complex graph structures and affects their performance on graph-level tasks [18, 19, 20].

Thanks to the ability to model complex relationships, graph kernel methods have been powerful tools for analyzing and learning from graph-structured data [21]. By employing mechanisms such as substructure matching and topological feature extraction, these graph kernels have been able to effectively capture the structural information of graphs [22]. Despite their great successes, graph kernels rely on handcrafted kernel functions, which pose challenges in adapting flexibly to diverse task requirements. On the one hand, most graph kernel methods measure graph similarity in an implicit manner, thereby limiting their ability to incorporate with neural networks for learning task-relevant representations. On the other hand, certain kernel methods, such as the Neighborhood Hash Kernel [23] and Graphlet [24], employ the approach of explicit feature mappings. However, these methods typically represent graphs using the frequency of substructure occurrences or symbolic

[*]Corresponding author.

labels, ignoring the semantic information crucial for accurate graph similarity matching [25]. Unlike kernel methods, neural networks are capable of extracting task-relevant features from the data driven by the objective function. Moreover, the high computational complexity of kernel methods renders them impractical for large-scale datasets. Consequently, it is critically important to develop a novel framework that combines the structural feature-capturing advantages of graph kernels with the representation learning capabilities of neural networks.

An intuitive solution is to design a framework that facilitates the negotiation between the graph kernel learning and neural network optimization processes. Within this framework, a kernel serves as a measurement of graph similarity, while the neural network is incorporated to integrate this process with the task objective for task-specific representation learning. To fulfill this, there are two key challenges to be addressed, i.e., 1) the selected kernel should capture both the semantic and structural information of the graph; and 2) the similarity measurement process driven by the kernel is supposed to be differentiable, enabling the network to be end-to-end trainable. For the first challenge, inspired by the findings from previous work [26, 27] that distance encoding is more expressive than the 1-WL algorithm, we attempt to design the kernel based on the shortest paths. Since the nodes on the shortest paths provide rich semantic information, the positions of different nodes on the shortest path reflect the topological information of the path. For the second challenge, we explicitly map the graph into a vector space where similarity can be measured using the vector inner product. Since the vector inner product is differentiable, it facilitates the joint learning between kernel-induced features and the neural network to obtain more useful representations for specific tasks [28].

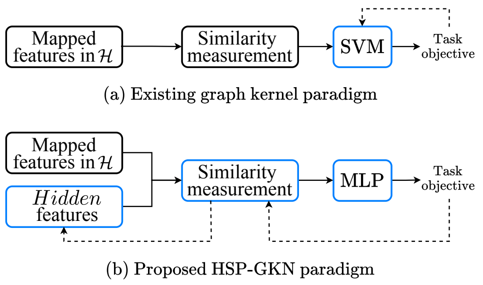
Mapped features in $\mathcal{H}$
Similarity measurement
SVM
Task objective
(a) Existing graph kernel paradigm
Mapped features in $\mathcal{H}$
$Hidden$ features
Similarity measurement
MLP
Task objective
(b) Proposed HSP-GKN paradigm

Figure 1: Optimization pipeline comparison of existing graph kernel methods and proposed HSP-GKN framework.

In this paper, we propose a simple yet effective kernel-based graph learning framework, termed Hierarchical Shortest-Path Graph Kernel Network (HSP-GKN). Specifically, we first design a computationally efficient graph kernel based on the shortest paths in the graph, which constructs an explicit feature mapping that transforms the graph into hierarchical features to preserve both the semantic and structural information of the given graph. Next, we integrate task objective and similarity measurement into a unified optimization framework. In this framework, we first define a set of trainable *hidden* graph features and then use the designed kernel to measure their similarity with the hierarchical features. The resultant similarity vector serves as the graph embedding that is fed into a multilayer perceptron (MLP) for training. Unlike previous graph kernel methods that isolate the learning processes of graph representations and objective optimization [29], the proposed HSP-GKN integrates them into a unified framework to learn representations that are more relevant to specific tasks, as illustrated in Figure 1. The dashed lines represent the optimization process of the task objective, with the blue borders indicating trainable components and $\mathcal{H}$ denoting a Hilbert space.

Our contributions include: (1) We design a computationally efficient kernel based on the shortest paths. It can be flexibly integrated with neural networks while preserving both the semantic and structural information of a given graph. (2) We develop a novel framework termed HSP-GKN which integrates the kernel we proposed with neural networks, enabling objective-driven learning of representations beneficial for specific tasks. (3) Experimental results on graph classification datasets across various domains demonstrate the proposed kernel and framework outperform traditional graph kernel methods and state-of-the-art models in terms of both performance and computational efficiency.

2 Related Work

Graph kernels. Most graph kernels follow the R-convolution framework [30], which decomposes graphs into substructures and computes similarities via kernel functions, aggregating them to yield the overall graph similarity. Different graph kernels are computed in distinct ways, we categorize them into explicitly and implicitly computed graph kernels in this work. The idea of explicitly computing graph kernels involves mapping a graph into a feature vector and measuring the similarity between the feature vectors of two graphs [23, 24, 31, 32]. These methods typically map graphs into feature vectors composed of the frequency of substructure occurrences, focusing mainly on graph topology while neglecting node semantic information. Unlike these methods, implicit computed kernels measure similarity between substructures that are hard to represent explicitly. The RW kernel [33] is one of the first methods that implicitly measures graph similarity based on the number of common walks between two graphs. Unlike random walks, shortest paths capture both local and global structures, while node attributes along paths provide rich semantics, motivating various shortest-path-based kernels [31, 34, 35, 36]. Despite their success, the implicit similarity computation limits the ability to integrate neural networks for learning task-specific representations. Moreover, the high computational complexity (e.g., SP kernel [34] has a complexity of $\mathcal{O}(n^4)$ with respect to the sample size n) restricts their applicability to large-scale datasets. In this paper, we design a graph kernel with a computational complexity of $\mathcal{O}(n^2 \log n)$. It employs explicit computation, allowing the similarity measurement process to be optimized end-to-end, facilitating both model efficiency and task-specific learning.

Graph kernel neural networks (GKNNs). Recent research has explored the integration of graph kernels and neural networks, leading to notable performance improvements [37, 38, 39, 40]. For example, RWNN [37] introduces random walk-based mechanisms to capture higher-order dependencies and improve node representation by incorporating both local and global structural information. GNN-lofi [39] leverages localized feature-based histogram intersection to enhance the representation learning of graph by focusing on local feature distribution matching. The success of GKNNs lies in their ability to combine the complementary strengths of graph kernels and neural networks [38, 41]. Kernel functions allow GKNNs to effectively capture structural similarities between graphs, while neural networks provide the capacity to learn complex and abstract graph representations. Despite their success, GKNNs still face several challenges. Scalability remains a major concern, particularly when applied to large-scale graph datasets. Moreover, existing methods either rely on implicitly computed kernels, which prevent the model from learning task-relevant representations, or focus solely on substructures while overlooking rich semantic information in graphs. In this paper, we construct an computationally efficient explicit feature mapping that incorporates both structural and semantic information, mapping graphs into vector space amenable to neural network processing.

3 Methodology

In this section, we first introduce key notations employed throughout the paper. Next, we describe the proposed Hierarchical Shortest-Path (HSP) graph kernel, which serves as the theoretical foundation of our framework. Finally, we present the end-to-end learning framework, i.e., Hierarchical Shortest-Path Graph Kernel Networks (HSP-GKN).

3.1 Notations

Let $G = (V, E, a)$ be an undirected graph, where V is the vertex set and E is the edge set, and $a\colon V \to \Sigma$ is a function that assigns attributes, either discrete or continuous, from a set Σ to nodes in the graph. Given two nodes $v_a, v_b \in V$, a path π of length $n-1$ from v_a to v_b in G is defined as a sequence of nodes $\pi = [v_1, v_2, \ldots, v_n]$, where $v_1 = v_a$, $v_n = v_b$, and $[v_i, v_{i+1}] \in E$ for all $i = 1, \ldots, n-1$. Let $\pi(i) = v_i$ denote the i-th node encountered while traversing along the path. In particular, we denote by $a(\pi)$ the concatenation of node attributes along this path. Let $|\pi|$ be the discrete length of π, and the diameter $\delta(G)$ of G is the maximum length of the shortest path between any two nodes in the graph. We use $\mathbf{X} \in \mathbb{R}^{n \times d}$ to denote the node attribute matrix of a graph, where n is the number of nodes and d is the dimension of node attribute. The attribute of a given node v_i corresponds to the i-th row of $\mathbf{X}$.

3.2 Hierarchical Shortest-Path Graph Kernel

In this subsection, we introduce the Hierarchical Shortest Path-Graph Kernel, which encodes both the semantic and topological information of graphs into explicit features and describes an efficient method for its computation.

Definition 3.1. GraphHopper (GH) kernel [36]. Given two graphs, G and G', let $\mathcal{SP}_l\left(G\right)$ and $\mathcal{SP}_l\left(G'\right)$ denote the set of all shortest paths of length l in graph G and G'. We can then define the

GH kernel as:

$$k\left(G,G'\right)=\sum_{l=1}^{L}\left[\sum_{\pi\in\mathcal{SP}_l(G)}\sum_{\pi'\in\mathcal{SP}_l(G')}k_{path}\left(\pi,\pi'\right)\right], \tag{1}$$

where L denotes the results of $min\ \left(\delta\left(G\right),\delta\left(G'\right)\right)$, and k_{path} the sum of the kernel over all nodes along the shortest paths.

However, all shortest paths need to be compared in GH kernel, and a graph can have at most $|V|^2$ shortest paths, the computational complexity of $k(G,G')$ results in $\mathcal{O}(|V|^2|V'|^2)$. This makes computation infeasible for large graphs. Moreover, like other kernels based on shortest paths, the GH kernel performs implicit computation, resulting in a computational complexity that is quadratic in relation to the number of graphs. Consequently, we design a computationally efficient instance of the GH kernel, referred to as the **Hierarchical Shortest-Path (HSP) Graph Kernel**.

In the HSP kernel, we restrict the node-level kernel k_{node} to a linear kernel, that is:

$$k_{path}(\pi,\pi')=\sum_{i=1}^{|\pi|+1}k_{node}\ \left(\pi\left(i\right),\pi'\left(i\right)\right),\ \text{with}\ k_{node}(v,v')=\left\langle a\left(v\right),a\left(v'\right)\right\rangle. \tag{2}$$

Since the linear kernel is positive definite and the sum of linear kernels is still a linear kernel, k_{path} remains a linear kernel and is also positive definite. By exploiting the linear additivity property of the linear kernel, we construct a hierarchical feature mapping that aggregates the set of shortest paths of the same length into a *cohesive shortest-path*.

3.3 Hierarchical Graph Feature Mapping

Differing from previous graph feature mapping, which maps the graph to a fixed-dimensional feature, we transform the graph into a feature set organized by the shortest-path length. The hierarchical graph feature mapping steps are grouping all shortest paths of the same length and aggregating the node attributes at the corresponding indices along the shortest paths, therefore preserving the semantic information of nodes and their relative positions along the shortest-path.

Definition 3.2. Given a graph G, let $\mathcal{SP}_l(G)$ denote the set of shortest paths of length l in G. We define the hierarchical graph feature mapping as:

$$\Phi(G)=\{\phi_l(G)\mid l=1,2,\ldots,\delta(G)\},\ \text{with}\ \phi_l(G)=\sum_{\pi\in\mathcal{SP}_l(G)}a(\pi). \tag{3}$$

This formula aggregates the shortest paths of the same length into a single *cohesive shortest-path*, and we can view the graph G as one that contains only $\delta(G)$ shortest paths. This approach eliminates the need to compare each pair of shortest paths of the same length between the two graphs, thereby significantly reducing computational cost. We can now simplify Eq. (1) as follows:

$$k\left(G,G'\right)=\sum_{l=1}^{L}\left\langle\phi_l\left(G\right),\phi_l\left(G'\right)\right\rangle, \tag{4}$$

where L denotes the results of $min\ \left(\delta\left(G\right),\delta\left(G'\right)\right)$. The derivation of Eq. (4) see Appendix A.1. In this case, the computational complexity of $k\left(G,G'\right)$ is $\mathcal{O}\big(\delta(G)^2\delta(G')^2\big)$, where $\delta(G)$ is the diameter of graph G, in real-world graphs $\delta(G)$ is typically much smaller than $|V|$. However, computing $\Phi(G)$ requires at most $|V|^2$ addition operations on $a(\pi)$, which remains a high-complexity operation. In the following, we further discuss how to compute $\Phi(G)$ efficiently.

Given the node attribute matrix $\mathbf{X}$ and the set of all shortest paths of length l in the graph G, denoted as $\mathcal{SP}_l(G)$. $\phi_l(G)$ essentially represents the weighted sum of all nodes on the shortest paths of length l. Therefore, when searching for the shortest paths in the graph, we simultaneously maintain a matrix $\mathbf{M}_l\in\mathbb{R}^{n\times(l+1)}$, where the entry $[\mathbf{M}_l]_{ij}$ counts how many times node i appears at the j-th coordinate of the shortest paths in $\mathcal{SP}_l(G)$. To avoid confusion, it is worth noting that M_l is different from $M(v)$ as defined in the GH kernel, where $[M(v)]_{i,j}$ represents the number of times v appears as the i^{th} node on a shortest path of discrete length j. From this, it is straightforward to derive that:

$$\boldsymbol{\phi}_l(G)=\text{vec}(\mathbf{X}^\top\mathbf{M}_l), \tag{5}$$

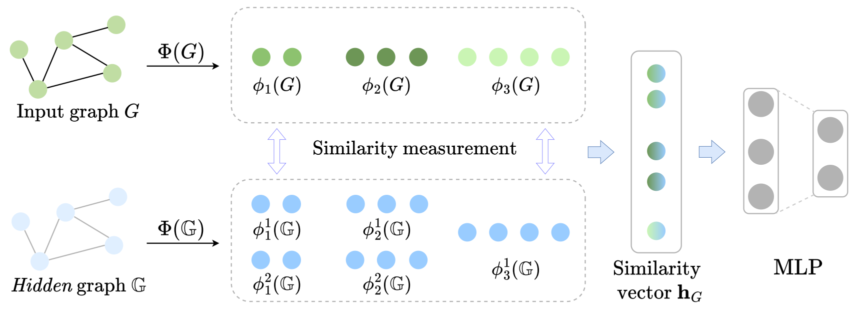

Figure 2: An overview of HSP-GKN. $\Phi(\cdot)$ is the hierarchical graph feature mapping.

where $\mathrm{vec}(\cdot)$ denotes the vectorization operator, which transforms a matrix into a vector by stacking the columns of the matrix one after another. Now, computing $\Phi(G)$ only need to perform $\delta(G) \times |V|$ multiplication and addition operations, which is much smaller than the $|V|^2$ additions required.

Explicit feature mapping. It is known that if a positive definite kernel $k(\cdot, \cdot)$ is defined on $\mathcal{X} \times \mathcal{X}$, there exists a mapping $\phi : \mathcal{X} \to \mathcal{H}$ that maps the set $\mathcal{X}$ into a Hilbert space $\mathcal{H}$ with inner product $\langle \cdot, \cdot \rangle$, such that for $\forall x_i, x_j \in \mathcal{X}$, it holds that:

$$k\left(x_i, x_j\right) = \left\langle \phi\left(x_i\right), \phi\left(x_j\right) \right\rangle_{\mathcal{H}}. \tag{6}$$

Therefore, for a positive definite kernel, there exists a feature space where similarity can be computed using an inner product. However, the hierarchical feature mapping that we define represents graphs in the form of sets, which cannot be computed using standard inner products. Therefore, the set-based representation does not constitute the feature space associated with our kernel.

Since our goal is to construct an explicit feature mapping that transforms graphs into vector representations suitable for neural network processing, we next explore how to explicitly map this set into a vector space corresponding to the HSP kernel.

Lemma 3.1. *The hierarchical shortest-path graph kernel is positive definite.*

Lemma 3.2. *If a graph contains the shortest path of length l, then the shortest paths of length* $1, 2, \ldots, l-1$ *must also exist.*

The proof of Lemma 3.1 and Lemma 3.2 can be found in Appendix A.2. According to Lemma 3.2, we can conclude that although some graphs have different diameters, sorting the shortest paths of two graphs in ascending order of length allows their path lengths to overlap partially within a certain range. Now, we arrange $\boldsymbol{\phi}_l(G)$ into an ordered set by sorting according to the corresponding shortest-path length l in ascending order. Subsequently, by applying a concatenation operation, we obtain the feature vector representation of G:

$$\boldsymbol{\Phi}(G) = \mathrm{CONCAT}\big(\boldsymbol{\phi}_1(G), \boldsymbol{\phi}_2(G), \ldots, \boldsymbol{\phi}_{\delta(G)}(G)\big), \tag{7}$$

where $\mathrm{CONCAT}(\cdot)$ is the vertical concatenation of vectors. It follows that $\boldsymbol{\phi}_l(G)$ and $\boldsymbol{\phi}_l(G')$ have corresponding positions in $\boldsymbol{\Phi}(G)$ and $\boldsymbol{\Phi}(G')$. Let $\odot$ denote the aligned inner product of vectors of possibly different lengths, e.g.

$$[a, b, c] \odot [d, e] = [a, b] \cdot [d, e]. \tag{8}$$

Referring to Eq. (4), we obtain:

$$k\left(G, G'\right) \ = \boldsymbol{\Phi}(G) \odot \boldsymbol{\Phi}(G'). \tag{9}$$

We now further discuss the feature vector representation of G. If we zero-extend $\boldsymbol{\Phi}(G)$ to a sufficiently large dimension that exceeds the dimension of any possible graph, Eq. (9) can be extended to the standard inner product computation. Therefore, the extended vector is essentially the mapped feature vector induced by the HSP kernel in the feature space $\mathcal{H}$.

3.4 Graph Kernel Network Based on HSP

In this subsection, we present our proposed learning framework, HSP-GKN, and explain how to integrate the similarity measurement of the HSP kernel with objective optimization into a unified framework. An overview of HSP-GKN is illustrated in Figure 2.

As defined previously, hierarchical graph feature mapping transforms a graph into a set of graph features, referred to as hierarchical features. The main steps of HSP-GKN begin with defining a set of learnable *hidden* graph features. Then, a similarity vector is obtained by measuring the similarity between these learnable features and the hierarchical features. Finally, we feed this similarity vector as the graph embedding into MLP and optimize the framework via an objective function.

Hidden graph features. From Eq. (9), we know that the similarity measurement process in HSP kernels is non-learnable, as each graph is mapped to a fixed vector representation. Inspired by RWNN [37], we extend the concept of *hidden* graph features, i.e., parameterized graph features into paths. Specifically, the set of *hidden* graph features is defined as:

$$\{\phi_l^{(i)}(\mathbb{G}) \mid i = 1, \ldots, n_l;\ l = 1, \ldots, L\},\ \text{with}\ n_1 + n_2 + \cdots + n_L = N. \tag{10}$$

Here, L and N are hyperparameters to adjust the hierarchy and number of *hidden* graph features, n_l and the number of shortest paths of length l in all the graphs of the training set are proportional. It is worth noting that $\mathbb{G}$ is not an actual graph; instead, it was introduced to facilitate the definition. We can consider this set as consisting of features derived from N learnable graphs, each obtained through the feature mapping.

End-to-end optimization framework. After defining the *hidden* graph features, we use the linear kernel used by the HSP kernel to measure the similarity between graph G and these features, resulting in a similarity vector:

$$\mathbf{h}_G = [\left\langle \phi_l(G), \phi_l^{(i)}(\mathbb{G}) \right\rangle \mid i = 1, \ldots, n_l;\ l = 1, \ldots, L]. \tag{11}$$

In what follows, we provide more details about the implementation of the proposed HSP-GKN framework.

Given an input graph G, we first obtain the hierarchical feature vector representations set of the graph $\{\boldsymbol{\phi}_l(G) \mid l = 1, 2, \ldots, \delta(G)\}$ through Eq. (5). Then, we define a set of trainable parameters that correspond to the set of *hidden* graph features:

$$\mathcal{W} = \{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_L\}, \tag{12}$$

where $\mathbf{W}_l \in \mathbb{R}^{[(l+1)\times d]\times n_l}$, and d is the dimension of the node attributes.The i-th column of $\mathbf{W}_l$ represents the vector representation of $\phi_l^{(i)}(\mathbb{G})$. Subsequently, we can obtain the similarity vector between the graph G and the feature vectors:

$$\mathbf{h}_G = [\boldsymbol{\phi}_1(G)^\top \mathbf{W}_1, \boldsymbol{\phi}_2(G)^\top \mathbf{W}_2, \ldots, \boldsymbol{\phi}_L(G)^\top \mathbf{W}_L],$$

with

$$\boldsymbol{\phi}_l(G) = \mathbf{0} \quad \text{if} \quad l > \delta(G). \tag{13}$$

This similarity vector is then used as the graph embedding and fed into the MLP for learning. Since the inner product calculation of vectors is differentiable, the entire framework can be trained end-to-end. Under the guidance of the objective function, HSP-GKN can learn *hidden* graph features that are better aligned with downstream tasks, achieving improved performance compared to fixed similarity measurements.

3.5 Computational Complexity Analysis

For feature mapping, we use Dijkstra's algorithm to compute the all-pairs shortest paths for each graph, with a time complexity of $\mathcal{O}\left(n^2 \log n + nm\right)$, where n is the number of vertices and m is the number of edges in the graph. When generating the hierarchical feature vector representation, matrix multiplication $\mathbf{X}^\top\left(\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_{\delta(G)}\right)$ is required, where the time complexity is $\mathcal{O}\left(\ (\delta+1)\, nd\ \right)$, where δ is the graph diameter and d is the dimension of node attributes. Similarity computation uses the inner product of vectors with a time complexity of $\mathcal{O}\big((\delta+1)\delta d\big)$. Therefore,

Table 1: Classification accuracy (± standard deviation) of our HSP, HSP-GKN, and the baselines on the TUDatasets. Best performance is highlighted in **bold**. The second-best performance is indicated with an <u>underline</u>. OOT means cannot completed within 24 hours. We use * to indicate the best performance among kernel methods.

	MUTAG	**NCI1**	**PTC_MR**	**DD**	**BZR**	**PROTEINS**
SP	82.4 ± 5.5	72.5 ± 2.0	60.2 ± 9.4	77.9 ± 4.5	83.7 ± 4.5	74.9 ± 3.6
WL-SP	81.4 ± 8.7	60.8 ± 2.4	54.5 ± 9.8	76.0 ± 3.5	OOT	72.1 ± 3.1
GH	82.5 ± 5.8	71.0 ± 2.3	60.2 ± 9.4	OOT	82.3 ± 7.2	74.8 ± 2.4
CORE-SP	85.1 ± 6.8	73.8 ± 1.4	57.3 ± 9.7	79.3 ± 3.8	OOT	76.5 ± 3.9
GraphSAGE	83.6 ± 9.6	76.0 ± 1.8	61.7 ± 4.9	72.9 ± 2.0	81.2 ± 4.2	74.3 ± 3.8
DiffPool	79.8 ± 6.7	76.9 ± 1.9	61.1 ± 5.6	75.0 ± 3.5	84.5 ± 4.2	72.5 ± 3.5
DGCNN	84.0 ± 7.1	76.4 ± 1.7	59.5 ± 6.9	76.6 ± 4.3	81.8 ± 4.4	73.2 ± 3.2
GIN	84.7 ± 6.7	80.0 ± 1.4	59.1 ± 7.0	75.3 ± 2.9	85.4 ± 5.1	72.8 ± 3.6
KerGNN	84.2 ± 5.1	<u>82.8 ± 1.8</u>	68.4 ± 2.7	<u>78.9 ± 3.5</u>	81.8 ± 3.9	76.5 ± 3.9
PathNN	87.3 ± 6.1	81.1 ± 1.2	65.6 ± 2.4	77.0 ± 3.1	86.6 ± 4.2	75.2 ± 3.9
MMD	<u>91.5 ± 6.5</u>	58.4 ± 2.6	62.8 ± 1.6	60.2 ± 2.0	<u>91.0 ± 11</u>	<u>77.6 ± 2.5</u>
GPN	79.3 ± 5.7	80.8 ± 0.5	67.7 ± 1.2	77.0 ± 0.8	84.2 ± 1.1	75.6 ± 0.8
Cosmo'2024 [40]	85.2 ± 2.2	71.5 ± 1.2	60.1 ± 1.9	75.3 ± 2.7	85.5 ± 1.9	75.3 ± 1.1
DHAKR	89.9 ± 1.0	77.3 ± 1.3	<u>68.8 ± 0.9</u>	74.6 ± 1.7	87.2 ± 2.8	77.5 ± 0.6
HSP	$85.8 \pm 6.3^*$	$75.2 \pm 1.5^*$	$62.4 \pm 7.2^*$	78.3 ± 2.7	$84.8 \pm 5.1^*$	73.3 ± 3.8
HSP-GKN	**97.5 ± 2.2**	**83.6 ± 1.9**	**76.3 ± 3.4**	**84.2 ± 2.6**	**93.4 ± 3.6**	**79.5 ± 1.7**

	ENZYMES	**COLLAB**	**IMDB-B**	**IMDB-M**	**REDDIT-B**	**REDDIT-M-5K**
SP	37.3 ± 8.7	58.8 ± 1.2	58.2 ± 4.7	39.2 ± 2.3	81.7 ± 2.5	47.9 ± 1.9
WL-SP	27.3 ± 7.4	58.8 ± 1.2	58.2 ± 4.7	39.2 ± 2.3	OOT	OOT
GH	67.7 ± 6.5	60.0 ± 1.4	59.4 ± 3.4	39.5 ± 2.6	OOT	OOT
CORE-SP	39.5 ± 9.3	OOT	68.5 ± 3.9	49.4 ± 0.5	<u>91.0 ± 1.8</u>	54.35 ± 0.1
GraphSAGE	58.2 ± 6.0	73.9 ± 1.7	68.8 ± 4.5	47.6 ± 3.5	84.3 ± 1.9	50.0 ± 1.3
DiffPool	59.5 ± 5.6	68.9 ± 2.0	68.4 ± 3.3	45.6 ± 3.4	89.1 ± 1.6	53.8 ± 1.4
DGCNN	38.9 ± 5.7	71.2 ± 1.9	69.2 ± 3.0	45.6 ± 3.4	87.8 ± 2.5	49.2 ± 1.2
GIN	59.6 ± 4.5	75.6 ± 2.3	71.2 ± 3.9	48.5 ± 3.3	89.9 ± 1.9	**56.1 ± 1.7**
KerGNN	62.1 ± 5.5	75.1 ± 2.3	74.4 ± 4.3	51.6 ± 3.1	89.5 ± 1.6	52.7 ± 1.2
PathNN	**73.0 ± 5.2**	76.9 ± 3.4	72.6 ± 3.3	50.8 ± 4.5	89.2 ± 1.1	53.9 ± 1.3
MMD	48.4 ± 3.3	57.9 ± 1.5	59.1 ± 2.7	37.2 ± 2.1	71.3 ± 2.7	37.0 ± 1.2
GPN	65.6 ± 1.3	**83.6 ± 0.5**	75.1 ± 2.2	49.4 ± 0.9	82.7 ± 1.4	46.9 ± 0.9
Cosmo'2024 [40]	66.4 ± 3.2	73.6 ± 1.1	69.9 ± 1.4	45.6 ± 1.2	84.9 ± 2.5	42.6 ± 1.1
DHAKR	64.4 ± 4.7	73.8 ± 1.7	<u>75.2 ± 2.6</u>	<u>52.1 ± 2.5</u>	87.4 ± 1.3	49.2 ± 1.6
HSP	54.6 ± 6.8	$75.4 \pm 2.7^*$	$71.5 \pm 4.9^*$	$50.7 \pm 3.1^*$	83.6 ± 3.6	51.3 ± 2.2
HSP-GKN	<u>70.4 ± 3.1</u>	<u>80.1 ± 1.8</u>	**78.5 ± 2.8**	**55.4 ± 1.7**	**91.7 ± 1.2**	<u>55.7 ± 1.3</u>

the overall time complexity for computing the similarity is $\mathcal{O}\big(n^2 \log n + nm + (\delta + 1)(\delta + n)d\big)$. Since real-world graphs are typically sparse (i.e., $m \ll n^2$) and the graph diameter δ is usually much smaller than n [42], the computational cost of the HSP kernel primarily depends on the number of nodes in the graph. Furthermore, since each graph requires feature mapping computation only once, the computational complexity of average similarity calculation is significantly lower than the aforementioned complexity. Our experiments show a linear in the number of graphs and quadratic in the number of nodes.

4 Experiments

In this section, we first evaluate the performance of the HSP kernel and its corresponding HSP-GKN framework on graph classification datasets. Next, we study the runtime behaviour of proposed methods on synthetic graphs and real-world datasets. Finally, we conduct ablation experiments and hyperparameter analysis on HSP-GKN.

4.1 Graph Classification

Datasets. We evaluate the proposed kernel and framework on 12 datasets from different domains and of varying scales, all included in the TUDatasets collection [43]. These datasets are categorized as follows: small molecule datasets (BZR, MUTAG, NCI1, PTC_MR), bioinformatics datasets (DD, ENZYMES, PROTEINS), and social network datasets (COLLAB, IMDB-B, IMDB-M, REDDIT-B, REDDIT-M-5K). More details about these datasets can be found in the Appendix B.

In the HSP-GKN framework, each graph is transformed into a vector of dimension $(L+1)Ld/2$ before being fed into the model for training, which is much more compact than the original graph. Although additional preprocessing time is required, the HSP avoids processing the entire graph at each iteration. Therefore, it is comparable with GNNs in terms of overall training time.

Baselines. We compare the proposed kernel and model against the following graph kernels based on shortest path: shortest path kernel (SP) [34], Weisfeiler-Lehman shortest path kernel (WL-SP) [31], graphHopper kernel (GH) [36] and core shortest path kernel (CORE-SP) [35]. We also compare our model against various GNNs, including (1) classical GNNs such as GraphSAGE [44], DiffPool [45], DGCNN [46] and GIN [16]; (2) state-of-the-art GNNs such as PathNN [47], MMD [41], and GPN [48]. Furthermore, we compare our method against recently GKNNs including kerGNN [38], Cosmo'2024 [40], and DHAKR [49].

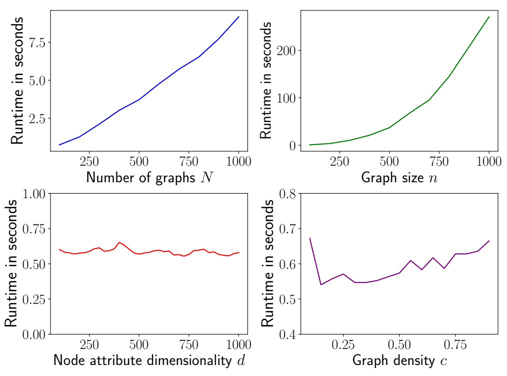

Figure 3: Dependence of runtime on N, n, d and c on synthetic graphs.

Table 2: Running time of different graph kernels on TUDatasets (seconds). OOT: timeout over 24 hours.

	ENZYMES	NCI1	REDDIT-M-5K
SP	7.92	16.33	12944.24
WL-SP	49.47	367.21	OOT
GH	356.80	2935.79	OOT
CORE-SP	26.48	49.87	31779.70
HSP	1.71	12.70	3869.14

Table 3: One-epoch running time of different GNNs on TUDatasets (seconds).

	ENZYMES	NCI1	REDDIT-M-5K
GraphSAGE	0.326	0.902	5.420
DiffPool	2.195	13.785	26.427
DGCNN	0.292	0.974	4.093
GIN	0.412	2.322	5.193
HSP-GKN	0.013	0.009	0.011

Configurations. We follow [50] and evaluate TUDatasets using a 10-fold cross-validation using their provided data splits for a fair comparison. For datasets without predefined splits, we adopt the splitting method provided in [50] and apply the same split across all baseline models. We run each experiment 10 times to obtain the average accuracy and standard deviation. For the HSP-GKN model configuration, we use a two-layer MLP, where each layer's dimension is half of the previous layer's dimension. For the selection of hyperparameters, we use Optuna [51] to tune the hyperparameters through an automated search process.

Result analysis. Table 1 illustrates average prediction accuracies and standard deviations. We observe that the proposed HSP-GNN framework outperforms the baselines on 9 out of the 12 datasets while it provides the second-best accuracy on the remaining 3 datasets. On the MUTAG, PTC_MR, and DD datasets, our framework achieves absolute accuracy improvements of 6.0%, 7.5%, and 5.3%, respectively, over the second-best accuracy. It is worth noting that, although we restrict the node kernel of HSP to a linear kernel, it still outperforms the shortest-path-based kernel baselines on 7 out of the 12 datasets. Furthermore, HSP-GKN achieves significantly higher classification accuracy than HSP across all datasets, demonstrating the effectiveness of our end-to-end framework. Overall, our results show that HSP-GKN achieves high performance levels on the TUDatasets.

4.2 Runtime Behaviour Study

Here, we experimentally examine the runtime performance of the HSP kernel and HSP-GKN framework.

Datasets. We assessed the behaviour on randomly generated graphs with respect to four parameters: dataset size N, graph size n, node attribute dimensionality d, and graph density c. For read-world

Table 4: Classification accuracy improvement over the HSP kernel. HSP-MLP refers to removing *hidden* graph features and directly inputting the graph feature vectors into the MLP for training.

	ENZYMES	NCI1	REDDIT-M-5K	PROTEINS
HSP-MLP	7.5 ↑	2.1 ↑	0.3 ↑	2.8 ↑
HSP-GKN	15.8 ↑	8.4 ↑	4.4 ↑	6.2 ↑

datasets, we selected 3 datasets from the TUDatasets. These datasets include large-scale dataset COLLAB, large graph dataset REDDIT-M-5K, and continuous attribute dataset ENZYMES.

Baselines. We compared the runtime with other graph kernel methods on real-world datasets, including SP, WL-SP, GH and CORE-SP kernel. We also evaluate the runtime in comparison to classical GNNs, including GraphSAGE, DiffPool, DGCNN and GIN.

Configurations. For randomly generated graphs, we kept 3 out of 4 parameters fixed at their default values and varied the fourth parameter. The default values we used were 100 for N, 100 for n, 100 for d, and 0.3 for the graph density c. All experiments were conducted on a system equipped with an 8-core Intel Xeon E5-2667v4@3.2GHz with 256 GB of RAM and an NVIDIA 3090 GPU. For the graph kernel methods used for comparison, we utilized the library provided by [52] to measure the runtime. All graph kernel methods were executed on the CPU using a single core.

Results analysis. On synthetic graphs, as shown in Figure 3, we observe that the HSP kernel scales linearly with dataset size N. When varying the number of nodes n per graph, we observe that the runtime of the HSP kernel scales quadratically with n, as mentioned in the complexity analysis. We also observe that changing the dimension of node attributes d does not significantly increase the computation time. However, when varying the graph density c, the computation time peaks at a density of 0.1 and 0.9. This is because, at low density, the graph diameter δ becomes large, while at high density, the number of edges m is significantly higher than n.

With real-world datasets, as shown in Table 2, our kernel shows significantly lower computation time across all datasets, particularly on large graphs. For example, REDDIT-M-5K contains 4,999 graphs, with average node counts of 508. Some kernels even fail to complete computation within 24 hours. Since the HSP kernel is insensitive to the dimension of node attributes, it exhibits high scalability on ENZYMES datasets. In the comparison with GNNs, as shown in Table 3. Even though our method requires preprocessing, which takes slightly less time than computing the kernel matrix (see Table 2), the overall training time is significantly lower than other models on the ENZYMES and NCI1 datasets. Although preprocessing takes longer on the REDDIT-M-5K dataset due to the larger average node count, it can be parallelized to reduced preprocessing time.

4.3 Ablation Study

To validate the effectiveness of the proposed *hidden* graph features, we removed the *hidden* graph features and directly input the graph feature vectors into the MLP for training. As shown in Table 4, HSP-GKN achieves a more significant performance improvement over the HSP kernel. This underscores the importance of employing a trainable similarity measurement and confirms that integrating the HSP with *hidden* graph features facilitates learning task-specific graph representations that offer better performance.

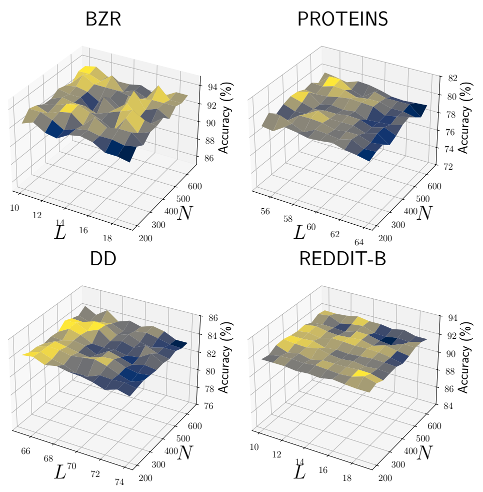

Figure 4: The sensitivity of HSP-GKN with the variation of L and N on four datasets.

4.4 Analysis of Hyperparameter L and N

As can be seen in Eq. (10), HSP-GKN introduces two hyperparameters L and N. We conduct experiments to show the effect of this parameter on four datasets. Figure 4 illustrates the performance variation of HSP-GKN when L and N varies. From these figures, we observe that 1) as the size of dataset increases, the performance of the method becomes less sensitive to hyperparameters; 2) the performance of the method is overall stable across a wide range of L and N.

5 Conclusion and Future Discussion

We propose the Hierarchical Shortest-Path Graph Kernel Network (HSP-GKN). In our method, we introduce the *hidden* graph features to integrate the HSP kernel similarity measurement and downstream tasks into a unified framework. Moreover, our proposed HSP kernel simultaneously preserves both the semantic and structural information of the graphs. Extensive experimental results demonstrate the effectiveness and superiority of the HSP kernel and HSP-GKN framework compared to state-of-the-art methods. In the future, we plan to extend our methods to graph regression and other tasks.

Limitations and broader impacts. HSP incorporates the node attribute while ignoring the edge features in graphs. We mainly focus on leveraging node features and graph topology in this work, making it broadly applicable across a wide range of datasets. However, the framework is flexible and can be extended to support edge attributes in several ways. One possible extension is concatenating the edge attributes along the shortest path with the corresponding node attributes into a unified vector representation. Since the shortest paths are already computed as part of the kernel, this concatenation would introduce minimal additional computational overhead. Alternatively, we can define a separate kernel for edge features to measure the similarity between two shortest paths based on their edge attributes. This edge-based similarity can then be combined with node-based similarity to provide a more comprehensive comparison between shortest paths. We leave this as a promising direction for future work and believe that such an extension can further enhance the model's expressiveness on attributed graphs with rich edge information. The end-to-end manner in graph kernel methods is inspiring for the community.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (NSFC) (Grant No. 62506102, 62562026), the Key Research and Development Program of Hainan Province (Grant No. ZDYF2024GXJS014, ZDYF2023GXJS163), and the Natural Science Foundation of Hainan University (No. XJ2400009401).

References

[1] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32(1):4–24, 2021.

[2] Jingxin Liu, Renda Han, Wenxuan Tu, Haotian Wang, Junlong Wu, and Jieren Cheng. Federated node-level clustering network with cross-subgraph link mending. In *Proceedings of the International Conference on Machine Learning*, 2025.

[3] Lele Fu, Bowen Deng, Sheng Huang, Tianchi Liao, Shirui Pan, and Chuan Chen. Less is more: Federated graph learning with alleviating topology heterogeneity from a causal perspective. In *Proceedings of the Forty-second International Conference on Machine Learning*, 2025.

[4] Sheng Huang, Lele Fu, Tianchi Liao, Bowen Deng, Chuanfu Zhang, and Chuan Chen. Fedbg: Proactively mitigating bias in cross-domain graph federated learning using background data. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 5408–5416, 2025.

[5] Wenxuan Tu, Renxiang Guan, Sihang Zhou, Chuan Ma, Xin Peng, Zhiping Cai, Zhe Liu, Jieren Cheng, and Xinwang Liu. Attribute-missing graph clustering network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 15392–15401, 2024.

[6] Wenxuan Tu, Sihang Zhou, Xinwang Liu, Xifeng Guo, Zhiping Cai, En Zhu, and Jieren Cheng. Deep fusion clustering network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 9978–9987, 2021.

[7] Wenxuan Tu, Sihang Zhou, Xinwang Liu, Yue Liu, Zhiping Cai, En Zhu, Changwang Zhang, and Jieren Cheng. Initializing then refining: A simple graph attribute imputation network. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 3494–3500, 2022.

[8] Renxiang Guan, Wenxuan Tu, Siwei Wang, Jiyuan Liu, Dayu Hu, Chang Tang, Yu Feng, Junhong Li, Baili Xiao, and Xinwang Liu. Structure-adaptive multi-view graph clustering for remote sensing data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 16933–16941, 2025.

[9] Yaowen Hu, Wenxuan Tu, Yue Liu, Miaomiao Li, Wenpeng Lu, Zhigang Luo, Xinwang Liu, and Ping Chen. Divide-then-rule: A cluster-driven hierarchical interpolator for attribute-missing graphs. In *Proceedings of the ACM International Conference on Multimedia*, 2025.

[10] Zhao Kang, Chong Peng, Qiang Cheng, Xinwang Liu, Xi Peng, Zenglin Xu, and Ling Tian. Structured graph learning for clustering and semi-supervised classification. *Pattern Recognition*, 110:107627, 2021.

[11] Luca Cosmo, Anees Kazi, Seyed-Ahmad Ahmadi, Nassir Navab, and Michael M. Bronstein. Latent-graph learning for disease prediction. In *Medical Image Computing and Computer Assisted Intervention*, pages 643–653, 2020.

[12] Wenxuan Tu, Sihang Zhou, Xinwang Liu, Zhiping Cai, Yawei Zhao, Yue Liu, and Kunlun He. Wage: Weight-sharing attribute-missing graph autoencoder. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(7):5760–5777, 2025.

[13] Wenxuan Tu, Sihang Zhou, Xinwang Liu, Chunpeng Ge, Zhiping Cai, and Yue Liu. Hierarchically contrastive hard sample mining for graph self-supervised pre-training. *IEEE Transactions on Neural Networks and Learning Systems*, 35(11):16748–16761, 2024.

[14] Yaowen Hu, Wenxuan Tu, Yue Liu, Xinhang Wan, Junyi Yan, Taichun Zhou, and Xinwang Liu. Scalable attribute-missing graph clustering via neighborhood differentiation. In *Proceedings of the International Conference on Machine Learning*, 2025.

[15] Wenxuan Tu, Bin Xiao, Xinwang Liu, Sihang Zhou, Zhiping Cai, and Jieren Cheng. Revisiting initializing then refining: An incomplete and missing graph imputation network. *IEEE Transactions on Neural Networks and Learning Systems*, 36(3):3244–3257, 2025.

[16] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proceedings of the International Conference on Learning Representations*, 2019.

[17] Jiarui Feng, Lecheng Kong, Hao Liu, Dacheng Tao, Fuhai Li, Muhan Zhang, and Yixin Chen. Extending the design space of graph neural networks by rethinking folklore weisfeiler-lehman. In *Advances in Neural Information Processing Systems*, pages 9029–9064, 2023.

[18] Kamilia Zaripova, Luca Cosmo, Anees Kazi, Seyed-Ahmad Ahmadi, Michael M Bronstein, and Nassir Navab. Graph-in-graph (gig): Learning interpretable latent graphs in non-euclidean domain for biological and healthcare applications. *Medical Image Anal.*, 88:102839, 2023.

[19] Jingxin Liu, Jieren Cheng, Renda Han, Wenxuan Tu, Jiaxin Wang, and Xin Peng. Federated graph-level clustering network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 18870–18878, 2025.

[20] Peng Zhou, Xinwang Liu, Liang Du, and Xuejun Li. Self-paced adaptive bipartite graph learning for consensus clustering. *ACM Transactions on Knowledge Discovery from Data*, 17(5):1–35, 2023.

[21] Wei Ju, Zheng Fang, Yiyang Gu, Zequn Liu, Qingqing Long, Ziyue Qiao, Yifang Qin, Jianhao Shen, Fang Sun, Zhiping Xiao, Junwei Yang, Jingyang Yuan, Yusheng Zhao, Yifan Wang, Xiao Luo, and Ming Zhang. A comprehensive survey on deep graph representation learning. *Neural Networks*, 173:106207, 2024.

[22] Zhengyang Mao, Wei Ju, Siyu Yi, Yifan Wang, Zhiping Xiao, Qingqing Long, Nan Yin, Xinwang Liu, and Ming Zhang. Learning knowledge-diverse experts for long-tailed graph classification. *ACM Transactions on Knowledge Discovery from Data*, 19(2):1–24, 2025.

[23] Shohei Hido and Hisashi Kashima. A linear-time graph kernel. In *IEEE International Conference on Data Mining*, pages 179–188, 2009.

[24] Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. Efficient graphlet kernels for large graph comparison. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 488–495, 2009.

[25] Yaxu Xie, Alain Pagani, and Didier Stricker. SG-PGM: partial graph matching network with semantic geometric fusion for 3d scene graph alignment and its downstream tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 28401–28411, 2024.

[26] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. *Advances in Neural Information Processing Systems*, pages 4465–4478, 2020.

[27] Caterina Graziani, Tamara Drucks, Fabian Jogl, Monica Bianchini, Franco Scarselli, and Thomas Gärtner. The expressive power of path-based graph neural networks. In *Proceedings of the International Conference on Machine Learning*, 2024.

[28] Wei Ju, Siyu Yi, Yifan Wang, Zhiping Xiao, Zhengyang Mao, Hourun Li, Yiyang Gu, Yifang Qin, Nan Yin, Senzhang Wang, Xinwang Liu, Xiao Luo, Philip S. Yu, and Ming Zhang. A survey of graph neural networks in real world: Imbalance, noise, privacy and OOD challenges. *CoRR*, abs/2403.04468, 2024.

[29] Zhenyu Yang, Ge Zhang, Jia Wu, Jian Yang, Quan Z. Sheng, Shan Xue, Chuan Zhou, Charu Aggarwal, Hao Peng, Wenbin Hu, Edwin R. Hancock, and Pietro Liò. State of the art and potentialities of graph-level learning. *ACM Comput. Surv.*, 57(2):28:1–28:40, 2025.

[30] David Haussler et al. Convolution kernels on discrete structures. Technical report, Citeseer, 1999.

[31] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12(9):2539–2561, 2011.

[32] Xinwang Liu, Jianping Yin, Lei Wang, Lingqiao Liu, Jun Liu, Chenping Hou, and Jian Zhang. An adaptive approach to learning optimal neighborhood kernels. *IEEE transactions on cybernetics*, 43(1):371–384, 2012.

[33] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the international conference on machine learning*, pages 321–328, 2003.

[34] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Proceedings of the IEEE international conference on data mining*, pages 8–pp, 2005.

[35] Giannis Nikolentzos, Polykarpos Meladianos, Stratis Limnios, and Michalis Vazirgiannis. A degeneracy framework for graph similarity. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 2595–2601, 2018.

[36] Aasa Feragen, Niklas Kasenburg, Jens Petersen, Marleen de Bruijne, and Karsten M. Borgwardt. Scalable kernels for graphs with continuous attributes. *Advances in neural information processing systems*, pages 216–224, 2013.

[37] Giannis Nikolentzos and Michalis Vazirgiannis. Random walk graph neural networks. *Advances in Neural Information Processing Systems*, pages 16211–16222, 2020.

[38] Aosong Feng, Chenyu You, Shiqiang Wang, and Leandros Tassiulas. Kergnns: Interpretable graph neural networks with graph kernels. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 6614–6622, 2022.

[39] Alessandro Bicciato, Luca Cosmo, Giorgia Minello, Luca Rossi, and Andrea Torsello. Gnn-lofi: A novel graph neural network through localized feature-based histogram intersection. *Pattern Recognit.*, 148:110210, 2024.

[40] Luca Cosmo, Giorgia Minello, Alessandro Bicciato, Michael M. Bronstein, Emanuele Rodolà, Luca Rossi, and Andrea Torsello. Graph kernel neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 36(4):6257–6270, 2025.

[41] Yan Sun and Jicong Fan. MMD graph kernel: Effective metric learning for graphs via maximum mean discrepancy. In *Proceedings of the International Conference on Learning Representations*, 2024.

[42] Maximilien Danisch, Oana Balalau, and Mauro Sozio. Listing k-cliques in sparse real-world graphs. In *Proceedings of the World Wide Web Conference on World Wide Web*, pages 589–598, 2018.

[43] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *CoRR*, 2020.

[44] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, pages 1024–1034, 2017.

[45] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, 2018.

[46] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, pages 4438–4445, 2018.

[47] Gaspard Michel, Giannis Nikolentzos, Johannes F. Lutzeyer, and Michalis Vazirgiannis. Path neural networks: Expressive and accurate graph neural networks. In *Proceedings of the International Conference on Machine Learning*, pages 24737–24755, 2023.

[48] Yunchong Song, Siyuan Huang, Xinbing Wang, Chenghu Zhou, and Zhouhan Lin. Graph parsing networks. In *Proceedings of the International Conference on Learning Representations*, 2024.

[49] Feifei Qian, Lu Bai, Lixin Cui, Ming Li, Ziyu Lyu, Hangyuan Du, and Edwin R. Hancock. DHAKR: learning deep hierarchical attention-based kernelized representations for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 19995–20003, 2025.

[50] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *Proceedings of the International Conference on Learning Representations*, 2020.

[51] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, 2019.

[52] Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, Christos Giatsidis, Konstantinos Skianis, and Michalis Vazirgiannis. Grakel: A graph kernel library in python. *Journal of Machine Learning Research*, 21(54):1–5, 2020.

A Proof of the Statements in Section 3.2

A.1 Derivation of Eq. (4)

Given two shortest path $\pi_l = [v_1, v_2, \ldots, v_{l+1}]$ and $\pi'_l = [v'_1, v'_2, \ldots, v'_{l+1}]$. As mentioned in preliminary work, we have $a(\pi) = \text{concat}(a(v_1), a(v_2), \ldots, a(v_{l+1}))$. From Eq. (2), we know

$$k_{path}(\pi, \pi') = \sum_{i=1}^{|\pi|+1} \langle a(v_i), a(v'_i) \rangle. \tag{14}$$

Now we have

$$k_{path}(\pi, \pi') = \langle a(\pi), a(\pi') \rangle. \tag{15}$$

By substituting into Eq. (1), we can get

$$\begin{aligned} k_{\text{hsp}}(G, G') &= \sum_{l=1}^{L} \left[\sum_{\pi \in \mathcal{SP}_l(G)} \sum_{\pi' \in \mathcal{SP}(G')} \langle a(\pi), a(\pi') \rangle \right] \\ &= \sum_{l=1}^{L} \left\langle \sum_{\pi \in \mathcal{SP}_l(G)} a(\pi), \sum_{\pi' \in \mathcal{SP}(G')} a(\pi') \right\rangle \\ &= \sum_{l=1}^{L} \langle \phi_l(G), \phi_l(G') \rangle . \end{aligned} \tag{16}$$

Eq. (4) has been derived.

A.2 Proof of Lemma 3.1

Definition A.1. (Positive definite kernel) Let $k : \mathcal{X}$ be a set. $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a positive definite kernel if $k(x, y) = k(y, x)$ and for any $n \in \mathbb{N}$, $x_1, \ldots, x_n \in \mathcal{X}$ and $c_1, \ldots, c_n \in \mathbb{R}$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) \geq 0. \tag{17}$$

In Eq. (9), we mentioned that

$$k(G_i, G_j) = \mathbf{\Phi}(G_i) \odot \mathbf{\Phi}(G_j).$$

Now we zero-extend $\mathbf{\Phi}(G_j)$ and $\mathbf{\Phi}(G_j)$ to a sufficiently large dimension D that exceeds the dimension of any possible graph, that is, $\mathbf{\Phi}_{pad}(G) = [\mathbf{\Phi}(G), 0] \in \mathbb{R}^D$, this operation does not change the process of feature mapping. Now we have

$$k(G_i, G_j) = \langle \mathbf{\Phi}_{pad}(G_i), \mathbf{\Phi}_{pad}(G_j) \rangle . \tag{18}$$

First, $k(G_i, G_j)$ is obviously symmetric. Given any $n \in \mathbb{N}$, non-empty set of graphs $\{G_1, G_2, \ldots, G_n\}$ and $c_1, \ldots, c_n \in \mathbb{R}$. We first define the quadratic form:

$$Q = \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(G_i, G_j).$$

With refernce to Eq. (18), we have:

$$\begin{aligned} Q &= \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j \langle \mathbf{\Phi}_{\text{pad}}(G_i), \mathbf{\Phi}_{\text{pad}}(G_j) \rangle \\ &= \left\langle \sum_{i=1}^{n} c_i \mathbf{\Phi}_{\text{pad}}(G_i), \sum_{j=1}^{n} c_j \mathbf{\Phi}_{\text{pad}}(G_j) \right\rangle \\ &= \left\langle \sum_{i=1}^{n} c_i \mathbf{\Phi}_{\text{pad}}(G_i), \sum_{i=1}^{n} c_i \mathbf{\Phi}_{\text{pad}}(G_i) \right\rangle . \end{aligned} \tag{19}$$

Table 5: Dataset statistics and properties of TUDatasets.

Dataset	**Properties**					
	Number of graphs	Number of classes	Avg number of vertices	Avg number of edges	Vertex labels	Continuous vertex attributes
MUTAG	188	2	17.93	19.79	✓	×
NCI1	4110	2	29.87	32.30	✓	×
PTC_MR	344	2	14.29	14.69	✓	×
DD	1178	2	284.32	715.66	✓	×
BZR	405	2	35.75	38.36	✓	✓
PROTEINS	1113	2	39.06	72.82	✓	✓
ENZYMES	600	6	32.6	62.14	✓	✓
COLLAB	5000	3	74.49	2457.78	×	×
IMDB-BINARY	1000	2	19.77	96.53	×	×
IMDB-MULTI	1500	3	13.00	65.94	×	×
REDDIT-BINARY	2000	2	429.63	497.75	×	×
REDDIT-MULTI-5K	4999	5	508.52	594.87	×	×

Let $v = \sum_{i=1}^{n} c_i \mathbf{\Phi}_{pad}(G_i)$, now we have $Q = \langle v, v \rangle = \|v\|^2$. Clearly, $Q \geq 0$, therefore $k(G_i, G_j)$ is positive definite.

A.3 Proof of Lemma 3.2

Assume there is a shortest-path of length l from vertex v_1 to v_{l+1}, denoted as $\pi_l = [v_1, \ldots, v_l, v_{l+1}]$. Then, there exists a path of length $l-1$ from vertex v_1 to v_l, $\pi_{l-1} = [v_1, v_2, \ldots, v_l]$. If π_{l-1} is not the shortest-path in graph G, meaning there is a shorter shortest-path from v_1 to v_l, then there must exist a shortest-path from v_1 to v_{l+1} with length less than l, which contradicts the assumption. Therefore, π_{l-1} is the shortest-path of length $l-1$ from vertex v_1 to v_l. The proof for other lengths is similar.

B Dataset statistics.

We evaluated the proposed methods on 12 publicly available graph classification datasets including 4 small molecule datasets: BZR, MUTAG, NCI1 and PTC_MR, 3 bioinformatics datasets: DD, ENZYMES and PROTEINS, as well as 6 social network datasets: COLLAB, IMDB-B, IMDB-M, REDDIT-B, REDDIT-M-5K).

A summary of the 12 datasets is given in Table 5.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our contributions include: (1) We design a computationally efficient kernel based on the shortest paths. It can be flexibly integrated with neural networks while preserving both the semantic and structural information of a given graph; (2) We develop a novel framework termed HSP-GKN which integrates the kernel we proposed with neural networks, enabling objective-driven learning of representations beneficial for specific tasks. These contributions are covered in the abstract and introduction.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of our work in terms of applicability on graphs with edge attributes in Section 5.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: All proofs are provided in the Appendix.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide all the details of our method in Section 3 and Section 4. Furthermore, we provide the code of our proposed method in the supplementary material.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We include the code of our proposed method in the supplementary material.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The complete hyperparameter configurations can be found in our code.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: To ensure statistical reliability, we provide variance measurements for all reported results. All experimental results were obtained from 10 independent runs with variance calculated.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide the compute workers in Section 4. Time of execution is also reported in Section 4.2.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We conduct the research with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We provide the broader impacts of our work in Section 5

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our work does not pose such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All datasets, models, and code involved in our paper are open source.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: Our paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Our paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Our paper does not involve participants.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: This research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.