RETHINKING LARGE-SCALE GRAPH LEARNING WITH LAPLACIAN KERNEL MACHINES

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph-based semi-supervised learning (SSL) leverages the relational structure among samples to exploit both labeled and unlabeled data. Existing approaches differ in where the graph enters the learning problem. Among them, Laplacian Regularized Kernel Methods (LapKMs), which inject the graph as a regularizer on the predictor, are renowned for rigorous theory, yet have been overlooked for a long time due to the scalability bottleneck. This paper argues that the long-standing stagnation of LapKMs arises from an *optimization* gap rather than a model deficiency: the coupling of the kernel matrix and the graph Laplacian induces prohibitive computation and numerical instability. We propose LAPTOP, a Tractable optimization scheme for large-scale LapKM, which reformulates LapKM via a tailored Alternating Direction Method of Multipliers (ADMM) and decouples it into two well-structured subproblems, i.e., graph Laplacian label smoothing and kernel ridge regression. Together with a memory-efficient block coordinate descent solver, LAPTOP trains *exact* kernel models on million-node graphs using a single consumer-grade GPU, and supports more diverse loss functions than the existing LapKMs. Experiments on six large-scale benchmarks show that LAPTOP ranks within the top three against strong graph neural network (GNN) and graph Transformer baselines on all datasets, with less than half the memory of the strongest baseline. Our results call for a re-examination of LapKMs as a principled complement to deep graph learning.

1 INTRODUCTION

Graph-based semi-supervised learning (SSL) leverages the relational structure among samples to exploit both labeled and unlabeled data, and has gained significant attention in applications such as social network analysis, recommendation systems, and biological network modeling (Wu et al., 2021). As modern benchmarks grow to millions of nodes (Hu et al., 2020), scalability has become the key bottleneck of the field. The dominant response has been an explosion of architectural designs, from sampling-based graph neural networks (Hamilton et al., 2017; Zeng et al., 2020) to scalable graph Transformers (Chen et al., 2023). As the architectures grow ever richer, a contrarian question arises: *what is the minimal machinery required to exploit graph structure at scale?*

To answer this question, we first clarify where the graph can enter the learning pipeline. In brief, existing approaches fall into three paradigms.

- *Label-side*: the graph defines a diffusion dynamics on the labels, and the prediction primarily relies on label propagation alone, without an explicit mapping from node features to labels. This paradigm is thus inherently transductive (Zhu et al., 2003; Zhou et al., 2004).
- *Feature-side*: the graph is embedded into the feature learning process, or more precisely, into each layer of a neural network, enabling end-to-end transductive or inductive prediction. This path is popularized by Kipf & Welling (2017) as graph neural networks (GNNs) today.
- *Regularizer-side*: the graph regularizes the predictor's smoothness over the manifold (Belkin et al., 2006). Laplacian regularized kernel models (LapKMs) build upon this paradigm with kernel methods, and are natively inductive.

The three paradigms also differ in how much machinery they attach to the graph, from no predictor at all to a graph-coupled deep architecture (Figure 1). However, the three paradigms have not developed

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

equally. GNNs dominate the current literature, and the label-propagation research has also been revitalized recently (Calder et al., 2020; Huang et al., 2021). In contrast, LapKMs have remained scarcely developed. Despite the grounded theory (Belkin et al., 2006), they are still restricted to small-scale datasets (Li et al., 2019), and their behavior on modern large-scale benchmarks remains unknown. *Is this stagnation a fundamental inferiority of the model?*

In this paper, we answer this question by showing that the bottleneck is computational rather than statistical. We identify two challenges, which rooted in the *coupling* of the kernel matrix $\boldsymbol{K}$ and the graph Laplacian $\boldsymbol{L}$ in the LapKM objective: (i) the coupled system requires $O(n^3)$ operations and $O(n^2)$ storage, and (ii) The ill-conditioned nature of the coupled system makes the inversion numerically unstable. Moreover, the conventional remedy of low-rank Nyström approximation (Li et al., 2019) forfeits most of the graph gain (Section 4), indicating that the coupled problem must be solved exactly rather than approximated. In other words, the issue of LapKM is an optimization gap. Interestingly, the answer is as minimal as the question demands. We propose LAPTOP, a Tractable optimization framework for large-scale LapKM. Our contributions are as follows:

Framework: The optimization gap is ideentified, and a tailored Alternating Direction Method of Multipliers (ADMM) reformulation decouples the objective into two well-structured subproblems, a graph Laplacian label smoothing step and a kernel ridge regression (KRR) step, both solved by memory-efficient solvers. The design trains exact kernel models with $O(n+b^2)$ memory and supports diverse convex losses beyond the squared and hinge losses.

Benchmark: We enable, to our knowledge, the first benchmark of LapKMs on modern large-scale graph datasets, where LAPTOP matches representative GNNs on a single consumer-grade GPU. These results validate LapKMs as an effective tool for modern graph learning.

Implication: With the optimization gap closed, the theoretically grounded regularizer-side paradigm finally becomes practical at scale. It completes the landscape of graph learning and reconnects classical regularization theory with modern large-scale applications.

2 PRELIMINARY

In this section, we present the frequent notations, the related work of graph-based SSL, and the problem settings of LapKM.

2.1 NOTATIONS

Let the sample space and the label space be $\mathcal{X}$ and $\mathcal{Y}$, respectively. Denote $\mathcal{K}(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ a positive definite kernel function. Let $\mathcal{H}$ be the reproducing kernel Hilbert space (RKHS) induced by $\mathcal{K}$, with endowed inner product $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ and norm $\|f\|_{\mathcal{H}} := \langle f, f\rangle_{\mathcal{H}}^{1/2}$. Then we have $f(\boldsymbol{x}) = \langle f, \mathcal{K}(\boldsymbol{x},\cdot)\rangle_{\mathcal{H}}$ for any $f \in \mathcal{H}$ and $\boldsymbol{x} \in \mathcal{X}$ (Schölkopf & Smola, 2002). We use the notation $[n] := \{1, 2, \cdots, n\}$. Let $\mathcal{I} \subseteq [n]$ be an index and $\boldsymbol{a} \in \mathbb{R}^n$ be a vector, we denote $\boldsymbol{a}_{\mathcal{I}}$ the subvector of $\boldsymbol{a}$ indexed by $\mathcal{I}$. For a matrix $\boldsymbol{A}$, similarly, we use $\boldsymbol{A}_{\mathcal{I},\mathcal{J}}$ to indicate the submatrix indexed by $\mathcal{I}$ and $\mathcal{J}$, and $\boldsymbol{A}_{\mathcal{I},:} := \boldsymbol{A}_{\mathcal{I},[n]}$.

2.2 RELATED WORK

2.2.1 GRAPH-BASED SSL

In graph-based SSL, data points are connected via a graph structure. We focus on the common homophily assumption, i.e., connected nodes are likely to share similar labels. Existing approaches can be organized into three paradigms according to the position of graph in the learning pipeline.

Label-side. Early work propagated labels from labeled to unlabeled nodes via the graph Laplacian (Zhu et al., 2003; Zhou et al., 2004), where the solution is the stationary state of a label diffusion dynamics on the graph. This line has been revitalized recently: Poisson learning (Calder et al., 2020; Zhou et al., 2024) improve the low-label-rate and extremely sparse labels regime, while Huang et al. (2021) show that label propagation on top of simple base predictors can outperform GNNs. However, these methods are inherently transductive: the prediction is a vector on the observed nodes, and unseen nodes require re-running the propagation.

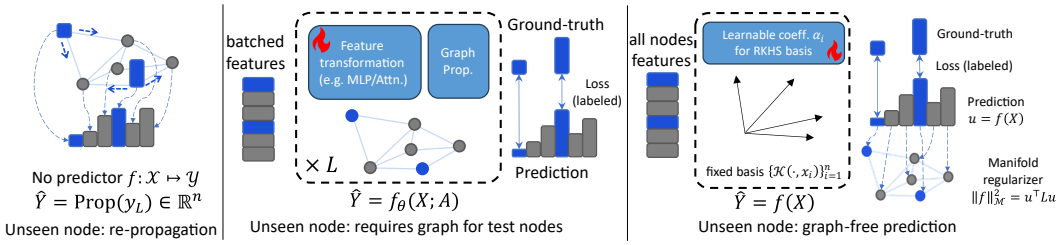


Figure 1: Three paradigms of graph-based SSL. Left: label-side methods (e.g., Laplacian Learning), middle: feature-side methods (e.g., GNNs), right: regularizer-side methods (e.g., LapKMs).

Feature-side. GNNs embed the graph convolution operator into deep neural networks (Kipf & Welling, 2017) and have become the dominant approach, with significant advancements on over-smoothing (Liu et al., 2020; He et al., 2021), large-graph sampling (Hamilton et al., 2017; Zeng et al., 2020), and graph Transformers (Ying et al., 2021; Wu et al., 2022; Chen et al., 2023) with linear-complexity variants scaling to million-node graphs (Wu et al., 2025; Deng et al., 2024). Decoupled GNNs separate the graph propagation from the feature transformation layers and also reach promising performance (Wu et al., 2019; Gasteiger et al., 2019), and carefully tuned classic GNNs remain competitive with graph Transformers on standard benchmarks (Luo et al., 2024). The optimization perspective also provides a new way to design novel graph operator for GNN layers (Zhu et al., 2021; Yang et al., 2021; 2025). In these designs, the training objective remains the task loss on labeled nodes, and the graph serves as an operator on intermediate features. Thus, the inductive inference still relies on graph connections of newly observed nodes.

Regularizer-side. Belkin et al. (2006) proposed LapKMs by incorporating graph geometry into kernel models via manifold regularization, which encourages the predictor to be smooth over the graph. Then, the predictor can be expressed as a pure kernel expansion over the training samples by Representer Theorem (see (2)). This enables convex optimization and *graph-free* inductive prediction on unseen samples, which is LapKMs' key difference from GNNs. However, the involved matrix inverses and quadratic programming are costly on large-scale graphs (Zhang et al., 2016). Although (Melacci & Belkin, 2011) solves primal LapSVM with Newton's method, it still relies on expensive preconditioned CG. Li et al. (2019) scales up LapRLS via Nyström approximation and preconditioned CG, while suffering from the memory bottleneck of storing the preconditioner. In this paper, we address the memory and computation issues of LapKMs to enable large-scale graph-based SSL, with a unified optimization framework for diverse loss functions beyond LapSVM and LapRLS.

2.2.2 Large-scale kernel methods

Kernel methods are powerful while the $O(n^3)$ computation becomes unaffordable on large-scale datasets. Nyström method (Williams & Seeger, 2000) and Random Fourier feature (Rahimi & Recht, 2007) approximate the kernel matrix to reduce the cost. EigenPro (Ma & Belkin, 2017; 2019) accelerates kernel SGD by preconditioning, and FALKON (Rudi et al., 2017) solves KRR efficiently via Nyström approximation and preconditioned CG, while requiring $O(s^2)$ storage for s Nyström centers, which limits large s on memory-constrained devices. To alleviate this, Abedsoltan et al. (2023; 2025) leveraged projection gradient descent to reduce the memory burden. On the other hand, Rathore et al. (2026); Zhang & Lai (2025) proposed to use the memory-efficient BCD solver for kernel methods. However, LapKM involves a manifold regularization term beyond standard kernel methods, which brings additional computation and numerical challenges as discussed in Section 3.1. In this paper, we present a concise ADMM framework that elegantly resolves these challenges.

2.3 Laplacian regularized kernel models (LapKM)

Consider a partially labeled training dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^l \cup \{\boldsymbol{x}_i\}_{i=l+1}^n$, where $\boldsymbol{x}_i \in \mathcal{X}$ is the i-th sample $(i \in [n])$, and $y_i \in \mathcal{Y}$ is the label for $\boldsymbol{x}_i, i \in [l]$. Denote $\boldsymbol{A} \in \{0,1\}^{n \times n}$ the adjacency matrix illustrating the undirected graph relationship of data, where $A_{ij} = 1$ indicates the edge connection between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, and $\boldsymbol{D}$ the degree matrix with $D_{ii} = \sum_{j=1}^n A_{ij}$. Let $\xi_y(\cdot)$ be a convex loss with $y \in \mathcal{Y}$ as ground-truth, and $f(\cdot)$ be the predictor to be learned. LapKM incorporates the graph

structure into the learning process by a manifold regularization encouraging smoothness of the learned function over the graph. The optimization problem for LapKM is:

$$\min_{f\in\mathcal{H}} \frac{1}{2}\|f\|_{\mathcal{H}}^2 + \frac{c_M}{2}\|f\|_{\mathcal{M}}^2 + c_L \sum_{i=1}^{l} \xi_{y_i}(f(\boldsymbol{x}_i)), \quad \|f\|_{\mathcal{M}}^2 := \frac{1}{2}\sum_{i,j}\left(\frac{f(\boldsymbol{x}_i)}{\sqrt{D_{ii}}} - \frac{f(\boldsymbol{x}_j)}{\sqrt{D_{jj}}}\right)^2 A_{ij}. \quad (1)$$

Here, $\|f\|_{\mathcal{M}}^2$ is the manifold regularization term indicating the smoothness of f on the graph and is defined as the normalized Dirichlet energy that penalizes the discrepancies of predictions between connected nodes. In this case, $\|f\|_{\mathcal{M}}^2 = \boldsymbol{u}^\top \boldsymbol{L}\boldsymbol{u}$ with $\boldsymbol{u} := [f(\boldsymbol{x}_1), \cdots, f(\boldsymbol{x}_n)]^\top$, and $\boldsymbol{L} := \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ being the normalized Laplacian (Belkin et al., 2006). Using Representer Theorem (Belkin et al., 2006), the optimal predictor is expressed as the linear combination of $\{\mathcal{K}(\boldsymbol{x}_i, \cdot)\}_{i=1}^n$, and thus (1) is equivalently reduced to an n-dimensional optimization problem. Specifically, the desired predictor is expressed as:

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i^\star \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}), \quad \text{where} \quad \boldsymbol{\alpha}^\star = \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^n} \frac{1}{2}\boldsymbol{\alpha}^\top \boldsymbol{K}\boldsymbol{\alpha} + \frac{c_M}{2}\boldsymbol{\alpha}^\top \boldsymbol{K}\boldsymbol{L}\boldsymbol{K}\boldsymbol{\alpha} + \xi_{\boldsymbol{y}}(\boldsymbol{K}\boldsymbol{\alpha}), \quad (2)$$

where $\xi_{\boldsymbol{y}}(\boldsymbol{u}) := c_L \sum_{i=1}^{l} \xi_{y_i}(u_i)$, and $\boldsymbol{K} \in \mathbb{R}^{n\times n}$ is the kernel matrix with $K_{ij} = \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)$. LapRLS and LapSVM are two well-known LapKMs in (Belkin et al., 2006).

Example 2.1. *LapRLS applies the squared loss $\xi_y(u) = (u-y)^2/2$ yielding the optimal solution:*

$$\boldsymbol{\alpha}^\star = c_L \left(\boldsymbol{I} + c_L \boldsymbol{J}\boldsymbol{K} + c_M \boldsymbol{L}\boldsymbol{K}\right)^{-1} \boldsymbol{J}\boldsymbol{y}. \quad (3)$$

where $\boldsymbol{J} \in \mathbb{R}^{n\times n}$ is a diagonal matrix with $J_{ii} = 1$ if $\boldsymbol{x}_i$ is labeled and $J_{ii} = 0$ otherwise, and $\boldsymbol{y} \in \mathbb{R}^n$ is zero-padded on the unlabeled samples.

Example 2.2. *LapSVM corresponds to the hinge loss $\xi_y(u) = \max\{0, 1-yu\}$, where $y \in \{-1, 1\}$. The solution is given by:*

$$\boldsymbol{\alpha}^\star = \boldsymbol{Q}\boldsymbol{J}\boldsymbol{\beta}^\star, \ \ \boldsymbol{\beta}^\star = \operatorname*{argmin}_{0\le y_i\beta_i\le c_L, i\in[l]} \frac{1}{2}\boldsymbol{\beta}^\top \boldsymbol{J}\boldsymbol{K}\boldsymbol{Q}\boldsymbol{J}\boldsymbol{\beta} - \boldsymbol{\beta}^\top \boldsymbol{y}, \quad \boldsymbol{Q} = \left(\boldsymbol{I} + c_M \boldsymbol{L}\boldsymbol{K}\right)^{-1}, \quad (4)$$

where $\boldsymbol{J}$ and $\boldsymbol{y}$ follow the definitions in Example 2.1.

3 LapTop

In this section, we first identify the challenges of applying LapKM to large-scale datasets, which motivates the proposed LapTop framework and the ADMM optimization. Then, we present the efficient solvers of the two key subproblems in LapTop, and analyze its computational costs.

3.1 Motivation: why has LapKM fallen behind?

LapRLS and LapSVM are the two classical LapKMs. However, their applications are limited in small datasets (Li et al., 2019). We argue that this stagnation stems from an *optimization* gap rather than a model deficiency. Specifically, we identify two critical reasons why LapKM fails to scale up.

Expensive computation. The inverses in (3) and (4) cost $O(n^3)$ operations and thus are not suitable for the large-scale dataset. More importantly, unlike the vanilla kernel methods that only require computing $\boldsymbol{K}\boldsymbol{v}$ on-the-fly, (4) requires $O(n^2)$ space to store $\boldsymbol{Q}$ to avoid repeated computing the inverse, which becomes expensive when n is large.

Numerical instability. The kernel matrix $\boldsymbol{K}$ typically exhibits a rapid spectral decay, e.g., $\sigma_k = O(k^{-d/2})$ for NTK on the d-dimensional sphere (Geifman et al., 2020) and $\sigma_k = O(\exp(-ck^{1/d}))$ for the Gaussian kernel (Minh et al., 2006), leading to an extremely high condition number $\mathsf{cond}(\boldsymbol{K}) := \sigma_1/\sigma_n$. The ill-conditioning is inherited and often aggravated by $\boldsymbol{L}\boldsymbol{K}$, making the inverses in (3) and (4) numerically unstable for large n. In addition, $\boldsymbol{L}\boldsymbol{K}$ is not symmetric positive definite, so efficient solvers designed for definite systems (e.g., CG) may be inapplicable, and merely adding identity regularization may be insufficient to stabilize the inversion.

Previously, Li et al. (2019) scales up LapRLS via Nyström approximation and preconditioned CG, obtaining remarkable results on datasets with n up to 10^5. However, its preconditioner requires $O(s^2)$

space with s the number of Nyström landmarks, which becomes a bottleneck to increase s for a good approximation (Abedsoltan et al., 2023; Zhang & Lai, 2025). More importantly, restricting the solution to the span of $s \ll n$ landmarks forfeits most of the graph gain (Section 4): the graph regularizer takes effect only with the full representer expansion, not a low-rank surrogate of it.

In essence, the reason behind the above challenges is the coupling of graph Laplacian $\boldsymbol{L}$ and kernel matrix $\boldsymbol{K}$. Hence, the key to scaling up LapKM is to decouple the two components during optimization. **First, to ensure numerical stability,** we leverage a compact ADMM reformulation strategy, thus solving two well-structured subproblems separately. As shown in the following sections, this strategy further allows for diverse loss functions beyond LapRLS and LapSVM. **Second, to mitigate expensive computation,** we propose Block Coordinate Descent (BCD) solver for the kernel subproblem, enabling memory-efficient kernel solvers without relying on approximation, thus fully unleashing the power of kernel methods on large-scale datasets.

3.2 Optimization: an ADMM framework

ADMM (Boyd et al., 2011) is a standard solver to split a difficult optimization problem into easier subproblems. Despite its simplicity, it is an elegant solution to alleviate the computational and numerical challenges in LapKM. First, with an auxiliary variable, we reformulate problem (2) as:

$$\min_{\boldsymbol{\alpha},\boldsymbol{u}} \frac{1}{2}\boldsymbol{\alpha}^\top \boldsymbol{K}\boldsymbol{\alpha} + \frac{c_M}{2}\boldsymbol{u}^\top \boldsymbol{L}\boldsymbol{u} + \xi_{\boldsymbol{y}}(\boldsymbol{u}), \;\; \textit{s.t.} \;\; \boldsymbol{u} = \boldsymbol{K}\boldsymbol{\alpha}, \tag{5}$$

Then the augmented Lagrangian function is defined as:

$$\mathsf{L}_\rho(\boldsymbol{\alpha}, \boldsymbol{u}, \boldsymbol{\lambda}) := \frac{1}{2}\boldsymbol{\alpha}^\top \boldsymbol{K}\boldsymbol{\alpha} + \frac{c_M}{2}\boldsymbol{u}^\top \boldsymbol{L}\boldsymbol{u} + \xi_{\boldsymbol{y}}(\boldsymbol{u}) + \boldsymbol{\lambda}^\top(\boldsymbol{u} - \boldsymbol{K}\boldsymbol{\alpha}) + \frac{\rho}{2}\|\boldsymbol{u} - \boldsymbol{K}\boldsymbol{\alpha}\|^2.$$

A standard ADMM generates a sequence $\{(\boldsymbol{u}_k, \boldsymbol{\alpha}_k, \boldsymbol{\lambda}_k)\}$ via the following iteration:

$$\boldsymbol{u}_{k+1} = \operatorname*{argmin}_{\boldsymbol{u}} \mathsf{L}_\rho(\boldsymbol{\alpha}_k, \boldsymbol{u}, \boldsymbol{\lambda}_k), \tag{6a}$$

$$\boldsymbol{\alpha}_{k+1} = \operatorname*{argmin}_{\boldsymbol{\alpha}} \mathsf{L}_\rho(\boldsymbol{\alpha}, \boldsymbol{u}_{k+1}, \boldsymbol{\lambda}_k), \tag{6b}$$

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \rho(\boldsymbol{u}_{k+1} - \boldsymbol{K}\boldsymbol{\alpha}_{k+1}). \tag{6c}$$

The optimization in LapKM is decoupled into two easier subproblems (6a) and (6b), which can be formulated as **graph Laplacian label smoothing** and **KRR**, respectively, as shown in the following analysis. Therefore, LapTop can be seen as performing label smoothing and model fitting alternately. In the sequel, we elucidate on these optimization steps.

3.2.1 $\boldsymbol{u}$-step: Graph Laplacian Label smoothing

In this step, we optimize (6a), i.e.,

$$\min_{\boldsymbol{u}} \frac{c_M}{2}\boldsymbol{u}^\top \boldsymbol{L}\boldsymbol{u} + \xi_{\boldsymbol{y}}(\boldsymbol{u}) + \frac{\rho}{2}\left\|\boldsymbol{u} - \boldsymbol{K}\boldsymbol{\alpha}^k + \frac{\boldsymbol{\lambda}^k}{\rho}\right\|^2, \tag{7}$$

Intuitively, $\boldsymbol{u}$ can be interpreted as the smoothed labels of all data points, where the labeled parts tend to fit the observed labels via minimizing $\xi_{\boldsymbol{y}}(\boldsymbol{u})$, and the unlabeled parts are refined via minimizing the manifold regularization. Moreover, the loss function is decoupled from the model parameters $\boldsymbol{\alpha}$ in this step, which enables us to handle various loss functions flexibly.

Remark 3.1 (Label propagation as the first iterate of LapTop). The $\boldsymbol{u}$-step recovers classical label propagation as its first iterate. Specifically, initializing $\boldsymbol{\alpha}_0 = \mathbf{0}$, $\boldsymbol{\lambda}_0 = \mathbf{0}$ and adopting the squared loss, (7) becomes a quadratic problem whose normal equation reads $(c_M\boldsymbol{L} + c_L\boldsymbol{J} + \rho\boldsymbol{I})\,\boldsymbol{u} = c_L\boldsymbol{J}\boldsymbol{y}$, which coincides with the stationary system of label propagation (Zhou et al., 2004) up to the ridge term $\rho\boldsymbol{I}$, and Poisson learning (Calder et al., 2020) corresponds to the same system with a reweighted source term. Hence, label propagation can be regarded as the first iteration of LapTop, and the subsequent ADMM iterations close the loop between the graph-smoothed labels and the kernel predictor.

In sharp contrast to the coupled operator $\boldsymbol{L}\boldsymbol{K}$ in (3), the Hessian of (7) is $\rho\boldsymbol{I} + c_M\boldsymbol{L} + \nabla^2\xi_{\boldsymbol{y}}(\boldsymbol{u})$, which is sparse, symmetric positive definite, and well-conditioned. We therefore solve (7) by two off-the-shelf schemes chosen according to the loss property: for twice-differentiable losses (e.g.,

squared and Logistic losses), an inexact Newton method whose Newton system is solved by truncated CG with a ratio-based stopping rule (Eisenstat & Walker, 1996), where the sparsity of $\boldsymbol{L}$ and the diagonality of $\nabla^2 \xi_{\boldsymbol{y}}$ yield cheap matrix-vector products; for prox-friendly losses (e.g., hinge and squared hinge losses), Accelerated Proximal Gradient Descent (APGD) with element-wise closed-form proximal operators. Both schemes cost $O(n+E)$ per iteration with E the number of edges. Note that APGD and Newton are classic and standard solvers, so we leave the details to Appendix B.

3.3 α-STEP: KRR SUBPROBLEM

Let $\boldsymbol{q}_k := \boldsymbol{u}_{k+1} + \boldsymbol{\lambda}_k/\rho$, and with some algebra, (6b) can be rewritten as:

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^\top \boldsymbol{K}\boldsymbol{\alpha} + \frac{\rho}{2}\left\|\boldsymbol{K}\boldsymbol{\alpha} - \boldsymbol{q}_k\right\|^2, \tag{8}$$

It can be seen that (8) resembles the KRR problem (Schölkopf & Smola, 2002) with labels $\boldsymbol{q}_k$ and regularization level ρ^{-1}. We can write its first-order optimality condition as:

$$(\boldsymbol{K} + \boldsymbol{I}/\rho)\boldsymbol{\alpha}_{k+1} = \boldsymbol{q}_k. \tag{9}$$

For convenience, the discussion below omits the subscription k. Unlike (7) with sparse Hessian, (9) is usually dense and more ill-conditioned. Thus, CG becomes inefficient from two aspects. *Computationally*, each CG iteration requires $O(n^2)$ operations, which is prohibitive for large n. *Numerically*, preconditioning is necessary to ensure CG's convergence, while it requires extra computational and memory overhead (Frangella et al., 2023; Epperly et al., 2025).

Block coordinate descent (BCD) has recently emerged as a memory-efficient solver for large-scale kernel methods (Rathore et al., 2026; Zhang & Lai, 2025), which enjoys cheap per-iteration cost and exploits modern parallel computing hardware. Let b be the block size. The full index set is partitioned into disjoint blocks of size b: $[n] = \bigcup_{j=1}^{\lceil n/b \rceil} \mathcal{B}_j$, In each iteration, we pick a block $\mathcal{B} \in \{\mathcal{B}_j\}_{j=1}^{\lceil n/b \rceil}$ and update the coordinates $\boldsymbol{\alpha}_{\mathcal{B}}$ while fixing others. Let the current iterate $\boldsymbol{\alpha}_{\mathcal{B}}$ be updated to $\boldsymbol{\alpha}_{\mathcal{B}} + \boldsymbol{s}^\star$, where $\boldsymbol{s}^\star$ is the exact block minimizer given by the block normal equation $(\boldsymbol{K}_{\mathcal{B},\mathcal{B}} + \boldsymbol{I}/\rho)\boldsymbol{s} = \boldsymbol{e}_{\mathcal{B}}$ with the block residual $\boldsymbol{e}_{\mathcal{B}} := \boldsymbol{q}_{\mathcal{B}} - \boldsymbol{K}_{\mathcal{B},:}\boldsymbol{\alpha} - \boldsymbol{\alpha}_{\mathcal{B}}/\rho$. The block update thus reads:

$$\boldsymbol{\alpha}_{\mathcal{B}} \leftarrow \boldsymbol{\alpha}_{\mathcal{B}} + (\boldsymbol{K}_{\mathcal{B},\mathcal{B}} + \boldsymbol{I}/\rho)^{-1}\boldsymbol{e}_{\mathcal{B}}. \tag{10}$$

In this way, we transform the large-scale problem (9) into a series of small-scale linear systems with size b. To further avoid $O(b^3)$ costs of matrix inverse in (10), we apply truncated CG to solve this block system inexactly. *Computationally*, the complexity is reduced to $O(b^2)$, and truncation avoids over-solving. *Numerically*, $\boldsymbol{K}_{\mathcal{B},\mathcal{B}}$ is better conditioned than $\boldsymbol{K}$ due to its smaller size, thus ensuring TCG's fast convergence without preconditioning. See Appendix B.1 for more details.

Note that Rathore et al. (2026) also leverages BCD to solve KRR (8). They apply Nyström method to approximately solve the block-wise linear system (10) and adaptively tuned step size, which yields additional computation. In contrast, our TCG solver is more straightforward and can also ensure proper step size via truncation, making the proposed BCD more efficient and easier to implement.

3.4 COMPUTATIONAL COMPLEXITY ANALYSIS

The convergence of the ADMM (6) is guaranteed under mild conditions (Lin et al., 2022). In practice, LAPTOP operates as an inexact ADMM scheme to balance the accuracy and efficiency. We discuss the computation complexity of subproblems in detail as below.

There are two scenarios in $\boldsymbol{u}$-step. The Newton and APGD schemes cost $O(T_{\mathsf{NT}} T_{\mathsf{CG}}(n+E))$ and $O(T_{\mathsf{APGD}}(n+E))$, respectively, where $T_{\mathsf{NT}}, T_{\mathsf{APGD}}, T_{\mathsf{CG}}$ are the number of iteration of Newton, APGD and the inner truncated CG solver, and E is the number of edges in the graph. T_{CG} is typically small due to the truncation strategy. The strong convexity of (7) ensures linear convergence of both solvers: the inexact Newton method converges super-linearly, with the condition number affecting only the inner TCG iterations (Galli & Lin, 2022), while APGD attains the accelerated rate $O(\sqrt{c_M/\rho}\log(1/\epsilon))$ when paired with an adaptive restart or a condition-number-dependent constant momentum. Generally, $T_{\mathsf{NT}} \ll T_{\mathsf{APGD}}$ for the same precision level. Nevertheless, the second-order information is not mandatory for the APGD, making it more versatile than Newton's method.

For the $\boldsymbol{\alpha}$-step, the space complexity of the $\boldsymbol{\alpha}$-step is $O(b^2)$ due to the storage of $K_{\mathcal{B},\mathcal{B}}$, significantly lower than $O(n^2)$ space required by conventional LapKM. Note that BCD is applied to the equivalent

Table 1: Large-scale node classification benchmark results. The top three results are highlighted with the **first**, **second**, and third place, respectively. The timeout is 24 hours. Laplacian and Poisson learning have no random seeds and thus no standard deviation.

Methods		Flickr $n \approx 0.09$M	Proteins $n \approx 0.13$M	Arxiv $n \approx 0.17$M	Reddit $n \approx 0.23$M	Yelp $n \approx 0.72$M	Products $n \approx 2.45$M
		ACC (%) ↑	AUC (%) ↑	ACC (%) ↑	ACC (%) ↑	F1 (%) ↑	ACC (%) ↑
Label Propagation	Laplacian	47.43	78.58	66.73	92.52	31.02	76.64
	Poisson	42.21	74.65	48.84	87.71	21.64	69.62
GNN/GT baselines	GCN	49.23±0.30	72.51±0.35	71.74±0.29	93.34±0.02	37.84±0.10	75.64±0.21
	GraphSAGE	50.10±0.11	77.68±0.20	71.49±0.27	95.40±0.10	**63.40±0.61**	78.29±0.16
	GraphSAINT	51.03±0.23	76.17±0.28	70.56±0.22	**96.60±0.14**	**65.30±0.31**	80.27±0.26
	NAGphormer	51.71±0.23	73.61±0.33	70.13±0.55	93.58±0.05	62.15±0.14	73.55±0.21
	Polynormer	51.97±0.18	**78.97±0.47**	**73.46±0.16**	94.13±0.03	62.50±0.19	**83.82±0.11**
	DIFFormer	51.09±0.12	**79.49±0.44**	69.86±0.25	94.76±0.11	60.45±0.21	74.16±0.31
LapKM baselines	LapSVM-Primal	45.38±0.25	Timeout	Timeout	Timeout	Timeout	Timeout
	LapRLS-NyPCG	44.22±0.32	61.66±0.17	54.70±0.17	71.26±0.13	58.09±0.20	59.01±0.21
LapTops (ours)	LapTop-RLS	51.92±0.09	78.94±0.03	71.60±0.08	**95.90±0.02**	62.64±0.09	79.98±0.09
	LapTop-SVM	52.02±0.03	78.95±0.07	**71.75±0.12**	95.45±0.15	63.02±0.15	**80.56±0.16**
	LapTop-SVM^2	**52.53±0.09**	78.74±0.11	71.73±0.08	95.80±0.03	62.85±0.12	80.31±0.11
	LapTop-Log	**52.28±0.08**	78.16±0.09	71.64±0.13	94.65±0.15	62.91±0.02	80.12±0.16

quadratic potential $\phi(\boldsymbol{\alpha}) := \frac{1}{2}\boldsymbol{\alpha}^\top(\boldsymbol{K} + \boldsymbol{I}/\rho)\boldsymbol{\alpha} - \boldsymbol{q}_k^\top\boldsymbol{\alpha}$, whose stationarity condition matches the linear system (9). In each iteration, computing $\boldsymbol{K}_{\mathcal{B},:}\boldsymbol{\alpha}$ block-wisely and approximately solving linear system (10) costs $O(nb + T_{\mathsf{CG}}b^2)$ operations. BCD in our setting converges within $O(\frac{n\rho\sigma_1}{b}\log(1/\epsilon))$ iterations (Richtárik & Takáč, 2020; Tappenden et al., 2016), where σ_1 is the largest eigenvalue of $\boldsymbol{K}$. Therefore, we can either increase the block size b or decrease the penalty parameter ρ to accelerate convergence. Here, the benefit of a larger b mainly comes from the improved parallelism on modern hardware, since the total arithmetic cost per epoch is nearly independent of b. A complexity comparison of LapTop and other LapKMs is shown in Table A.1 in Appendix B.

4 Experiments

We show that LapTop substantially outperforms existing LapKM baselines and is competitive with strong GNNs on six large-scale benchmarks, and analyze the source of the gains, the effect of prior mismatch, inductive inference, and the convergence of the proposed solver.

4.1 Experimental Setup

We focus on the node classification tasks on large-scale graph to align with the prevalent application of GNNs. Specifically, the datasets used in the experiments include the social media networks: Reddit (Hamilton et al., 2017), Flickr, Yelp (Zeng et al., 2020), and the open graph benchmarks (OGB) datasets: Proteins, Arxiv, and Products (Hu et al., 2020), which are prevalent in benchmarking GNNs in recent research. We use the default splits provided by the datasets. The processing details are provided in Appendix C. To our knowledge, we are the first to benchmark LapKM on these large-scale graph datasets for comparison with GNN baselines.

To highlight that LapTop can support diverse loss functions, we implement four instances: LapTop-RLS with squared loss, LapTop-SVM(^{2}) with (squared) hinge loss, and LapTop-Log with Logistic loss. For LapKM baselines, we compare against LapSVM-Primal (Melacci & Belkin, 2011) and LapRLS-NyPCG (Li et al., 2019). We set the number of Nyström landmarks to the maximum that fits the memory and keeps the numerical convergence. As transductive graph-only baselines, we include Laplacian label propagation (Zhou et al., 2004) and Poisson learning (Calder et al., 2020). To posit our method within the modern graph learning landscape, we compare against the feature-side baselines, GNNs and graph Transformers (GTs), GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GraphSAINT (Zeng et al., 2020), NAGphormer (Chen et al., 2023), DIFFormer (Wu et al., 2025), Polynormer (Deng et al., 2024). See Appendix C for more details.

Table 2: Peak memory (GB) and time (min) comparison. LAPTOP is profiled with the SVM loss.

	Flickr		Yelp		Products	
Methods	Mem	Time	Mem	Time	Mem	Time
LapSVM-Primal	6.3	338	14.3	**>1440**	20.3	**>1440**
LapRLS-NyPCG	**23.5**	**4**	**21.3**	**5**	19.1	**19**
LapTop	**1.7**	10	**8.8**	59	**10.1**	173
Polynormer	19.3	298	21.3	367	**22.1**	443

Table 3: Inductive accuracy (%). Test nodes and edges are unseen during training.

Test graph	Method	Products	Arxiv
w/o	Plain KRR	58.82	56.35
	LAPTOP	65.02	57.43
w/	Polynormer	60.50	68.12
	KRR + smooth	75.77	68.60
	LAPTOP + smooth	**77.12**	**68.97**

4.2 RESULTS AND ANALYSIS

Benchmarking results. Table 1 summarizes the results. LAPTOP outperforms the LapKM baselines by large margins on all datasets: LapSVM-Primal times out beyond Flickr due to its full-batch Newton steps, and LapRLS-NyPCG scales further via Nyström approximation but lags far behind, for reasons analyzed below. Laplacian propagation, using the graph alone, exceeds several GNNs on Products, but remains clearly behind LAPTOP once node features become informative (e.g., by four points on Products). Against GNN/GT baselines, LAPTOP ranks within the top three on all six datasets and first on Flickr, with the four loss variants staying within 1.3 points of each other. A gap remains to the linear graph Transformer Polynormer on Arxiv and Products, which we examine next. To our knowledge, these are the first results that validate LapKMs on modern large-scale benchmarks, showing that a well-optimized kernel model can be competitive without deep architectures. LAPTOP is also efficient in practice (Table 2): it trains with the lowest memory footprint among the compared methods—e.g., 10.1 GB on Products, less than half of Polynormer—while being 2.6–30× faster than Polynormer. LapRLS-NyPCG is the fastest to train, but requires expensive storage of 19–24 GB.

Where do the gains come from? Since LAPTOP fixes the representation by a kernel and injects the graph only through the manifold regularizer, the contribution of the graph can be isolated by comparing with the LAPTOP without graph ($c_M = 0$, i.e., plain KRR) in Figure 2. On Products and Flickr, the manifold regularizer improves plain KRR by more than 10 points, closing most of the gap to the best GNN/GT baseline on Products and surpassing all of them on Flickr. The residual gap thus indicates the remaining headroom for learned representations and more flexible graph operators, echoing the drift of the GNN literature towards decoupled propagation (Wu et al., 2019; Gasteiger et al., 2019; Huang et al., 2021). Realizing this gain requires the exact kernel model. Specifically, LapRLS-NyPCG targets the same model as LAPTOP-RLS, but restricts the solution to the span of $s(\ll n)$ Nyström landmarks; the resulting gap is comparable to the graph gain itself, and LapRLS-NyPCG on Products is barely above plain KRR (59.01% vs. 58.82%). This suggests that a low-rank solution space can hardly express functions that vary smoothly along the graph, so the graph prior is largely lost. Retaining the full representer expansion (2) at scale is precisely what the coupled problem previously rendered intractable.

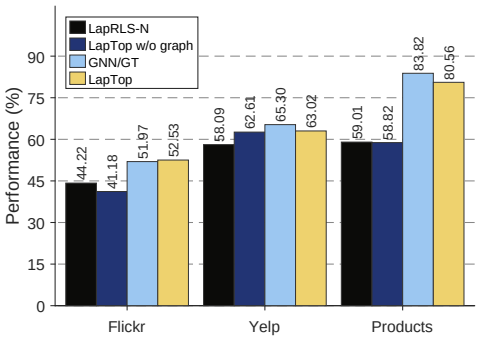
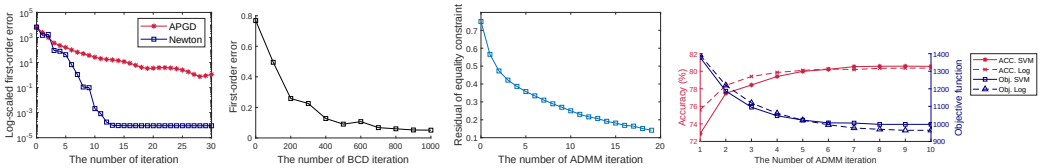


Figure 2: Ablation for performance gains. Yelp reports F1; others report accuracy.

When the prior mismatches the graph. Yelp is such a case: the edges are friendships while the labels are reviewed business categories, and adjacent label sets share an average Jaccard similarity of only 0.22 (Zhao et al., 2023), i.e., low homophily. This mismatch is not a defect specific to LAPTOP, but is shared by GNNs, LapKMs, and Laplacian learning methods built on the global homophily assumption. LAPTOP degrades gracefully here: a small selected c_M essentially recovers plain KRR (+0.4 points), whereas GCN collapses far below plain KRR. The locality-aware GNNs such as GraphSAGE and GraphSAINT (Zeng et al., 2021) stay above plain KRR. Still, c_M only scales the global propagation operator $(\rho \boldsymbol{I} + c_M \boldsymbol{L})^{-1}$, so it can weaken the prior but not localize



(a) $\boldsymbol{u}$-step (b) $\boldsymbol{\alpha}$-step (c) $\|\boldsymbol{u} - \boldsymbol{K}\boldsymbol{\alpha}\|_1/n$ (d) Objective and accuracy

Figure 3: Convergence behavior of LAPTOP. The subproblem convergence (a,b), the feasibility residual (c), and outer-loop convergence (d) are separately shown.

it, and heterophilic graphs remain a limitation of LAPTOP. Localizing the prior, e.g., Laplacians on sampled subgraphs in the spirit of Zeng et al. (2021), is a natural future direction.

Inductive inference. We further train on the subgraph induced by training and validation nodes, and reveal test nodes and their edges only at inference. LAPTOP supports two inference modes: graph-free prediction $f(\boldsymbol{x}^*)$ via the representer expansion, and, when the test graph is available, a single $\boldsymbol{u}$-step on it initialized at $f(\boldsymbol{x}^*)$, which smooths the predictions without retraining. As shown in Table 3, with access to the test graph, LAPTOP outperforms Polynormer on both datasets, by a large margin on Products, where Polynormer falls far below its transductive accuracy. This may be attributed to the structural shift of the sales-rank split, which trains on popular products and tests on the long tail: graph operators learned on the training subgraph transfer poorly, whereas the smoothing step involves no learned parameters and adapts to the revealed graph. Applying the same smoothing to plain KRR also surpasses Polynormer on Products, while the kernel predictor remains the stronger base both before and after smoothing. Even without the test graph, LAPTOP improves over plain KRR on Products, indicating that part of the graph prior is distilled into the kernel predictor.

Convergence behavior. Figure 3 summarizes the optimization dynamics of LAPTOP. For the $\boldsymbol{u}$-step with smooth losses, the inexact Newton method reaches a high-precision solution within about 15 inner TCG steps in a single outer step, while APGD converges relatively slowly but remains applicable to non-smooth losses (Figure 3(a)). For the $\boldsymbol{\alpha}$-step, the KRR residual $\|\nabla\phi(\boldsymbol{\alpha})\|$ exhibits an overall descending trend, albeit non-monotonically, since BCD monotonically decreases the potential ϕ rather than the residual itself (Figure 3(b)). The objective decreases monotonically and stabilizes within about ten ADMM iterations, with the accuracy saturating alongside (Figure 3(d)), and the equality-constraint residual decreases stably throughout iterations (Figure 3(c)). Notably, the accuracy saturates while the residual is still mild. The reason may be the robustness of ADMM against the inexact subproblem solvers (Boyd et al., 2011).

5 CONCLUSION AND FUTURE WORK

In this paper, we revisited Laplacian regularized kernel models (LapKMs), the regularizer-side paradigm that has been overlooked for two decades. We showed that its long-standing stagnation arises from an optimization gap: the coupling of the kernel matrix and the graph Laplacian induces prohibitive computation and numerical instability. The proposed LAPTOP resolves this gap via a tailored ADMM framework with efficient subproblem solvers, scaling LapKM to million-node graphs with exact kernels and diverse loss functions. Empirical results demonstrate that LAPTOP is competitive with GNN baselines on million-node datasets, suggesting that output-side graph smoothing contributes the dominant predictive gain on homophilous benchmarks. We hope this work encourages the community to re-examine theoretically grounded kernel models as a principled complement to deep graph learning.

The standard LapKM relies on the homophily assumption, where the manifold regularization acts as a low-pass graph filter. It may fail on heterophilous graphs, where connected nodes are potentially dissimilar and high-frequency signals are useful (Luan et al., 2022). Adaptive graph filters or localized graph smoothing within the label-smoothing subproblem are a promising direction (Zeng et al., 2021). On the other hand, graph neural tangent kernels (Du et al., 2019) inject the graph into the kernel itself, i.e., feature-side in kernel language, whereas LapKM keeps a generic kernel with an explicit, tunable graph term; combining the two injection points is an interesting open problem.

REPRODUCIBILITY STATEMENT

To support reproducibility, we point to the following parts of the submission rather than restating experimental details here. The ADMM formulation, the $\boldsymbol{u}$- and $\boldsymbol{\alpha}$-subproblems, and the complexity analysis are given in Section 3; the full training procedure is stated in Algorithm A.2, with Newton-TCG and APGD details in Appendices B.1 and B.2 and proximal operators in Appendix A. Dataset statistics and processing, kernels, and implementation notes appear in Appendix C, the inductive protocol in Appendix C.5, and the hyperparameter grid, selected configurations, random seeds, and baseline settings in Appendix C.6 and Table A.4. Anonymous source code is linked from Appendix C.

AI USE STATEMENT

In this work, we used generative AI tools to help draft the written interpretation of experimental results. We have not used generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate mathematical claims, provide ingredients for or assist in writing proofs, propose or refine scientific hypotheses, design the research methodology or experiments, implement the proposed method, translate the manuscript, or clean and reformat datasets; qualitative or thematic data analysis is not applicable to this work. Additionally, we used generative AI tools (coding assistants) to draft and edit manuscript text for readability, to typeset tables and layout, to write and revise plotting scripts and scientific figures, to brainstorm wording and section structure, and to assist with literature lookup and reference formatting. All algorithms, experimental runs, numerical results, and scientific claims were produced and verified by the authors. LLM-assisted prose was checked against the reported tables and the cited literature; plotting code was executed and the resulting figures were inspected by the authors. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Amirhesam Abedsoltan, Mikhail Belkin, and Parthe Pandit. Toward large kernel models. In *International Conference on Machine Learning (ICML)*, volume 202, pp. 61–78. PMLR, 2023.

Amirhesam Abedsoltan, Siyuan Ma, Parthe Pandit, and Mikhail Belkin. Fast training of large kernel models with delayed projections. In *Advances in Neural Information Processing Systems (NeurIPS)*, October 2025.

Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7 (85):2399–2434, 2006.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, July 2011. ISSN 1554-0669.

Jeff Calder, Brendan Cook, Matthew Thorpe, and Dejan Slepcev. Poisson learning: Graph based semi-supervised learning at very low label rates. In *International Conference on Machine Learning (ICML)*, volume 119, pp. 1306–1316. PMLR, 2020.

Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. NAGphormer: A tokenized graph transformer for node classification in large graphs. In *International Conference on Learning Representations (ICLR)*, 2023.

Lin Chen and Sheng Xu. Deep neural tangent kernel and Laplace kernel have the same RKHS. In *International Conference on Learning Representations (ICLR)*, 2021.

Youngmin Cho and Lawrence Saul. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 22. Curran Associates, Inc., 2009.

Chenhui Deng, Zichao Yue, and Zhiru Zhang. Polynormer: Polynomial-expressive graph transformer in linear time. In *International Conference on Learning Representations (ICLR)*, 2024.

Simon S. Du, Kangcheng Hou, Barnabás Póczos, Ruslan Salakhutdinov, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Stanley C. Eisenstat and Homer F. Walker. Choosing the forcing terms in an inexact Newton method. *SIAM Journal on Scientific Computing*, 17(1):16–32, 1996. doi: 10.1137/0917003.

Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. Embrace rejection: Kernel matrix approximation by accelerated randomly pivoted cholesky. *SIAM Journal on Matrix Analysis and Applications*, 46(4):2527–2557, December 2025. ISSN 0895-4798. doi: 10.1137/24M1699048.

Zachary Frangella, Joel A. Tropp, and Madeleine Udell. Randomized Nyström preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 44(2):718–752, June 2023. ISSN 0895-4798. doi: 10.1137/21M1466244.

Leonardo Galli and Chih-Jen Lin. A study on truncated Newton methods for linear classification. *IEEE Transactions on Neural Networks and Learning Systems*, 33(7):2828–2841, July 2022. ISSN 2162-2388. doi: 10.1109/TNNLS.2020.3045836.

Damien Garreau, Wittawat Jitkrittum, and Motonobu Kanagawa. Large sample analysis of the median heuristic, 2017. URL https://arxiv.org/abs/1707.07269.

Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized PageRank. In *International Conference on Learning Representations (ICLR)*, 2019.

Amnon Geifman, Abhay Yadav, Yoni Kasten, Meirav Galun, David Jacobs, and Ronen Basri. On the similarity between the Laplace and neural tangent kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 1451–1461. Curran Associates, Inc., 2020.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1024–1034, 2017.

Mingguo He, Zhewei Wei, Zengfeng Huang, and Hongteng Xu. BernNet: Learning arbitrary graph spectral filters via Bernstein approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 14239–14251, 2021.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 22118–22133, 2020.

Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin Benson. Combining label propagation and simple models out-performs graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2021.

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

Jian Li, Yong Liu, Rong Yin, and Weiping Wang. Approximate manifold regularization: Scalable algorithm and generalization analysis. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2887–2893, 2019.

Zhouchen Lin, Huan Li, and Cong Fang. *Accelerated Optimization for Machine Learning: First-Order Algorithms*. Springer, Singapore, 2020. doi: 10.1007/978-981-15-2910-8.

Zhouchen Lin, Huan Li, and Cong Fang. *Alternating direction method of multipliers for machine learning*. Springer, 2022.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pp. 338–348, August 2020.

Zirui Liu, Kaixiong Zhou, Fan Yang, Li Li, Rui Chen, and Xia Hu. EXACT: Scalable graph neural networks training via extreme activation compression. In *International Conference on Learning Representations (ICLR)*, 2022.

Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. Revisiting heterophily for graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 1362–1375, December 2022.

Yuankai Luo, Lei Shi, and Xiao-Ming Wu. Classic GNNs are strong baselines: Reassessing GNNs for node classification. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024.

Siyuan Ma and Mikhail Belkin. Diving into the shallows: A computational perspective on large-scale shallow learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pp. 3778–3787, 2017.

Siyuan Ma and Mikhail Belkin. Kernel machines that adapt to GPUs for effective large batch training. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 1, pp. 360–373, April 2019.

Stefano Melacci and Mikhail Belkin. Laplacian support vector machines trained in the primal. *Journal of Machine Learning Research*, 12(31):1149–1184, 2011. ISSN 1533-7928.

Ha Quang Minh, Partha Niyogi, and Yuan Yao. Mercer's theorem, feature maps, and smoothing. In *Learning Theory*, volume 4005, pp. 154–168. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-35294-5 978-3-540-35296-9. doi: 10.1007/11776420_14.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 20, pp. 1177–1184. Curran Associates, Inc., 2007.

Pratik Rathore, Zachary Frangella, Jiaming Yang, Michał Dereziński, and Madeleine Udell. Have ASkotch: A neat solution for large-scale kernel ridge regression. *Journal of Machine Learning Research*, 27(162):1–48, 2026.

Peter Richtárik and Martin Takáč. Stochastic reformulations of linear systems: Algorithms and convergence theory. *SIAM Journal on Matrix Analysis and Applications*, 41(2):487–524, January 2020. ISSN 0895-4798. doi: 10.1137/18M1179249.

Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. FALKON: An optimal large scale kernel method. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pp. 3888–3898. Curran Associates, Inc., 2017.

Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, 2002. doi: 10.7551/mitpress/4175.001.0001.

Rachael Tappenden, Peter Richtárik, and Jacek Gondzio. Inexact coordinate descent: Complexity and preconditioning. *Journal of Optimization Theory and Applications*, 170(1):144–176, 2016.

Christopher Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 13, pp. 682–688. MIT Press, 2000.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning (ICML)*, pp. 6861–6871. PMLR, May 2019.

Qitian Wu, Wentao Zhao, Zenan Li, David P Wipf, and Junchi Yan. NodeFormer: A scalable graph structure learning transformer for node classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 27387–27401, 2022.

Qitian Wu, David Wipf, and Junchi Yan. Transformers from diffusion: A unified framework for neural message passing. *Journal of Machine Learning Research*, 26(129):1–47, 2025.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, January 2021. ISSN 2162-2388. doi: 10.1109/TNNLS.2020.2978386.

Yongyi Yang, Tang Liu, Yangkun Wang, Jinjing Zhou, Quan Gan, Zhewei Wei, Zheng Zhang, Zengfeng Huang, and David Wipf. Graph neural networks inspired by classical iterative algorithms. In *International Conference on Machine Learning (ICML)*, pp. 11773–11783, 2021.

Yongyi Yang, Tang Liu, Yangkun Wang, Zengfeng Huang, and David Wipf. Implicit vs unfolded graph neural networks. *Journal of Machine Learning Research*, 26(82):1–46, 2025.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 28877–28888, 2021.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graph-SAINT: Graph sampling based inductive learning method. In *International Conference on Learning Representations (ICLR)*, 2020.

Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. Decoupling the depth and scope of graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Junhong Zhang and Zhihui Lai. Joker: Joint optimization framework for lightweight kernel machines. In *International Conference on Machine Learning (ICML)*, volume 267, pp. 75297–75313. PMLR, 2025.

Yan-Ming Zhang, Xu-Yao Zhang, Xiao-Tong Yuan, and Cheng-Lin Liu. Large-scale graph-based semi-supervised learning via tree Laplacian solver. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 30, March 2016. doi: 10.1609/aaai.v30i1.10218.

Tianqi Zhao, Thi Ngan Dong, Alan Hanjalic, and Megha Khosla. Multi-label node classification on graph-structured data. *Transactions on Machine Learning Research*, 2023.

Dengyong Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 16, 2004.

Xiong Zhou, Xianming Liu, Hao Yu, Jialiang Wang, Zeke Xie, Junjun Jiang, and Xiangyang Ji. Variance-enlarged Poisson learning for graph-based semi-supervised learning with extremely sparse labeled data. In *International Conference on Learning Representations (ICLR)*, 2024.

Meiqi Zhu, Xiao Wang, Chuan Shi, Houye Ji, and Peng Cui. Interpreting and unifying graph neural networks with an optimization framework. In *Proceedings of the Web Conference (WWW)*, pp. 1215–1226, 2021.

Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *International Conference on Machine Learning (ICML)*, pp. 912–919, 2003.

Appendix

A PROXIMAL OPERATOR OF THE LOSS FUNCTION

1. Hinge loss

The hinge function is defined as $f(x) = \max\{0, 1 - yx\}$ with $y \in \{-1, 1\}$.

$$\mathsf{prox}_{\gamma,f}(x) = \operatorname*{argmin}_u \frac{1}{2\gamma}(u - x)^2 + \max\{0, 1 - yu\}. \tag{A.1}$$

Its proximal operator is:

$$\mathsf{prox}_{\gamma,f}(x) = \begin{cases} x, & \text{If } xy \geq 1, \\ y, & \text{If } 1 - \gamma \leq xy < 1, \\ \gamma y + x & \text{Otherwise.} \end{cases} \tag{A.2}$$

Proof. First, consider the case $uy \geq 1$, then $1 - uy \leq 0$, and the last term of (A.1) vanishes. In this case,

$$\mathsf{prox}_{\gamma,f}(x) = \operatorname*{argmin}_u \frac{1}{2\gamma}(u - x)^2, \quad \textit{s.t. } uy \geq 1. \tag{A.3}$$

We can see $u = x$ is an inactive minimizer (i.e., the minimizer when we ignore the constraints). Therefore, if $xy \geq 1$, the solution $u = x$ satisfies $uy = xy \geq 1$. This proves the first case of (A.2).

Second, consider $uy < 1$, and (A.1) becomes

$$\min_u \frac{1}{2\gamma}(u - x)^2 + 1 - uy, \quad \textit{s.t. } uy < 1. \tag{A.4}$$

Let the derivative be zero, we have the inactive minimizer $u = \gamma y + x$. This solution only holds for $uy < 1$. To ensure this, we have:

$$uy = \gamma y^2 + xy < 1. \tag{A.5}$$

That is, if $xy < 1 - \gamma y^2 = 1 - \gamma$, we have $u = \gamma y + x$. The last line of (A.2) is proven.

Finally, the remaining scenario is $1 - \gamma \leq xy < 1$. If $y = 1$, we have $1 - \gamma \leq x < 1$, then the problem becomes

$$\min_u \frac{1}{2\gamma}(u - x)^2 + \max\{0, 1 - u\}. \tag{A.6}$$

The optimality condition is that there exists an $u^\star$ such that

$$0 \in \frac{1}{\gamma}(u^\star - x) + \partial(\max\{0, 1 - u^\star\}) \Leftrightarrow -\frac{u^\star - x}{\gamma} \in \partial(\max\{0, 1 - u^\star\}).$$

Note that at $u = 1$, the subgradient term is $\partial(\max\{0, 1 - u\}) = [-1, 0]$. On the other hand, by $1 - \gamma \leq x < 1$ we have

$$-1 \leq -\frac{1 - x}{\gamma} < 0$$

Meaning that $-(1 - x)/\gamma$ belongs to $[-1, 0)$, and further belongs to $[-1, 0]$:

$$-\frac{1 - x}{\gamma} \in \partial(\max\{0, 1 - u\})|_{u=1} = [-1, 0], \tag{A.7}$$

i.e., the optimality is reached. So $u = 1$ is the solution. Similarly, we can obtain $u = -1$ if $y = -1$ and $1 - \gamma \leq -x < 1$. Thus, the second line of (A.2) is proved. □

2. Squared hinge loss

The squared hinge loss is defined as $f(x) = \max\{0, 1 - yx\}^2/2$ with $y \in \{-1, 1\}$. Its proximal operator is:

$$\mathsf{prox}_{\gamma,f}(x) = \begin{cases} x, & \text{If } xy \geq 1, \\ \frac{x+\gamma y}{1+\gamma} & \text{Otherwise.} \end{cases} \tag{A.8}$$

Proof. The optimization problem is

$$\min_u \frac{1}{2\gamma}(u - x)^2 + \frac{1}{2}\max\{0, 1 - yu\}^2. \tag{A.9}$$

First, if $uy \geq 1$, the second term is zero. The problem becomes $\min_u \frac{1}{2\gamma}(u - x)^2$ s.t. $uy \geq 1$. The unconstrained minimizer is $u = x$. If $xy \geq 1$, then $u = x$ satisfies the constraint, so $u = x$ is the solution.

Second, if $uy < 1$, the second term is $\frac{1}{2}(1 - yu)^2$. The problem is unconstrained locally:

$$0 = \frac{1}{\gamma}(u - x) - y(1 - yu) \iff u - x - \gamma y + \gamma u = 0, \tag{A.10}$$

where we use $y^2 = 1$. The solution is $u = \frac{x+\gamma y}{1+\gamma}$. Checking the condition for this case:

$$uy = \frac{xy + \gamma}{1 + \gamma} < 1 \iff xy < 1. \tag{A.11}$$

Thus, when $xy < 1$, the solution is $\frac{x+\gamma y}{1+\gamma}$. □

B Solver details

B.1 Newton-TCG for twice-differentiable losses

For twice-differentiable losses $\xi_y(\cdot)$ (e.g., the squared loss $\xi_y(u) = (u - y)^2/2$ and the Logistic loss $\xi_y(u) = \log(1 + \exp(-yu))$), we leverage the Hessian for Newton's method. Let the objective function of (7) be $J(\boldsymbol{u})$ and the t-th iterate be $\boldsymbol{u}_{k,t}$. The Newton step $\boldsymbol{s}_t$ is given by minimizing the second-order Taylor expansion of $J(\boldsymbol{u}_{k,t})$:

$$\boldsymbol{s}_t = \operatorname*{argmin}_{\boldsymbol{s}} \frac{1}{2}\boldsymbol{s}^\top \nabla^2 J(\boldsymbol{u}_{k,t})\boldsymbol{s} + \nabla J(\boldsymbol{u}_{k,t})^\top \boldsymbol{s}, \tag{A.12}$$

i.e., $\boldsymbol{s}_t = -\nabla^2 J(\boldsymbol{u}_{k,t})^{-1}\nabla J(\boldsymbol{u}_{k,t})$, and the iterate is updated as $\boldsymbol{u}_{k,t+1} = \boldsymbol{u}_{k,t} + \boldsymbol{s}_t$. Here,

$$\nabla^2 J(\boldsymbol{u}_{k,t}) = \boldsymbol{B} + \nabla^2 \xi_{\boldsymbol{y}}(\boldsymbol{u}_{k,t}), \quad \nabla J(\boldsymbol{u}_{k,t}) = \boldsymbol{g}(\boldsymbol{u}_{k,t}) + \nabla \xi_{\boldsymbol{y}}(\boldsymbol{u}_{k,t}), \tag{A.13}$$

where

$$\boldsymbol{B} := \rho \boldsymbol{I} + c_M \boldsymbol{L}, \quad \boldsymbol{g}(\boldsymbol{u}_{k,t}) := \boldsymbol{B}\boldsymbol{u}_{k,t} - (\rho \boldsymbol{K}\boldsymbol{\alpha}_k - \boldsymbol{\lambda}_k). \tag{A.14}$$

Note that the loss term is supported on the labeled nodes: $[\nabla \xi_{\boldsymbol{y}}(\boldsymbol{u})]_i = c_L \nabla \xi_{y_i}(u_i)$ if $\boldsymbol{x}_i$ is labeled and 0 otherwise, and $[\nabla^2 \xi_{\boldsymbol{y}}(\boldsymbol{u})]_{ii} = c_L \nabla^2 \xi_{y_i}(u_i)$ if $\boldsymbol{x}_i$ is labeled and 0 otherwise.

The exact Newton step involves the expensive matrix inversion $\nabla^2 J(\boldsymbol{u}_{k,t})^{-1}$. In addition, enforcing high precision in the early stages of optimization often leads to *over-solving*, i.e., solving the inner subproblem to high precision when the current iterate is far from the optimum, where a coarse approximate solution suffices for global convergence. To address both issues, we adopt the truncated CG method (TCG) in Algorithm A.1. Specifically, for a linear system $\boldsymbol{A}\boldsymbol{s} = \boldsymbol{b}$, TCG inherits the classical CG iterations but terminates when the following ratio-based condition is satisfied:

$$\frac{\|\boldsymbol{A}\boldsymbol{s} - \boldsymbol{b}\|_1}{\|\boldsymbol{b}\|_1} \leq \epsilon := \min\{0.5, \|\boldsymbol{b}\|_1^{0.5}\},$$

where the threshold ratio ϵ is suggested by (Eisenstat & Walker, 1996), which ensures sufficient accuracy of the approximate solution while avoiding over-solving. The matrix-vector product $\nabla^2 J(\boldsymbol{u}_{k,t})\boldsymbol{p}$ is efficient due to the sparsity of $\boldsymbol{L}$ and the diagonality of $\nabla^2 \xi_{\boldsymbol{y}}(\boldsymbol{u}_{k,t})$, yielding cheap TCG iterations. Moreover, the Newton step with TCG is proven to achieve a superlinear convergence rate (Galli & Lin, 2022).

Algorithm A.1: Truncated Conjugate Gradient (TCG)

Input : Matrix $\boldsymbol{A} \in \mathbb{R}^{m\times m}$, right-hand side $\boldsymbol{b} \in \mathbb{R}^m$, iteration limitation T_{CG}.
Define the threshold ratio $\epsilon := \min\{0.5, \|\boldsymbol{b}\|_1^{0.5}\}$;
Initialize: $\boldsymbol{s}_0 = \boldsymbol{0}, \boldsymbol{r}_0 \leftarrow \boldsymbol{b}, \boldsymbol{p}_0 \leftarrow \boldsymbol{r}_0$;
for $t = 0, \cdots, T_{\mathsf{CG}} - 1$ **do**
  $\omega_t \leftarrow (\boldsymbol{r}_t^\top \boldsymbol{r}_t)/(\boldsymbol{p}_t^\top \boldsymbol{A}\boldsymbol{p}_t)$;
  $\boldsymbol{s}_{t+1} \leftarrow \boldsymbol{s}_t + \omega_t \boldsymbol{p}_t$;
  $\boldsymbol{r}_{t+1} \leftarrow \boldsymbol{r}_t - \omega_t \boldsymbol{A}\boldsymbol{p}_t$;
  if $\|\boldsymbol{r}_{t+1}\|_1 \le \epsilon \|\boldsymbol{b}\|_1$ **then**
    return $\boldsymbol{s}_{t+1}$;
  end
  $\beta_t \leftarrow (\boldsymbol{r}_{t+1}^\top \boldsymbol{r}_{t+1})/(\boldsymbol{r}_t^\top \boldsymbol{r}_t)$;
  $\boldsymbol{p}_{t+1} \leftarrow \boldsymbol{r}_{t+1} + \beta_t \boldsymbol{p}_t$;
end
return $\boldsymbol{s}_{T_{\mathsf{CG}}}$;

B.2 APGD FOR PROX-FRIENDLY LOSSES

For non-smooth losses whose Hessian is ill-defined, the objective of (7) decomposes into a smooth part and a prox-friendly part:

$$\min_{\boldsymbol{u}} \underbrace{\frac{c_M}{2}\boldsymbol{u}^\top \boldsymbol{L}\boldsymbol{u} + \frac{\rho}{2}\left\|\boldsymbol{u} - \boldsymbol{K}\boldsymbol{\alpha}^k + \frac{\boldsymbol{\lambda}^k}{\rho}\right\|^2}_{=:h(\boldsymbol{u}),\ \text{smooth}} + \underbrace{\xi_{\boldsymbol{y}}(\boldsymbol{u})}_{\text{prox-friendly}}.$$

Here, $h(\boldsymbol{u})$ is quadratic and is $(c_M\sigma_{\max}(\boldsymbol{L}) + \rho)$-smooth, where $\sigma_{\max}(\boldsymbol{L})$ is the largest eigenvalue of $\boldsymbol{L}$. This scenario includes nonsmooth losses, e.g., the hinge loss $\xi_y(u) = \max\{0, 1 - yu\}$ with $y \in \{-1, 1\}$ and the squared hinge loss $\xi_y(u) = \max\{0, 1 - yu\}^2/2$. We use the Accelerated Proximal Gradient Descent (APGD) method. At iteration t, given the current point $\boldsymbol{u}_{k,t}$ and the previous point $\boldsymbol{u}_{k,t-1}$, we apply a FISTA-style update:

$$\boldsymbol{v}_{k,t} = \boldsymbol{u}_{k,t} + \frac{\theta_t - 1}{\theta_{t+1}}(\boldsymbol{u}_{k,t} - \boldsymbol{u}_{k,t-1}), \quad \boldsymbol{u}_{k,t+1} = \mathsf{prox}_{\eta,\xi_{\boldsymbol{y}}}\left(\boldsymbol{v}_{k,t} - \eta\nabla h(\boldsymbol{v}_{k,t})\right), \tag{A.15}$$

where θ_t is the momentum parameter, typically updated as $\theta_{t+1} = (1 + \sqrt{1 + 4\theta_t^2})/2$ with $\theta_0 = 1$, and the step size is ideally $\eta = 1/(c_M\sigma_{\max}(\boldsymbol{L}) + \rho)$ according to the smoothness of $h(\cdot)$ (Lin et al., 2020).[1] The gradient $\nabla h(\boldsymbol{v}_{k,t})$ at the extrapolation point is given by (A.14) directly. Due to the separability of $\xi_{\boldsymbol{y}}(\cdot)$, the proximal operator decomposes into element-wise operations: for the labeled data points, $[\mathsf{prox}_{\eta,\xi_{\boldsymbol{y}}}(\boldsymbol{z})]_i = \mathsf{prox}_{\eta c_L,\xi_{y_i}}(z_i) = \mathrm{argmin}_u\left(\frac{1}{2}(u - z_i)^2 + \eta c_L \xi_{y_i}(u)\right)$, and $[\mathsf{prox}_{\eta,\xi_{\boldsymbol{y}}}(\boldsymbol{z})]_i = z_i$ for unlabeled data. The closed-form proximal operators for the hinge and squared hinge losses are derived in Appendix A and summarized in Table A.3.

B.3 COMPLEXITY COMPARISON

Table A.1: Complexity of different LapKMs. b is the block size and s is the number of Nyström landmarks ($b \ll s \ll n$), and $T_1 := \max\{T_{\mathsf{NT}}T_{\mathsf{CG}}, T_{\mathsf{APGD}}\}$. The operations of LapSVM-Primal and LapRLS-NyPCG are reported per inner (P)CG iteration, while those of LAPTOP are reported per outer ADMM iteration.

Methods	Space	Operations per iteration
LapSVM-Primal	n^2	$2n^2 + (n + E)$
LapRLS-NyPCG	$s^2 + ns$	$2s^2 + ns + (n + E)$
LAPTOP	$n + b^2$	$T_1(n + E) + T_{\mathsf{BCD}}(nb + T_{\mathsf{CG}}b^2)$

[1] In practice, since $\sigma_{\max}(\boldsymbol{L}) \le 2$, we use $\eta = 1/(2c_M + \rho)$ to avoid computing the largest eigenvalue.

Table A.2: The statistics of datasets used in the experiments (Liu et al., 2022).

Dataset	Task	Nodes	Edges	Features	Classes	Label Rates	Metrics
Flickr	multi-class	89,250	449,878	500	7	50.00%	Accuracy
Proteins	multi-label	132,534	39,561,252	8	112	65.00%	ROC-AUC
Arxiv	multi-class	169,343	1,157,799	128	40	53.70%	Accuracy
Reddit	multi-class	232,965	11,606,919	602	41	65.86%	Accuracy
Yelp	multi-label	716,847	6,977,409	300	100	75.00%	F1-micro
Products	multi-class	2,449,029	61,859,076	100	47	8.03%	Accuracy

C Experimental details

C.1 Dataset

The datasets used in the experiments are summarized in Table A.2. We follow the node classification setting of GNN literature to classify the unlabeled data. The information on the used datasets is as follows:

- **Flickr**[2] (Zeng et al., 2020). It is a multi-class node classification dataset. Unlike typical social networks, each node represents an image uploaded to Flickr, and edges connect images that share common properties (e.g., same location, gallery, or tags). The node features are extracted from the images using visual descriptors (e.g., SIFT).
- **Proteins**[3] (Hu et al., 2020). It is a multi-label protein-function prediction dataset on a protein–protein association network. Each node represents a protein, and edges indicate biologically meaningful associations (e.g., physical interactions, co-expression, or homology). The 8-dimensional node features are obtained by averaging the incident edge features.
- **Arxiv**[4] (Hu et al., 2020). It is a citation network dataset where nodes represent papers from the Computer Science category of arXiv, and edges represent citation relationships. The node features are 128-dimensional vectors obtained by averaging the Word2Vec embeddings of words in the paper's title and abstract.
- **Reddit**[5] (Hamilton et al., 2017). It is a multi-class node classification dataset collected from Reddit. Each node represents a post, and edges connect posts if the same user comments on both. The node features are derived from the post content (GloVe embeddings).
- **Yelp**[6] (Zeng et al., 2020). It is a multi-label node classification dataset. Each node represents a customer (user), and edges represent friendship relationships between users. The node features are extracted from the user's reviews and profile information.
- **Products**[7] (Hu et al., 2020). It is a multi-class node classification dataset collected from Amazon. Each node represents a product, and edges represent the co-purchasing relationships. The node features are derived from the product descriptions using the bag-of-words model followed by Principal Component Analysis (PCA).

C.2 Kernels

Three popular kernels, Gaussian, Matern-0.5, and Arc-cosine, are used in the experiments for their strong representation power (Cho & Saul, 2009; Chen & Xu, 2021).

- Gaussian kernel is defined as:

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2\sigma^2}\right), \quad \sigma > 0. \tag{A.16}$$

[2] https://www.dgl.ai/dgl_docs/api/python/dgl.data.FlickrDataset
[3] https://ogb.stanford.edu/docs/nodeprop/#ogbn-proteins
[4] https://ogb.stanford.edu/docs/nodeprop/#ogbn-arxiv
[5] https://www.dgl.ai/dgl_docs/generated/dgl.data.RedditDataset.html
[6] https://www.dgl.ai/dgl_docs/generated/dgl.data.YelpDataset.html
[7] https://ogb.stanford.edu/docs/nodeprop/#ogbn-products

Algorithm A.2: LAPTOP

Input : Graph Laplacian $\boldsymbol{L}$, labels $\boldsymbol{y}$, kernel function $\mathcal{K}(\cdot,\cdot)$, loss function $\xi_{\boldsymbol{y}}$, parameters $c_M, c_L, \rho > 0$, block size b, iteration limitations T_{admm} and T_{bcd}.

Initialize: $\boldsymbol{\alpha}_0 = \mathbf{0}, \boldsymbol{u}_0 = \mathbf{0}, \boldsymbol{\lambda}_0 = \mathbf{0}$, and the accumulator $\boldsymbol{v} \leftarrow \mathbf{0}$ (which maintains $\boldsymbol{v} = \boldsymbol{K}\boldsymbol{\alpha}$);

for $k = 0, \cdots, T_{\mathsf{admm}}$ **do**

  // $\boldsymbol{u}$-step: Laplacian regularized loss minimization

  Update $\boldsymbol{u}_{k+1}$ by optimizing (7) via Newton or APGD;

  Let $\boldsymbol{q}_k \leftarrow \boldsymbol{u}_{k+1} + \boldsymbol{\lambda}_k/\rho$;

  // $\boldsymbol{\alpha}$-step: KRR subproblem

  for $t = 0, \cdots, T_{\mathsf{bcd}}$ **do**

    Randomly pick a block $\mathcal{B} \in \{\mathcal{B}_1, \cdots, \mathcal{B}_{\lceil n/b \rceil}\}$;

    $\boldsymbol{A} \leftarrow \boldsymbol{K}_{\mathcal{B},\mathcal{B}} + \boldsymbol{I}/\rho$, $\boldsymbol{e}_{\mathcal{B}} \leftarrow \boldsymbol{q}_{\mathcal{B}} - \boldsymbol{v}_{\mathcal{B}} - \boldsymbol{\alpha}_{\mathcal{B}}/\rho$;

    $\boldsymbol{s} \leftarrow \mathtt{TCG}(\boldsymbol{A}, \boldsymbol{e}_{\mathcal{B}})$ via Algorithm A.1;

    Update $\boldsymbol{\alpha}_{\mathcal{B}} \leftarrow \boldsymbol{\alpha}_{\mathcal{B}} + \boldsymbol{s}$, and the accumulator $\boldsymbol{v} \leftarrow \boldsymbol{v} + \boldsymbol{K}_{:,\mathcal{B}}\boldsymbol{s}$;

  end

  // Update dual variable $\boldsymbol{\lambda}$

  Update $\boldsymbol{\lambda}_{k+1} \leftarrow \boldsymbol{\lambda}_k + \rho(\boldsymbol{u}_{k+1} - \boldsymbol{v})$;

end

return $\boldsymbol{\alpha}$.

The bandwidth σ is set to the median of pairwise distances of (a subset of) training data, known as the median heuristic (Garreau et al., 2017).

- Matern05 kernel is defined as:

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2}{\sigma}\right), \quad \sigma > 0. \tag{A.17}$$

  This kernel is shown to have the same RKHS as the NTK of ReLU neural networks (Chen & Xu, 2021). The bandwidth is set by the same median heuristic.

- Arc-cosine kernels are closely related to infinite-width neural networks (Cho & Saul, 2009), and we use the zeroth-order version defined as:

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = 1 - \frac{1}{\pi}\arccos\left(\frac{\boldsymbol{x}^\top \boldsymbol{x}'}{\|\boldsymbol{x}\|_2 \|\boldsymbol{x}'\|_2}\right). \tag{A.18}$$

C.3 IMPLEMENTATION DETAILS

The experiments were conducted using a single RTX 4090 GPU with 24GB memory. The implementation of LAPTOP is based on PyTorch and follows Algorithm A.2. We provide a minimal reproducible code at https://anonymous.4open.science/r/LapTop-paper-BCB3.

We use the one-versus-rest strategy for multi-class datasets. Especially, we transform the multi-class labels into binary labels for each class, i.e., the label matrix $\boldsymbol{Y} \in \{-1, 1\}^{n \times \kappa}$, where κ is the number of classes, is defined as:

$$Y_{ij} = \begin{cases} 1, & \text{If node } i \text{ belongs to class } j, \\ -1, & \text{Otherwise.} \end{cases} \tag{A.19}$$

Then in model training, we train κ predictors: $\{f_1, \cdots, f_\kappa\}$ in parallel, where f_j is the predictor for class j. Similarly, in the test phase, the final predicted label is the class with the largest prediction value:

$$\hat{y} = \underset{j \in [\kappa]}{\operatorname{argmax}}\, f_j(\boldsymbol{x}).$$

For the multi-label datasets Yelp and Proteins, we use the following label matrix:

$$Y_{ij} = \begin{cases} 1, & \text{If node } i \text{ has label } j, \\ -1, & \text{Otherwise.} \end{cases} \tag{A.20}$$

In the test phase, Yelp assigns label j to node i if $f_j(\boldsymbol{x}_i) > 0$, while Proteins is evaluated by ROC-AUC on the continuous scores $\{f_j(\boldsymbol{x}_i)\}$.

C.4 LABEL-PROPAGATION BASELINES

We report harmonic/Laplacian label propagation (Zhou et al., 2004) and Poisson learning (Calder et al., 2020), solved by conjugate gradient with residual tolerance 10^{-4}. Metrics match Table 1: accuracy on Flickr, Arxiv, Reddit, and Products; mean ROC-AUC on Proteins; micro-F1 on Yelp. Poisson learning diverges on Yelp and Products: the residual rises from 251 to 4.97×10^4 by step 20 on Yelp, and from 19 to 8×10^3 on Products. Both graphs have many connected components (6,288 on Yelp), and the normalized Laplacian is incompatible across components. On Products, Laplacian attains 76.64% test accuracy overall, 78.71% on the 2,152,259 test nodes connected to training labels and 3.62% on the remaining 60,832 disconnected nodes. Arxiv is a single connected component, where Poisson Learning is capped at 160 CG steps with a final residual of 9.8×10^{-4} (above the 10^{-4} tolerance). Validation scores (Laplacian / Poisson) are $47.22\%/42.16\%$ on Flickr, $92.53\%/87.63\%$ on Reddit, $83.77\%/78.91\%$ ROC-AUC on Proteins, and 30.99% micro-F1 for Laplacian on Yelp. On Arxiv, Laplacian/Poisson report test $66.73\%/48.84\%$, validation $68.34\%/54.44\%$, and training $100\%/78.48\%$. On Products, Laplacian reports test 76.64%, validation 90.54%, and training 100%.

C.5 INDUCTIVE INFERENCE PROTOCOL

For the inductive evaluation in Table 3, all models are trained on the subgraph induced by the training and validation nodes, and the test nodes together with their incident edges are revealed only at inference. All hyperparameters are selected by the validation performance within the training subgraph, and test labels are never used at any stage. For the smoothing variants (KRR + smooth and LAPTOP + smooth), a single $\boldsymbol{u}$-step is performed on the revealed graph, initialized at the base predictor's scores, with the smoothing hyperparameters (c_L, c_M, and the iteration budget) selected on validation as well; the same smoothing step and selection protocol are applied to both base predictors. Polynormer is trained on the same training subgraph without access to test nodes or edges; at inference, it is evaluated on the revealed graph with its global phase enabled, using the hyperparameters tuned on the validation split.

C.6 BASELINES, BUDGETS, AND PROFILING

Baseline sources. The results of GCN on Yelp and GraphSAINT on Reddit are taken from the original papers, and all other GNN/GT results in Table 1 are produced by our own runs. Our own GNN runs use the standard hyperparameters from the official implementations; we note that Luo et al. (2024) report higher scores for classic GNNs with extensive tuning (e.g., GraphSAGE on Products), which are not directly comparable to the standard-protocol numbers in Table 1.

Paired graph-free variants. The graph-free variant ($c_M = 0$, plain KRR) in Figure 2 and Table 3 is paired with its LAPTOP counterpart under the same kernel and the same hyperparameter search space: LAPTOP-SVM^2 on Flickr, and LAPTOP-SVM on Yelp and Products.

Budgets and seeds. We set the number of Nyström landmarks of LapRLS-NyPCG to the largest value that fits the GPU memory and keeps the numerical convergence, i.e., $s = 22000$ on Flickr, Proteins, Arxiv, Reddit, and $s = 12000$ for Yelp, and Products. The inner budgets of LAPTOP are $T_{\mathsf{BCD}} = 1000$ block updates per ADMM iteration and $T_{\mathsf{CG}} = 25$ per linear solve. All randomized results are averaged over 5 independent seeds.

Profiling. Table 2 profiles LAPTOP-SVM; the memory is the peak GPU memory during training, and the time is the total wall-clock training time.

In the experiments, the four LAPTOP models are summarized in Table A.3. We fix $\rho = 16$ for our experiments, which is shown to be a moderate value balancing convergence rate and the final performance. The hyperparameters c_L, c_M are tuned from the set $\{2^i : i = -5, -4, \cdots, 10\}$ with grid search. Table A.4 shows the hyperparameters for the LAPTOP models.

C.7 EXTENDED DISCUSSION ON CONVERGENCE

As shown in Figure 3(d), the objective of LAPTOP decreases monotonically during the ADMM iterations and converges within 10 ADMM iterations on Products for both LAPTOP-SVM and

Table A.3: Lookup table of LAPTOP models in the experiments. Denote $\sigma(z) = 1/(1+\exp(-z))$ the sigmoid function, and $\mathbb{I}_{(\cdot)}$ the indicator function that returns 1 if the condition holds and 0 otherwise.

Model	Loss function	$\boldsymbol{u}$-step solver	$\nabla\xi_y$	$\nabla^2\xi_y$	$\mathsf{prox}_{\gamma,\xi_y}(z)$
LAPTOP-RLS	$(u-y)^2/2$	Newton/APGD	$u-y$	1	$\frac{1}{1+\gamma}z + \frac{\gamma}{1+\gamma}y$
LAPTOP-SVM	$\max\{0, 1-uy\}$	APGD	$-y\mathbb{I}_{yu<1}$	Not applicable	See (A.2)
LAPTOP-SVM^2	$\max\{0, 1-uy\}^2/2$	Newton/APGD	$-y\max\{0, 1-uy\}$	$\mathbb{I}_{yu<1}$	See (A.8)
LAPTOP-Log	$\log(1+\exp(-uy))$	Newton	$-y\sigma(-uy)$	$\sigma(uy)\sigma(-uy)$	Not prox-friendly

Table A.4: The tuned hyperparameters for LAPTOP.

Methods	Parameters	Datasets					
		Flickr	Proteins	Arxiv	Reddit	Yelp	Products
LAPTOP-RLS	c_L	128	512	512	1024	1024	512
	c_M	64	32	4	64	2^{-5}	128
	ADMM iter.	5	10	10	3	30	10
	Block size	1024	8192	4096	4096	8192	8192
	Kernel	Gaussian	Matern05	arccos	arccos	Matern05	arccos
LAPTOP-SVM	c_L	128	256	512	1024	1024	512
	c_M	64	16	4	64	2^{-5}	128
	ADMM iter.	5	10	10	3	30	10
	Block size	1024	4096	4096	4096	8192	8192
	Kernel	Gaussian	Matern05	arccos	arccos	Matern05	arccos
LAPTOP-SVM^2	c_L	128	256	512	512	512	512
	c_M	64	16	4	64	2^{-5}	64
	ADMM iter.	5	10	10	3	30	10
	Block size	1024	4096	4096	4096	8192	8192
	Kernel	Gaussian	Matern05	arccos	arccos	Matern05	arccos
LAPTOP-Log	c_L	128	256	512	1024	1024	512
	c_M	64	16	4	128	2^{-5}	32
	ADMM iter.	5	10	10	3	30	10
	Block size	1024	8192	4096	4096	8192	8192
	Kernel	Gaussian	Matern05	arccos	arccos	Matern05	arccos

LAPTOP-Log, validating the effectiveness and numerical stability of the proposed optimization algorithm.

We further study the convergence of the two subproblems, taking LAPTOP-RLS on Products as an example. Figure 3(a) compares APGD and Newton's method for the $\boldsymbol{u}$-step with a log-scaled y-axis. It can be observed that APGD converges linearly as expected. Note that for LAPTOP-RLS, the $\boldsymbol{u}$-step subproblem (7) is quadratic, so Newton's method converges in a single outer step; the Newton curve in Figure 3(a) reports the inner TCG iterations for solving this single Newton system, which reaches a high-precision solution within 15 iterations. Therefore, Newton's method is preferred for smooth loss functions. Nevertheless, APGD has less per-iteration cost and is generic for various losses with linear convergence. Figure 3(b) shows the convergence of BCD with block size 8192. In practice, we find that BCD almost converges within 800 block updates, i.e., roughly 3 passes (epochs) over all variables.

IMPACT STATEMENT

This work revisits and revitalizes Laplacian Regularized Kernel Methods (LapKM), the regularizer-side paradigm that has been overlooked in the era of deep graph learning. By resolving the long-standing computational bottlenecks and numerical instability, the proposed LAPTOP framework successfully scales classical kernel methods to millions of nodes without low-rank kernel approximations. This suggests that the scalability limit of kernel methods was not a theoretical bound but an optimization challenge, which we have overcome via our ADMM-based decoupling strategy. Empirically, it demonstrates promising performance matching GNNs on million-scale datasets, providing a strong alternative to deep models for large-scale graph learning tasks. This may encourage

the community to re-examine the potential of theoretically grounded shallow models, fostering a synergistic research ecosystem where classical regularization theory complements and co-evolves with deep representation learning.