# Convexified Message-Passing Graph Neural Networks

**Saar Cohen**
Department of Computer Science
Bar-Ilan University, Israel

**Noa Agmon**
Department of Computer Science
Bar-Ilan University, Israel

**Uri Shaham**
Department of Computer Science
Bar-Ilan University, Israel

## Abstract

Graph Neural Networks (GNNs) are key tools for graph representation learning, demonstrating strong results across diverse prediction tasks. In this paper, we present ***Convexified Message-Passing Graph Neural Networks*** (CGNNs), a novel and general framework that combines the power of message-passing GNNs with the tractability of *convex* optimization. By mapping their nonlinear filters into a reproducing kernel Hilbert space, CGNNs transform training into a convex optimization problem, which projected gradient methods can solve both efficiently and optimally. Convexity further allows CGNNs' statistical properties to be analyzed accurately and rigorously. For two-layer CGNNs, we establish rigorous generalization guarantees, showing convergence to the performance of an optimal GNN. To scale to deeper architectures, we adopt a principled layer-wise training strategy. Experiments on benchmark datasets show that CGNNs significantly exceed the performance of leading GNN models, obtaining 10–40% higher accuracy in most cases, underscoring their promise as a powerful and principled method with strong theoretical foundations. In rare cases where improvements are not quantitatively substantial, the convex models either slightly exceed or match the baselines, stressing their robustness and wide applicability. Though over-parameterization is often used to enhance performance in non-convex models, we show that our CGNNs yield shallow convex models that can surpass non-convex ones in accuracy and model compactness.

Proceedings of the 29[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300. 

## 1 INTRODUCTION

Graphs serve as versatile models across various domains, such as social networks (Wang et al., 2018), protein design (Ingraham et al., 2019), and medical diagnosis (Li et al., 2020). Graph representation learning has thus attracted widespread attention in recent years, with Graph Neural Networks (GNNs) emerging as powerful tools. Message Passing Neural Networks (MPNNs) (Gilmer et al., 2017; Hamilton et al., 2017; Veličković et al., 2018), a most widely adopted class of GNNs, provide a scalable and elegant architecture that effectively exploits a message-passing mechanism to extract useful information and learn high-quality representations from graph data.

In this paper, we present the new and general framework of ***Convexified Message-Passing Graph Neural Networks*** (CGNNs), reducing the training of message-passing GNNs to a *convex* optimization problem solvable efficiently and optimally via projected gradient methods. The convexity of CGNNs offers a distinct advantage: it ensures *global* optimality and computational tractability without relying on, e.g., over-parameterization, unlike many existing approaches (Allen-Zhu et al., 2019; Du and Lee, 2018; Du et al., 2018; Zou et al., 2020). This convexity also enables a precise and theoretically grounded analysis of generalization and statistical behavior.

We exhibit our framework by convexifying two-layer Graph Convolution Networks (GCNs) (Defferrard et al., 2016; Kipf and Welling, 2017), a widely-used GNN model for learning over graph-structured data. Inspired by the convexification of convolutional neural networks by Zhang et al. (2017), we adapt their methodology to the graph domain. A high-level overview of our convexification pipeline is summarized in Figure 1, explained as follows. The key challenge lies in handling GCNs' nonlinearity: we address this by first relaxing the space of GCN filters to a reproducing kernel Hilbert space (RKHS) (step (1) in Figure 1), effectively reducing the model to one with a linear activation function (step (2)). We then show that GCNs

with RKHS filters admit a low-rank matrix representation (step (3)). Finally, by relaxing the non-convex rank constraint into a convex nuclear norm constraint (step (4)), we arrive at our CGCN formulation.

Theoretically, we establish an upper bound on the expected generalization error of our class of two-layer CGCNs, composed of the best possible performance attainable by a two-layer GCN with infinite data and a complexity term that decays to zero polynomially with the number of training samples. Our result dictates that the generalization error of a CGCN provably converges to that of an optimal GCN, offering both strong statistical guarantees and practical relevance. Moreover, our analysis naturally extends to deeper models by applying the argument layerwise, providing insights into the statistical behavior of multi-layer CGCNs as well. Finally, we perform an extensive empirical analysis over various benchmark datasets, where our framework exhibits outstanding results. First, we apply our framework to a wide range of leading message-passing GNN models, including GAT (Veličković et al., 2018), GATv2 (Brody et al., 2022), GIN (Xu et al., 2019), GraphSAGE (Hamilton et al., 2017), and ResGatedGCN (Bresson and Laurent, 2017). We also extended our framework to hybrid transformer models, specifically GraphGPS (Rampášek et al., 2022) and GraphTrans (Shi et al., 2021; Wu et al., 2021), which integrate local message passing GNNs with global attention mechanisms. Remarkably, in most cases, the convex variants outperform the original non-convex counterparts by a large magnitude, obtaining 10–40% higher accuracy, highlighting the practical power of our approach alongside its theoretical appeal. Even in the minority of cases where improvements are modest, the convexified models still match or slightly exceed the baseline performance, further validating the robustness and broad applicability of our method. While over-parameterization is commonly used to boost performance in non-convex models (Allen-Zhu et al., 2019; Du and Lee, 2018; Du et al., 2018; Zou et al., 2020), we show that our CGNNs framework enables the construction of shallow convex architectures that outperform over-parameterized non-convex ones in both accuracy and model compactness.

## 2 RELATED WORK

**Message-Passing GNNs.** Early developments of message-passing GNNs include GCN (Defferrard et al., 2016; Kipf and Welling, 2017), GAT (Veličković et al., 2018), GIN (Xu et al., 2019), and GraphSAGE (Hamilton et al., 2017). Such methods effectively exploit a message-passing mechanism to recursively aggregate useful neighbor information (Gilmer et al., 2017), enabling them to learn high-quality representations from graph data. Those models have proven effective across diverse tasks, obtaining state-of-the-art performance in various graph prediction tasks (see, e.g., (Chen et al., 2022; Hamilton et al., 2017; Rampášek et al., 2022; Veličković et al., 2018; Xu et al., 2019; You et al., 2021)). Our CGNNs harness the power of message-passing GNNs by transforming training into a *convex* optimization problem. This shift not only enables efficient computation of globally optimal solutions, but also allows an accurate and rigorous analysis of statistical properties. In Appendix A, we provide more details about hybrid transformer models.

**Convergence of Gradient Methods to Global Optima.** A vast line of research analyzed the convergence of gradient-based methods to global optima when training neural networks, but under restrictive conditions. Several works rely on over-parameterization, often requiring network widths far exceeding the number of training samples (Allen-Zhu et al., 2019; Du et al., 2018; Zou et al., 2020), with milder assumptions in shallow networks (Du and Lee, 2018; Soltanolkotabi et al., 2018). However, the convexity of our CGNNs enables global optimality and computational efficiency without relying on any of the above restrictive assumptions.

**Convex Neural Networks.** The area of convex neural networks is relatively underexplored. Zhang et al. (2016a) proposed a convex relaxation step for fully-connected neural networks, which inspired the convexification of convolutional neural networks (CNNs) (Zhang et al., 2017); see Appendix A.2 for an extensive comparison to (Zhang et al., 2017). Other works study linear CNNs (Gunasekar et al., 2018) and linear GNNs (Shin et al., 2023), which are of less interest in practice since they collapse all layers into one. Recently, Cohen and Agmon (2021a) proposed convex versions of specific GNN models termed as aggregation GNN, which apply only to a signal with temporal structure (Gama et al., 2018). Their work is narrowly tailored and focused on a specific application in swarm robotics (Cohen and Agmon, 2021c,b, 2020), where convexity arises from modeling assumptions tailored to domain adaptation. In contrast, our CGNN framework establishes a general and theoretically principled convexification of diverse message-passing GNNs by transforming the standard architecture itself, independent of application-specific constraints. Unlike their formulation that does not preserve, e.g., standard GNN modularity, our approach retains these key structural features and applies broadly to a wide range of graph learning tasks. In particular, our contribution resolves the key limitations of Cohen and Agmon (2021a) through three fundamental distinctions: (1)

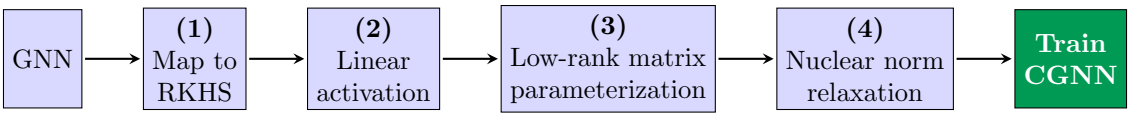


Figure 1: Convexification Procedure of a message-passing GNN into a CGNN.

**Scope**, by enabling training on variable-sized graphs (inductive setting) versus their fixed-topology restriction; (2) **Theory**, by extending analysis to general non-polynomial filters and proving strictly tighter nuclear norm bounds; and (3) **Methodology**, by utilizing an RKHS formulation that supports modern optimizers and generic modular tasks. In Appendix A.3, we supply an extensive comparison to Cohen and Agmon (2021a), detailing the fundamental distinctions from their work. As such, CGNNs represent a scalable and domain-agnostic foundation for convex GNN training, grounded in a fundamentally different mathematical framework.

**Classical Graph Kernels.** Our framework offers a unique theoretical bridge between modern deep graph learning and classical graph kernels. While GNNs have largely superseded kernel methods in popularity, establishing the formal connections between them contextualizes the theoretical advances of the CGNN framework, particularly regarding the role of Reproducing Kernel Hilbert Spaces (RKHS) and optimization landscapes. We supply a detailed comparison to classic graph kernels in Appendix A.5.

**GNNs as Unified Optimization Problems.** Our framework relates to and fundamentally differs from prior work on GNNs as unified optimization problems. Specifically, Chen et al. (2023b) classify spatial GNNs based on how they aggregate neighbor information, while discussing how graph convolution can be depicted as an optimization problem. They also discuss how filters of spectral GNNs can be approximated via either linear, polynomial or rational approximation. The works by He et al. (2021), Wang and Zhang (2022) and Guo and Wei (2023) focus on polynomial approximation, treating spectral filters as parameterized by a polynomial basis (see Appendix A.4 for details). However, in all those works training remains ***non-convex***. Our contribution is thus orthogonal and complementary: rather than deriving architectures, we ***convexify the training problem*** itself for general, non-convex message-passing GNNs (not only spectral ones). Training thus becomes a convex optimization problem, ensuring global optimality as well as an accurate and rigorous analysis of statistical properties. Further, our framework applies directly to general message-passing GNNs and hybrid transformer architectures like GraphGPS and GraphTrans, whereas The works by He et al. (2021), Wang and Zhang (2022) and Guo and Wei (2023) only cover spectral GNNs. In fact, the optimization-oriented analysis of Wang and Zhang (2022) is restricted to *linear* GNNs, while we focus on convexifying ***non-convex*** GNNs. We will add a dedicated discussion explicitly contrasting these settings.

**Notations.** For brevity, $[n]$ denotes the discrete set $\{1, 2, \ldots, n\}$ for any $n \in \mathbb{N}_{>0}$. For a rectangular matrix $A$, let $\|A\|_*$ be its nuclear norm (i.e., the sum of $\mathcal{A}_\ell$'s singular values), and $\|A\|_2$ be its spectral norm (i.e., maximal singular value). Let $\ell^2(\mathbb{N})$ be the set of countable dimensional of square-summable sequences $v$, i.e., $\sum_{i=1}^{\infty} v_i^2 < \infty$.

## 3 PRELIMINARIES

Network data can be captured by an underlying undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, where $\mathcal{V} = \{1, ..., n\}$ denotes the vertex set, $\mathcal{E}$ denotes the edge set and $\mathcal{W} : \mathcal{E} \to \mathbb{R}$ denotes the edge weight function, while a missing edge $(i, j)$ is depicted by $\mathcal{W}(i, j) = 0$. The *neighborhood* of node $i \in \mathcal{V}$ is denoted by $N_i := \{j \in \mathcal{V} | (j, i) \in \mathcal{E}\}$. The data on top of this graph is modeled as a *graph signal* $\mathcal{X} \in \mathbb{R}^{n \times F}$, whose $i^{\text{th}}$ row $[\mathcal{X}]_i = \mathbf{x}_i$ is an $F$-dimensional *feature vector* assigned to node $i \in \mathcal{V}$. We leverage the framework of *graph signal processing* (GSP) (Ortega et al., 2018) as the mathematical foundation for learning from graph signals. For relating the graph signal $\mathcal{X}$ with the underlying structure of $\mathcal{G}$, we define a *graph shift operator* (GSO) by $\mathcal{S} \in \mathbb{R}^{n \times n}$, which satisfies $[\mathcal{S}]_{ij} = s_{ij} = 0$ if $(j, i) \notin \mathcal{E}$ for $j \neq i$. Commonly utilized GSOs include the **adjacency** matrix (Sandryhaila and Moura, 2013, 2014), the **Laplacian** matrix (Shuman et al., 2013), and their normalized counterparts (Defferrard et al., 2016; Gama et al., 2019a). By viewing $\mathcal{S}$ as a linear operator, the graph signal $\mathcal{X}$ can be *shifted* over the nodes, yielding that the output at node $i$ for the $f^{\text{th}}$ feature becomes $[\mathcal{S}\mathcal{X}]_{if} = \sum_{j=1}^{n} [\mathcal{S}]_{ij} [\mathcal{X}]_{jf} = \sum_{j \in N_i} s_{ij} x_{jf}$. Note that, whereas $\mathcal{S}\mathcal{X}$ corresponds to a local information exchange between a given node and its direct neighbors, applying this linear operator recursively aggregates information from nodes in further hops along the graph. Namely, the *$k$-shifted signal* $\mathcal{S}^k\mathcal{X}$ aggregates information from nodes that are $k$-hops away.

**Graph Convolutional Networks** (GCNs) stack

multiple graph (convolutional) layers to extract high-level node representations from graph signals (Gama et al., 2019b). To formally describe the creation of convolutional features in a GCN with $L \in \mathbb{N}$ layers, let the output of layer $\ell-1$ (and thus the input to the $\ell^{\text{th}}$ layer) be $\mathcal{X}_{\ell-1}$, comprising of $F_{\ell-1}$ feature vectors. The input to layer 0 is $\mathcal{X}_0 = \mathcal{X}$, yielding $F_0 = F$, while the final output has $G := F_L$ features. Given a set of *filters* $A := \{\mathcal{A}_{\ell k} \in \mathbb{R}^{F_{\ell-1} \times F_\ell} | k \in [K] \cup \{0\}, \ell \in [L]\}$, the *graph (convolutional) filter* at layer $\ell$ processes the $F_{\ell-1}$ features of the input graph signal $\mathcal{X}_{\ell-1}$ in parallel by $F_\ell$ graph filters to output the following $F_\ell$ convolutional features:

$$\Psi^{\mathcal{A}_\ell}(\mathcal{X}; \mathcal{S}) := \sum_{k=0}^{K} \mathcal{S}^k \mathcal{X}_{\ell-1} \mathcal{A}_{\ell k}, \tag{1}$$

where $\mathcal{A}_\ell := (\mathcal{A}_{\ell k})_{k=0}^K$ concatenates the filters at layer $\ell$. Note that all nodes share the same parameters to weigh equally the information from all $k$-hop away neighbors for any $k \in [K]$, which corresponds to the *parameter sharing* of GCNs. A GCN is then defined as a nonlinear mapping between graph signals $\Phi : \mathbb{R}^{n \times F} \to \mathbb{R}^{n \times G}$, which is obtained by feeding the aggregated features through a pointwise nonlinear activation function $\sigma : \mathbb{R} \to \mathbb{R}$ (which, for brevity, denotes its entrywise application in (2)), producing the feature vectors of the $\ell^{\text{th}}$ layer's output:

$$\Phi(\mathcal{X}; \mathcal{S}, A) := \mathcal{X}_L; \quad \mathcal{X}_\ell := \sigma(\Psi^{\mathcal{A}_\ell}(\mathcal{X}; \mathcal{S})). \tag{2}$$

We denote the $f^{\text{th}}$ column of the filters $\mathcal{A}_{\ell k}$ as $\mathbf{a}_{\ell k}^f = [a_{\ell k}^{fg}]_{g \in [F_{\ell-1}]}$. Given some positive constants $\{R_\ell\}_{\ell \in [L]}$, we consider the following *non-convex* function class of (regularized) GCNs:

$$\mathcal{F}_{\text{gcn}} := \{\Phi \text{ as in (2)} : \max_{0 \le k \le K} \|\mathbf{a}_{\ell k}^f\|_2 \le R_\ell \ \forall f, \ell\}. \tag{3}$$

By setting $R_\ell = \infty$ for any $\ell \in [L]$, this function class reduces to the class of (unregularized) GCNs.

### 3.1 Training GCNs

We focus on the *graph classification* task with $G$ classes. Here, we are given a training set $\mathcal{T} := \{(\mathcal{G}^{(j)}, \mathcal{X}^{(j)}, y^{(j)})\}_{j=1}^m$ of $m$ samples, where the $j^{\text{th}}$ sample consists of a graph $\mathcal{G}^{(j)}$, a graph signal $\mathcal{X}^{(j)} \in \mathbb{R}^{n \times F}$ over $\mathcal{G}^{(j)}$ and the graph's true label $y^{(j)} \in [G]$. Without loss of generality, we assume that each graph $\mathcal{G}^{(j)}$ has $n$ nodes (otherwise, we can add isolated dummy nodes). We then train a GCN (as formulated in (2)), while node features from the last layer are aggregated to obtain the graph-level representation via a fixed permutation invariant function, such as summation or a more sophisticated graph-level pooling function (Ying et al., 2018b; Zhang et al., 2018). Training is done by solving the following optimization problem for some loss function $J : \mathbb{R}^{n \times G} \times [G] \to \mathbb{R}$:

$$\Phi_{\text{gcn}}^* \in \operatorname*{arg\,min}_{\Phi \in \mathcal{F}_{\text{gcn}}} \frac{1}{m} \sum_{j=1}^{m} J(\Phi(\mathcal{X}^{(j)}; \mathcal{S}, A), y^{(j)}). \tag{4}$$

We assume that $J$ is convex and $M$-Lipschitz in its first argument (predictions) for any fixed second argument (labels). The number of parameters to be trained are determined by the number of layers $L$, the number of filters $K$ and features $F_\ell$ in each layer $\ell$. By (2), each GCN $\Phi \in \mathcal{F}_{\text{gcn}}$ depends nonlinearly on the parameters, making the optimization problem in (4) non-convex. Next, we thus present a relaxation of the class $\mathcal{F}_{\text{gcn}}$, enabling a convex formulation of the associated problem in (4).

**Remark 3.1.** *As our focus is on understanding generalization of convexified message-passing GNNs, we restricted evaluations to graph classification. This is exactly as in prior works that only consider a single task (see, e.g., (Lachi et al., 2025; Chen et al., 2023a; Wu et al., 2022)). However,* ***our convexification framework directly extends to node-level prediction****, by using the node features from the last layer instead of aggregating them to obtain the graph-level representation. While this results in a node-level instead of graph-level loss, our convexification procedure remains identical.*

## 4 CONVEXIFICATION OF GCNS

As $\mathcal{F}_{\text{gcn}}$ in (3) is a non-convex set and the activation function is nonlinear, the ERM problem on GCNs is non-convex, raising the challenges mentioned in Section 1. To overcome them, we propose a procedure for making both the ERM's predictions domain and loss function *convex*. Following Zhang et al. (2017), we develop the class of ***Convexified GCNs*** **(CGCNs)**. In Section 4.1, we first depict our method for *linear* activation functions. Given that the linear activation function would collapse all layers into one, this case is of no interest in practice. However, it supplies us with some insight into the general kernelization scheme of GCNs. In the nonlinear case (Section 4.2), we reduce the problem to the linear case by proposing a relaxation to a suitably chosen *Reproducing Kernel Hilbert Space* (RKHS), enabling us to obtain a convex formulation of the associated ERM problem. We then present an algorithm for learning CGCNs (Section 4.3), discuss its scalability and time complexity (Section 4.4), and establish its generalization guarantees (Section 4.5).

### 4.1 *Linear* Activation Functions

To provide the intuition for our method, we employ the nuclear norm in the simple case of the linear activation

function $\sigma(\mathbf{w}) = \mathbf{w}$, which provides us with a convex function class, unlike (3). In this case, the output of layer $\ell$ is simply $\mathcal{X}_\ell = \sum_{k=0}^{K} \mathcal{S}^k \mathcal{X}_{\ell-1} \mathcal{A}_{\ell k}$ due to (1)-(2). Hence, $\mathcal{X}_\ell$ linearly depends on the filters $\{\mathcal{A}_{\ell k}\}_{k=0}^{K}$ of layer $\ell$, i.e., the convolution function $\Psi^{\mathcal{A}_\ell} : \mathbb{R}^{n \times F_{\ell-1}} \to \mathbb{R}^{n \times F_\ell}$ in (1) linearly depends on the parameters.

**Low-Rank Constraints.** The filters matrix $\mathcal{A}_{\ell k} \in \mathbb{R}^{F_{\ell-1} \times F_\ell}$ has rank at most $F_{\ell-1}$. As $\mathcal{A}_\ell := (\mathcal{A}_{\ell k})_{k=0}^{K}$ concatenates the filters at layer $\ell$, then $\text{rank}(\mathcal{A}_\ell) \leq F_{\ell-1}$. Hence, the class $\mathcal{F}_{\text{gcn}}$ of GCNs in (3) corresponds to a collection of functions depending on norm and low-rank constraints imposed on the filters. We next define:

$$\mathcal{F}_{\text{linear-gcn}} := \{\Psi^{\mathcal{A}_\ell} : \max_{0 \leq k \leq K} \|\mathbf{a}_{\ell k}^f\|_2 \leq R_\ell \quad \text{and} \quad (5a)$$

$$\text{rank}(\mathcal{A}_\ell) = F_{\ell-1} \quad \forall f, \ell\}, \quad (5b)$$

which is the specific form of our original class of GCNs in (3) for the linear case. While (5a) holds even when the parameters are *not* shared over all nodes, the low-rank constraint (5b) will no longer be satisfied in this case. Namely, (5b) is achieved via GCNs' parameter sharing: each node processes its neighborhood up to $K$ hops away using the same filters, and thus the number of independent rows of parameters remains at most $F_{\ell-1}$, regardless of the number of nodes in the graph.

**Nuclear Norm Relaxation.** However, filters that satisfy the constraints (5a) and (5b) form a *non-convex* set. A standard convex relaxation of a rank constraint involves replacing it with a nuclear norm constraint. That is, controlling the sum of singular values instead of the number of non-zero ones, similarly to how the $\ell_1$-norm constraint can be viewed as a convex relaxation of the $\ell_0$-norm constraint.

**Upper Bounding the Nuclear-Norm.** In Appendix B, we prove that:

**Lemma 4.1.** *Assume that the filters $\mathcal{A}_\ell$ at each layer $\ell$ satisfy* (5a) *and* (5b)*. Then, $\mathcal{A}_\ell$'s nuclear norm is upper bounded as $\|\mathcal{A}_\ell\|_* \leq \mathcal{B}_\ell$ for some $\mathcal{B}_\ell > 0$ that is a function of $R_\ell, F_\ell, F_{\ell-1}, K$.*

Using Lemma 4.1, we can replace (5a) and (5b) with a nuclear norm constraint, thus forming a convex function class. Namely, we define the class of ***Convexified GCNs*** (CGCNs) as:

$$\mathcal{F}_{\text{cgcn}} := \{\{\Psi^{\mathcal{A}_\ell}\}_{\ell=1}^{L} : \|\mathcal{A}_\ell\|_* \leq \mathcal{B}_\ell \quad \forall \ell\}, \quad (6)$$

while guaranteeing that $\mathcal{F}_{\text{gcn}} \subseteq \mathcal{F}_{\text{cgcn}}$. Thus, solving the ERM in (4) over $\mathcal{F}_{\text{cgcn}}$ instead of $\mathcal{F}_{\text{gcn}}$ yields a convex optimization problem over a wealthier class of functions. Next, for the more general setting of *nonlinear* activation functions, we devise an algorithm capable of solving this convex program.

### 4.2 *Nonlinear* Activation Functions

For certain nonlinear activation functions $\sigma$ and suitably chosen kernel function $\kappa$ such as the Gaussian RBF kernel, we next prove how the class $\mathcal{F}_{\text{gcn}}$ of GCNs in (3) can be relaxed to a *Reproducing Kernel Hilbert Space* (RKHS), reducing the problem to the linear activation case (Readers may refer to (Schölkopf et al., 2001) for a brief overview on RKHS). Beforehand, we rephrase the filter at layer $\ell$ from (1)-(2) as follows. Denoting the $f^{\text{th}}$ column of the filters $\mathcal{A}_{\ell k}$ as $\mathbf{a}_{\ell k}^f$, in Appendix D we prove that:

**Lemma 4.2.** *A GCN induces, at each layer $\ell$, a nonlinear filter $\tau_\ell^f$ for the $f^{th}$ feature, which operates on the structured aggregation of neighborhood information and is given as (the activation $\sigma$ is applied entrywise):*

$$\tau_\ell^f : \mathcal{Z} \in \mathbb{R}^{(K+1) \times F_{\ell-1}} \mapsto \sigma\left(\sum_{k=0}^{K} \langle [\mathcal{Z}]_{k+1}, \mathbf{a}_{\ell k}^f \rangle\right). \quad (7)$$

For any graph signal $\mathcal{S}^k\mathcal{X}$, let $\mathbf{z}_i^k(\mathcal{X}) := [\mathcal{S}^k\mathcal{X}]_i$ be the $i^{\text{th}}$ row of $\mathcal{S}^k\mathcal{X}$, which is the information aggregated at node $i$ from $k$-hops away. Defining the matrix $\mathcal{Z}_{i,\ell}(\mathcal{X}_{\ell-1}) \in \mathbb{R}^{(K+1) \times F_{\ell-1}}$ whose $(k+1)^{\text{th}}$ row is $[\mathcal{Z}_{i,\ell}(\mathcal{X}_{\ell-1})]_{k+1} = \mathbf{z}_i^k(\mathcal{X}_{\ell-1})$ for $k \in [K] \cup \{0\}$, the output at layer $\ell$ is thus given by $\tau_\ell^k(\mathcal{Z}_{i,\ell}(\mathcal{X}_{\ell-1}))$.

**Kernelization of GCNs.** In the sequel, we consider a training data $\mathcal{T} = \{(\mathcal{G}^{(j)}, \mathcal{X}^{(j)}, y^{(j)})\}_{j=1}^{m}$, some layer of index $\ell$ and a positive semi-definite kernel function $\kappa_\ell : \mathbb{R}^{F_{\ell-1}} \times \mathbb{R}^{F_{\ell-1}} \to \mathbb{R}$ for layer $\ell$. Thus, for brevity, we omit indices involving $\ell$ and $\mathcal{X}_{\ell-1}^{(j)}$ when no ambiguity arise. Namely, we denote, e.g., $\mathcal{Z}_{i,j} := \mathcal{Z}_{i,\ell}(\mathcal{X}_{\ell-1}^{(j)})$ and $\mathbf{z}_{i,j}^k := \mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)})$.

First, we prove in Lemma 4.3 that the filter specified by (7) resides in the RKHS induced by the kernel (See Appendix E for a proof that adapts the one by Zhang et al. (2017) to GCNs). Zhang et al. (2017) prove that convolutional neural networks can be contained within the RKHS induced by properly chosen kernel and activation functions. For certain choices of kernels (e.g., the Gaussian RBF kernel) and a sufficiently smooth $\sigma$ (e.g., smoothed hinge loss), we can similarly prove that the filter $\tau_\ell^f$ in (7) is contained within the RKHS induced by the kernel function $\kappa_\ell$. See Appendices C.1-C.2 for possible choices of kernels and Appendix C.3 for valid activation functions, as well as an elaboration on sufficient smoothness in Remark C.2.

**Lemma 4.3.** *Mercer's theorem (Steinwart, 2008, Theorem 4.49) implies that the filter $\tau_\ell^f$ at layer $\ell$ for the $f^{th}$ feature can be reparametrized as a linear combination of kernel products, reducing the problem to the*

*linear activations case. Formally:*

$$\tau_\ell^f(\mathcal{Z}_{i,j}) = \sum_{k=0}^{K} \sum_{(i',j')\in[n]\times[m]} \alpha^f_{k,(i',j')} \cdot \kappa_\ell(\mathbf{z}^k_{i,j}, \mathbf{z}^k_{i',j'}), \tag{8}$$

*where* $\{\alpha^f_{k,(i',j')}\}_{(i',j',k)\in[n]\times[m]\times([K]\cup\{0\})}$ *are some coefficients.*

**Remark 4.4.** *Lemma 4.3 suggests that filters of the form* (8) *are parameterized by a finite set of coefficients* $\{\alpha^f_{k,(i',j')}\}$*, allowing optimization to be carried over these coefficients instead of the original parameters* $\mathcal{A}_\ell$ *(See Section 4.3 for details on solving this optimization problem).*

To avoid rederiving all results in the kernelized setting, we reduce the problem to the linear case in Section 4.1. Thus, consider the (symmetric) kernel matrix $\mathcal{K}^k_\ell \in \mathbb{R}^{nm\times nm}$, whose entry at row $(i,j)$ and column $(i',j')$ equals to $\kappa_\ell(\mathbf{z}^k_{i,j}, \mathbf{z}^k_{i',j'})$. We factorize the kernel matrix as $\mathcal{K}^k_\ell = Q^k_\ell (Q^k_\ell)^\top$ with $Q^k_\ell \in \mathbb{R}^{nm\times P}$ (as in (Zhang et al., 2017), we can use, e.g., Cholesky factorization (Dereniowski and Kubale, 2003) with $P = nm$).

**Remark 4.5.** *The* $(i,j)$*-th row of* $Q^k_\ell$*, i.e.,* $\mathbf{q}^k_{i,j} := [Q^k_\ell]_{(i,j)} \in \mathbb{R}^P$*, can be interpreted as a feature vector instead of the original* $\mathbf{z}^k_{i,j} \in \mathbb{R}^{F_{\ell-1}}$ *(the information aggregated at node* $i$ *from* $k$*-hops away).*

Thus, using Remark 4.5, the reparameterization in (8) of the filter at layer $\ell$ for the $f^{\text{th}}$ feature can be rephrased as:

$$\begin{aligned} \tau_\ell^f(\mathcal{Z}_{i,j}) &= \sum_{k=0}^{K} \langle \mathbf{q}^k_{i,j}, \hat{\mathbf{a}}^f_{\ell k}\rangle; \\ \hat{\mathbf{a}}^f_{\ell k} &:= \sum_{(i',j',k)\in[n]\times[m]\times([K]\cup\{0\})} \alpha^f_{k,(i',j')} \mathbf{q}^k_{i',j'} \end{aligned} \tag{9}$$

Next, we outline the learning and testing phases of CGCNs (See Appendix E for more details).

**Remark 4.6.** *Relaxing the filters to the induced RKHS reduces the problem to the linear case. The resulting relaxed filters satisfy a low-rank constraint and have a bounded nuclear norm. By Section 4.1, convexly relaxing the low-rank constraint via the nuclear norm yields a convex function class.*

To summarize, we describe the forward pass operation of a single layer in Algorithm 1. It processes an input graph signal $\mathcal{X}$ and transforms it in two steps:

**Step 1: Arranging Input to the Layer.** Each layer $\ell$ takes as input the graph signal $\mathcal{X}_{\ell-1}$ from the previous layer and computes intermediate graph signals $Z^k_\ell(\mathcal{X}_{\ell-1})$ for different hop distances $k$ (line 2). For each node $i$, information is aggregated from its $k$-hop neighbors (line 4), from which the $i^{\text{th}}$ row of $Z^k_\ell(\mathcal{X}_{\ell-1})$ is then obtained based on kernel products (lines 4-6).

**Step 2: Computing the Layer Output.** Using the intermediate representations and the trainable filters $\hat{\mathcal{A}}_{\ell k}$, the layer's output signal $\mathcal{X}_\ell$ is calculated (line 8). Specifically, each node's output feature vector at layer $\ell$ is computed via a weighted sum of the transformed signals (line 9).

**Algorithm 1** Forward Pass of the $\ell^{\text{th}}$ CGCN Layer

**Input:** Graph signal $\mathcal{X} \in \mathbb{R}^{n\times F}$, filters $\hat{\mathcal{A}}_\ell = (\hat{\mathcal{A}}_{\ell k})_{k=0}^K$

1: **Step 1: Arrange input to layer $\ell$**
2: Given an input graph signal $\mathcal{X}_{\ell-1} \in \mathbb{R}^{n\times F}$ from the previous layer, compute a new graph signal $Z^k_\ell(\mathcal{X}_{\ell-1}) \in \mathbb{R}^{n\times P}$ for any $\ell \in [L]$ and $k \in [K]\cup\{0\}$ as follows.
3: Compute $\mathbf{z}^k_i(\mathcal{X}^{(j)}_{\ell-1}) := [\mathcal{S}^k \mathcal{X}^{(j)}_{\ell-1}]_i$, the signal aggregated at node $i$ from $k$-hops away with respect to the training graph signal $\mathcal{X}^{(j)}$.
4: As usual in kernel methods, we need to compute $\kappa_\ell(\mathbf{z}, \mathbf{z}^k_i(\mathcal{X}_{\ell-1}))$, where $\mathbf{z} \in \mathbb{R}^{F_{\ell-1}}$ is some input graph signal.
5: Form the vector $\mathbf{v}^k_\ell(\mathbf{z}^k_i(\mathcal{X}_{\ell-1}))$ whose elements are kernel products (See Appendix E).
6: Compute the $i^{\text{th}}$ row of $Z^k_\ell(\mathcal{X}_{\ell-1})$ using $Z^k_\ell(\mathcal{X}_{\ell-1})_i = (Q^k)^\dagger \mathbf{v}^k_\ell(\mathbf{z}^k_i(\mathcal{X}_{\ell-1}))$, where $(Q^k)^\dagger$ is the pseudo-inverse of $Q^k$ (See Appendix E for more details).
7: **Step 2: Compute the layer's output**
8: Using the computed signals $Z^k_\ell(\mathcal{X}_{\ell-1})$, construct the output graph signal $\mathcal{X}_\ell$ for layer $\ell$.
9: Layer $\ell$ induces a *linear* mapping between graph signals $\hat{\Psi}^{\hat{\mathcal{A}}_\ell} : \mathbb{R}^{n\times F_{\ell-1}} \to \mathbb{R}^{n\times F_\ell}$ s.t. the value of the $f^{\text{th}}$ feature of node $i$ at layer $\ell$ is $[\mathcal{X}_\ell]_{if} = [\hat{\Psi}^{\hat{\mathcal{A}}_\ell}(\mathcal{X}_{\ell-1})]_{if} = \sum_{k=0}^K \langle \hat{\mathbf{z}}^k_i(\mathcal{X}_{\ell-1}), \hat{\mathbf{a}}^f_{\ell k}\rangle$, where $\hat{\mathbf{a}}^f_{\ell k}$ are the trainable parameters (filters) of the CGCN layer from (9).

**Output:** Output graph signal $\mathcal{X}_\ell \in \mathbb{R}^{n\times F_\ell}$

### 4.3 The Algorithm for Learning Multi-Layer CGCNs

Algorithm 2 summarizes the training process of a multi-layer CGCN, estimating the parameters of each layer from bottom to top (i.e., in a layer-wise fashion). As such, to learn the nonlinear filter $\tau^f_\ell$ at layer $\ell$, we should learn the $P$-dimensional vector $\hat{\mathbf{a}}^f_{\ell k}$. For this sake, we define the graph signal $Z^k_j \in \mathbb{R}^{n\times P}$ for any $k \in [K]\cup\{0\}$ and $j \in [m]$ whose $i^{\text{th}}$ row is $[Q^k]_{(i,j)}$. We can then apply the nuclear norm relaxation from Section 4.1. Solving the resulting optimization problem yields a matrix of filters $\hat{\mathcal{A}}_{\ell k} \in \mathbb{R}^{P\times F_\ell}$ for layer $\ell$ whose $f^{\text{th}}$ column is $\hat{\mathbf{a}}^f_{\ell k}$.

**Algorithm 2** Training of CGCNs

**Input**: Training set $\mathcal{T} = \{(\mathcal{G}^{(j)}, \mathcal{X}^{(j)}, y^{(j)})\}_{j=1}^{m}$; Kernels $\{\kappa_\ell\}_{\ell=1}^{L}$; Regularization parameters $\{\mathcal{B}_\ell\}_{\ell=1}^{L}$; Number of graph filters $\{F_\ell\}_{\ell=1}^{L}$

1: **for each** layer $2 \leq \ell \leq L$ **do**
2: Taking $\{(\mathcal{G}^{(j)}, \mathcal{X}_{\ell-1}^{(j)}, y^{(j)})\}_{j=1}^{m}$ as training samples, construct a kernel matrix $\mathcal{K}_\ell^k \in \mathbb{R}^{nm \times nm}$ as in Section 4.2.
3: Compute a factorization or an approximation of $\mathcal{K}_\ell^k$ to obtain $Q_\ell^k \in \mathbb{R}^{nm \times P}$.
4: For any sample $\mathcal{X}_{\ell-1}^{(j)}$, form a graph signal $Z_\ell^k(\mathcal{X}_{\ell-1}^{(j)}) \in \mathbb{R}^{n \times P}$ via line 6 of Algorithm 1.
5: Solve (10) via (11) to obtain a the filters $\hat{\mathcal{A}}_\ell$.
6: Compute the output $\mathcal{X}_\ell^{(j)}$ of layer $\ell$ via Algorithm 1.

**Output**: Predictors $\{\hat{\Psi}^{\hat{\mathcal{A}}_\ell}(\cdot)\}_{\ell=1}^{L}$ and the output $\mathcal{X}_L^{(j)}$.

Given regularization parameters $\{\mathcal{B}_\ell\}_{\ell=1}^{L}$ for the filters' nuclear norms, we thus aim to solve the following optimization problem at each layer $\ell$ due to line 9 of Algorithm 1:

$$\hat{\mathcal{A}}_\ell \in \underset{\|\hat{\mathcal{A}}_\ell\|_* \leq \mathcal{B}_\ell}{\arg\min}\ \tilde{J}(\hat{\mathcal{A}}_\ell), \quad \text{where } \tilde{J}(\hat{\mathcal{A}}_\ell) := \frac{1}{m}\sum_{j=1}^{m} J(\hat{\Psi}^{\hat{\mathcal{A}}_\ell}(\mathcal{X}_{\ell-1}^{(j)}), y^{(j)}). \tag{10}$$

This can be easily solved via *projected gradient descent*: at round $t$, with step size $\eta^t > 0$, compute:

$$\hat{\mathcal{A}}_\ell^{t+1} = \Pi_{\mathcal{B}_\ell}(\hat{\mathcal{A}}_\ell^t - \eta^t \nabla_{\hat{\mathcal{A}}_\ell} \tilde{J}(\hat{\mathcal{A}}_\ell^t)) \tag{11}$$

where $\nabla_{\hat{\mathcal{A}}_\ell}\tilde{J}$ is the gradient of $\tilde{J}$ from (10) and $\Pi_{\mathcal{B}_\ell}$ denotes the Euclidean projection onto the nuclear norm ball $\{\hat{\mathcal{A}}_\ell : \|\hat{\mathcal{A}}_\ell\|_* \leq \mathcal{B}_\ell\}$. This projection can be done by first computing the singular value decomposition of $\hat{\mathcal{A}}$ and then projecting the vector of singular values onto the $\ell_1$-ball via an efficient algorithm (see, e.g., (Duchi et al., 2008, 2011; Xiao and Zhang, 2014)). The problem in (10) can also be solved using a projected version of any other optimizer (e.g., Adam (Kingma and Ba, 2015)).

### 4.4 Scalability, Time Complexity and Challenges of Learning CGCNs

Algorithm 2's runtime highly depends on how the factorization matrix $Q_\ell^k$ is computed and its width $P$ (line 3). Naively picking $P = nm$ requires solving the exact kernelized problem, which can be costly. To gain scalability, we compute a *Nyström approximation* (Williams and Seeger, 2001) in $\mathcal{O}(P^2 nm)$ time by randomly sampling training examples, computing their kernel matrix and representing each training example by its similarity to the sampled examples and kernel matrix. This avoids computing or storing the full kernel matrix, thus requiring only $\mathcal{O}(Pmn)$ space. While this approach is efficient (Williams and Seeger, 2001; Drineas and Mahoney, 2005), it relies on the sampled subset adequately representing the entire data; poorly represented examples may degrade approximation quality. See Appendix F for more details and alternative methods.

### 4.5 Convergence and Generalization Error

We now analyze Algorithm 2's generalization error, focusing on two-layer models ($L = 1$) in the binary classification case where the output dimension is $G = 2$. Our main result can be easily extended to the multiclass case by a standard one-versus-all reduction to the binary case. In case of multiple hidden layers, our analysis then applies to each hidden layer separately. However, a global depth-wide bound is an important future direction.

Specifically, by an approach similar to Zhang et al. (2017), we prove that the generalization error of Algorithm 2 converges to the optimal generalization error of a GCN, as it is bounded above by this optimal error plus an additive term that decays polynomially to zero with the sample size. For any $\ell \in [L]$ and $k \in [K]$, we use the random kernel matrix $\mathcal{K}_\ell^k(\mathcal{X}) \in \mathbb{R}^{n \times n}$ induced by a graph signal $\mathcal{X} \in \mathbb{R}^{n \times F}$ drawn randomly from the population, i.e., the $(i, i')$-th entry of $\mathcal{K}_\ell^k(\mathcal{X})$ is $\kappa_\ell(\mathbf{z}_i^k(\mathcal{X}), \mathbf{z}_{i'}^k(\mathcal{X}))$, where $\mathbf{z}_i^k(\mathcal{X}_{\ell-1}) := [\mathcal{S}^k \mathcal{X}_{\ell-1}]_i$ aggregates information at node $i$ from $k$-hops away.

**Theorem 4.7.** *Consider two-layer models in the binary classification case (i.e., $L = 1, G = 2$). Let $J(\cdot, \mathbf{y})$ be $M$-Lipchitz continuous for any fixed $\mathbf{y} \in [G]^n$ and let $\kappa_L$ be either the inverse polynomial kernel or the Gaussian RBF kernel (Recall Appendices C.1-C.2). For any valid activation function $\sigma$ (Recall Appendix C.3), there is a constant radius $R_L > 0$ such that the expected generalization error is at most:*

$$\mathbb{E}_{\mathcal{X},\mathbf{y}}[J(\hat{\Psi}^{\hat{\mathcal{A}}_L}(\mathcal{X}), \mathbf{y})] \leq \inf_{\Phi \in \mathcal{F}_{gcn}} \mathbb{E}_{\mathcal{X},\mathbf{y}}[J(\Phi(\mathcal{X}), \mathbf{y})] + \frac{Const}{\sqrt{m}},$$

*where $Const$ is some constant that depends on $R_L$, but not on the sample size $m$.*

*Proof.* (*Sketch*) The full proof in Appendix G comprises of two steps: When $L = 1, G = 2$, we first consider a wealthier function class, denoted as $\mathcal{F}_{\text{cgcn}}$, which is proven to contain the class of GCNs $\mathcal{F}_{\text{gcn}}$ from (2). The richer class relaxes the class of GCNs in two aspects: (1) The graph filters are relaxed to belong to

the RKHS induced by the kernel, and (2) the $\ell_2$-norm constraints in (2) are substituted by a single constraint on the Hilbert norm induced by the kernel. Though Algorithm 2 never explicitly optimizes over the function class $\mathcal{F}_{\text{cgcn}}$, we prove that the predictor $\hat{\Psi}^{\hat{\mathcal{A}}_L}$ generated at the last layer is an empirical risk minimizer in $\mathcal{F}_{\text{cgcn}}$. Then, we prove that $\mathcal{F}_{\text{cgcn}}$ is not "too big" by upper bounding its *Rademacher complexity*, a key concept in empirical process theory that can be used to bound the generalization error in our ERM problem (see, e.g., (Bartlett and Mendelson, 2002) for a brief overview on Rademacher complexity). Combining this upper bound with standard Rademacher complexity theory (Bartlett and Mendelson, 2002), we infer that the generalization error of $\hat{\Psi}^{\hat{\mathcal{A}}_L}$ converges to the best generalization error of $\mathcal{F}_{\text{cgcn}}$. Since $\mathcal{F}_{\text{gcn}} \subset \mathcal{F}_{\text{cgcn}}$, the latter error is bounded by that of GCNs, completing the proof. □

# 5 EXPERIMENTAL EVALUATION

**Datasets.** We evaluate our framework through extensive experiments on several benchmark graph classification datasets (Morris et al., 2020), including the four bioinformatics datasets MUTAG, PROTEINS, PTC-MR, NCI1 and the chemical compounds dataset Mutagenicity. We also evaluated our models on the `ogbg-molhiv` molecular property prediction dataset from the Open Graph Benchmark (Hu et al., 2020), originally sourced from MoleculeNet (Wu et al., 2018) and among its largest benchmarks. Following many prior works (Ma et al., 2019; Ying et al., 2018a), we randomly split each dataset into three parts: 80% as training set, 10% as validation set and the remaining 10% as test set. The statistics of the above datasets appear in Appendix H.1.

**Baselines.** We apply our convexification procedure to the message-passing mechanisms of popular GNNs and hybrid graph transformers, creating convex counterparts. Our framework is integrated into a wide range of leading models, including GCN (Defferrard et al., 2016; Kipf and Welling, 2017), GAT (Veličković et al., 2018), GATv2 (Brody et al., 2022), GIN (Xu et al., 2019), GraphSAGE (Hamilton et al., 2017), ResGatedGCN (Bresson and Laurent, 2017), as well as the hybrid transformers GraphGPS (Rampášek et al., 2022) and GraphTrans (Shi et al., 2021; Wu et al., 2021). In Appendix I.5, we include additional experiments for DirGNN (Rossi et al., 2024). All models share the same architecture design for fair comparison; see Appendix H.2 for more details. **Our code is available at (Cohen et al., 2026).**

We would like to clarify that many recent graph learning papers in influential venues continue using the same baselines or older ones (see, e.g., (Armgaan et al., 2025; Yehudai et al., 2025; Hordan et al., 2025; Ma et al., 2023; Chen et al., 2023a)). Further, our baselines are still listed in leaderboards of modern datasets like Open Graph Benchmark (OGB) (Hu et al., 2020). Thus, our baselines match what current peer-reviewed GNN work uses for molecular datasets and graph classification, especially for message-passing comparisons. We also clarify that our goal is parallel: to compare convexified GNNs to their own non-convex counterparts, *not* to propose a new SOTA model.

**Results and Analysis.** Table 1 presents our main empirical results for the graph classification benchmarks, comparing two-layer convex models against two-layer and six-layer non-convex models. On the `ogbg-molhiv` dataset, our convex models consistently obtained around 98% accuracy, surpassing their non-convex counterparts, which reached about 96%. Impressively, in most cases, the two-layer convex variants substantially surpass their non-convex counterparts by 10–40% accuracy, showcasing the practical strength of our framework alongside its theoretical benefits. Even when improvements are modest, our convex models match or slightly surpass their non-convex versions, underscoring the robustness and broad applicability of our approach. Notable examples are GATv2 and GraphGPS, whose convex versions significantly outperform their non-convex ones in 4 out of 5 datasets, with slight improvements on Mutagenicity. Our convex models' superiority over the hybrid transformer GraphGPS, which combines message-passing with self-attention, highlights the power of convexified GNNs over non-convex ones, even when enhanced by self-attention. In Appendix I, we visualize these trends more holistically.

Over-parameterization is known to aid gradient-based methods to reach global minima that generalize well in non-convex models (Allen-Zhu et al., 2019; Du et al., 2018; Zou et al., 2020), making it a common practice for obtaining strong performance. Our results challenge this approach, showing that our shallow convex models often surpass their more over-parameterized non-convex counterparts and obtain better results with fewer layers and parameters. Namely, we surpass them in both performance and model compactness. These gains are most notable in small-data regimes, which are prone to overfitting, where we suspect that local optima obtained by non-convex models may hinder generalization. However, convex models avoid such pitfalls, offering more stable and reliable learning. Our results show that rich representations can be attained without over-parameterization by exploiting convexity in shallow models.

In Appendix I.4, we give more results for ***deeper*** con-

Table 1: Accuracy ($\% \pm \sigma$) of 2-layer convex versions of base models against their 2- and 6-layer non-convex counterparts on various graph classification datasets over 4 runs, each with different seeds. Datasets are ordered from left to right in ascending order of size. Convex variants start with 'C'. A 2-layer model is marked by (2L), while a 6-layer one by (6L). Results of convex models surpassing their non-convex variants by a large gap ($\sim$10–40%) are in **bold** and modest improvements are underlined.

| Model | MUTAG | PTC-MR | PROTEINS | NCI1 | Mutagenicity |
|---|---|---|---|---|---|
| CGCN (2L) | $\mathbf{85.0 \pm 3.5}$ | $\underline{67.6 \pm 5.8}$ | $\mathbf{75.0 \pm 3.2}$ | $\mathbf{77.1 \pm 2.8}$ | $\underline{80.6 \pm 3.3}$ |
| GCN (2L) | $50.0 \pm 20.3$ | $48.5 \pm 10.4$ | $59.5 \pm 4.1$ | $61.0 \pm 4.0$ | $69.7 \pm 2.4$ |
| GCN (6L) | $65.0 \pm 19.6$ | $63.0 \pm 11.8$ | $58.7 \pm 6.1$ | $65.0 \pm 5.9$ | $72.7 \pm 1.6$ |
| CGIN (2L) | $\mathbf{82.5 \pm 5.5}$ | $\mathbf{66.4 \pm 3.4}$ | $\mathbf{77.7 \pm 1.8}$ | $\underline{74.8 \pm 9.4}$ | $\mathbf{77.9 \pm 4.5}$ |
| GIN (2L) | $50.0 \pm 11.7$ | $49.3 \pm 10.0$ | $50.4 \pm 12.0$ | $61.0 \pm 1.9$ | $67.8 \pm 4.6$ |
| GIN (6L) | $61.2 \pm 14.3$ | $51.9 \pm 14.5$ | $61.6 \pm 1.2$ | $66.3 \pm 2.4$ | $69.1 \pm 3.2$ |
| CGAT (2L) | $\mathbf{83.7 \pm 5.4}$ | $\mathbf{70.1 \pm 9.9}$ | $\mathbf{72.1 \pm 4.7}$ | $\underline{68.5 \pm 4.7}$ | $\underline{75.3 \pm 2.6}$ |
| GAT (2L) | $62.0 \pm 22.7$ | $58.7 \pm 18.3$ | $60.0 \pm 3.3$ | $67.1 \pm 3.9$ | $69.3 \pm 3.0$ |
| GAT (6L) | $75.0 \pm 16.2$ | $61.0 \pm 17.6$ | $63.1 \pm 1.8$ | $67.3 \pm 2.6$ | $72.8 \pm 2.0$ |
| CGATv2 (2L) | $\mathbf{87.5 \pm 2.5}$ | $\mathbf{69.4 \pm 7.5}$ | $\mathbf{73.4 \pm 1.7}$ | $\underline{66.2 \pm 1.5}$ | $\underline{80.6 \pm 1.0}$ |
| GATv2 (2L) | $68.7 \pm 4.1$ | $56.3 \pm 13.6$ | $61.6 \pm 1.6$ | $62.2 \pm 6.2$ | $70.1 \pm 2.7$ |
| GATv2 (6L) | $70.0 \pm 7.9$ | $59.2 \pm 17.4$ | $63.3 \pm 2.1$ | $63.5 \pm 4.0$ | $74.9 \pm 1.9$ |
| CResGatedGCN (2L) | $\mathbf{83.7 \pm 5.4}$ | $\mathbf{73.0 \pm 5.0}$ | $\mathbf{78.5 \pm 2.2}$ | $\underline{69.8 \pm 3.2}$ | $\underline{79.1 \pm 3.5}$ |
| ResGatedGCN (2L) | $43.7 \pm 17.4$ | $50.7 \pm 7.1$ | $47.9 \pm 9.9$ | $67.2 \pm 2.7$ | $70.6 \pm 0.9$ |
| ResGatedGCN (6L) | $55.0 \pm 12.2$ | $55.4 \pm 20.5$ | $55.8 \pm 8.7$ | $67.3 \pm 2.5$ | $71.5 \pm 1.6$ |
| CGraphSAGE (2L) | $\mathbf{82.5 \pm 2.4}$ | $\mathbf{72.3 \pm 6.1}$ | $\mathbf{73.6 \pm 3.1}$ | $\underline{69.3 \pm 2.8}$ | $\mathbf{80.1 \pm 3.6}$ |
| GraphSAGE (2L) | $65.0 \pm 16.5$ | $56.2 \pm 17.0$ | $62.5 \pm 0.8$ | $61.4 \pm 4.1$ | $70.6 \pm 1.8$ |
| GraphSAGE (6L) | $55.0 \pm 12.7$ | $61.5 \pm 8.9$ | $63.1 \pm 1.8$ | $64.7 \pm 2.0$ | $70.8 \pm 2.3$ |
| CGraphTrans (2L) | $\mathbf{81.2 \pm 4.1}$ | $\mathbf{68.1 \pm 8.3}$ | $\mathbf{75.4 \pm 3.2}$ | $\underline{68.9 \pm 2.5}$ | $\underline{73.8 \pm 3.4}$ |
| GraphTrans (2L) | $62.5 \pm 16.7$ | $55.3 \pm 13.8$ | $60.7 \pm 2.6$ | $68.9 \pm 0.8$ | $68.0 \pm 4.1$ |
| GraphTrans (6L) | $57.5 \pm 19.5$ | $61.4 \pm 16.9$ | $61.6 \pm 1.2$ | $70.0 \pm 4.3$ | $72.1 \pm 3.8$ |
| CGraphGPS (2L) | $\mathbf{83.7 \pm 2.1}$ | $\mathbf{65.5 \pm 9.3}$ | $\mathbf{81.6 \pm 7.2}$ | $\mathbf{60.1 \pm 6.1}$ | $\underline{59.4 \pm 4.1}$ |
| GraphGPS (2L) | $50.9 \pm 19.6$ | $49.2 \pm 20.1$ | $42.0 \pm 1.9$ | $50.8 \pm 2.6$ | $41.4 \pm 8.7$ |
| GraphGPS (6L) | $52.5 \pm 2.1$ | $46.4 \pm 2.3$ | $48.8 \pm 1.5$ | $49.4 \pm 1.9$ | $55.1 \pm 5.5$ |

vex models, depicting how accuracy can be improved as depth increases. This directly addresses how our convexified models handle over-smoothing: Appendix I.4 shows that the accuracy of convex models can further improve with additional layers, which is consistent with our theoretical guarantees in Theorem 4.7. Hence, this illustrates that errors remain controlled as depth increases, offering strong empirical support for the practical effectiveness of multi-layer CGNNs, which is crucial because state-of-the-art GNN applications often rely on deeper architectures. Intuitively, convexified models handle over-smoothing well thanks to the nuclear norm relaxation used by our model. Specifically, the nuclear norm projection reduces the effective capacity of filters and, in practice, suppresses small singular-value directions that often correspond to high-frequency/noisy modes over the graph. This prevents representation collapse in shallow models and empirically reduces over-smoothing in deeper convex models. See Appendix I.6 for more details.

## 6 CONCLUSION

We presented CGNNs, which transform the training of message-passing GNNs to a convex optimization problem. They ensure global optimality and computational efficiency, while having provable generalization and statistical guarantees. Empirically, we showed that our CGNNs can effectively capture complex patterns without the need for over-parameterization or extensive data. This insight offers a more efficient and data-scarce alternative to traditional deep learning approaches, with substantial improvements across various benchmark datasets. As noted in Section 4.4, CGNNs' training is heavily affected by the factorization of the kernel matrix in terms of time and space complexity. Addressing these challenges is thus a key direction for future work.

**Acknowledgments**

This research was funded in part by ISF grant 1563/22 (NA) and MOST grant number 8491/25 (US).

**References**

Allen-Zhu, Z., Li, Y., and Song, Z. (2019). A convergence theory for deep learning via over-parameterization. In *International conference on machine learning*, pages 242–252. PMLR.

Armgaan, B., Jain, E., Pandey, H., Chandran, M., and Ranu, S. (2025). Gnnxemplar: Exemplars to explanations - natural language rules for global GNN interpretability. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Bartlett, P. L. and Mendelson, S. (2002). Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482.

Bresson, X. and Laurent, T. (2017). Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*.

Brody, S., Alon, U., and Yahav, E. (2022). How attentive are graph attention networks? In *International Conference on Learning Representations*.

Chen, D., O'Bray, L., and Borgwardt, K. (2022). Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning*, volume 162, pages 3469–3489.

Chen, J., Gao, K., Li, G., and He, K. (2023a). NAGphormer: A tokenized graph transformer for node classification in large graphs. In *The Eleventh International Conference on Learning Representations*.

Chen, Z., Chen, F., Zhang, L., Ji, T., Fu, K., Zhao, L., Chen, F., Wu, L., Aggarwal, C., and Lu, C.-T. (2023b). Bridging the gap between spatial and spectral domains: A unified framework for graph neural networks. *ACM Computing Surveys*, 56(5):1–42.

Cohen, S. and Agmon, N. (2020). Converging to a desired orientation in a flock of agents. *arXiv preprint arXiv:2010.04686*.

Cohen, S. and Agmon, N. (2021a). Convexified graph neural networks for distributed control in robotic swarms. In *International Joint Conferences on Artificial Intelligence Organization*, pages 2307–2313.

Cohen, S. and Agmon, N. (2021b). Recent advances in formations of multiple robots. *Current Robotics Reports*, 2(2):159–175.

Cohen, S. and Agmon, N. (2021c). Spatial consensus-prevention in robotic swarms. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 359–367.

Cohen, S., Agmon, N., and Shaham, U. (2026). Convexified message-passing graph neural networks. `https://github.com/saarcohen30/cgcn/`.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29:3844–3852.

Dereniowski, D. and Kubale, M. (2003). Cholesky factorization of matrices in parallel and ranking of graphs. In *International Conference on Parallel Processing and Applied Mathematics*, pages 985–992. Springer.

Drineas, P. and Mahoney, M. W. (2005). Approximating a gram matrix for improved kernel-based learning. In *International Conference on Computational Learning Theory*, pages 323–337. Springer.

Du, S. and Lee, J. (2018). On the power of over-parametrization in neural networks with quadratic activation. In *International conference on machine learning*, pages 1329–1338. PMLR.

Du, S. S., Hou, K., Póczos, B., Salakhutdinov, R., Wang, R., and Xu, K. (2019). Graph neural tangent kernel: Fusing graph neural networks with graph kernels. *Advances in neural information processing systems*, 32:5723–5733.

Du, S. S., Zhai, X., Poczos, B., and Singh, A. (2018). Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representations*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159.

Duchi, J., Shalev-Shwartz, S., Singer, Y., and Chandra, T. (2008). Efficient projections onto the $\ell_1$-ball for learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 272–279.

Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428*.

Gama, F., Marques, A. G., Leus, G., and Ribeiro, A. (2018). Convolutional neural network architectures for signals supported on graphs. *IEEE Transactions on Signal Processing*, 67(4):1034–1049.

Gama, F., Ribeiro, A., and Bruna, J. (2019a). Diffusion scattering transforms on graphs. In *International Conference on Learning Representations*.

Gama, F., Ribeiro, A., and Bruna, J. (2019b). Stability of graph scattering transforms. *Advances in Neural Information Processing Systems*, 32.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR.

Gunasekar, S., Lee, J. D., Srebro, N., and Soudry, D. (2018). Implicit bias of gradient descent on linear convolutional networks. *Advances in Neural Information Processing Systems*, 31:9461–9471.

Guo, Y. and Wei, Z. (2023). Graph neural networks with learnable and optimal polynomial bases. In *International conference on machine learning*, pages 12077–12097. PMLR.

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 30.

He, M., Wei, Z., Xu, H., et al. (2021). Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. *Advances in neural information processing systems*, 34:14239–14251.

Hordan, S., Bechler-Speicher, M., Lifshitz, G., and Dym, N. (2025). Spectral graph neural networks are incomplete on graphs with a simple spectrum. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. (2020). Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133.

Hussain, M. S., Zaki, M. J., and Subramanian, D. (2022). Global self-attention as a replacement for graph convolution. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 655–665.

Ingraham, J., Garg, V. K., Barzilay, R., and Jaakkola, T. (2019). Generative models for graph-based protein design. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, volume 32, pages 15820–15831.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V., and Tossou, P. (2021). Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629.

Krishnagopal, S. and Ruiz, L. (2023). Graph neural tangent kernel: Convergence on large graphs. In *International Conference on Machine Learning*, pages 17827–17841. PMLR.

Lachi, V., Ferrini, F., Longa, A., Lepri, B., Passerini, A., and Jaeger, M. (2025). Bridging theory and practice in link representation with graph neural networks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Le, Q., Sarlos, T., and Smola, A. (2013). Fastfood - computing hilbert space expansions in loglinear time. In *International Conference on Machine Learning*, volume 28, pages 244–252.

Li, Y., Qian, B., Zhang, X., and Liu, H. (2020). Graph neural network-based diagnosis prediction. *Big data*, 8(5):379–390.

Ma, L., Lin, C., Lim, D., Romero-Soriano, A., Dokania, P. K., Coates, M., Torr, P., and Lim, S.-N. (2023). Graph inductive biases in transformers without message passing. In *International Conference on Machine Learning*, pages 23321–23337. PMLR.

Ma, Y., Wang, S., Aggarwal, C. C., and Tang, J. (2019). Graph convolutional networks with eigenpooling. In *ACM SIGKDD international conference on knowledge discovery & data mining*, pages 723–731.

Minsker, S. (2017). On some extensions of bernstein's inequality for self-adjoint operators. *Statistics & Probability Letters*, 127:111–119.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. (2020). Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*. `www.graphlearning.io`.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. (2019). Weisfeiler and leman go neural: Higher-order graph neural networks. *Proceedings of the AAAI conference on artificial intelligence*, 33(01):4602–4609.

Ortega, A., Frossard, P., Kovačević, J., Moura, J. M., and Vandergheynst, P. (2018). Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F.,

Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32, pages 8024–8035.

Rahimi, A. and Recht, B. (2007). Random features for large-scale kernel machines. In *Advances in neural information processing systems*, volume 20, pages 1177–1184.

Ramakrishna, R., Wai, H.-T., and Scaglione, A. (2020). A user guide to low-pass graph signal processing and its applications: Tools and applications. *IEEE Signal Processing Magazine*, 37(6):74–85.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. (2022). Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515.

Rossi, E., Charpentier, B., Di Giovanni, F., Frasca, F., Günnemann, S., and Bronstein, M. M. (2024). Edge directionality improves learning on heterophilic graphs. In *Learning on graphs conference*, pages 25–1. PMLR.

Sandryhaila, A. and Moura, J. M. (2013). Discrete signal processing on graphs. *IEEE transactions on signal processing*, 61(7):1644–1656.

Sandryhaila, A. and Moura, J. M. (2014). Discrete signal processing on graphs: Frequency analysis. *IEEE Transactions on Signal Processing*, 62(12):3042–3054.

Schölkopf, B., Herbrich, R., and Smola, A. J. (2001). A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9).

Shi, Y., Huang, Z., Feng, S., Zhong, H., Wang, W., and Sun, Y. (2021). Masked label prediction: Unified message passing model for semi-supervised classification. In *International Joint Conferences on Artificial Intelligence Organization*, pages 1548–1554.

Shin, S., Shomorony, I., and Zhao, H. (2023). Efficient learning of linear graph neural networks via node subsampling. *Advances in Neural Information Processing Systems*, 36:55479–55501.

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98.

Soltanolkotabi, M., Javanmard, A., and Lee, J. D. (2018). Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *IEEE Transactions on Information Theory*, 65(2):742–769.

Steinwart, I. (2008). *Support Vector Machines*. Springer.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *International Conference on Learning Representations*.

Wang, X. and Zhang, M. (2022). How powerful are spectral graph neural networks. In *International conference on machine learning*, pages 23341–23362. PMLR.

Wang, Z., Chen, T., Ren, J., Yu, W., Cheng, H., and Lin, L. (2018). Deep reasoning with knowledge graph for social relationship understanding. In *International Joint Conference on Artificial Intelligence*, pages 1021–1028.

Williams, C. K. and Seeger, M. (2001). Using the nyström method to speed up kernel machines. In *Advances in neural information processing systems*, pages 682–688.

Wu, Q., Zhao, W., Li, Z., Wipf, D. P., and Yan, J. (2022). Nodeformer: A scalable graph structure learning transformer for node classification. *Advances in Neural Information Processing Systems*, 35:27387–27401.

Wu, Z., Jain, P., Wright, M., Mirhoseini, A., Gonzalez, J. E., and Stoica, I. (2021). Representing long-range context for graph neural networks with global attention. *Advances in neural information processing systems*, 34:13266–13279.

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. (2018). MoleculeNet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530.

Xiao, L. and Zhang, T. (2014). A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *International Conference on Learning Representations*.

Yehudai, G., Sanford, C., Bechler-Speicher, M., Fischer, O., Gilad-Bachrach, R., and Globerson, A. (2025). Depth-width tradeoffs for transformers on graph tasks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. (2021). Do transform-

ers really perform badly for graph representation? *Advances in neural information processing systems*, 34:28877–28888.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018a). Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. (2018b). Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31.

You, Y., Chen, T., Shen, Y., and Wang, Z. (2021). Graph contrastive learning automated. In *International conference on machine learning*, pages 12121–12132. PMLR.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. (2018). An end-to-end deep learning architecture for graph classification. *Proceedings of the AAAI conference on artificial intelligence*, 32(1).

Zhang, Y., Lee, J. D., and Jordan, M. I. (2016a). $\ell_1$-regularized neural networks are improperly learnable in polynomial time. In *International Conference on Machine Learning*, volume 48, pages 993–1001.

Zhang, Y., Lee, J. D., and Jordan, M. I. (2016b). l1-regularized neural networks are improperly learnable in polynomial time. In *International Conference on Machine Learning*, pages 993–1001.

Zhang, Y., Liang, P., and Wainwright, M. J. (2017). Convexified convolutional neural networks. In *International Conference on Machine Learning*, pages 4044–4053. PMLR.

Zou, D., Cao, Y., Zhou, D., and Gu, Q. (2020). Gradient descent optimizes over-parameterized deep relu networks. *Machine learning*, 109:467–492.

## Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes. Our mathematical setting and model are described in Section 3, while in Sections 4.2–4.5 we clarify that certain theoretical claims, such as inclusion in the RKHS and generalization guarantees in Theorem 4.7, hold only for certain choices of kernel functions and sufficiently smooth activation functions (see Appendices C.1-C.3). Our algorithm for learning multi-layer CGCNs is fully described in Section 4.3.]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes. Section 4.4 and Appendix F discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [No. The paper fully discloses all the information needed to reproduce the main experimental results. Further, all code is publicly available at (Cohen et al., 2026) for reproducibility as well as further inspection.]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes. In Sections 4.2–4.5, we clarify that certain theoretical claims, such as inclusion in the RKHS and generalization guarantees in Theorem 4.7, hold only for certain choices of kernel functions (e.g., Gaussian RBF) and sufficiently smooth activation functions.]
   (b) Complete proofs of all theoretical results. [Yes. Lemma 4.1 is proven in Appendix B, in Appendix D we prove Lemma 4.2, Lemma 4.3 in proven in Appendix E, and Theorem 4.7 is discussed in Section 4.5 along with a proof sketch, while a complete proof appears in Appendix G.]
   (c) Clear explanations of any assumptions. [Yes. See Appendices C.1-C.2 for possible choices of kernels and Appendix C.3 for valid activation functions, as well as an elaboration on sufficient smoothness in Remark C.2.]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [No. The paper fully discloses all the information needed to reproduce the main experimental results. All code is publicly available at (Cohen et al., 2026) for reproducibility as well as further inspection.]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes. In Section 5 and Appendix H.2, we specify all the training and test details (e.g., data

splits, hyperparameters, how they were chosen, type of optimizer, etc.). The statistics of the considered datasets appear in Appendix H.1, while Appendix H.2 includes additional implementation details.]

(c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes. In Table 1 we explicitly report standard deviations. Similarly, the bar plots in Figure 3 report mean accuracies over four runs along with error bars that denote the standard deviation.]

(d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes. The paper provides sufficient information on the computing resources needed to reproduce the experiments in Appendix H.2.]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:

(a) Citations of the creator If your work uses existing assets. [Yes. The creators and original owners of assets used in the paper are properly credited.]

(b) The license information of the assets, if applicable. [Yes. The license and terms of use are explicitly mentioned and properly respected. We also state which version of the asset is used and, where possible, include a URL.]

(c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable. The paper does not release new assets.]

(d) Information about consent from data providers/curators. [Yes. As mentioned earlier, the license and terms of use are explicitly mentioned and properly respected.]

(e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable.]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:

(a) The full text of instructions given to participants and screenshots. [Not Applicable. The paper does not involve crowdsourcing nor research with human subjects.]

(b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable. The paper does not involve crowdsourcing nor research with human subjects.]

(c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable. The paper does not involve crowdsourcing nor research with human subjects.]

# Convexified Message-Passing Graph Neural Networks: Supplementary Materials

## A ADDITIONAL RELATED WORK

### A.1 Graph Transformers

Instead of aggregating local neighborhoods, Graph Transformers capture interaction information between any pair of nodes via a single self-attention layer. Some works design specific attention mechanisms or positional encodings (Hussain et al., 2022; Kreuzer et al., 2021; Ma et al., 2023; Ying et al., 2021), while others combine message-passing GNNs to build hybrid architectures (Chen et al., 2022; Rampášek et al., 2022; Shi et al., 2021; Wu et al., 2021). Those methods enable nodes to interact with any other node, facilitating direct modeling of long-range relations. Our framework broadly applies to such hybrid methods, as we exhibit on the leading models GraphGPS (Rampášek et al., 2022) and GraphTrans (Shi et al., 2021; Wu et al., 2021). Empirically, applying our framework consistently improves performance in graph classification tasks, with significant gains on GraphGPS, showcasing the versatility and effectiveness of our approach.

### A.2 Comparison to Convexified CNNs (Zhang et al., 2017)

There is a fundamental mathematical gap between CNNs (Zhang et al., 2017) and message-passing GNNs studied in our work. The transition from grids to graphs is not merely an application of existing theory; it requires solving a distinct set of theoretical problems related, which Zhang et al. (2017) do not address. Our work thus differs from that by Zhang et al. (2017) in several fundamental dimensions, such as:

1. **Irregular Domains:** Zhang et al. (2017) rely on the fixed, grid-like structure of images where "neighbors" are deterministic; the input has a constant dimension and regular structure. They cannot account for a domain where the "receptive field" (neighborhood) size varies from node to node as in GNNs. Further, Zhang et al. (2017) cannot handle information exchange between nodes. In contrast, our work focuses on irregular domains consisting of graphs with varying numbers of nodes and edges, which cannot be derived from the fixed-grid analysis of Zhang et al. This lack of stationarity breaks many simplifications used in the CNN setting.
2. **Differences in Convexification Procedures:** Zhang et al. (2017) use RKHS constructions tied to convolutional filters and patch extraction on grids. The kernel acts on image patches and the analysis exploits fixed patch geometry. Conversely, our work requires a different RKHS construction designed to explicitly model the recursive aggregation of information across multiple hops.
3. **Nuclear norm relaxation:** Unlike Zhang et al. (2017), our nuclear norm relaxation is applied to matrix representations of graph filters that concatenate hop-wise filters across features. The low-rank structure thus has node-hop-feature semantics (not just patches in a grid).

Finally, we show how the convexification can be applied to a wide family of message-passing GNNs and hybrid transformer models, which cannot be treated by Zhang et al. (2017).

### A.3 Fundamental Distinctions from Cohen and Agmon (2021a)

While our work and that by Cohen and Agmon (2021a) use convexification in graph contexts, the architectural foundations, scope and mathematical formulations are fundamentally different.

- **Modeling and Scope:** Cohen and Agmon (2021a) proposed convex versions of specific GNN models termed as aggregation GNN, which apply only to a signal with temporal structure; particularly, their

formulation is restricted to a **fixed** communication topology. Their work is also narrowly tailored and focused on a specific application in swarm robotics, where convexity arises from modeling assumptions tailored to domain adaptation. In contrast, our CGNN framework establishes a general and theoretically principled convexification of diverse message-passing GNNs by transforming the standard architecture itself, independent of application-specific constraints. In other words, we resolve key mathematical limitations of their work:

1. Cohen and Agmon (2021a): Their formulation is mathematically restricted to a **fixed communication topology**. Consequently, **their method is undefined if the number of nodes changes between training and inference**. It cannot be applied to standard graph classification benchmarks.
2. Our CGNN Framework: In contrast, our framework is explicitly designed for **multiple, variable-sized graphs**. By relaxing the message-passing filters directly (rather than polynomial coefficients of a fixed operator), our method supports training on **multiple, heterogeneous graphs**. As detailed in Appendix H, our implementation handles datasets with varying numbers of nodes and edges, **a capability that is mathematically impossible under the formulation of Cohen and Agmon (2021a)**.

- **Modularity and Generalization:** Unlike Cohen and Agmon (2021a), whose formulation does not preserve standard GNN modularity and is limited to a **single fixed graph and a specific control setting**, our framework retains the usual GNN architectural structure and supports **multiple** graphs during training. **Our convexification also extends to, e.g., node-level tasks** by applying the same procedure with a node-level loss on the final-layer features; no further reformulation is required. In contrast, their model cannot be directly adapted to generic node or graph prediction without substantial redesign. Hence, CGNNs provide a scalable, domain-agnostic convexification framework with a fundamentally different mathematical basis. In summary, the key modularity differences are:

  1. Cohen and Agmon (2021a): Narrowly tailored to a specific control setting (swarm robotics) and requires substantial redesign to adapt to other tasks.
  2. Our Work: Our framework preserves the modularity of standard GNNs; extending our convexification to node-level tasks (as opposed to graph-level) requires only a change in the final loss function, not a reformulation of the architecture.

- **Tighter Theoretical Foundations and Different Models:** Our nuclear norm relaxation step and derived upper bound on relaxed filters' nuclear norm are fundamentally different, much tighter and more general than those presented by Cohen and Agmon (2021a).

  1. The analysis by Cohen and Agmon (2021a) is limited to **polynomial** filters. After the nuclear norm relaxation of **polynomial** filters, Cohen and Agmon (2021a) prove that the nuclear norm of the filters at layer $\ell$ is at most $R_\ell F_\ell F_{\ell-1}\sqrt{\sum_i(\sum_k \lambda_i^k)^2}$. Here, $\lambda_1, \ldots, \lambda_n$ are the distinct eigenvalues of the shift operator $\mathcal{S}$, $R_\ell, F_\ell, K$ are as in Section 3.
  2. Our analysis is fundamentally different: it is **not restricted to polynomial filters**, but applies to **general** filters. After the nuclear norm relaxation, Lemma 4.1 gives an upper bound on the filters' nuclear norm, whose derivation is simpler and yields a much tighter upper bound for general filters. Namely, we prove that the nuclear norm of ***any*** possible filter at layer $\ell$ is at most $R_\ell\sqrt{F_\ell F_{\ell-1}(K+1)}$. Eventually, this results in a smaller generalization gap bound, as observed in Theorem 4.7.

- **Algorithmic Differences:** Hence, our relaxation to a reproducing kernel Hilbert space (RKHS) and its analysis also substantially differ from Cohen and Agmon (2021a).

  1. The methods of Cohen and Agmon (2021a) are limited to learning a relaxation of the frequency representation of **polynomial** filters. Further, their algorithm is restricted to using classic projected gradient descent instead of more modern optimizers. Also, the implementation of their framework is confined to graphs with the **same** number of nodes.
  2. As before, our analysis **is not restricted to polynomial filters**. Rather, it applies to **general non-convex** filters, by directly learning the relaxed filters themselves. Our algorithm can also employ a projected version of any modern optimizer (e.g., Adam). Additionally, as detailed in Appendix H, our implementation utilizes several components that allow use to train our convex models on datasets with graphs containing **varying numbers of nodes and edges**.

In short, the algorithmic differences are:

1. Cohen and Agmon (2021a): **Restricted to learning a relaxation of the frequency representation of polynomial filters** using classic projected gradient descent.
2. Our Work: Our relaxation to a Reproducing Kernel Hilbert Space (RKHS) allows us to **directly learn the relaxed filters using any modern optimizer** (e.g., Adam), ensuring better scalability and convergence in practice.

### A.4 GNNs as Unified Optimization Problems

Our convexification framework relates to and fundamentally differs from prior work on GNNs as unified optimization problems. Chen et al. (2023b) first classify spatial GNNs based on how they aggregate neighbor information, while discussing how graph convolution can be depicted as an optimization problem. They also discuss how filters of spectral GNNs can be approximated via either linear, polynomial or rational approximation. The works by He et al. (2021), Wang and Zhang (2022) and Guo and Wei (2023) focus on polynomial approximation, treating spectral filters as parameterized by a polynomial basis. Specifically:

- BernNet (He et al., 2021) parameterizes an arbitrary spectral filter in a Bernstein polynomial basis. BernNet is analyzed from the perspective of graph optimization, showing that any polynomial filter that attempts to approximate a valid filter has to take the form of BernNet.
- Wang and Zhang (2022) notice that spectral GNNs with different polynomial bases of the spectral filters have the same expressive power but different empirical performance. Thus, for **linear** GNNs, they analyze the effect of polynomial basis from an optimization perspective. Specifically, they show that a set of orthonormal bases whose weight function corresponds to the graph signal density can enable linear GNNs to achieve the highest convergence rate.
- Guo and Wei (2023) continue that work, by first proposing for learning an orthonormal polynomial basis. Then, they tackle the definition of optimal basis from an optimization perspective in [3], proposing a model that computes the optimal basis for a given graph structure and graph signal.

However, in all those works training remains ***non-convex***. Our contribution is thus orthogonal and complementary: rather than deriving architectures, we ***convexify the training problem*** itself for general, non-convex message-passing GNNs (not only spectral ones). Training thus becomes a convex optimization problem, ensuring global optimality as well as an accurate and rigorous analysis of statistical properties. Further, our framework applies directly to general message-passing GNNs and hybrid transformer architectures like GraphGPS and GraphTrans, whereas The works by He et al. (2021), Wang and Zhang (2022) and Guo and Wei (2023) only cover spectral GNNs. In fact, the optimization-oriented analysis of Wang and Zhang (2022) is restricted to *linear* GNNs, while we focus on convexifying ***non-convex*** GNNs. We will add a dedicated discussion explicitly contrasting these settings.

### A.5 Classic Graph Kernels

1. **The relation to Weisfeiler-Lehman:** The most direct link lies in the Weisfeiler-Lehman (WL) graph kernel (Shervashidze et al., 2011). It is well-established that standard message-passing GNNs are functionally equivalent to the 1-WL isomorphism test in terms of expressive power (Xu et al., 2019). Classical WL kernels operate by explicitly mapping graphs to high-dimensional sparse feature vectors containing counts of color-refined subtrees, which are then typically optimized via Support Vector Machines (SVMs).
    - **The WL kernel approach:** The kernel method separates feature extraction (WL refinement) from learning (SVM). The "learning" is convex (quadratic programming), but the features are fixed and discrete.
    - **Our Approach:** Our CGNN framework internalizes this process. By mapping the nonlinear message-passing filters themselves into an RKHS, we do not merely extract fixed features; we learn the propagation mechanism itself. Crucially, we achieve this while retaining the hallmark benefit of the SVM era: ***convexity***. CGNNs thus guarantee global optima, eliminating the local minima issues plaguing standard non-convex GNN training.

2. **Implicit vs. Explicit Infinite-Dimensional Spaces:** Classical graph kernels rely on the kernel trick to implicitly operate in high-dimensional spaces without computing coordinates. Conversely, widely-used GNNs (e.g., GCN, GIN) operate in explicit, low-dimensional Euclidean spaces. Convexified GNNs occupy a middle ground that leverages the best of both worlds. We utilize the RKHS formalism to first parameterize graph filters in a countable-dimensional feature space. By the representer theorem, we give a reduction of the original ERM problem to a ***finite***-dimensional one. Then, we solve the learning problem via projected gradient descent. This mirrors the Infinite-Width GNN (or Neural Tangent Kernel) regime, but with a tractable, finite-data implementation. While Graph Neural Tangent Kernels (GNTK) (Du et al., 2019; Krishnagopal and Ruiz, 2023) are often static and used as fixed priors, CGNNs remain adaptable, learning the representation within the convex constraint set.

3. **Dataset Implications and Generalization:** The connection is empirically relevant given the standard evaluation benchmarks. Datasets such as TUDataset (e.g., MUTAG, PROTEINS, NCI1) were originally curated to benchmark graph kernels. For years, kernel methods (like the WL-subtree kernel and Deep Graph Kernels) outperformed early GNNs on these tasks due to the small sample sizes, where over-parameterized GNNs were prone to overfitting. By convexifying GNNs, our method introduces rigorous regularization inherent to the RKHS norm. This explains the strong performance of CGNNs on these "kernel-dominated" datasets: we provide the adaptivity of a neural network but with the complexity control and generalization guarantees typically associated with SVM-based kernel methods. Thus, CGNNs can be viewed as the natural neural evolution of the WL-kernel: preserving the global optimization guarantees while enabling end-to-end learning of continuous message-passing dynamics.

# B UPPER BOUNDING THE NUCLEAR NORM

**Lemma 4.1.** *For any layer $\ell$, $\|\mathcal{A}_\ell\|_* \leq \mathcal{B}_\ell$ for some constant $\mathcal{B}_\ell > 0$ dependent on $R_\ell, F_\ell, F_{\ell-1}, K$.*

*Proof.* Note that the nuclear norm of any matrix $B$ can be bounded as $\|B\|_* \leq \|B\|_F \sqrt{\text{rank}(B)}$, where $\|B\|_F$ is the Frobenius norm which is identical to the standard Euclidean norm of the vectorized version of the matrix. Hence, the nuclear norm of the filters $\mathcal{A}_\ell$ satisfying (5a) and (5b) is upper bounded by

$$\|\mathcal{A}_\ell\|_*^2 \leq \|\mathcal{A}_\ell\|_F^2 F_{\ell-1} = \sum_{k=0}^{K} \sum_{f\in[F_\ell]} \sum_{g\in[F_{\ell-1}]} |a_{\ell k}^{fg}|^2 \cdot F_{\ell-1} \leq R_\ell^2 F_\ell F_{\ell-1}(K+1).$$

That is, $\|\mathcal{A}_\ell\|_* \leq \mathcal{B}_\ell$ for $\mathcal{B}_\ell := R_\ell \sqrt{F_\ell F_{\ell-1}(K+1)}$. □

# C CHOICE OF KERNELS AND VALID ACTIVATIONS

In this section, we provide properties regarding two types of kernels: the *inverse polynomial kernel* (Section C.1) and the *Gaussian RBF kernel* (Section C.2). We show that the reproducing kernel Hilbert space (RKHS) corresponding to these kernels contain the nonlinear filter at each layer $\ell$ for the $f^{\text{th}}$ feature of node $i$ for certain choices of an activation function $\sigma$. Finally, we summarize the possible choices for an activation function $\sigma$ that result from our discussion (Section C.3).

## C.1 Inverse Polynomial Kernel

For $F$ features, the *inverse polynomial kernel* $\kappa^{IP} : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}$ is given as follows:

$$\kappa^{IP}(\mathbf{z}, \mathbf{z}') := \frac{1}{2 - \langle \mathbf{z}, \mathbf{z}' \rangle}; \quad \|\mathbf{z}\|_2 \leq 1, \|\mathbf{z}'\|_2 \leq 1 \tag{12}$$

Zhang et al. (2017) prove that $\kappa^{IP}$ indeed constitutes a legal kernel function. We include the proof to make the paper self-contained: they provide a feature mapping $\varphi : \mathbb{R}^F \to \ell^2(\mathbb{N})$ for which $\kappa^{IP}(\mathbf{z}, \mathbf{z}') = \langle \varphi(\mathbf{z}), \varphi(\mathbf{z}') \rangle$. The $(k_1, \ldots, k_j)$-th coordinate of $\varphi(\mathbf{z})$, where $j \in \mathbb{N}$ and $k_1, \ldots, k_j \in [F_\ell]$, is given by $2^{-\frac{j+1}{2}} x_{k_1} \ldots x_{k_j}$. Thus:

$$\langle \varphi(\mathbf{z}), \varphi(\mathbf{z}') \rangle = \sum_{j=0}^{\infty} 2^{-(j+1)} \sum_{(k_1,\ldots,k_j)\in[F_\ell]^j} z_{k_1} \ldots z_{k_j} z'_{k_1} \ldots z'_{k_j} \tag{13}$$

Since $\|\mathbf{z}\|_2 \leq 1$ and $\|\mathbf{z}'\|_2 \leq 1$, the series above is absolutely convergent. Combined with the fact that $|\langle \mathbf{z}, \mathbf{z}' \rangle| \leq 1$, this yields a simplified form of (13):

$$\langle \varphi(\mathbf{z}), \varphi(\mathbf{z}') \rangle = \sum_{j=0}^{\infty} 2^{-(j+1)} (\langle \mathbf{z}, \mathbf{z}' \rangle)^j = \frac{1}{2 - \langle \mathbf{z}, \mathbf{z}' \rangle} = \kappa^{IP}(\mathbf{z}, \mathbf{z}') \tag{14}$$

as desired. Combining (7) with the proof of Lemma 1 in Appendix A.1 of Zhang et al. (Zhang et al., 2016b), the RKHS associated with $\kappa^{IP}$ is comprised of the class of nonlinear graph filters in (7). It can be formulated as follows:

**Corollary C.1.** *Assume that $\sigma$ has a polynomial expansion $\sigma(t) = \sum_{j=0}^{\infty} a_j t^j$. Let $C_\sigma(t) := \sqrt{\sum_{j=0}^{\infty} 2^{j+1} a_j^2 t^{2j}}$. If $C_\sigma(\|\mathbf{a}_{\ell k}^f\|_2) < \infty$, then the RKHS induced by $\kappa^{IP}$ contains the function $\tau_\ell^f : \mathcal{Z} \in \mathbb{R}^{(K+1)\times F_{\ell-1}} \mapsto \sigma(\sum_{k=0}^{K} \langle [\mathcal{Z}]_{k+1}, \mathbf{a}_{\ell k}^f \rangle)$ with Hilbert norm $\|\tau_\ell^f\|_{\mathcal{A}} \leq C_\sigma(\sum_{k=0}^{K} \|\mathbf{a}_{\ell k}^f\|_2)$.*

Consequently, Zhang et al. (Zhang et al., 2016b) state that upper bounding $C_\sigma(t)$ for a particular activation function $\sigma$ is sufficient. To enforce $C_\sigma(t) < \infty$, the scalars $\{a_j\}_{j=0}^{\infty}$ must rapidly converge to zero, implying that $\sigma$ must be sufficiently smooth. For polynomial functions of degree $d$, the definition of $C_\sigma$ yields $C_\sigma(t) = \mathcal{O}(t^d)$. For the *sinusoid* activation $\sigma(t) := sin(t)$, we have:

$$C_\sigma(t) = \sqrt{\sum_{j=0}^{\infty} \frac{2^{2j+2}}{((2j+1)!)^2} (t^2)^{2j+1}} \leq 2e^{t^2} \tag{15}$$

For the *erf* function $\sigma_{erf}(t) := \frac{2}{\sqrt{\pi}} \int_0^t e^{z^2} dz$ and the *smoothed hinge loss* function $\sigma_{sh}(t) := \int_{-\infty}^t \frac{1}{2}(\sigma_{erf}(t)+1)dz$, Zhang et al. (Zhang et al., 2016b) proved that $C_\sigma(t) = \mathcal{O}(e^{ct^2})$ for universal numerical constant $c > 0$.

**Remark C.2.** *Generally, a function $f$ is* ***sufficiently smooth*** *if it is differentiable* sufficient *number of times. For instance, if we were to consider $\sigma(t) := \sin(t)$, its taylor expansion is provided by $\sin(t) = \sum_{j=0}^{\infty} \frac{(-1)^j}{(2j+1)!} t^{2j+1}$. Denoting $b_j := \frac{(-1)^j}{(2j+1)!} t^{2j+1}$, the ratio test yields $\lim_{j\to\infty} |\frac{b_{j+1}}{b_j}| = 0$ for all $t \in IR$. Hence, the radius of convergence of the expansion is the set of all real numbers.*

*Regarding Corollary C.1, as well as Corollary C.3 which follows in the next subsection, for enforcing $C_\sigma(t) < \infty$, the coefficients $\{a_j\}_{j=0}^{\infty}$ in $\sigma$'s polynomial expansion must quickly converge to zero as illustrated above, thus requiring $\sigma$ to be sufficiently smooth. Such activations include* **polynomial** *functions, the* **sinusoid**, *the* **erf** *function and the* **smoothed hinge loss**.

### C.2 Gaussian RBF Kernel

For $F$ features, the *Gaussian RBF kernel* $\kappa^{RBF} : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}$ is given as follows:

$$\kappa^{RBF}(\mathbf{z}, \mathbf{z}') := e^{-\gamma \|\mathbf{z}-\mathbf{z}'\|_2^2}; \quad \|\mathbf{z}\|_2 = \|\mathbf{z}'\|_2 = 1 \tag{16}$$

Combining (7) with the proof of Lemma 2 in Appendix A.2 of Zhang et al. (Zhang et al., 2016b), the RKHS associated with $\kappa^{RBF}$ is comprised of the class of nonlinear input features. It can be formulated as follows:

**Corollary C.3.** *Assume that $\sigma$ has a polynomial expansion $\sigma(t) = \sum_{j=0}^{\infty} a_j t^j$. Let $C_\sigma(t) := \sqrt{\sum_{j=0}^{\infty} \frac{j! e^{2\gamma}}{(2\gamma)^j} a_j^2 t^{2j}}$. If $C_\sigma(\|\mathbf{a}_{\ell k}^f\|_2) < \infty$, then the RKHS induced by $\kappa^{RBF}$ contains the function $\tau_\ell^f : \mathcal{Z} \in \mathbb{R}^{(K+1)\times F_{\ell-1}} \mapsto \sigma(\sum_{k=0}^K \langle [\mathcal{Z}]_{k+1}, \mathbf{a}_{\ell k}^f \rangle)$ with Hilbert norm $\|\tau_\ell^f\|_{\mathcal{A}} \le C_\sigma(\sum_{k=0}^K \|\mathbf{a}_{\ell k}^f\|_2)$.*

Comparing Corollary C.1 and Corollary C.3, $\kappa^{RBF}$ requires a stronger condition on the smoothness of the activation function. For polynomial functions of degree $d$, $C_\sigma(t) = \mathcal{O}(t^d)$ remains intact. However, for the *sinusoid* activation $\sigma(t) := \sin(t)$, we have:

$$C_\sigma(t) = \sqrt{e^{2\gamma} \sum_{j=0}^{\infty} \frac{1}{(2j+1)!} (\frac{t^2}{2\gamma})^{2j+1}} \le e^{\frac{t^2}{4\gamma}+\gamma} \tag{17}$$

In contrast, for both the *erf* function $\sigma_{erf}(t) := \frac{2}{\sqrt{\pi}} \int_0^t e^{z^2} dz$ and the *smoothed hinge loss* function $\sigma_{sh}(t) := \int_{-\infty}^t \frac{1}{2}(\sigma_{erf}(t)+1)dz$, $C_\sigma(t)$ is **infinite**. Hence, **the RKHS induced by $\kappa^{RBF}$ does not contain input features activated by those two functions**.

### C.3 Valid Activation Functions

Now, we summarize the possible valid choices of an activation function $\sigma$:

1. Arbitrary polynomial functions.
2. The sinusoid activation function $\sigma(t) := \sin(t)$.
3. The *erf* function $\sigma_{erf}(t) := \frac{2}{\sqrt{\pi}} \int_0^t e^{\tau^2} \mathrm{d}\tau$, which approximates the sigmoid function.
4. The *smoothed hinge loss* function $\sigma_{sh}(t) := \int_{-\infty}^t \frac{1}{2}(\sigma_{erf}(\tau)+1)\mathrm{d}\tau$, which approximates the ReLU function.

Recall that the *erf* and *smoothed hinge loss* activation functions are unsuited for the RBF kernel.

## D REPHRASING THE NONLINEAR FILTER AT EACH LAYER

*Proof.* For any graph signal $\mathcal{S}^k\mathcal{X}$, let $\mathbf{z}_i^k(\mathcal{X}) := [\mathcal{S}^k\mathcal{X}]_i$ be the $i^{\text{th}}$ row of $\mathcal{S}^k\mathcal{X}$, which is the information aggregated at node $i$ from $k$-hops away. We also denote the $f^{\text{th}}$ column of the filters $\mathcal{A}_{\ell k}$ as $\mathbf{a}_{\ell k}^f = [a_{\ell k}^{fg}]_{g\in[F_{\ell-1}]}$. By (1), the

value of the $f^{\text{th}}$ feature of node $i$ at layer $\ell$ can be thus written as:

$$[\Psi^{\mathcal{A}_\ell}(\mathcal{X};\mathcal{S})]_{if} = \sum_{k=0}^{K}\sum_{g=1}^{F_{\ell-1}}[\mathbf{z}_i^k(\mathcal{X}_{\ell-1})]_g[\mathbf{a}_{\ell k}^f]_g = \sum_{k=0}^{K}\langle \mathbf{z}_i^k(\mathcal{X}_{\ell-1}), \mathbf{a}_{\ell k}^f\rangle. \tag{18}$$

□

# E KERNELIZATION FOR NONLINEAR ACTIVATIONS

In this section, we elaborate on the kernelization for the case of nonlinear activation functions, briefly discussed in Section 4.2. Recall that $\sigma$ denotes, for brevity, its entrywise application in (2). Further, the of the $f^{\text{th}}$ feature of node $i$ at layer $\ell$'s output is given by $\sigma(\sum_{k=0}^{K}\langle \mathbf{z}_i^k(\mathcal{X}_{\ell-1}), \mathbf{a}_{\ell k}^f\rangle)$ by (18). According to Appendix C, consider a sufficiently smooth valid activation function $\sigma$, and a proper choice of kernel $\kappa_\ell : \mathbb{R}^{F_{\ell-1}} \times \mathbb{R}^{F_{\ell-1}} \to \mathbb{R}$. Without loss of generality, we assume that, each input input graph signal $\mathcal{X}^{(j)}$, the feature vector $[\mathcal{X}^{(j)}]_i = \mathbf{x}_i^{(j)}$ of each node $i$ resides in the unit $\ell_2$-ball (which can be obtain via normalization). Combined with $\kappa_\ell$'s being a continuous kernel with $\kappa_\ell(\mathbf{z},\mathbf{z}) \le 1$, Mercer's theorem (Steinwart, 2008, Theorem 4.49) implies that there exists a feature mapping $\varphi_\ell : \mathbb{R}^{F_{\ell-1}} \to \ell^2(\mathbb{N})$ for which $\kappa_\ell(\mathbf{z},\mathbf{z}') = \langle \varphi_\ell(\mathbf{z}), \varphi_\ell(\mathbf{z}')\rangle$, which yields the following for each feature $f$ of any node $i$ at layer $\ell$:

$$\sigma\left(\sum_{k=0}^{K}\langle \mathbf{z}_i^k(\mathcal{X}_{\ell-1}), \mathbf{a}_{\ell k}^f\rangle\right) = \sum_{k=0}^{K}\langle \varphi_\ell(\mathbf{z}_i^k(\mathcal{X}_{\ell-1})), \varphi_\ell(\mathbf{a}_{\ell k}^f)\rangle =: \sum_{k=0}^{K}\langle \varphi_\ell(\mathbf{z}_i^k(\mathcal{X}_{\ell-1})), \bar{\mathbf{a}}_{\ell k}^f\rangle \tag{19}$$

where $\bar{\mathbf{a}}_{\ell k}^f \in \ell^2(\mathbb{N})$ is a countable-dimensional vector, due to the fact that $\varphi$ itself is a countable sequence of functions. By Corollary C.1 and Corollary C.3, we have that $\|\bar{\mathbf{a}}_{\ell k}^f\|_2 \le C_{\sigma,\ell}(\|\mathbf{a}_{\ell k}^f\|_2)$, provided a monotonically increasing $C_\sigma$ which depends on the kernel $\kappa_\ell$. Accordingly, $\varphi_\ell(\mathbf{z})$ may be utilized as the vectorized representation of each output features $\mathbf{z}_i^k(\mathcal{X}_{\ell-1})$, where $\bar{\mathbf{a}}_{\ell k}^f$ constitutes the linear graph filters, thus reducing the problem to training a GCN with the identity activation function.

Each graph filter is now parametrized by the *countable*-dimensional vector $\bar{\mathbf{a}}_{\ell k}^f$. Subsequently, we supply a reduction of the original ERM problem to a *finite*-dimensional one. Output on the training data $\mathcal{T} := \{(\mathcal{G}^{(j)}, \mathcal{X}^{(j)}, y^{(j)})\}_{j=1}^m$ should be solely considered when solving the ERM problem, i.e., $\sum_{k=0}^{K}\langle \varphi_\ell(\mathbf{z}_i^k(\mathcal{X}_{\ell-1})), \bar{\mathbf{a}}_{\ell k}^f\rangle$ for each feature $f$ of any node $i$ at layer $\ell$. Let $\mathcal{P}_{\ell-1}^k$ be the orthogonal projector onto the linear subspace spanned by $\Gamma_{\ell-1}^k := \{\mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)}) : i \in [n], j \in [m], k \in [K]\cup\{0\}\}$. Therefore, for any $(i,j) \in [n]\times[m]$:

$$\sum_{k=0}^{K}\langle \varphi_\ell(\mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)})), \bar{\mathbf{a}}_{\ell k}^f\rangle = \sum_{k=0}^{K}\langle \mathcal{P}_{\ell-1}^k\varphi_\ell(\mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)})), \bar{\mathbf{a}}_{\ell k}^f\rangle = \sum_{k=0}^{K}\langle \varphi_\ell(\mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)})), \mathcal{P}_{\ell-1}^k\bar{\mathbf{a}}_{\ell k}^f\rangle \tag{20}$$

where the last step stems from the fact that the orthogonal projector $\mathcal{P}_{\ell-1}^k$ is self-adjoint. Without loss of generality, for the sake of solving the ERM, we assume that $\bar{\mathbf{a}}_{\ell k}^f$ belongs to $\Gamma_{\ell-1}^k$. Similarly to (8), we can thus reparameterize it by:

$$\bar{\mathbf{a}}_{\ell k}^f = \sum_{(i,j)\in[n]\times[m]}\beta_{\ell k,(i,j)}^f\varphi_\ell(\mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)})) \tag{21}$$

where we denote $\beta_{\ell k}^f \in \mathbb{R}^{nm}$ as a vector of the coefficients, whose $(i,j)$-th coordinate is $\beta_{\ell k,(i,j)}^f$. To estimate $\bar{\mathbf{a}}_{\ell k}^f$, it thus suffices to estimate the vector $\beta_{\ell k}^f$. By definition, the relation $(\beta_{\ell k}^f)^T\mathcal{K}_\ell^k\beta_{\ell k}^f = \|\bar{\mathbf{a}}_{\ell k}^f\|_2^2$ holds, where $\mathcal{K}_\ell^k \in \mathbb{R}^{nm\times nm}$ is the symmetric kernel matrix from Section 4.2, whose rows and columns are indexed by some triple $(i,j) \in [n]\times[m]$. The entry at row $(i,j)$ and column $(i',j')$ equals to $\kappa_\ell(\mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)}), \mathbf{z}_{i'}^k(\mathcal{X}_{\ell-1}^{(j')}))$. A suitable factorization of $\mathcal{K}_\ell^k$ such that $\mathcal{K}_\ell^k = Q_\ell^k(Q_\ell^k)^\top$ for $Q_\ell^k \in \mathbb{R}^{nm\times P}$ then yields the following norm constraint:

$$\|(Q_\ell^k)^\top\beta_{\ell k}^f\|_2 = \sqrt{(\beta_{\ell k}^f)^T\mathcal{K}_\ell^k\beta_{\ell k}^f} = \|\bar{\mathbf{a}}_{\ell k}^f\|_2^2 \le C_{\sigma,\ell}(\|\mathbf{a}_{\ell k}^f\|_2) \le C_{\sigma,\ell}(R_\ell) \tag{22}$$

where the last inequality is due to $\|\mathbf{a}_{\ell k}^f\|_2 \le R_\ell$ due to (3).

Let $\mathbf{v}_\ell^k(\mathbf{z}) \in \mathbb{R}^{nm}$ be a vector whose $(i,j)$-th coordinate equals to $\kappa_\ell(\mathbf{z}, \mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)}))$. By (19) and (21), we can write:

$$\begin{aligned}\sigma\left(\sum_{k=0}^{K}\langle \mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)}), \mathbf{a}_{\ell k}^f\rangle\right) \equiv \sum_{k=0}^{K}\langle \varphi_\ell(\mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)})), \bar{\mathbf{a}}_{\ell k}^f\rangle &\equiv \sum_{k=0}^{K}\langle \mathbf{v}_\ell^k(\mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)})), \beta_{\ell k}^f\rangle \\ &\equiv \sum_{k=0}^{K}\langle \beta_{\ell k}^f, \mathbf{v}_\ell^k(\mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)}))\rangle\end{aligned} \tag{23}$$

For each $\mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)})$, the vector $\mathbf{v}_\ell^k(\mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)}))$ resides in the column space of the symmetric kernel matrix $\mathcal{K}_\ell^k$. Thus, given that $(Q_\ell^k)^\dagger$ denotes the pseudo-inverse of $Q_\ell^k$, we infer the following for any $(i,j) \in [n] \times [m]$:

$$\begin{aligned}\sum_{k=0}^{K}\langle \beta_{\ell k}^f, \mathbf{v}_\ell^k(\mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)}))\rangle &= \sum_{k=0}^{K}(\beta_{\ell k}^f)^\top Q_\ell^k (Q_\ell^k)^\dagger \mathbf{v}_\ell^k(\mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)})) \\ &= \sum_{k=0}^{K}\left\langle ((Q_\ell^k)^\top)^\dagger (Q_\ell^k)^\top \beta_{\ell k}^f, \mathbf{v}_\ell^k(\mathbf{z}_i^k(\mathcal{X}_{\ell-1}^{(j)}))\right\rangle\end{aligned} \tag{24}$$

Accordingly, replacing the vector $\beta_{\ell k}^f$ on the right-hand side of (23) by $((Q_\ell^k)^\top)^\dagger (Q_\ell^k)^\top \beta_{\ell k}^f$ won't affect the empirical risk. For any matrix $\mathcal{Z} \in \mathbb{R}^{(K+1)\times F_{\ell-1}}$ whose $(k+1)^{\text{th}}$ row is given by $[\mathcal{Z}]_{k+1} := \mathbf{z}_k$ for $k \in [K] \cup \{0\}$, we thereby propose the following parameterization function:

$$\tau_\ell^f(\mathcal{Z}) := \sum_{k=0}^{K}\left\langle ((Q_\ell^k)^\top)^\dagger (Q_\ell^k)^\top \beta_{\ell k}^f, \mathbf{v}_\ell^k(\mathbf{z}_k)\right\rangle = \sum_{k=0}^{K}\langle (Q_\ell^k)^\dagger \mathbf{v}_\ell^k(\mathbf{z}_k), (Q_\ell^k)^\top \beta_{\ell k}^f\rangle \tag{25}$$

Consequently, the value of the $f^{\text{th}}$ feature of node $i$ at layer $\ell$ is given by $\tau_\ell^f(\mathcal{Z}_{i,\ell}(\mathcal{X}_{\ell-1}))$, where $\mathcal{Z}_{i,\ell}(\mathcal{X}_{\ell-1}) \in \mathbb{R}^{(K+1)\times F_{\ell-1}}$ is the matrix whose $(k+1)^{\text{th}}$ row is $[\mathcal{Z}_{i,\ell}(\mathcal{X}_{\ell-1})]_k = \mathbf{z}_i^k(\mathcal{X}_{\ell-1})$ for $k \in [K] \cup \{0\}$. That is, $(Q_\ell^k)^\top \beta_{\ell k}^f$ now act as the filters under the reparameterization in (25) which satisfy the following constraints:

**Norm Constraint.** $\|(Q_\ell^k)^\top \beta_{\ell k}^f\|_2 \leq C_{\sigma,\ell}(R_\ell)$ due to (22).

**Rank Constraint.** Its rank is at most $F_{\ell-1}$.

Therefore, similarly to the linear case in Section 4.1, those constraints can be relaxed to the nuclear norm constraint:

$$\|(Q_\ell^k)^\top \beta_{\ell k}^f\|_* \leq C_{\sigma,\ell}(R_\ell)\sqrt{F_\ell F_{\ell-1}(K+1)} \tag{26}$$

Comparing the constraint (6) from the linear activation function case with (26), we notice that the sole distinction is that the term $R_\ell$ is replaced by $C_{\sigma,\ell}(R_\ell)$, which is required due to our use of the kernel trick for reducing the case of nonlinear activation functions to the linear setting.

# F MORE DETAILS ON MATRIX APPROXIMATION AND FACTORIZATION METHODS

As mentioned in Section 4.4, the time complexity of Algorithm 2 largely depends on the width $P$ of each factorization matrix $Q_\ell^k$ and the method used to obtain it (line 3). A naive choice of $P = nm$ corresponds to solving the full kernelized problem, which can be computationally expensive. For instance, using Cholesky factorization with $P = nm$ (Dereniowski and Kubale, 2003) incurs a complexity of $\mathcal{O}(m^3n^3)$ and space complexity of $\mathcal{O}(m^2n^2)$.

However, to gain scalability, we use *Nyström approximation* (Williams and Seeger, 2001): an approximation $\mathcal{K}_\ell^k \approx Q_\ell^k (Q_\ell^k)^\top$ for $Q_\ell^k \in \mathbb{R}^{nm\times P}$ is obtained by randomly sampling $P$ rows/columns from the original $\mathcal{K}_\ell^k$, which takes $\mathcal{O}(P^2nm)$ time. In practice, it randomly samples training examples, computes their kernel matrix and represents each training example by its similarity to the sampled examples and kernel matrix. It is well-known that this method can significantly accelerate our computations (Williams and Seeger, 2001; Drineas and Mahoney, 2005), which is attained by using the approximate kernel matrix instead of the full one. Another advantage is that it is not necessary to compute or store the full kernel matrix, but only a submatrix requiring space complexity of $\mathcal{O}(Pmn)$.

The Nyström method works effectively so long as the sampled training examples and the kernel matrix adequately represent the entire graph. However, if some training examples are distant from the randomly sampled ones, they may not be well-represented.

Another possible approximation method is the *random feature approximation* (Rahimi and Recht, 2007), which can be executed in $\mathcal{O}(nmP \log F_\ell)$ time (Le et al., 2013). Overall, using approximations significantly accelerates our algorithm's performance in terms of time and space complexity.

# G PROOF OF THEOREM 4.7

**Theorem 4.7.** *Consider models with a single hidden layer for binary classification (i.e., $L = 1, G = 2$). Let $J(\cdot, \mathbf{y})$ be $M$-Lipchitz continuous for any fixed $\mathbf{y} \in [G]^n$ and let $\kappa_L$ be either the inverse polynomial kernel or the Gaussian RBF kernel (Recall Appendices C.1-C.2). For any valid activation function $\sigma$ (Recall Appendix C.3), there is a constant $C_{\sigma,L}(\mathcal{B}_L)$ such that with the radius $R_L := C_{\sigma,L}(\mathcal{B}_L)\sqrt{2F}$, the expected generalization error is at most:*

$$\mathbb{E}_{\mathcal{X},\mathbf{y}}[J(\hat{\Psi}^{\hat{\mathcal{A}}_L}(\mathcal{X}), \mathbf{y})] \leq \inf_{\Phi \in \mathcal{F}_{gcn}} \mathbb{E}_{\mathcal{X},\mathbf{y}}[J(\Phi(\mathcal{X}), \mathbf{y})] + \frac{cMR_L\sqrt{\log(n(K+1))\sum_{k=0}^{K}\mathbb{E}[||K_L^k(\mathcal{X})||_2]}}{\sqrt{m}} \tag{27}$$

*where $c > 0$ is a universal constant.*

We begin by proving several properties that apply to the general multi-class case with models comprising of multiple hidden layers, and then we specialize them to single-hidden-layer models in the binary classification case. First, a wealthier function class shall be considered, which contains the class of GCNs. For each layer $\ell$, let $\mathcal{H}_\ell$ be the RKHS induced by the kernel function $\kappa_\ell$ and $||\cdot||_{\mathcal{H}_\ell}$ be the associated Hilbert norm. Recall that, given any matrix $\mathcal{Z} \in \mathbb{R}^{(K+1)\times F_{\ell-1}}$, we denote its $(k+1)^{\text{th}}$ row as $[\mathcal{Z}]_{k+1} := \mathbf{z}_k$ for any $k \in [K] \cup \{0\}$. Using this notation, consider the following function class:

$$\begin{aligned}
\mathcal{F}_{\text{cgcn}} &:= \{\Phi^{\mathcal{A}} = \phi^{\mathcal{A}_L} \circ \cdots \circ \phi^{\mathcal{A}_1} : \phi^{\mathcal{A}_\ell} \in \mathcal{F}_{\ell,cgcn} \quad \forall \ell \in [L]\}, \text{ where} \\
\mathcal{F}_{\ell,cgcn} &:= \Bigg\{ \phi^{\mathcal{A}_\ell} : \mathbb{R}^{n\times F^\star_{\ell-1}} \to \mathbb{R}^{n\times F_\ell} : F^\star_{\ell-1} < \infty \text{ and } [\phi^{\mathcal{A}_\ell}(\mathcal{X})]_{if} = \tau_\ell^f(\mathcal{Z}_{i,\ell}(\mathcal{X})) \\
&\qquad \text{where } \tau_\ell^f : \mathcal{Z} \in \mathbb{R}^{(K+1)\times F_{\ell-1}} \mapsto \sum_{k=0}^{K}\sum_{g=1}^{F^\star_{\ell-1}} [\mathbf{z}_k]_g [\mathbf{a}_{\ell k}^f]_g \text{ and} \\
&\qquad\qquad \|\tau_\ell^f\|_{\mathcal{H}_\ell} \leq C_{\sigma,\ell}(R_\ell)(K+1)\sqrt{F_\ell F_{\ell-1}} \Bigg\}
\end{aligned} \tag{28}$$

where $C_{\sigma,\ell}(R_\ell)$ depends solely on the chosen activation function $\sigma$ and the regularization parameter $R_\ell$. We further consider the function class of all graph filters at layer $\ell$ of a standard GCN:

$$\mathcal{F}_{\ell,gcn} := \left\{ \Psi^{\mathcal{A}_\ell} : \Psi^{\mathcal{A}_\ell}(\mathcal{X}) := \sigma\left(\sum_{k=0}^{K} \mathcal{S}^k \mathcal{X}_{\ell-1}\mathcal{A}_{\ell k}\right) \text{ and } \max_{0\leq k\leq K} \|\mathbf{a}_{\ell k}^f\|_2 \leq R_\ell \ \forall f \right\} \tag{29}$$

Next, we show that the class $\mathcal{F}_{\text{cgcn}}$ contains the class of GCNs $\mathcal{F}_{\text{gcn}}$, making the former a reacher class of functions. Particularly, the class $\mathcal{F}_{\ell,cgcn}$ of kernelized filters at layer $\ell$ contains the class $\mathcal{F}_{\ell,gcn}$ of all graph filters at layer $\ell$ corresponding to a standard GCN.

**Lemma G.1.** *For any valid activation function $\sigma$, there is some $C_{\sigma,\ell}(R_\ell)$ that solely depends on $\sigma$ and $R_\ell$ such that $\mathcal{F}_{gcn} \subset \mathcal{F}_{cgcn}$. In particular, $\mathcal{F}_{\ell,gcn} \subset \mathcal{F}_{\ell,cgcn}$.*

*Proof.* Recall that that any GCN $\Phi \in \mathcal{F}_{\text{gcn}}$ induces a nonlinear filter at each layer $\ell$ for the $f^{\text{th}}$ which can be characterized by $\tau_\ell^f : \mathcal{Z} \in \mathbb{R}^{(K+1)\times F_{\ell-1}} \mapsto \sigma(\sum_{k=0}^{K}\langle[\mathcal{Z}]_{k+1}, \mathbf{a}_{\ell k}^f\rangle)$, i.e., the GCN $\Phi$ is given by the following composition of functions $\Phi = \tau_L \circ \cdots \circ \tau_1$, where we defined $\tau_\ell : \mathbf{z} \mapsto (\tau_\ell^f(\mathbf{z}))_{f\in[F_\ell]}$. Given any valid activation function, Corollary C.1 and Corollary C.3 yield that $\tau_\ell^f$ resides within the reproducing kernel Hilbert space $\mathcal{H}_\ell$ and its Hilbert norm is upper bounded as $\|\tau_\ell^f\|_{\mathcal{H}_\ell} \leq C_\sigma(\|\sum_{k=0}^{K}\mathbf{a}_{\ell k}^f\|_2) \leq C_\sigma(R_\ell)\sqrt{K+1}$, which clearly satisfies the constraint in (28) for any layer $\ell$, as desired. □

Recalling that $\hat{\Psi}^{\hat{\mathcal{A}}_\ell}(\cdot)$ is the predictor at each layer $\ell$ produced by Algorithm 2, we now formally prove that the predictor $\hat{\Psi}^{\hat{\mathcal{A}}_L}(\cdot)$ generated at the last layer is an empirical risk minimizer in the function class $\mathcal{F}_{\text{cgcn}}$.

**Lemma G.2.** *By running Algorithm 2 with the regularization parameter $\mathcal{B}_\ell = C_{\sigma,\ell}(R_\ell)\sqrt{F_\ell F_{\ell-1}}$ at layer $\ell$, the predictor $\hat{\Psi}^{\hat{\mathcal{A}}_L}(\cdot)$ generated at the last layer satisfies:*

$$\hat{\Psi}^{\hat{\mathcal{A}}_L} \in \underset{\Phi\in\mathcal{F}_{cgcn}}{\arg\min} \frac{1}{m}\sum_{j=1}^{m} J(\Phi(\mathcal{X}^{(j)}), y^{(j)}) \tag{30}$$

*Proof.* Consider the function class of all CGCNs whose filters at layer $\ell$ have a nuclear norm of at most $\mathcal{B}_\ell$, given by:

$$\mathcal{F}_{\mathcal{B}_\ell} := \left\{ \Phi^{\mathcal{A}} = \Psi^{\mathcal{A}_L} \circ \cdots \circ \Psi^{\mathcal{A}_1} \,\middle|\, \Psi^{\mathcal{A}_\ell} : \mathcal{X} \mapsto \sum_{k=0}^{K} \mathcal{S}^k \mathcal{X}_{\ell-1} \mathcal{A}_{\ell k} \text{ and } \mathcal{A}_{\ell k} \in \mathbb{R}^{F_{\ell-1}\times F_\ell} \text{ and } \mathcal{A}_\ell\|_* \le \mathcal{B}_\ell \right\} \tag{31}$$

Our goal is thus proving that $\mathcal{F}_{\mathcal{B}_\ell} \subset \mathcal{F}_{\text{cgcn}}$, and that any empirical risk minimizer in $\mathcal{F}_{\text{cgcn}}$ is also in $\mathcal{F}_{\mathcal{B}_\ell}$. We begin with showing that $\mathcal{F}_{\mathcal{B}_\ell} \subset \mathcal{F}_{\text{cgcn}}$. Let $\Phi^{\mathcal{A}} = \Psi^{\mathcal{A}_L} \circ \cdots \circ \Psi^{\mathcal{A}_1} \in \mathcal{F}_{\mathcal{B}_\ell}$ such that $\Psi^{\mathcal{A}_\ell}(\mathcal{X}) = \sum_{k=0}^{K} \mathcal{S}^k \mathcal{X}_{\ell-1}\mathcal{A}_{\ell k}$ with $\|\mathcal{A}_\ell\|_* \le \mathcal{B}_\ell$ for any $\ell \in [L]$. We shall prove that $\Psi^{\mathcal{A}_\ell} \in \mathcal{F}_{\ell,cgcn}$. Indeed, consider the singular value decomposition (SVD) of $\mathcal{A}_{\ell k}$, which we assume is given by $\mathcal{A}_{\ell k} = \sum_{s\in[F^\star_{\ell-1}]} \lambda_{\ell ks}\mathbf{w}_{\ell ks}\mathbf{u}_{\ell ks}^\top$ for some $F^\star_{\ell-1} < \infty$, where each left-singular vector $\mathbf{w}_{\ell ks}$ and each right-singular vector $\mathbf{u}_{\ell ks}$ are both unit column vectors, while each singular value $\lambda_{\ell ks}$ is a real number. Therefore, the filters corresponding to the $f^{\text{th}}$ feature can be written as:

$$\mathbf{a}^f_{\ell k} = \sum_{s\in[F^\star_{\ell-1}]} \lambda_{\ell ks}[\mathbf{w}_{\ell ks}]_f \cdot \mathbf{u}_{\ell ks}^\top = \sum_{s\in[F^\star_{\ell-1}]} \lambda_{\ell ks} w^f_{\ell ks} \cdot \mathbf{u}_{\ell ks}^\top \tag{32}$$

As such, the value of the $f^{\text{th}}$ feature of node $i$ under $\Psi^{\mathcal{A}_\ell}$ can be rephrased as:

$$[\Psi^{\mathcal{A}_\ell}(\mathcal{X})]_{if} = \sum_{k=0}^{K}\sum_{g=1}^{F_{\ell-1}} [\mathbf{z}^k_i(\mathcal{X}_{\ell-1})]_g [\mathbf{a}^f_{\ell k}]_g = \sum_{k=0}^{K} \left\langle \mathbf{z}^k_i(\mathcal{X}_{\ell-1}), \sum_{s\in[F^\star_{\ell-1}]} \lambda_{\ell ks} w^f_{\ell ks}\mathbf{u}_{\ell ks}^\top \right\rangle \tag{33}$$

Let $\mathbf{v}^k_\ell(\mathbf{z}) \in \mathbb{R}^{nm}$ be a vector whose $(i,j)$-th coordinate equals to $\kappa_\ell(\mathbf{z}, \mathbf{z}^k_i(\mathcal{X}^{(j)}_{\ell-1}))$. Similarly to Appendix E, denote $\tau^f_\ell(\mathbf{z}) := \sum_{k=0}^{K}\langle (Q^k_\ell)^\dagger \mathbf{v}^k_\ell(\mathbf{z}), \sum_{s\in[F^\star_{\ell-1}]} \lambda_{\ell ks} w^f_{\ell ks}\mathbf{u}_{\ell ks}^\top\rangle$. For any matrix $\mathcal{Z} \in \mathbb{R}^{(K+1)\times F_{\ell-1}}$ whose $(k+1)^{\text{th}}$ row is given by $[\mathcal{Z}]_{k+1} := \mathbf{z}_k$ for any $k \in [K]\cup\{0\}$, note that $\tau^f_\ell$ can be rephrased as $\tau^f_\ell(\mathbf{z}) := \sum_{k=0}^{K}\langle \mathbf{v}^k_\ell(\mathbf{z}), ((Q^k_\ell)^\top)^\dagger \sum_{s\in[F^\star_{\ell-1}]} \lambda_{\ell ks} w^f_{\ell ks}\mathbf{u}_{\ell ks}^\top\rangle$. Thus, (22) implies that the Hilbert norm of the function $\tau^f_\ell$ satisfies:

$$\begin{aligned}
\|\tau^f_\ell\|_{\mathcal{H}_\ell} &\le \sum_{k=0}^{K} \left\| (Q^k_\ell)^\top ((Q^k_\ell)^\top)^\dagger \sum_{s\in[F^\star_{\ell-1}]} \lambda_{\ell ks} w^f_{\ell ks}\mathbf{u}_{\ell ks}^\top \right\|_2 \le \sum_{k=0}^{K} \left\| \sum_{s\in[F^\star_{\ell-1}]} \lambda_{\ell ks} w^f_{\ell ks}\mathbf{u}_{\ell ks}^\top \right\|_2 \\
&\le \sum_{k=0}^{K}\sum_{s\in[F^\star_{\ell-1}]} |\lambda_{\ell ks}| \cdot |w^f_{\ell ks}| \cdot \|\mathbf{u}_{\ell ks}^\top\|_2 \\
&\le \sum_{k=0}^{K}\sum_{s\in[F^\star_{\ell-1}]} |\lambda_{\ell ks}| = \sum_{k=0}^{K} \|\mathcal{A}_{\ell k}\|_* \le C_{\sigma,\ell}(R_\ell)(K+1)\sqrt{F_\ell F_{\ell-1}}
\end{aligned} \tag{34}$$

where we used the triangle inequality and the fact that the singular vectors are unit vectors. Since $[\Psi^{\mathcal{A}_\ell}(\mathcal{X})]_{if} = \tau^f_\ell(\mathcal{Z}_{i,\ell}(\mathcal{X}_{\ell-1}))$, we obtained that $\mathcal{F}_{\mathcal{B}_\ell} \subset \mathcal{F}_{\text{cgcn}}$, as desired.

To conclude the proof, we next prove that any empirical risk minimizer $\Phi^{\mathcal{A}}$ in $\mathcal{F}_{\text{cgcn}}$ is also in $\mathcal{F}_{\mathcal{B}_\ell}$. By our proof in Appendix E, the value of the $f^{\text{th}}$ feature of node $i$ at layer $\ell$ is then given by $\tau^f_\ell(\mathcal{Z}_{i,\ell}(\mathcal{X}^{(j)}_{\ell-1}))$, where, as specified in (25), $\tau^f_\ell(\mathcal{Z}) = \sum_{k=0}^{K}\langle (Q^k_\ell)^\dagger \mathbf{v}^k_\ell(\mathbf{z}_k), (Q^k_\ell)^\top \beta^f_{\ell k}\rangle$ is a reparameterization of the graph filters at layer $\ell$ for some vector of the coefficients $\beta^f_{\ell k} \in \mathbb{R}^{nm}$ whose $(i,j)$-th coordinate is $\beta^f_{\ell k,(i,j)}$. That is, $(Q^k_\ell)^\top \beta^f_{\ell k}$ now act

as the filters under this reparameterization. In Appendix E, we have shown that the Hilbert norm of $\tau_\ell^f$ is then $\|\tau_\ell^f\|_{\mathcal{H}_\ell} = \|(Q_\ell^k)^\top \beta_{\ell k}^f\|_2 \leq C_{\sigma,\ell}(R_\ell)$, where the last inequality is by (22). By (26), we further infer that the nuclear norm of the filters $(Q_\ell^k)^\top \beta_{\ell k}^f$ in the new parameterization satisfies $\|(Q_\ell^k)^\top \beta_{\ell k}^f\|_* \leq C_{\sigma,\ell}(R_\ell)\sqrt{F_\ell F_{\ell-1}(K+1)} \leq C_{\sigma,\ell}(R_\ell)(K+1)\sqrt{F_\ell F_{\ell-1}}$, meaning that $\Phi^{\mathcal{A}} \in \mathcal{F}_{\mathcal{B}_\ell}$, as desired. □

Next, we prove that $\mathcal{F}_{L,cgcn}$ is not "too big" by upper bounding its Rademacher complexity. The Rademacher complexity is a key concept in empirical process theory and can be used to bound the generalization error in our empirical risk minimization problem. Readers should refer to (Bartlett and Mendelson, 2002) for a brief overview on Rademacher complexity. The *Rademacher complexity* of a function class $\mathcal{F} = \{\phi : \mathcal{D} \to \mathbb{R}\}$ with respect to $m$ i.i.d samples $\{\mathcal{X}^{(j)}\}_{j=1}^m$ is given by:

$$\mathcal{R}_m(\mathcal{F}) := \mathbb{E}_{\mathcal{X},\varepsilon}\left[\sup_{\phi\in\mathcal{F}} \frac{1}{m}\sum_{j=1}^m \varepsilon_j \phi(\mathcal{X}^{(j)})\right] \tag{35}$$

where $\{\varepsilon_j\}_{j=1}^m$ are an i.i.d. sequence of uniform $\{\pm 1\}$-valued variables. Next, we provide the upper bound on the Rademacher complexity of the function class $\mathcal{F}_{L,cgcn}$ describing the graph filters. The lemma refers to the random kernel matrix $\mathcal{K}_\ell^k(\mathcal{X}) \in \mathbb{R}^{n\times n}$ induced by a graph signal $\mathcal{X} \in \mathbb{R}^{n\times F}$ drawn randomly from the population, i.e., the $(i,i')$-th entry of $\mathcal{K}_\ell^k(\mathcal{X})$ is $\kappa_\ell(\mathbf{z}_i^k(\mathcal{X}), \mathbf{z}_{i'}^k(\mathcal{X}))$. Furthermore, we consider the expectation $\mathbb{E}[||\mathcal{K}_\ell^k(\mathcal{X})||_2]$ of this matrix's spectral norm. While the previous lemmas consider the most general case where the number of features $F_L = G$ at the last layer may be any positive integer, observe that the following result focuses on the binary classification case where $F_L = 1, G = 2$. As mentioned in Section 4.5, Lemma G.3 can be readily extended to the multi-class case by employing a standard one-versus-all reduction to the binary case.

**Lemma G.3.** *In the binary classification case where $F_L = G = 2$, there exists a universal constant $c > 0$ such that:*

$$\mathcal{R}_m(\mathcal{F}_{L,cgcn}) \leq \frac{c\cdot C_{\sigma,\ell}(\mathcal{B}_L)\sqrt{2F_{L-1}(K+1)\cdot\log(n(K+1))\cdot\sum_{k=0}^K \mathbb{E}[||K_L^k(\mathcal{X})||_2]}}{\sqrt{m}} \tag{36}$$

*Proof.* We begin with some preliminaries that apply to the general case and then leverage them in order to obtain the result. For brevity, we denote $R_\ell = C_{\sigma,\ell}(\mathcal{B}_\ell)\sqrt{F_\ell F_{\ell-1}}$. Consider some function $\Phi^{\mathcal{A}} = \phi^{\mathcal{A}_L} \circ \cdots \circ \phi^{\mathcal{A}_1} \in \mathcal{F}_{\text{cgcn}}$. Without loss of generality, we assume that, each input input graph signal $\mathcal{X}^{(j)}$, the feature vector $[\mathcal{X}^{(j)}]_i = \mathbf{x}_i^{(j)}$ of each node $i$ resides in the unit $\ell_2$-ball (which can be obtain via normalization). Combined with $\kappa_\ell$'s being a continuous kernel with $\kappa_\ell(\mathbf{z},\mathbf{z}) \leq 1$, Mercer's theorem (Steinwart, 2008, Theorem 4.49) implies that there exists a feature map $\varphi_\ell : \mathbb{R}^{F_{\ell-1}} \to \ell^2(\mathbb{N})$ for which $\sum_{\xi=1}^\infty [\varphi_\ell(\mathbf{z})]_\xi [\varphi_\ell(\mathbf{z}')]_\xi = \sum_{\xi=1}^\infty \varphi_{\ell\xi}(\mathbf{z})\varphi_{\ell\xi}(\mathbf{z}')$ converges uniformly and absolutely to $\kappa_\ell(\mathbf{z},\mathbf{z}')$. Accordingly, it holds that $\kappa_\ell(\mathbf{z},\mathbf{z}') = \langle\varphi_\ell(\mathbf{z}), \varphi_\ell(\mathbf{z}')\rangle$, which yields the following for each feature $f$ of any node $i$ at layer $\ell$ is $\tau_\ell^f(\mathcal{Z}_{i,\ell}(\mathcal{X}_{\ell-1}))$, where for any matrix $\mathcal{Z} \in \mathbb{R}^{(K+1)\times F_{\ell-1}}$ whose $(k+1)^{\text{th}}$ row is given by $[\mathcal{Z}]_{k+1} := \mathbf{z}_k$ for any $k \in [K]\cup\{0\}$:

$$\tau_\ell^f(\mathcal{Z}) := \sum_{k=0}^K \langle\varphi_\ell(\mathbf{z}_k), \bar{\mathbf{a}}_{\ell k}^f\rangle \tag{37}$$

where $\bar{\mathbf{a}}_{\ell k}^f \in \ell^2(\mathbb{N})$ is a countable-dimensional vector, due to the fact that $\varphi$ itself is a countable sequence of functions. Particularly, the Hilbert norm of $\tau_\ell^f$ is $\|\tau_\ell^f\|_{\mathcal{H}_\ell} = \|\bar{\mathbf{a}}_{\ell k}^f\|_2$. Now, for any graph signal $\mathcal{X}$, let $\Theta_i(\mathcal{X})$ be the linear operator that maps any sequence $\theta \in \ell^2(\mathbb{N})$ to the vector $[\langle\varphi_\ell(\mathbf{z}_i^0(\mathcal{X})),\theta\rangle, \ldots, \langle\varphi_\ell(\mathbf{z}_i^K(\mathcal{X})),\theta\rangle]^\top \in \mathbb{R}^{K+1}$. Thereby, recalling that $\Phi^{\mathcal{A}} = \phi^{\mathcal{A}_L} \circ \cdots \circ \phi^{\mathcal{A}_1}$ where $\phi^{\mathcal{A}_\ell} \in \mathcal{F}_{\ell,cgcn}$ satisfies $[\phi^{\mathcal{A}_\ell}(\mathcal{X}_{\ell-1})]_{if} = \tau_\ell^f(\mathcal{Z}_{i,\ell}(\mathcal{X}_{\ell-1}))$, then the $f^{\text{th}}$ feature of node $i$ at layer $\ell$ can be rephrased as:

$$\begin{aligned}
[\phi^{\mathcal{A}_\ell}(\mathcal{X}_{\ell-1})]_{if} = \mathbf{1}_{K+1}^\top \Theta_i(\mathcal{X}_{\ell-1})\bar{\mathbf{a}}_{\ell k}^f = \text{trace}\left(\Theta_i(\mathcal{X}_{\ell-1})(\bar{\mathbf{a}}_{\ell k}^f \mathbf{1}_{K+1}^\top)\right) \\
\Rightarrow \phi^{\mathcal{A}_\ell}(\mathcal{X}_{\ell-1}) = \left[\text{trace}\left(\Theta_i(\mathcal{X}_{\ell-1})(\bar{\mathbf{a}}_{\ell k}^f \mathbf{1}_{K+1}^\top)\right)\right]_{i\in[n], f\in[F_\ell]}
\end{aligned} \tag{38}$$

where $\mathbf{1}_{K+1}$ is the all-ones vector of dimension $K+1$. Note that the matrix $\bar{\mathcal{A}}_{\ell k} := \bar{\mathbf{a}}_{\ell k}^f \mathbf{1}_{K+1}^\top$ satisfies the following:

$$\|\bar{\mathcal{A}}_{\ell k}\|_* = \|\bar{\mathbf{a}}_{\ell k}^f \mathbf{1}_{K+1}^\top\|_* \leq \|\mathbf{1}_{K+1}^\top\|_2 \cdot \|\bar{\mathbf{a}}_{\ell k}^f\|_2 = \|\tau_\ell^f\|_{\mathcal{H}_\ell}\sqrt{K+1} \leq R_\ell\sqrt{K+1} \tag{39}$$

Now, we obtain our upper bound for the binary classification case where $F_L = G = 2$. Combining (38) with (39), we obtain that the Rademacher complexity of $\mathcal{F}_{L,cgcn}$ is upper bounded as follows:

$$\begin{aligned}\mathcal{R}_m(\mathcal{F}_{L,cgcn}) &= \mathbb{E}\left[\sup_{\phi^{\mathcal{A}_L}\in\mathcal{F}_{L,cgcn}} \frac{1}{m}\sum_{j=1}^{m}\varepsilon_j \Phi^{\mathcal{A}_L}(\mathcal{X}^{(j)})\right] = \\ &= \frac{1}{m}\mathbb{E}\left[\sup_{\|\bar{\mathcal{A}}_{Lk}\|_*\leq R_L\sqrt{K+1}} \operatorname{trace}\left(\Big(\sum_{j=1}^{m}\varepsilon_j\Theta_i(\mathcal{X}_{L-1}^{(j)})\Big)\bar{\mathcal{A}}_{Lk}\right)\right] \\ &\leq \frac{R_L\sqrt{K+1}}{m}\mathbb{E}\left[\Big\|\sum_{j=1}^{m}\varepsilon_j\Theta_i(\mathcal{X}_{L-1}^{(j)})\Big\|_2\right]\end{aligned} \tag{40}$$

where the last equality uses Hölder's inequality while using the duality between the nuclear norm and the spectral norm. Next, note that $\sum_{j=1}^{m}\varepsilon_j\Theta_i(\mathcal{X}_{L-1}^{(j)})$ can be thought of as a matrix with $n$ rows and infinitely many columns. We denote its sub-matrix comprising of the first $r$ columns as $\Theta_i^{(r)}(\mathcal{X}_{L-1}^{(j)})$ and let $\Theta_i^{(-r)}(\mathcal{X}_{L-1}^{(j)})$ be the remaining sub-matrix. Therefore:

$$\mathbb{E}\left[\Big\|\sum_{j=1}^{m}\varepsilon_j\Theta_i(\mathcal{X}_{L-1}^{(j)})\Big\|_2\right] \tag{41a}$$

$$\leq \mathbb{E}\left[\Big\|\sum_{j=1}^{m}\varepsilon_j\Theta_i^{(r)}(\mathcal{X}_{L-1}^{(j)})\Big\|_2\right] + \left(\mathbb{E}\left[\Big\|\sum_{j=1}^{m}\varepsilon_j\Theta_i^{(-r)}(\mathcal{X}_{L-1}^{(j)})\Big\|_F^2\right]\right)^{1/2} \tag{41b}$$

$$\leq \mathbb{E}\left[\Big\|\sum_{j=1}^{m}\varepsilon_j\Theta_i^{(r)}(\mathcal{X}_{L-1}^{(j)})\Big\|_2\right] + \left(nm\mathbb{E}\left[\sum_{\xi=r+1}^{\infty}(\varphi_{\ell\xi}(\mathbf{z}))^2\right]\right)^{1/2} \tag{41c}$$

Since $\sum_{\xi=1}^{\infty}(\varphi_{\ell\xi}(\mathbf{z}))^2$ uniformly converges to $\kappa_\ell(\mathbf{z})$, the second term in (41c) converges to 0 as $r\to\infty$. Hence, it is sufficient to upper bound the first term in (41c) and take the limit $r\to\infty$. By Bernstein inequality due to (Minsker, 2017, Theorem 2.1), whenever $\operatorname{trace}(\Theta_i^{(r)}(\mathcal{X}_{L-1}^{(j)})\Theta_i^{(r)}(\mathcal{X}_{L-1}^{(j)})^\top)\leq C_1$, there is a universal constant $c>0$ such that the expected spectral norm is upper bounded by:

$$\begin{aligned}\mathbb{E}\left[\Big\|\sum_{j=1}^{m}\varepsilon_j\Theta_i^{(r)}(\mathcal{X}_{L-1}^{(j)})\Big\|_2\right] &\leq c\sqrt{\log(mC_1)}\mathbb{E}\left[\left(\sum_{j=1}^{m}\Big\|\Theta_i^{(r)}(\mathcal{X}_{L-1}^{(j)})\Theta_i^{(r)}(\mathcal{X}_{L-1}^{(j)})^\top\Big\|_2\right)^{1/2}\right] \\ &\leq c\sqrt{\log(mC_1)}(m\mathbb{E}[\|\Theta_i(\mathcal{X})\Theta_i(\mathcal{X})^\top\|_2])^{1/2} \\ &\leq c\sqrt{\log(mC_1)}\left(m\sum_{k=0}^{K}\mathbb{E}[||K_L^k(\mathcal{X})||_2]\right)^{1/2}\end{aligned} \tag{42}$$

Notice that, due to the uniform kernel expansion $\kappa_\ell(\mathbf{z},\mathbf{z}') = \sum_{\xi=1}^{\infty}\varphi_{\ell\xi}(\mathbf{z})\varphi_{\ell\xi}(\mathbf{z}')$, it holds that the trace norm is bounded as $\operatorname{trace}(\Theta_i^{(r)}(\mathcal{X}_{L-1}^{(j)})\Theta_i^{(r)}(\mathcal{X}_{L-1}^{(j)})^\top)\leq K+1$ since $\kappa_\ell(\mathbf{z},\mathbf{z}')\leq 1$. Combining this with (42) and (40) implies the desired in (36). □

We are ready to prove the key lemma that will yield our main result. Recall that our current focus is on binary classification (i.e., $F_L = G = 2$). While our prior results consider models with multiple hidden layers, our main result regards the case of a single hidden layer $L = 1$. As mentioned in Section 4.5, in the case of multiple hidden layers, our result applies to each individual layer separately. Using Lemmas G.1–G.3, we are capable of comparing the CGCN predictor obtained by Algorithm 2 against optimal model in the GCN class. Lemma G.2 yields that the predictor $\hat{\Psi}^{\hat{\mathcal{A}}_L}(\cdot)$ generated at the last layer is an empirical risk minimizer in the function class $\mathcal{F}_{\text{cgcn}}$. As such, by the theory of Rademacher complexity (see, e.g., (Bartlett and Mendelson, 2002)), we obtain that:

$$\mathbb{E}_{\mathcal{X},\mathbf{y}}[J(\hat{\Psi}^{\hat{\mathcal{A}}_L}(\mathcal{X}),\mathbf{y})] \leq \inf_{\Phi\in\mathcal{F}_{L,cgcn}}\mathbb{E}_{\mathcal{X},\mathbf{y}}[J(\Phi(\mathcal{X}),\mathbf{y})] + 2M\cdot\mathcal{R}_m(\mathcal{F}_{L,cgcn}) + \frac{c}{\sqrt{m}} \tag{43}$$

where $c$ is a universal constant a and we used the assumption that $J$ is $M$-Lipschitz. By Lemma G.1, we have that $\inf_{\Phi\in\mathcal{F}_{L,cgcn}}\mathbb{E}_{\mathcal{X},\mathbf{y}}[J(\Phi(\mathcal{X}),\mathbf{y})]\leq\inf_{\Phi\in\mathcal{F}_{L,gcn}}\mathbb{E}_{\mathcal{X},\mathbf{y}}[J(\Phi(\mathcal{X}),\mathbf{y})]$. Since $\mathcal{F}_{\text{cgcn}} = \mathcal{F}_{1,cgcn}$ due to $L=1$, this means that $\inf_{\Phi\in\mathcal{F}_{\text{cgcn}}}\mathbb{E}_{\mathcal{X},\mathbf{y}}[J(\Phi(\mathcal{X}),\mathbf{y})]\leq\inf_{\Phi\in\mathcal{F}_{\text{gcn}}}\mathbb{E}_{\mathcal{X},\mathbf{y}}[J(\Phi(\mathcal{X}),\mathbf{y})]$. Combined with (43), we obtain the desired.

**Remark G.4.** *The constant $C_{\sigma,L}(\mathcal{B}_L)$ depends on the convergence rate of the polynomial expansion of the activation function $\sigma$ (See Appendix C). Algorithmically, knowing the activation function is not required to run Algorithm 2. Theoretically, however, the choice of $\sigma$ matters for Theorem 4.7 when comparing the predictor produced by Algorithm 2 against the best GCN, as it affects the representation power of such GCN. If a GCN with an activation function $\sigma$ performs well, CGCN will similarly generalize strongly.*

# H ADDITIONAL EXPERIMENTAL DETAILS

## H.1 Dataset Descriptions

Table 2 provides a summary of the statistics and characteristics of datasets used in this paper. All datasets are accessed using the TUDataset class (Morris et al., 2020) in PyTorch Geometric (Fey and Lenssen, 2019) (MIT License).

Table 2: Statistics of the graph classification datasets used in our experiments.

| Dataset | #Graphs | Avg. Nodes | Avg. Edges | #Classes |
|---|---|---|---|---|
| MUTAG | 118 | 18 | 20 | 2 |
| PTC-MR | 344 | 26 | 51 | 2 |
| PROTEINS | 1,113 | 39 | 73 | 2 |
| NCI1 | 4,110 | 30 | 32 | 2 |
| Mutagenicity | 4,337 | 30 | 31 | 2 |
| `ogbg-molhiv` | 41,127 | 25 | 27 | 2 |

## H.2 Additional Implementation Details

**Hardware and Software.** All experiments could fit on one GPU at a time. Most experiments were run on a server with 4 NVIDIA L40S GPUs. We ran all of our experiments in Python, using the PyTorch framework (Paszke et al., 2019) (license URL). We also make use of PyTorch Geometric (PyG) (Fey and Lenssen, 2019) (MIT License) for experiments with graph data.

**Implementation Details.** Particularly, all models were implemented using PyTorch Geometric (PyG) (Fey and Lenssen, 2019). For GATv2 and GraphGPS, we use the official PyG implementations of their original papers, while for other baselines we use their PyG implementations, as standard in prior works (see, e.g., (Morris et al., 2019; Rampášek et al., 2022)). In a high level, each neural architecture with $L \in \{2, 6\}$ layers consists of an input encoder, stacked GNN layers with edge encoders, batch normalization as well as optional dropout and residual connections, followed by a global pooling step and a final output encoder (See Appendix H.3 for more details). The embedding dimension of each hidden layer is set to 32, while non-convex models use a ReLU activation function between layers. For convex models, we use the Gaussian RBF kernel $\kappa(\mathbf{z}, \mathbf{z}') := \exp(-\gamma\|\mathbf{z} - \mathbf{z}'\|_2^2)$ with $\gamma = 0.2$, and compute an approximate kernel matrix via Nyström approximation (Williams and Seeger, 2001) with dimension $P = 32$ (Recall Section 4.4 in the full paper). Further, since we do not focus on the impact of attention heads on performance, all convex and non-convex versions of GAT, GATv2, GraphTrans and GraphGPS use their default configurations of attention heads.

**Optimization.** All models are trained over 200 epochs with the Adam optimizer (Kingma and Ba, 2015) using the cross entropy loss, a starting learning rate of $10^{-3}$, a minimum learning rate of $10^{-6}$ and a weight decay of $10^{-5}$. We use a ReduceLROnPlateau scheduler, which cuts the learning rate in half with a patience of 20 epochs, and training ends when we reach the minimum learning rate.

## H.3 Detailed Implementation Details of Neural Architectures

**Node Input Encoder.** The first layer of each neural architecture consists of an input encoder implemented using an MLP, which transforms the raw data captured by the graph signal into a suitable representation for the GNN to process. Specifically:

- If raw node features are available, then we an MLP whose input dimension is the number of features per node and its output dimension is the embedding dimension of 32.
- If node features are absent, we use a learnable `DiscreteEncoder` that maps integer node identifiers to vectors to the embedding dimension of 32.

**Edge Feature Encoders.** We construct a separate edge encoder for each GNN layer. Each edge encoder maps raw edge features to the hidden dimension:

- If edge features are provided, then we use a single-layer MLP.
- If edge features are absent, we use a `DiscreteEncoder`.

**GNN Layers.** We use a stack of $L$ GNN layers, each operating over the hidden feature dimension. Each GNN layer $\ell$ takes node features encoded using the input encoder, edge features encoded using the edge encoder corresponding to layer $\ell$, and an edge index matrix. Batch normalization is then applied to the output of the GNN layer, followed by a ReLU activation function. An optional dropout and a residual connection are then applied. Each layer thus has the following general form:

$$\text{GNN Layer} \to \text{BatchNorm} \to \text{ReLU} \to \text{Dropout (Optional)} \to \text{Residual Connection (Optional)}$$

**Global Pooling.** After all GNN layers, we aggregate node features to obtain graph-level representations using `add` pooling to aggregate the learned node-level representations to a graph-level embedding and a two-layer MLP for the final classification.

**Output MLP.** The final graph representation is passed through a two-layer MLP for the final classification. The first layer applies a linear projection followed by optional normalization and a nonlinearity. The second layer outputs logits or regression scores without an activation function.

# I Additional Experimental Results

## I.1 Heatmap of Classification Accuracy

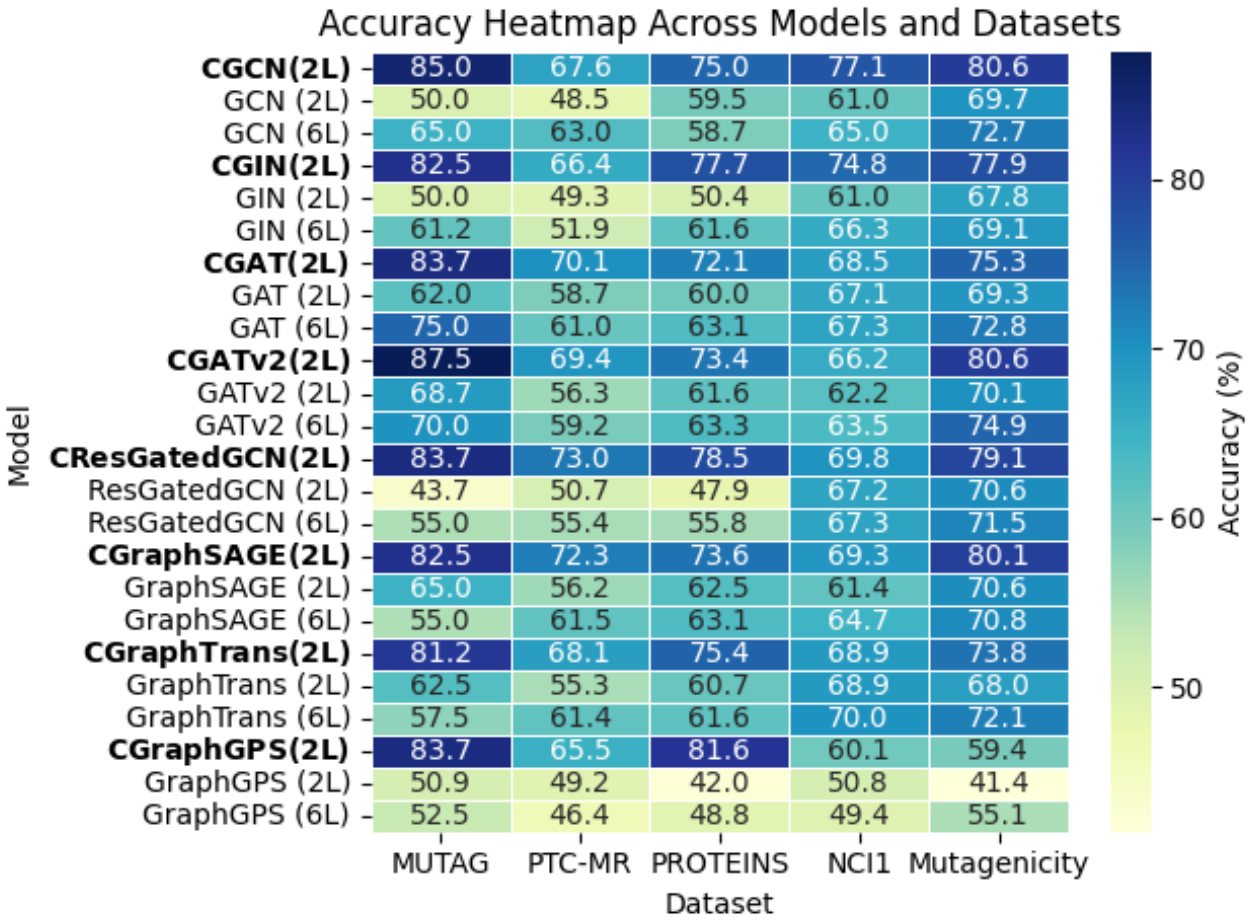


Figure 2: Heatmap of classification accuracy (%) across all models and datasets. Rows correspond to models, columns to datasets. The names of convex models are **bolded** to emphasize their performance.

Figure 2 provides a heatmap summarizing the accuracy of all models across the different datasets. It complements Table 1 in the full paper by visually reinforcing that convex models generally outperform their non-convex counterparts across nearly all datasets, with consistent and major improvements even for models like GATv2 and GraphGPS. This reaffirms the strength and generality of our convexification framework. For example, on MUTAG, the convex CGATv2 model reaches 87.5% accuracy, outperforming the two-layer (68.7%) and six-layer (70.0%) GINs. Similarly, CGraphGPS achieves 83.7%, exceeding the best non-convex variant by over 30%. This trend holds across other datasets like PTC-MR, PROTEINS, and NCI1, where convex versions of CGraphSAGE, CGATv2, and CGraphGPS outperform non-convex baselines by 10–40%.

### I.2 Bar Plots

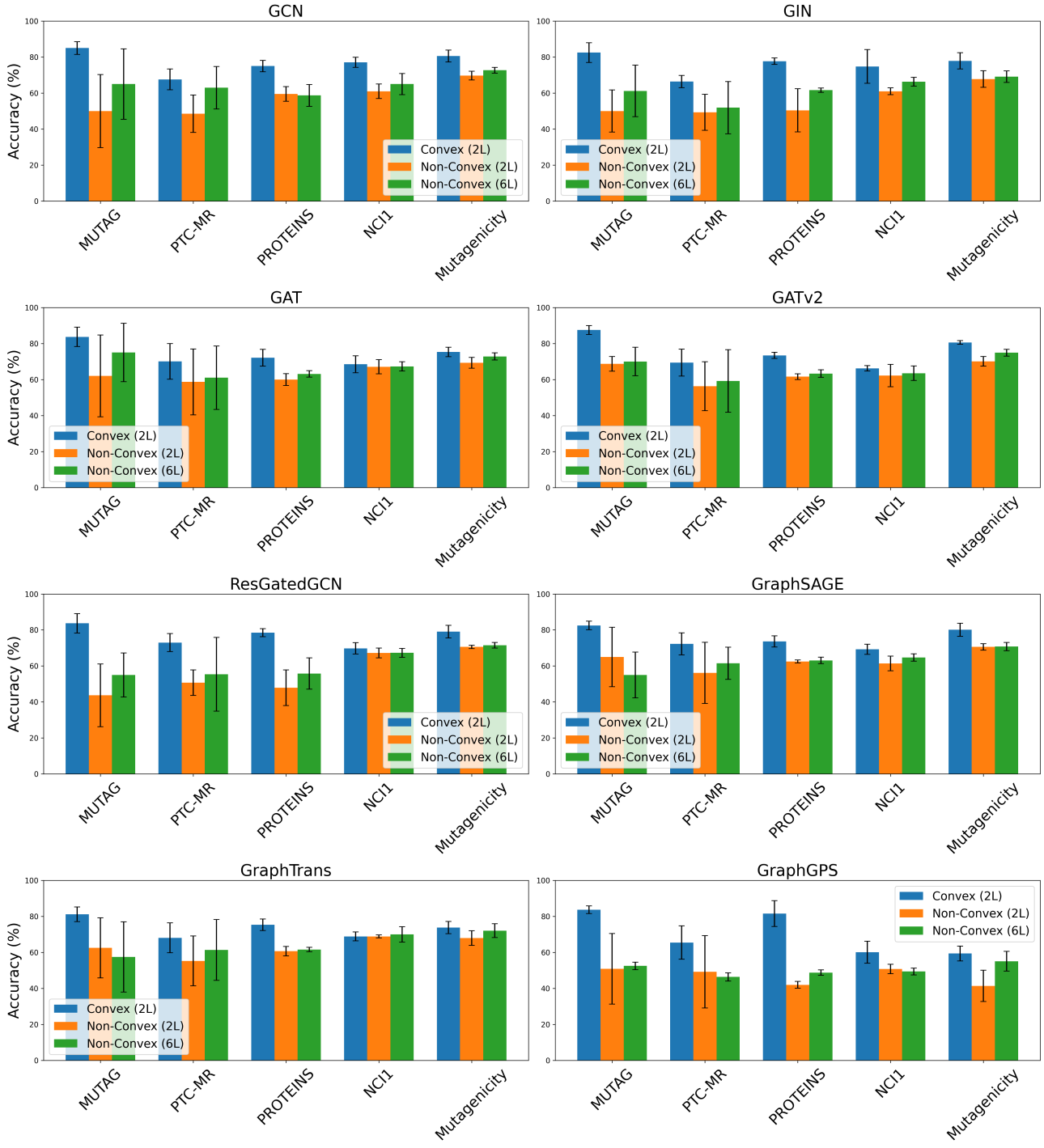


Figure 3: Accuracy comparison of two-layer convex GNNs (prefixed with 'C') against their standard two-layer and six-layer non-convex counterparts across five graph classification datasets. Bars represent the mean accuracy over four runs, and error bars denote the standard deviation. Datasets are ordered from left to right by increasing size.

The bar plots in Figure 3 highlight the consistent performance advantage of convex models over their non-convex counterparts across multiple datasets in a more straightforward way than numerical data alone. They further underscore how convex architectures provide not just dataset-specific improvements, but a general trend

of robustness and high accuracy, even in the absence of large training sets. This emphasizes that our convex models harness stable optimization landscapes, enabling reliable learning where non-convex models exhibit erratic performance due to training instabilities.

### I.3 Radar Plots

The radar plots in Figure 4 offer a holistic view of how convex models perform across diverse datasets. They reveal that convex variants not only perform strongly on small datasets like MUTAG and PTC-MR, but also remain competitive or superior on larger datasets such as NCI1 and Mutagenicity. This multi-dimensional consistency further challenges the notion that expressive representations require deep, non-convex architectures, and instead supports that convex shallow models, when carefully designed, can adapt well across a spectrum of tasks and data scales.

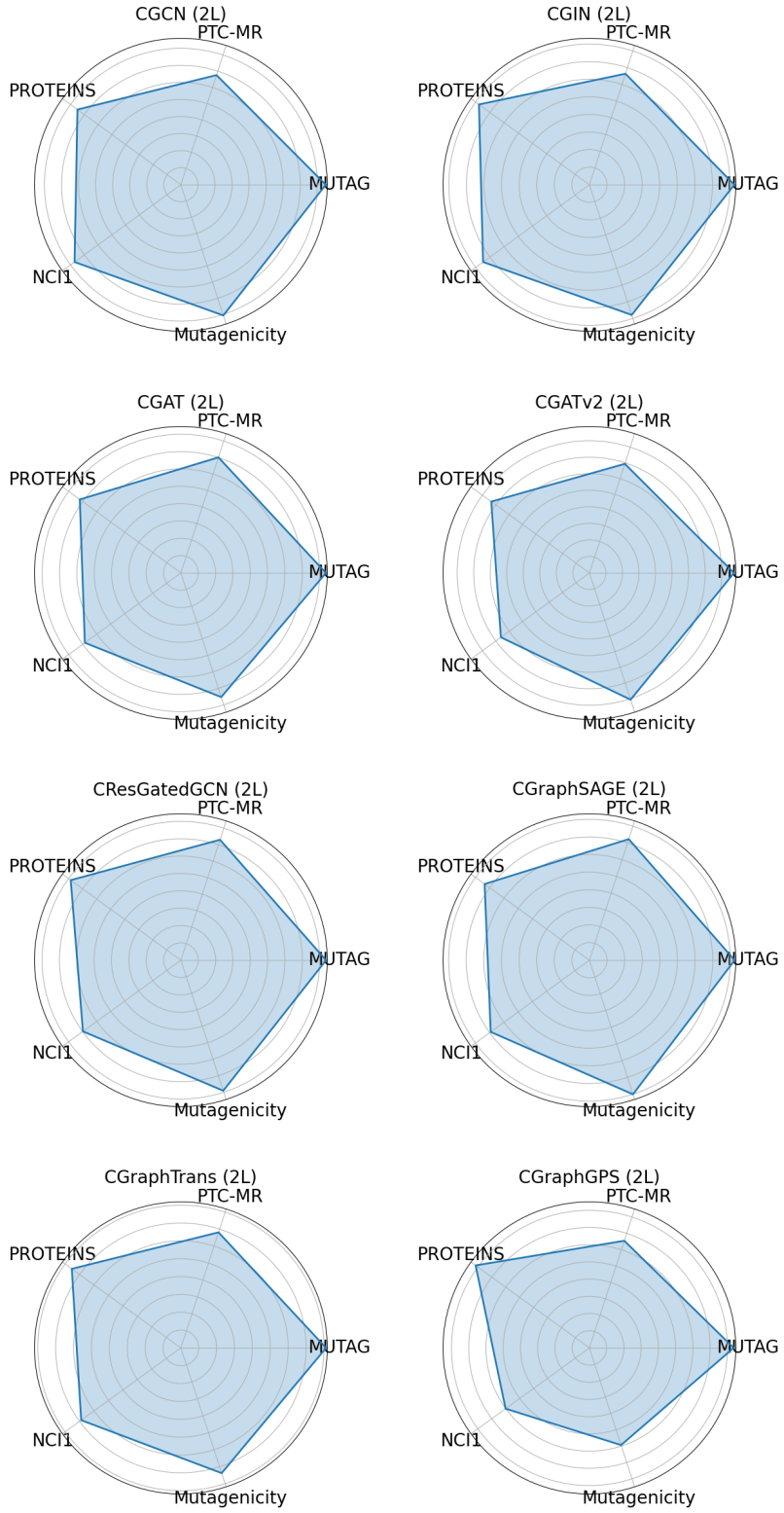


Figure 4: Radar plots showing the performance of each two-layer convex model across five graph classification datasets. Each axis corresponds to a dataset, with higher values indicating better classification accuracy. These plots illustrate the consistency and robustness of convex models across diverse graph types.

### I.4 Experimental Results for Deeper Convex Models

Table 3 presents additional empirical results on Mutagenicity, comparing two-layer and six-layer convex versions of base models against their two-layer and six-layer non-convex counterparts[1]. Our results in Table 3 show that the accuracy of convex models can further improve with additional layers, which is consistent with our theoretical guarantees. Specifically, recall that Theorem 4.7 bounds the generalization error of Algorithm 2 for single-hidden-layer models. As specified in Section 4.5, our analysis extends layer-wise in the multi-layer setting, i.e., it applies to each hidden layer separately. The results in Table 3 therefore illustrate that errors remain controlled as depth increases. This offers strong empirical support for the practical effectiveness of multi-layer CGNNs, which is crucial since state-of-the-art GNN applications typically rely on deeper architectures.

[1] As demonstrated in Section 5, two-layer convex models already exhibit superior performance compared to their two- and six-layer non-convex counterparts when all models incorporate DropOut, batch normalization, node input encoders, and edge feature encoders. To evaluate the performance of deeper GNN models in isolation, in this section we exclude these additional components.

Table 3: Accuracy ($\% \pm \sigma$) on Mutagenicity over 4 runs for two-layer and six-layer convex versions of base models against their two-layer and six-layer non-convex counterparts. Convex variants start with 'C'. A two-layer model is marked by (2L), while a six-layer one is marked by (6L). Results of convex models surpassing their non-convex variants by a large gap (∼10–40%) are in **bold**, modest improvements are underlined, and improvements of six-layer convex models over the corresponding two-layer convex models are in dark green.

| Model | Mutagenicity |
|---|---|
| CGCN (2L) | $54.7 \pm 2.1$ |
| CGCN (6L) | $61.9 \pm 2.7$ |
| GCN (2L) | $54.7 \pm 1.1$ |
| GCN (6L) | $54.7 \pm 1.9$ |
| CGIN (2L) | $56.8 \pm 1.8$ |
| CGIN (6L) | $59.1 \pm 1.8$ |
| GIN (2L) | $54.8 \pm 2.0$ |
| GIN (6L) | $55.8 \pm 2.0$ |
| CGAT (2L) | $55.7 \pm 2.5$ |
| CGAT (6L) | $58.8 \pm 3.6$ |
| GAT (2L) | $54.8 \pm 2.0$ |
| GAT (6L) | $54.8 \pm 2.0$ |
| CGATv2 (2L) | $56.6 \pm 2.7$ |
| CGATv2 (6L) | $62.3 \pm 4.2$ |
| GATv2 (2L) | $55.9 \pm 1.4$ |
| GATv2 (6L) | $59.6 \pm 6.2$ |
| CResGatedGCN (2L) | $56.2 \pm 2.7$ |
| CResGatedGCN (6L) | $59.6 \pm 3.7$ |
| ResGatedGCN (2L) | $56.1 \pm 2.1$ |
| ResGatedGCN (6L) | $56.1 \pm 2.1$ |
| CGraphSAGE (2L) | $55.1 \pm 2.4$ |
| CGraphSAGE (6L) | $66.3 \pm 5.3$ |
| GraphSAGE (2L) | $54.8 \pm 2.0$ |
| GraphSAGE (6L) | $60.0 \pm 3.3$ |
| CGraphTrans (2L) | $55.5 \pm 2.3$ |
| CGraphTrans (6L) | $63.2 \pm 4.7$ |
| GraphTrans (2L) | $54.8 \pm 2.0$ |
| GraphTrans (6L) | $59.9 \pm 4.6$ |
| CGPS (2L) | $57.7 \pm 4.5$ |
| CGPS (6L) | $59.4 \pm 4.1$ |
| GPS (2L) | $41.4 \pm 8.7$ |
| GPS (6L) | $55.1 \pm 5.5$ |

### I.5 Experimental Results for DirGNNs

We herein provide additional empirical results for the recent baseline DirGNN (Rossi et al., 2024), which is a generic wrapper for computing graph convolution on directed graphs. If the graph is originally undirected, applying DirGNN effectively treats it as directed. In our simulations, we extend GCN, GAT and GraphSAGE with DirGNN, obtaining DirGCN, DirGAT and DirGraphSAGE, respectively. In Table 4, we report results on the MUTAG and PROTEINS datasets, where the results are generated as described in Table 1. Particularly, convex architectures are highlighted in bold. Our two-layer convex models consistently outperform their non-convex two- and six-layer counterparts by 10-40% accuracy.

Table 4: Accuracy ($\% \pm \sigma$) on MUTAG and PROTEINS over 4 runs for two-layer convex versions of DirGNNs against their two-layer and six-layer non-convex counterparts. Convex variants start with 'C'. A two-layer model is marked by (2L), while a six-layer one is marked by (6L). Results of convex models surpassing their non-convex variants by a large gap (∼10–40%) are in **bold**, modest improvements are underlined, and improvements of six-layer convex models over the corresponding two-layer convex models are in dark green.

| Model | MUTAG | PROTEINS |
|---|---|---|
| CDirGCN (2L) | **$85.0 \pm 10.6$** | **$72.5 \pm 6.5$** |
| DirGCN (2L) | $50.0 \pm 10.0$ | $45.1 \pm 8.9$ |
| DirGCN (6L) | $57.5 \pm 18.2$ | $47.0 \pm 10.6$ |
| CDirGAT (2L) | **$75.0 \pm 6.1$** | **$73.6 \pm 4.3$** |
| DirGAT (2L) | $53.7 \pm 16.7$ | $59.7 \pm 5.8$ |
| DirGAT (6L) | $61.2 \pm 24.5$ | $61.1 \pm 10.5$ |
| CDirGraphSAGE (2L) | **$81.2 \pm 5.4$** | **$74.3 \pm 3.7$** |
| DirGraphSAGE (2L) | $60.0 \pm 14.5$ | $39.2 \pm 2.2$ |
| DirGraphSAGE (6L) | $41.2 \pm 18.8$ | $51.5 \pm 9.4$ |

### I.6 Why do Convexified Message-Passing GNNs Excel?

Next, we discuss why convexified GNNs perform well is valuable. We herein give a formal support for how each factor specified by the reviewer helps convexified models to excel, especially in small-data regimes.

1. Constraining filters by a nuclear-norm ball limits the hypothesis class and yields favorable generalization gap bounds. Specifically, under the conditions of Theorem 4.7, for each layer $\ell$ with graph filters $\mathcal{A}_\ell$ there exists a constant radius $B_\ell > 0$ such that, if $\|\mathcal{A}_\ell\|_\star \leq B_\ell$, then the generalization gap scales as $\mathcal{O}(\frac{MC_\sigma(B_\ell)}{\sqrt{m}})$ for a training set with $m$ samples and an $M$-Lipschitz continuous loss function, where $C_\sigma(B_\ell)$ depends on the activation function $\sigma$. In contrast, unconstrained nonconvex training may effectively access much larger function classes (higher variance), which hurts generalization in small-data regimes. Our Theorem 4.7 thus indicates that nuclear-norm constraints control generalization for our CGNN class.

2. Nuclear-norm projection acts as spectral smoothing, removing or attenuating directions with small singular values. Projecting the learned filter onto a nuclear-norm ball effectively suppresses small singular-value directions of the filter matrix. In particular, constraining nuclear norms encourages solutions that spread the singular-value mass in a way that uses fewer very large singular values, as nuclear norm sums them up, and penalizes many moderate ones. This eventually encourages low-rank solutions, similarly to trace-norm regularization. On graph-structured data, small singular-value directions often correspond to high-frequency components in the graph spectrum. In graph signal processing terms, low-rank approximations correspond to low-dimensional subspaces which omit high-frequency noise (see, e.g., (Ramakrishna et al., 2020)). Thus, the nuclear-norm projection yields a smoother operator that suppresses noise and prevents overfitting to spurious high-frequency patterns. Namely, this explains the empirical gains we observe in low-data regimes.