# Graph Machine Learning through the Lens of Bilevel Optimization

**Amber Yijia Zheng**[1]
Purdue University

**Tong He**
Amazon Web Services

**Yixuan Qiu**
Shanghai University of
Finance and Economics

**Minjie Wang**
Amazon Web Services

**David Wipf**
Amazon Web Services

## Abstract

Bilevel optimization refers to scenarios whereby the optimal solution of a lower-level energy function serves as input features to an upper-level objective of interest. These optimal features typically depend on tunable parameters of the lower-level energy in such a way that the entire bilevel pipeline can be trained end-to-end. Although not generally presented as such, this paper demonstrates how a variety of graph learning techniques can be recast as special cases of bilevel optimization or simplifications thereof. In brief, building on prior work we first derive a more flexible class of energy functions that, when paired with various descent steps (e.g., gradient descent, proximal methods, momentum, etc.), form graph neural network (GNN) message-passing layers; critically, we also carefully unpack where any residual approximation error lies with respect to the underlying constituent message-passing functions. We then probe several simplifications of this framework to derive close connections with non-GNN-based graph learning approaches, including knowledge graph embeddings, various forms of label propagation, and efficient graph-regularized MLP models. And finally, we present supporting empirical results that demonstrate the versatility of the proposed bilevel lens, which we refer to as *BloomGML*, referencing that *BiLevel Optimization Offers More Graph Machine Learning*. Our code is available at `https://github.com/amberyzheng/BloomGML`. Let graph ML *bloom*.

---

[1]Contribution during AWS Shanghai AI Lab internship.

## 1 Introduction

Graph machine learning covers a wide range of modeling tasks involving graph-structured data, where cross-instance/node dependencies are reflected by edges. As a classical example, label propagation (Zhou et al., 2003) and its many offshoots represent a semi-supervised learning approach whereby observed node labels are iteratively spread across the graph to unlabeled nodes. In a related vein, various forms of graph-regularized MLP models (Ando and Zhang, 2006; Hu et al., 2021; Zhang et al., 2023) share node representations across edges to penalize misalignment with network structure during training. And as a third example more narrowly focused on certain heterogeneous graphs, non-parametric knowledge graph embedding (KGE) models (Bordes et al., 2013) produce node and relation-type embeddings that have been trained to differentiate factual knowledge triplets, composed of head and tail nodes connected by an edge relation, from spurious ones.

More recently, graph neural networks (GNNs) have emerged as a promising class of predictive models for handling tasks such as node classification or link prediction (Kipf and Welling, 2016; Hamilton et al., 2017; Xu et al., 2019; Veličković et al., 2017; Zhou et al., 2020). Central to a wide variety of GNN architectures are layers composed of three components: a *message function*, which bundles information for sharing with neighbors, an *aggregation function* that fuses all the messages from neighbors, and an *update function* that computes the layer-wise output embedding for each node. Collectively, these functions enable the layer-by-layer propagation of information across the graph to facilitate downstream tasks (Kipf and Welling, 2016; Hamilton et al., 2017; Kearnes et al., 2016).

In the past, the graph ML models described above have primarily been motivated from diverse perspectives, without necessarily a single, transparent lens with which to evaluate their commonalities. To make strides towards a more cohesive graph ML narrative,

we propose to operate from a unifying vantage point afforded by bilevel optimization (Sinha et al., 2018; Wang et al., 2016). The latter refers to an optimization problem characterized by two interconnected levels, whereby the optimal solution of a *lower-level* objective serves as input features to an *upper-level* loss. Within the context of graph ML, we will demonstrate how the lower-level in particular can induce interpretable model architectures, while exposing underappreciated similarities across seemingly disparate paradigms. After providing relevant background material in Section 2, our primary contributions throughout the remainder of the paper can be summarized as follows:

- In Section 3 we introduce a broad class of lower-level energy functions that, when combined with appropriate optimization steps, give rise to efficient GNN message-passing layers inheriting interpretable inductive biases of energy minimizers. Moreover, subject to certain technical assumptions, we demonstrate that these optimization-induced layers can replicate arbitrary message and aggregation functions, while providing a flexible approximation for the update function. Hence we conditionally isolate any appreciable approximation error to the latter, and in doing so, articulate a transparent design space for GNN architectures produced by bilevel optimization.

- Based on the above, in Section 4 we propose a broadly-applicable bilevel optimization framework called BloomGML, and establish that several notable special cases effectively reproduce and unify non-GNN-based graph learning approaches. In particular, we demonstrate an equivalence between traditional KGE learning and the training of an implicit GNN, with parameters given by relational embeddings and a graph formed from positive and negative knowledge triplets. This association explains recent empirical findings in the literature while motivating new interpretable approaches to modeling over knowledge graphs.

- Finally, in Section 5 we present experiments that highlight the flexibility and explanatory power of candidate models motivated by our proposed bilevel optimization framework and attendant analysis thereof.

## 2 Background

In this section, we first introduce GNN architectures and their constituent message-passing layers as applied to node classification tasks (more general tasks will be addressed later). We then describe an existing, popular special case induced by bilevel optimization, followed by a discussion of both advantages and limitations.

### 2.1 Message Passing Graph Neural Networks

Let $\mathcal{G} = \{\mathcal{V}, \mathcal{R}, \mathcal{E}\}$ denote a heterogeneous graph with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{R}|$ edge types. Each edge $e \in \mathcal{E}$ is composed of a triplet via $e = (u, r, v)$, with nodes $u, v \in \mathcal{V}$ and relation type $r \in \mathcal{R}$. The more general heterogeneous case defaults to a homogeneous graph when $m = 1$. For any given node $v$, we also define the set of (1-hop) neighbors as $\mathcal{N}_v := \{(u, r) : (u, r, v) \in \mathcal{E}\}$. Additionally, associated with each node is a $d$-dimensional feature vector and a $c$-dimensional label vector which stack to form $\mathbf{X} \in \mathbb{R}^{n \times d}$ and $\mathbf{Y} \in \mathbb{R}^{n \times c}$, respectively. The latter reflects our focus on node classification tasks within this section for simplicity of exposition; the extension to link prediction (whereby labels correspond with supervision over edges rather than individual nodes) follows naturally.

When presented with a graph $\mathcal{G}$ so defined, a message-passing GNN (or MP-GNN) of depth $L$ produces a layer-wise sequence of node embeddings $\{\mathbf{H}^{(l)}\}_{l=1}^{L}$, where $\mathbf{H}^{(l)} = \{\mathbf{h}_v^{(l)}\}_{v \in \mathcal{V}} \in \mathbb{R}^{n \times d}$ represents the aggregated set of embeddings $\mathbf{h}_v^{(l)}$ at the $l$-th layer. Moreover, for each $l$, $\mathbf{H}^{(l)}$ is computed from $\mathbf{H}^{(l-1)}$ using a composition of three functions characteristic of MP-GNN architectures (Gilmer et al., 2017; Hamilton, 2020). We formalize this composition via the following definition:

**Definition 1** *An MP-GNN layer is formed via the composition of three functions:*

1. *A* <u>*message function*</u> *$f_M$ that computes $\boldsymbol{\mu}_{(u,r,v)}^{(l)} := f_M(\mathbf{h}_u^{(l-1)}, r, \mathbf{h}_v^{(l-1)}) \in \mathbb{R}^d$ for any edge $e = (u, r, v) \in \mathcal{E}$, i.e., given the head and tail nodes and the relation type, $f_M$ produces an embedding for the corresponding edge;*

2. *A permutation-invariant* <u>*aggregation function*</u> *$f_A$ that computes $\mathbf{a}_v^{(l)} := f_A(\{\boldsymbol{\mu}_{(u,r,v)}^{(l)} : (u, r) \in \mathcal{N}_v\}) \in \mathbb{R}^d$ for all $v \in \mathcal{V}$, meaning for each node the set function $f_A$ aggregates all messages associated with other nodes sharing an edge;*

3. *An* <u>*update function*</u> *$f_U$ that computes the embeddings of the next layer as $\mathbf{h}_v^{(l)} = f_U(\mathbf{h}_v^{(l-1)}, \mathbf{a}_v^{(l)}, \mathbf{x}_v)$ for all $v \in \mathcal{V}$, i.e., the embeddings from the previous layer are combined with the aggregated messages and (optionally) the original input node features $\mathbf{x}_v \in \mathbf{X}$.*

The permutation-invariance of $f_A$ notwithstanding, all three components defining the MP-GNN layer can in principle be any parameterized differentiable (almost everywhere) functions suitable for learning with SGD, etc. Hence different MP-GNN architectures are more-or-less tantamount to different selections for $f_M$, $f_A$,

and $f_U$. We adopt $\mathbf{W}$ to denote the bundled set of all trainable parameters within $\{f_M, f_A, f_U\}$.

Conditioned on executing $L$ layers per the schema of Definition 1 culminating in the embeddings $\mathbf{H}^{(L)}$, model training proceeds by minimizing the loss

$$\ell_{up}(\mathbf{W}, \Theta) := \sum_{v \in \mathcal{V}'} \mathcal{D}\left[ g\left(\mathbf{h}_v^{(L)}(\mathbf{W}); \Theta\right), \mathbf{y}_v\right], \quad (1)$$

where $\mathbf{h}_v^{(L)}(\mathbf{W}) \equiv \mathbf{h}_v^{(L)}$ reflects the explicit dependency of $\mathbf{H}^{(L)}$ on $\mathbf{W}$. In this expression, the output layer $g : \mathbb{R}^d \to \mathbb{R}^c$ is a differentiable node-wise function parameterized by $\Theta$, $\mathcal{V}' \subset \mathcal{V}$ indicates the subset of training instances with observable labels, and $\mathcal{D}$ is a discriminator function, e.g., cross-entropy for classification. Finally, the "up" in $\ell_{up}$ reflects the fact that this function will later serve as an *upper-level* loss when embedded within a bilevel optimization setting to be described next.

## 2.2 Graph-Centric Bilevel Optimization

Our generic MP-GNN setting thus far narrows to the realm of bilevel optimization (Sinha et al., 2018) when we infuse the latent embeddings $\{\mathbf{H}^{(l)}\}_{l=1}^{L}$ from above with additional layer-wise structure determined by a second, *lower-level* loss $\ell_{low}(\,\cdot\,; \mathbf{W}, \mathcal{G})$, which likewise depends on $\mathbf{W}$ and $\mathcal{G}$. Specifically, suppose that for all layers $l$, we enforce that

$$\ell_{low}\left(\mathbf{H}^{(l)}; \mathbf{W}, \mathcal{G}\right) \leq \ell_{low}\left(\mathbf{H}^{(l-1)}; \mathbf{W}, \mathcal{G}\right),$$
$$\text{and ideally} \quad \mathbf{H}^{(L)} \approx \arg\min_{\mathbf{H}} \ell_{low}\left(\mathbf{H}; \mathbf{W}, \mathcal{G}\right) \quad (2)$$

for $L$ sufficiently large. In other words, at each layer the composite embedding model $f_U \circ f_A \circ f_M$ that updates $\mathbf{H}^{(l)}$ per Definition 1 dually serves as an algorithm producing descent steps along the loss surface of $\ell_{low}$; hence the coincident dependency of both $\ell_{low}$ and $f_U \circ f_A \circ f_M$ on $\mathbf{W}$ and $\mathcal{G}$. When combined with (1), we arrive at our bilevel destination, with embeddings that iteratively minimize $\ell_{low}$ forming predictive features for end-to-end training within $\ell_{up}$.

**A Representative Special Case.** Motivated by Zhou et al. (2003) in the context of homogeneous graphs, we consider the energy $\ell_{low}(\mathbf{H}; \mathbf{W}, \mathcal{G}) :=$

$$\sum_{v \in \mathcal{V}} \tfrac{1}{2} \|\mathbf{h}_v - \pi(\mathbf{x}_v; \mathbf{W})\|_2^2 + \tfrac{\lambda}{2} \sum_{(u,v) \in \mathcal{E}} \|\mathbf{h}_u - \mathbf{h}_v\|_2^2, \quad (3)$$

where $\pi$ is a parameterized input model (e.g., an MLP with weights $\mathbf{W}$), and $\lambda > 0$ is a trade-off parameter. This factor balances local consistency w.r.t. the input model $\pi$ and global smoothness across the graph.

Next, starting from some $\mathbf{H}^{(0)}$, we may iterate $L$ gradient descent steps along (3) to obtain $\mathbf{H}^{(L)}$. And along

this trajectory for each layer $l$, it is straightforward to show (see Appendix B) that the resultant updates for each node $v$ adhere to the stipulations of Definition 1, and can be equivalently computed as

$$\boldsymbol{\mu}_{(u,r,v)}^{(l)} = \mathbf{h}_u^{(l-1)} - \mathbf{h}_v^{(l-1)}, \quad \mathbf{a}_v^{(l)} = \sum_{u \in \mathcal{N}_v} \boldsymbol{\mu}_{(u,r,v)}^{(l)}, \quad \text{and}$$

$$\mathbf{h}_v^{(l)} = (1 - \gamma)\mathbf{h}_v^{(l-1)} + \gamma\lambda\mathbf{a}_v^{(l)} + \gamma\pi(\mathbf{x}_v; \mathbf{W}), \quad (4)$$

where $\gamma$ is the gradient step-size parameter. From this expression, we observe that the effective message-passing layer involves weighted skip connections from the previous layer and input model, along with a permutation-invariant summation of the embeddings from neighboring nodes. And provided $\pi$ is differentiable w.r.t. $\mathbf{W}$, the update for $\mathbf{h}_v^{(l)}$ computed via (4) is differentiable as well. Hence we can substitute into (1) and train the entire system end-to-end using SGD over $\mathbf{W}$ and $\Theta$.

Using various preconditioners and reparameterizations, it has been widely noted that model layers constructed via (4), or variations thereof, directly correspond with various popular MP-GNN architectures (Chen et al., 2021; Ma et al., 2021; Pan et al., 2020; Xue et al., 2023; Yang et al., 2021; Zhang et al., 2020; Zhu et al., 2021b; Zhou et al., 2021; Di Giovanni et al., 2023). Hence this particular instance of bilevel optimization can be interpreted as training a GNN node classification model with layers designed to minimize (3).

**Why Construct GNN Architectures this Way?** As one notable example, across customer-facing applications it is often desirable to know which factors influence the output predictions of a GNN model (Ying et al., 2019). Relevant to providing such explanatory details, the embeddings produced by bilevel optimization as in Section 2.2 contain additional information when contextualized w.r.t. their role in the $\ell_{low}$. For instance, if some node embedding $\mathbf{h}_v^{(L)}$ within (3) is very far from $\pi(\mathbf{x}_v; \mathbf{W})$ relative to other nodes, it increases our confidence that subsequent predictions may be based on network effects rather than an irrelevant local feature $\mathbf{x}_v$. We will empirically explore this possibility in Section 5. Alternatively, in the context of heterophilic graphs, we can examine the distribution of $\|\mathbf{h}_v - \mathbf{h}_u\|_2$ across nodes sharing an edge to loosely calibrate the degree of heterophily, and possibly counteract its impact via appropriate modifications. And as a final dimension of motivation, the bilevel perspective provides natural entry points for the scalable training of models with arbitrary depth (Xue et al., 2023) or interpretable integration with offline sampling (Jiang et al., 2023).

### 2.3 Unresolved Limitations

Despite the widespread use of bilevel optimization in forming and/or analyzing GNN layers, prior work is largely confined to the exploration of narrow instances that follow from quite specific (usually quadratic) choices for $\ell_{low}$ paired with vanilla gradient descent, with application to homogeneous graphs; the example presented in Section 2.2 follows this paradigm. As of yet, there has been no clear delineation of a broad design space of possibilities. We take initial steps in this direction next.

## 3 Towards More Flexible GNNs from Bilevel Optimization

In the previous section, we applied a *specific* iterative algorithm, gradient descent with step-size $\gamma$, to a *particular* energy function, $\ell_{low}(\mathbf{H}; \mathbf{W}, \mathcal{G})$ from (3), and the resulting descent updates produced the GNN message-passing layers given by (4). Within this relatively narrow design space, the only algorithmic flexibility is in the choice of $\gamma$, and on the energy side, we are limited to tuning the trade-off parameter $\lambda$ and the input model $\pi$. Not surprisingly then, (4) is far less flexible/expressive relative to the general form from Definition 1. We now seek to reduce this gap via the following formulation.

To begin we require some additional notation. Let $\mathcal{L}(\mathcal{G})$ be a function space of interest dependent on graph $\mathcal{G}$, where each function $\ell_{low} \in \mathcal{L}(\mathcal{G})$ is specified as $\ell_{low} : \mathcal{H} \to \mathbb{R}$ over some node embedding domain $\mathcal{H} \subseteq \mathbb{R}^{n \times d}$. We then define $\mathcal{A} : \mathcal{L}(\mathcal{G}) \times \mathcal{H} \to \mathcal{H}$ as an arbitrary iterative mapping/algorithm that satisfies

$$\ell_{low} \left( \mathcal{A}[\ell_{low}, \mathbf{H}] \right) \leq \ell_{low}(\mathbf{H}) \quad \forall \ell_{low} \in \mathcal{L}(\mathcal{G}), \forall \mathbf{H} \in \mathcal{H}. \tag{5}$$

This expression merely entails that if we apply $\mathcal{A}$ to $\ell_{low}$ initialized at $\mathbf{H}$, the resulting iteration produces embeddings that reduce (or leave unchanged) $\ell_{low}$. We are then poised to ask a central motivating question:

*Can we find a sufficiently flexible pairing of some $\mathcal{A}$ and compatible $\ell_{low} \in \mathcal{L}(\mathcal{G})$ such that $\mathcal{A}[\ell_{low}, \mathbf{H}]$ closely approximates, to the extent possible, any composite message-passing layer $f_U \circ f_A \circ f_M$ which follows from the stipulations of Definition 1?*

To make initial progress in answering this question, we first convert Definition 1 to what we will refer to as a *canonical form* designed to minimize, without loss of generality, the non-uniqueness of the decomposition of $f_U \circ f_A \circ f_M$ into respective message, aggregation, and update functions. In so doing we obtain a more transparent entry point for isolating precisely where more-or-less exact alignment between $\mathcal{A}[\ell, \mathbf{H}]$ and $f_U \circ f_A \circ f_M$ is possible, and where non-trivial gaps remain.

### 3.1 Canonical Form of MP-GNN Layers

Given the composite function $f_U \circ f_A \circ f_M$, there is no unique decomposition into its constituent parts, even within the constraints of Definition 1. To this end, we introduce the following simplification of MP-GNN layers that, as we will later show, leads to a unique decomposition (up to inconsequential linear transformations) while maintaining expressiveness:

**Definition 2** *An MP-GNN layer in canonical form is defined as the composition of three functions:*

1. *A <u>message function</u> $\widetilde{f}_M$ that computes $\boldsymbol{\mu}^{(l)}_{(u,r,v)} := \widetilde{f}_M(\mathbf{h}^{(l-1)}_u, r, \mathbf{h}^{(l-1)}_v) \in \mathbb{R}^{\widetilde{d}}$ for any edge $e = (u, r, v) \in \mathcal{E}$;*

2. *The <u>aggregation function</u> $\widetilde{f}_A$ that computes $\mathbf{a}^{(l)}_v := \widetilde{f}_A(\{\boldsymbol{\mu}^{(l)}_{(u,r,v)} : (u,r) \in \mathcal{N}_v\}) = \sum_{(u,r) \in \mathcal{N}_v} \boldsymbol{\mu}^{(l)}_{(u,r,v)}$;*

3. *An <u>update function</u> $\widetilde{f}_U$ that computes the embeddings of the next layer as $\mathbf{h}^{(l)}_v = \widetilde{f}_U(\mathbf{h}^{(l-1)}_v, \mathbf{a}^{(l)}_v, \mathbf{x}_v)$ for all $v \in \mathcal{V}$.*

We remark that there still remains a degree of non-uniqueness in Definition 2 in that a linear transformation of $\boldsymbol{\mu}^{(l)}_{(u,r,v)}$ introduced by $\widetilde{f}_M$ could pass through $\widetilde{f}_A$ and be absorbed into $\widetilde{f}_U$ without changing the composite function $\widetilde{f}_U \circ \widetilde{f}_A \circ \widetilde{f}_M$. However, this ambiguity is inconsequential for our purposes, and nonlinear transformations cannot analogously pass through $\widetilde{f}_A$ without changing the form of the resulting composite function.

Additionally, although it may superficially appear as though we have lost expressive power in moving to the canonical form, this is actually not the case:

**Proposition 1** *For any $f_U \circ f_A \circ f_M$ adhering to Definition 1, there exists a canonical form $\widetilde{f}_U \circ \widetilde{f}_A \circ \widetilde{f}_M$ following Definition 2 that provides an arbitrarily close approximation.*

Appendix D contains the proof. So indeed, we can work with the canonical form without any appreciable loss of generality. From here, we will first narrow our scope in Section 3.2 to reproducing $f_M$ or $\widetilde{f}_M$ using some $\mathcal{A}[\ell_{low}, \cdot ]$; later in Section 3.3 we zoom out and address approximations to the full canonical form.

### 3.2 A Priori Constraint Approximating Message Functions $f_M$ or $\widetilde{f}_M$

To begin, we introduce the following definition:

**Definition 3** *We say that a message function $f_M$ satisfies a <u>gradient representation criteria</u> if*

$$
\begin{array}{rcl}
f_M(\mathbf{h}_u, r, \mathbf{h}_v) & = & \dfrac{\partial \zeta(\mathbf{h}_u, \mathbf{h}_v; r)}{\partial \mathbf{h}_v} \\[2mm]
f_M(\mathbf{h}_v, r^{-1}, \mathbf{h}_u) & = & \dfrac{\partial \zeta(\mathbf{h}_u, \mathbf{h}_v; r)}{\partial \mathbf{h}_u}
\end{array} \tag{6}
$$

*for some function $\zeta : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ with Lipschitz continuous gradients, where $r^{-1}$ is the inverse relation of $r$. Moreover, we extend this definition to the canonical form $\widetilde{f}_M$ via*

$$
\begin{array}{rcl}
\widetilde{f}_M(\mathbf{h}_u, r, \mathbf{h}_v) & = & \dfrac{\partial \widetilde{\zeta}(\Phi^\top \mathbf{h}_u, \mathbf{z}; r)}{\partial \mathbf{z}} \bigg|_{\mathbf{z} = \Phi^\top \mathbf{h}_v} \\[3mm]
\widetilde{f}_M(\mathbf{h}_v, r^{-1}, \mathbf{h}_u) & = & \dfrac{\partial \widetilde{\zeta}(\mathbf{z}, \Phi^\top \mathbf{h}_v; r)}{\partial \mathbf{z}} \bigg|_{\mathbf{z} = \Phi^\top \mathbf{h}_u}
\end{array} \tag{7}
$$

*for some $\Phi \in \mathbb{R}^{d \times \widetilde{d}}$ and function $\widetilde{\zeta} : \mathbb{R}^{\widetilde{d}} \times \mathbb{R}^{\widetilde{d}} \to \mathbb{R}$ with Lipschitz continuous gradients.*

This definition describes message functions that can be expressed in terms of the gradient of an energy term, i.e., $\zeta$ or $\widetilde{\zeta}$. Moreover, if a pair of bidirectional message functions do *not* satisfy this criteria, then they cannot generally be obtained with a gradient-based $\mathcal{A}$ applied to any possible $\ell_{low}$. To see this, consider a graph with just two nodes and no node features for simplicity. Moreover, assume that the aggregation function is identity (there is only one message in either direction) and the update function is given by $f_U(\mathbf{h}_v^{(l)}, \boldsymbol{\mu}_{(u,r,v)}^{(l)}) = \mathbf{h}_v^{(l-1)} - \gamma \boldsymbol{\mu}_{(u,r,v)}^{(l)}$ for some $\gamma > 0$. We then observe that any possible $\ell_{low}$ can be expressed w.l.o.g. as $\ell_{low}(\mathbf{H}; \mathbf{W}, \mathcal{G}) = \zeta(\mathbf{h}_1, \mathbf{h}_2; r)$ for some function $\zeta$. Therefore, the only source for producing the two required message functions, meaning for $1 \to 2$ and $2 \to 1$ is taking gradients of the first and second arguments of $\zeta$.

### 3.3 Approximating the Full Canonical Form via Energy Optimization

To proceed, we first require an adequately flexible form of energy function with terms that are, in a sense to be formally quantified, sufficiently aligned with the canonical form serving as our target. Intuitively, there is a trade-off here: If this energy is too complex, then iterative optimization algorithms applied to it can produce update rules that deviate from Definition 2, e.g., the update for any given node $v$ could potentially involve all other nodes in the graph, as opposed to merely those in $\mathcal{N}_v$. In contrast, if the energy is too narrowly defined, we will be unable to match the expressive power of the composite $\widetilde{f}_U \circ \widetilde{f}_A \circ \widetilde{f}_M$.

Motivated by these considerations, we propose the family of energies given by

$$
\ell_{low}(\mathbf{H}; \mathbf{W}, \mathcal{G}) := \tag{8}
$$
$$
\sum_{(u,r,v) \in \mathcal{E}} f(\mathbf{h}_u, \mathbf{h}_v; r) + \sum_{v \in \mathcal{V}} \left[ \kappa(\mathbf{h}_v; \mathbf{x}_v) + \eta(\mathbf{h}_v) \right],
$$

where $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and $\kappa : \mathbb{R}^d \to \mathbb{R}$ are arbitrary differentiable functions, while $\eta : \mathbb{R}^d \to \mathbb{R}$ has no such requirements, e.g., it can be nonsmooth and discontinuous. Additionally, $\mathbf{W}$ refers to all parameters that may be included within $f$, $\kappa$, and $\eta$.

The role of $\eta$ is to allow for the introduction of node-wise constraints. For instance, if $\eta(\mathbf{h}) := \mathcal{I}_\infty[\mathbf{h} < 0]$, meaning a discontinuous indicator function with infinite weight applied element-wise to the entries of $\mathbf{h}$ less than zero, then we are effectively enforcing the constraint that node embeddings must be non-negative (Yang et al., 2021). To algorithmically handle this possibility, we also require the notion of a proximal operator (Parikh and Boyd, 2014), which is defined as

$$
\mathbf{prox}_{\eta,\gamma}(\mathbf{h}) := \arg\min_{\mathbf{z}} \left( \eta(\mathbf{z}) + \frac{1}{2\gamma} \|\mathbf{z} - \mathbf{h}\|^2 \right). \tag{9}
$$

Proximal operators $\mathbf{prox}_{\eta,\gamma} : \mathbb{R}^d \to \mathbb{R}^d$ of this form are useful for deriving descent algorithms involving constraints or nonsmooth terms such as $\eta$. We are now prepared to present our main result of this section, with discussion and interpretations to follow:

**Proposition 2** *For any canonical message-passing layer with constituent $\widetilde{f}_U$, $\widetilde{f}_A$, and $\widetilde{f}_M$ abiding by Definition 2 and $\widetilde{f}_U$ satisfying Definition 3, there exists an algorithm $\mathcal{A}$ and functions $f$, $\kappa$, and $\eta$ defining $\ell_{low}$ from (8) such that*

$$
\mathcal{A}[\ell_{low}, \ \cdot \ ] = \hat{f}_U \circ \widetilde{f}_A \circ \widetilde{f}_M, \tag{10}
$$

*where $\hat{f}_U : \mathbb{R}^d \times \mathbb{R}^{\widetilde{d}} \times \mathbb{R}^d \to \mathbb{R}^d$ is defined as*

$$
\hat{f}_U(\mathbf{h}, \mathbf{a}, \mathbf{x}) := \mathbf{prox}_{\eta,\gamma} \left( \mathbf{h} - \gamma \sigma(\mathbf{a}) \left[ \Phi \mathbf{a} + \kappa'(\mathbf{h}; \mathbf{x}) \right] \right) \tag{11}
$$

*with $\sigma(\mathbf{a}) : \mathbb{R}^{\widetilde{d}} \to \mathbb{R}^+ > 0$ as an arbitrary positive function, $\Phi \in \mathbb{R}^{d \times \widetilde{d}}$ as an arbitrary matrix, and $\kappa'$ as the first-order derivative of $\kappa$. Moreover, there exists a $\gamma' > 0$ such that for any $\gamma \in (0, \gamma']$,*

$$
\ell_{low}(\mathcal{A}[\ell_{low}, \mathbf{H}]; \mathcal{G}) \leq \ell_{low}(\mathbf{H}; \mathcal{G}). \tag{12}
$$

Several salient points are worth highlighting as we unpack this result:

- Once we have granted that $\widetilde{f}_U$ adheres to Definition 3, all approximation error between the target

composite message passing function $\widetilde{f}_U \circ \widetilde{f}_A \circ \widetilde{f}_M$ and the proposed $\mathcal{A}[\ell_{low}, \ \cdot \ ]$ is confined to mismatch in the respective update functions $\widetilde{f}_U$ and $\hat{f}_U$, not the message and aggregation functions $\widetilde{f}_M$ and $\widetilde{f}_A$.

- While $\hat{f}_U$ cannot exactly represent all possible $\widetilde{f}_U$, it maintains considerable degrees of freedom for replicating important special cases, including many/most commonly adopted update functions. This flexibility comes from three primary sources: namely, $\eta$, $\sigma$, and $\kappa$ can all be freely chosen to widen the expressiveness of $\hat{f}_U$; we note that $\eta$ and $\kappa$ are determined by $\ell_{low}$, while $\sigma$ is associated with $\mathcal{A}$ (see Appendix E for further details).

- The function $\eta$, through its influence on $\mathbf{prox}_{\eta,\gamma}$, can introduce a wide variety of nonlinear activations into the message passing layer. And in general, since $\eta$ is arbitrary, $\mathbf{prox}_{\eta,\gamma}$ can be any arbitrary proximal operator. For example, $\mathbf{prox}_{\eta,\gamma}$ can implement any non-decreasing function of the elements of $\mathbf{h}$ (Gribonval and Nikolova, 2020), such as popular ReLU activations among other things.

- The function $\sigma$ allows us to introduce softmax self-attention in the message-passing layer. This capability is exactly analogous to how an optimization perspective can introduce self-attention into typical Transformer layers (Yang et al., 2022a).

- Meanwhile, the function $\kappa$ provides for arbitrary interactions between node embeddings and input node features. It also allows us to directly generalize (4). More concretely, when we choose $\eta(\mathbf{h}) = 0$ (such that $\mathbf{prox}_{\eta,\gamma}$ degenerates to an identity mapping), $\sigma(\mathbf{a}) = 1$, and $\kappa'(\mathbf{h}; \mathbf{x}) = \mathbf{h} - \pi(\mathbf{x}; \mathbf{W})$, we exactly recover the message passing layer from (4).

- Increasing the expressiveness of $\hat{f}_U$ to more tightly match an arbitrary $\widetilde{f}_U$ is challenging and possibly infeasible (at least using standard descent methods for $\mathcal{A}$). See Appendix C for further discussion.

While the proof of Proposition 2 is deferred to Appendix E, it is based on assigning $\mathcal{A}$ to be a form of proximal gradient descent with preconditioning. Although this class of algorithm has attractive convergence guarantees, there remains potential to improve the convergence rate. Within the present context, this is tantamount to reducing the actual number of message-passing layers needed to approach a minimum of $\ell_{low}$. We conclude by noting that it is possible to retain the expressiveness of (11) and the efficient message passing structure of Definition 2 while accelerating convergence through the use of momentum (Polyak, 1964) and related (Kingma and Ba, 2014); see Appendix F.

## 4 On Broader Graph ML Regimes

In the previous section, our focus was on deriving flexible MP-GNN layers obtained by applying an algorithm $\mathcal{A}$ to the minimization of a lower-level energy $\ell_{low}$. Through Proposition 2, we demonstrated that the operator $\mathcal{A}[\ell_{low}, \ \cdot \ ]$ is equivalent to a MP-GNN layer with unrestricted message function $\widetilde{f}_M$, aggregation function $\widetilde{f}_A$, and approximate update function $\hat{f}_U$ as defined by (11). Analogous to any other MP-GNN model, these components can be iterated for $L$ steps to produce final embeddings $\mathbf{H}^{(L)}$ that can be inserted into the supervised node classification loss as in (1) to complete an end-to-end bilevel optimization pipeline.

We now expand our scope to encompass other interrelated graph ML domains, building on the general optimization-based message-passing layers derived in Section 3.3. Because our motivation stems from the principle that *BiLevel Optimization Offers More Graph Machine Learning*, in the sequel we refer to our framework as *BloomGML*.

### 4.1 The General BloomGML Framework

We formalize BloomGML via the optimization problem

$$\min_{\mathbf{W},\Theta} \ell_{up}\left(\mathbf{H}^{(L)}(\mathbf{W}), \mathbf{Y}'; \Theta, \mathcal{G}\right) \qquad (13)$$

$$\text{s.t. } \mathbf{H}^{(l)} = \mathcal{A}\left[\ell_{low}(\ \cdot \ ; \mathbf{W}, \mathcal{G}), \mathbf{H}^{(l-1)}\right], \forall l = 1, \ldots, L,$$

where $\mathcal{A}$ and $\ell_{low}$ follow from Proposition 2 while $\ell_{up}$ is a generic upper-level loss that can be specified on an application-specific basis, e.g., (1) is a special case of this form. Additionally, as before we assume that rows of $\mathbf{Y}'$ are labels; however, for maximum generality we need not assume that these labels correspond with nodes. In this way, we can extend BloomGML to generic GNN tasks besides node classification. More importantly though, we can also form deep connections with non-GNN regimes equally as well. In this regard, we consider three such examples next: knowledge graph embeddings (KGEs), label propagation (LP), and what are often called graph-MLP models.

### 4.2 Knowledge Graph Embedding Models

**Basics.** Knowledge graphs are heterogeneous graphs, typically *without* node features or labels, where each triplet/edge $e = (u, r, v)$ contains relational information between two entities, e.g., $u = $ 'Paris', $v = $ 'France', and $r = $ 'capital of' (Ji et al., 2021). The goal of knowledge graph completion (KGC) is to predict unobserved/missing factual triplets given an existing knowledge graph of interest. A widely adopted approach to this fundamental problem is based on learning knowledge graph embeddings (KGEs) as follows. First, all

existing triplets $\mathcal{E}$ are treated as positive samples, while a complementary set of negative samples are obtained by randomly choosing triplets $e^- = (u^-, r^-, v^-) \notin \mathcal{E}$, with $u^-, v^- \in \mathcal{V}$ and $r^- \in \mathcal{R}$; we can assemble these in a negative graph $\mathcal{G}^- = \{\mathcal{V}, \mathcal{R}, \mathcal{E}^-\}$, where $\mathcal{E}^-$ denotes the set of false triplets.

We then train a KGE model by minimizing a binary cross-entropy loss of the form

$$
\begin{aligned}
\ell_{kge}(\mathbf{H}, \mathbf{E}) \quad := \quad & \sum_{e \in \mathcal{E}} \log \rho \left[ s\left( \mathbf{h}_u, \mathbf{e}_r, \mathbf{h}_v \right) \right] \qquad (14) \\
+ \quad & \sum_{e' \in \mathcal{E}^-} \log \left( 1 - \rho \left[ s\left( \mathbf{h}_{u^-}, \mathbf{e}_{r^-}, \mathbf{h}_{v^-} \right) \right] \right),
\end{aligned}
$$

where $\rho$ denotes a standard sigmoid function, $\mathbf{H} \in \mathbb{R}^{n \times d}$ are node embeddings as before, and $\mathbf{E} \in \mathbb{R}^{m \times d'}$ are embeddings for each relation type $r \in \mathcal{R}$. Finally, $s : \mathbb{R}^d \times \mathbb{R}^{d'} \times \mathbb{R}^d \to \mathbb{R}$ is a so-called score function, which evaluates the compatibility of the triplet embeddings, with a higher score indicating a greater likelihood of being a true triplet (Zheng et al., 2020). Additionally, (14) is frequently used as a training loss for traditional heterogeneous GNN models applied to knowledge graphs (Schlichtkrull et al., 2017). Note that at inference time, a candidate triplet can be evaluated by computing the score of the corresponding output-layer embeddings produced by the GNN.

**KGEs and BloomGML.** We now demonstrate how the above process emerges as a special case of BloomGML, and the interpretable implications of this association, in particular relative to GNN modeling. Let $\bar{\mathcal{G}} := \mathcal{G} \cup \mathcal{G}^- = \{\mathcal{V}, \bar{\mathcal{R}}, \bar{\mathcal{E}}\}$, where $\bar{\mathcal{E}} = \mathcal{E} \cup \mathcal{E}^-$. We also define $\bar{\mathcal{R}} = \mathcal{R} \cup \mathcal{R}^-$ where $\mathcal{R}^-$ denotes a dummy set of negative relations such that, for every $r \in \mathcal{R}$ there exists a $r^- \in \mathcal{R}^-$ representing that an original relation $r$ is now appearing in a negative triple. It follows that $|\bar{\mathcal{R}}| = 2m$. Also, we assume that $\mathbf{e}_r = \mathbf{e}_{r^-}$ for all $r$, i.e., the relation type embeddings are shared. For BloomGML, following (8) we define $\ell_{low}$ w.r.t. $\bar{\mathcal{G}}$ as

$$
\ell_{low}(\mathbf{H}; \mathbf{W}, \bar{\mathcal{G}}) = \sum_{\bar{e} \in \bar{\mathcal{E}}} f(\mathbf{h}_{\bar{u}}, \mathbf{h}_{\bar{v}}; \bar{r}), \qquad (15)
$$

where we choose $\mathbf{W} = \mathbf{E}$ and

$$
f(\mathbf{h}_{\bar{u}}, \mathbf{h}_{\bar{v}}; \bar{r}) = \begin{cases} \log \rho \left[ s\left( \mathbf{h}_u, \mathbf{e}_r, \mathbf{h}_v \right) \right], & \bar{e} \in \mathcal{E} \\ \log \left( 1 - \rho \left[ s\left( \mathbf{h}_{u^-}, \mathbf{e}_{r^-}, \mathbf{h}_{v^-} \right) \right] \right), & \bar{e} \in \mathcal{E}^- \end{cases}
$$
$$(16)$$

while implicitly assuming $\kappa$ and $\eta$ are zero. By constructing $\ell_{low}$ in this way, the lower-level minimization problem provides optimal node embeddings *conditioned on fixed relation embeddings* $\mathbf{E}$. Hence if we execute a descent algorithm $\mathcal{A}$ for $L$ steps (e.g., gradient descent), we can obtain some $\mathbf{H}^{(L)} \equiv \mathbf{H}^{(L)}(\mathbf{W}) \equiv \mathbf{H}^{(L)}(\mathbf{E})$.

To complete the full BloomGML specification per (13), we must also select $\ell_{up}$. In order to align with KGE learning, we choose $\Theta = \mathbf{W} = \mathbf{E}$, $\mathbf{Y}' = \{\emptyset\}$ and define

$$
\begin{aligned}
\ell_{up} \left( \mathbf{H}^{(L)}(\mathbf{W}), \mathbf{Y}'; \Theta, \bar{\mathcal{G}} \right) &\equiv \ell_{up} \left( \mathbf{H}^{(L)}(\mathbf{E}), \{\emptyset\}; \mathbf{E}, \bar{\mathcal{G}} \right) \\
&:= \ell_{kge} \left( \mathbf{H}^{(L)}(\mathbf{E}), \mathbf{E} \right), \quad (17)
\end{aligned}
$$

where $\ell_{kge}$ follows (14).

**Key Implications of this Reformulation.** Per Proposition 2 and the downstream BloomGML lens from above, it follows that if we minimize (17) over just $\mathbf{E}$, we are effectively reproducing a minimizer of the original KGE loss from (14), with $\mathbf{H}$ computed by $L$ steps of $\mathcal{A}$. *This implies that learning KGEs equates to training an $L$-layer MP-GNN on graph $\bar{\mathcal{G}}$ with parameters* $\mathbf{E}$. Of particular relevance here, recent work has been devoted to empirically showing that when training powerful heterogeneous GNN models using (14), contrary to conventional wisdom, completely removing the GNN message-passing layers does *not* actually harm performance. However, this observation is directly explained by the analysis herein: *Explicit external message-passing is not necessary since we are already training an implicit MP-GNN when solving (14) over unconstrained embeddings.* Beyond this finding, BloomGML offers an additional practical benefit. In brief, because we are free to choose $L$, in inductive settings we can pick a relatively small value such that $\mathbf{E}$ is learned to adjust accordingly. Hence when new nodes are introduced, *their embeddings can be quickly approximated via only $L$ steps of $\mathcal{A}$*, rather than full model retraining. We test this capability in Section 5.

### 4.3 Other Graph ML Regimes

BloomGML elucidates other graph ML scenarios as well, particularly LP and graph-regularized MLP models. While full details are deferred to Appendix G, we note here that LP variants can be recast within BloomGML when we treat observed labels as input features and $\mathcal{A}$ as proximal gradient descent. We also provide supplementary experiments that verify BloomGML predictions regarding the relationship between LP and special cases of graph-regularized MLPs.

## 5 Empirical Validation and Analysis

As BloomGML represents a conceptual framework that has independent value drawn from its explanatory/unifying nature, not just pushing SOTA per se, the motivations for our experiments are two-fold:

1. Showcase the versatility and interpretability of BloomGML across diverse testing regimes;

2. Demonstrate the explanatory power of the bilevel optimization lens applied to graph ML tasks.

Table 1: *Performance mitigating spurious input features.* Here the detect ratio is obtained by computing $\|\mathbf{h}_v - \pi(\mathbf{x}_v; \mathbf{W})\|_2$ for all $v \in \mathcal{V}$ and then segmenting out the percentage of corrupted nodes within the largest 20%.

| | | Cora | Citeseer | Pubmed | Arxiv |
|---|---|---|---|---|---|
| Accuracy | GCN | 51.40 | 46.40 | 59.00 | 64.03 |
| | Base from (3) | 54.97 | 38.98 | 45.67 | 47.63 |
| | BloomGML | **65.83** | **49.33** | **72.70** | **69.11** |
| Detect Ratio | BloomGML | 94.27 | 87.82 | 96.98 | 100.00 |

Despite these quite general aims, with limited space we only include a few representative use cases and co-incident analyses. That being said, our experiments still span quite diverse settings, where SOTA methods are different and often rely on domain-specific architectures and/or heuristics. Hence our strategy is simply to pick strong, domain-specific baselines for comparisons across each task and narrative purpose, noting that these baselines generally outperform standard/generic alternatives (on a task-by-task basis) per results in prior work. Additionally, throughout this section, we use *BloomGML* to describe models derived from (13), with differentiating descriptions accompanying each experiment. Also, please see Appendix H for comprehensive details of experimental setups as well as reproduction of result tables with error bars.

**Mitigating Spurious Input Features.** As motivated in Section 2.2, bilevel optimization with an appropriate $\ell_{low}$ should in principle be able to localize spurious input features, allowing the model to focus on more informative network effects when needed. However, to our knowledge, this capability has not as of yet been exploited. To this end, we compare the original base model formed via (3) and (4), with a more robust version drawn from the general BloomGML framework of (13). In particular, since quadratic regularizers can be sensitive to outliers, we modify (3) by choosing $\kappa(\mathbf{h}; \mathbf{x}) = \delta(\mathbf{h} - \pi(\mathbf{x}; \mathbf{W}))$ within BloomGML, where $\delta : \mathbb{R}^d \to \mathbb{R}^+$ represents a robust Huber loss penalty (Huber, 1992) applied to each input dimension and summed. We then conduct node classification experiments over datasets Cora, Citeseer, Pubmed, and ogbn-arxiv by randomly corrupting 20% of node features prior to training. Results are displayed in Table 1 where multiple conclusions are evident: The Huber loss improves the accuracy across all datasets, while correctly identifying the majority of outliers.

Table 2: *Node classification on heterophily graphs.* Following convention, accuracy (%) is reported for Roman and Amazon, while ROC-AUC (%) is used for Minesweeper, Tolokers, and Questions.

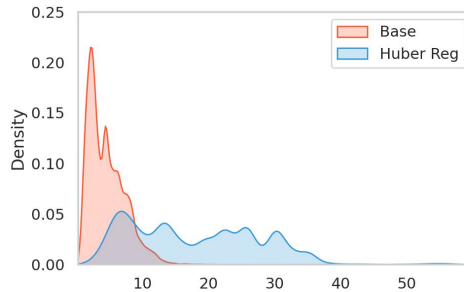| | Roman | Amazon | Minesweep | Tolokers | Questions | Avg. |
|---|---|---|---|---|---|---|
| FAGCN | 65.22 | 44.12 | 88.17 | 77.75 | 77.24 | 70.50 |
| FSGNN | 79.92 | 52.74 | 90.08 | 82.76 | **78.86** | 76.87 |
| GBK-GNN | 74.57 | 45.98 | 90.85 | 81.01 | 74.47 | 73.38 |
| JacobiConv | 71.14 | 43.55 | 89.66 | 68.66 | 73.88 | 69.38 |
| Base from (3) | 76.63 | 52.37 | 88.97 | 80.91 | 76.72 | 75.12 |
| BloomGML | 84.45 | **52.92** | 91.83 | 84.84 | 77.62 | 78.33 |
| BloomGML (w/Hub.) | **85.26** | 51.00 | **93.30** | **85.92** | 77.93 | **78.68** |



Figure 1: $\|\mathbf{h}_u^{(L)} - \mathbf{h}_v^{(L)}\|_2$ *density for* $(u, v) \in \mathcal{E}$ *but with different labels in the Roman dataset.*

**Comparison on Heterophily Graphs.** For graphs exhibiting heterophily (Zhu et al., 2021a), nodes sharing an edge are less likely to have the same label. Hence an energy function that pushes the respective node embeddings to a common value may be counterproductive, i.e., as in the edge-dependent term from (3). Instead, to address heterophily graphs using BloomGML, we form $\ell_{low}$ via $f(\mathbf{h}_v, \mathbf{h}_u; r) = \|\mathbf{h}_u \mathbf{C} - \mathbf{h}_v\|_2^2$, where $\mathbf{C} \in \mathbb{R}^{d \times d}$ is a trainable weight and $r$ is ignored for the homogeneous case. Additionally, to reduce sensitivity to less reliable node features that could potentially accompany heterophily graphs, we consider $\kappa(\mathbf{h}; \mathbf{x}) = \delta(\mathbf{h} - \pi(\mathbf{x}; \mathbf{W}))$ to contrast with $\kappa(\mathbf{h}; \mathbf{x}) = \|\mathbf{h} - \pi(\mathbf{x}; \mathbf{W})\|_2^2$. Finally, we select $\eta(\mathbf{h}) = \mathcal{I}_\infty[\mathbf{h} < 0]$ and $\sigma(\mathbf{a}) = 1$ and apply proximal gradient descent for $\mathcal{A}$. Table 2 displays results using the recent heterophily benchmarks Roman, Amazon, Minesweeper, Tolokers, and Questions from Platonov et al. (2023). We compare BloomGML against recent GNN models explicitly designed for handling heterophily, namely, FAGCN (Bo et al., 2021), FSGNN (Maurya et al., 2022), GBK-GNN (Du et al., 2022), and JacobiConv (Wang and Zhang, 2022). For reference, we also compare with the bilevel baseline formed from (3); additional GNN baselines from Platonov et al. (2023) are discussed in Appendix H. On average, we observe that BloomGML achieves the best accuracy here. Even so, our focus is not on solving heterophily problems per se, but rather, demonstrating that interpretable modifications to $\ell_{low}$ can induce predictable effects.

Regarding the latter, we present one additional visualization afforded by the BloomGML paradigm. On the Roman dataset, we incorporate $\delta$ within the function $f$, which allows greater freedom for embeddings sharing an edge but with different labels to deviate from one another. The distribution of $\|\mathbf{h}_u^{(L)} - \mathbf{h}_v^{(L)}\|_2$ for $(u, v) \in \mathcal{E}$ but $\mathbf{y}_v \neq \mathbf{y}_u$ (i.e., different labels) is shown in Figure 1. Clearly, the Huber loss allows greater deviation in embeddings sharing an edge as expected. Interestingly, the accuracy using Huber within $f$ increases to 88.12%.

**Comparison on Heterogeneous Graphs.** The base model formed via (3) has also recently been ex-

Table 3: *Node classification accuracy involving heterogeneous graphs.* BloomGML is modified from HALO to include robust regularization within $\kappa(\mathbf{h}; \mathbf{x})$.

| | AIFB | MUTAG | BGS | AM |
|---|---|---|---|---|
| HALO | 97.2 | 80.9 | 89.7 | 85.9 |
| BloomGML | 97.2 | **82.4** | **93.1** | **86.9** |

tended to heterogeneous graphs via the HALO model (Ahn et al., 2022). However, this extension remains limited in its strict adherence to the quadratic edge- and input feature-dependent penalty terms as in the original (3). To this end, we modify HALO within the confines of the BloomGML framework to explore non-quadratic penalties (Log-Cosh and Huber) while maintaining equivalence across all other aspects of model design. Node classification results are in Table 3, where we observe that all else being equal, these changes lead to an improvement in accuracy over HALO.

**Knowledge Graph Completion.** As mentioned at the end of Section 4.2, BloomGML is particularly well-suited for inductive KGE tasks such as KGC. To this end, we analytically and empirically compare against the recent RefactorGNN model (Chen et al., 2022), which includes related analysis showing how a particular score function, DistMult (Yang et al., 2014), when combined with a softmax training loss over positive samples, has some commonalities with GNN layer structure. From a practical standpoint, RefactorGNN involves training a KGE model while resetting node embeddings to initial values in regular intervals to facilitate the inductive setting (the edge embeddings are trained as usual). While certainly a noteworthy contribution, there nonetheless remain three limitations of RefactorGNN relative to BloomGML.

First, the RefactorGNN model does not actually induce an efficient/sparse MP-GNN, as there remains a global hub node to which all other nodes are connected and receive messages. Secondly, the supporting analysis provided in Chen et al. (2022) only directly addresses the softmax/DistMult pairing while excluding consideration of negative samples; in contrast, BloomGML serves as a general framework covering a wide variety of both KGE and non-KGE models alike, and within which we have explicitly accounted for negative samples through $\bar{\mathcal{G}}$. And lastly, the repeated reinitialization of node embeddings during RefactorGNN training can be viewed as a heuristic, with no guarantee of convergence; BloomGML sidesteps this issue altogether within an integrated bilevel optimization framework.

To conduct empirical comparisons with RefactorGNN, we must first specify (8) for BloomGML. We take inspiration from NBFNet (Zhu et al., 2021c) and design $\ell_{low}$ such that one step of proximal gradient descent

Table 4: *Inductive KGC tasks.* Reported results based on the Test Hits@10 metric. Sum and PNA are aggregators while T and D are TransE and DistMult score functions, respectively.

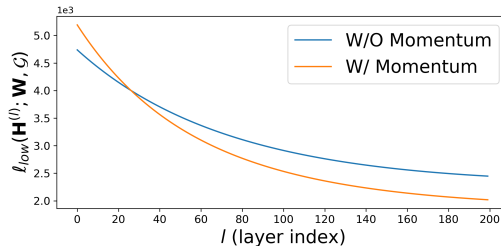| | WN18RR_v1 | | | | FB15k-237_v1 | | | |
|---|---|---|---|---|---|---|---|---|
| | PNA(T) | Sum(T) | PNA(D) | Sum(D) | PNA(T) | Sum(T) | PNA(D) | Sum(D) |
| RefactorGNN | / | / | / | 0.885 | / | / | / | 0.787 |
| BloomGML | 0.952 | 0.937 | **0.960** | 0.946 | 0.744 | 0.747 | **0.836** | 0.792 |



Figure 2: $\ell_{low}$ values versus propagation steps on Cora.

mimics an NBFNet model layer; see Appendix I for derivations showing how this process aligns with an MP-GNN layer when incorporated within BloomGML under the right circumstances. We also set $\kappa(\mathbf{h}; \mathbf{x}) = \eta(\mathbf{h}) = 0$, $\sigma(\mathbf{a}) = 1$ for simplicity. Inductive KGC results are shown in Table 4 on WordNet18RR_v1 and FB15K237_v1 datasets (Teru et al., 2020). Across different score functions TransE and DistMult, and aggregators Sum and PNA (Corso et al., 2020), BloomGML achieves strong performance relative to RefactorGNN. Note that in Table 4 we have reported the best RefactorGNN model from Chen et al. (2022) as there is presently no publicly-available code for reproducibiliy.

**Incorporation of Momentum.** We demonstrate how the incorporation of momentum within $\mathcal{A}$, as proposed at the end of Section 3.3 and detailed in Appendix F, can maintain MP-GNN structure while expediting convergence. Results are shown in Figure 2 and Table 5 using a simple base model akin to (3), where we observe that momentum can potentially improve BloomGML performance by reducing $\ell_{low}$ more quickly.

Table 5: *Node classification with momentum.* The optimization algorithm for BloomGML is SGD with momentum.

| | Cora | Citeseer | Pubmed | Arxiv |
|---|---|---|---|---|
| SGD | 80.1 | 73.2 | 78.0 | 72.0 |
| BloomGML | **83.4** | **74.0** | **80.7** | **72.6** |

**Additional Results.** Finally, please see Appendix J for additional experiments including an ablation over learning $\lambda$, the trade-off parameter from (3), as well as further demonstration of the versatility of BloomGML.

## 6 Conclusion

In this work we have introduced the BloomGML framework, which provides a novel lens for understanding various graph ML paradigms and introducing interpretable, targeted enhancements. *Let graph ML bloom.*

# References

Ahn, H., Yang, Y., Gan, Q., Moon, T., and Wipf, D. (2022). Descent steps of a relation-aware energy produce heterogeneous graph neural networks.

Ando, R. and Zhang, T. (2006). Learning on graph with laplacian regularization. *Advances in neural information processing systems*, 19.

Bo, D., Wang, X., Shi, C., and Shen, H. (2021). Beyond low-frequency information in graph convolutional networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35.

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26.

Chen, J., Mueller, J., Ioannidis, V. N., Adeshina, S., Wang, Y., Goldstein, T., and Wipf, D. (2021). Does your graph need a confidence boost? Convergent boosted smoothing on graphs with tabular node features. In *International Conference on Learning Representations*.

Chen, Y., Mishra, P., Franceschi, L., Minervini, P., Stenetorp, P., and Riedel, S. (2022). Refactor gnns: Revisiting factorisation-based models from a message-passing perspective.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. (2020). Principal neighbourhood aggregation for graph nets. In *Advances in Neural Information Processing Systems*, volume 33.

Di Giovanni, F., Rowbottom, J., Chamberlain, B. P., Markovich, T., and Bronstein, M. M. (2023). Understanding convolution on graphs via energies. *Transactions on Machine Learning Research*.

Du, L., Shi, X., Fu, Q., Ma, X., Liu, H., Han, S., and Zhang, D. (2022). Gbk-gnn: Gated bi-kernel graph neural networks for modeling both homophily and heterophily. In *Proceedings of the ACM Web Conference 2022*, pages 1550–1558.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7).

Eswaran, D., Günnemann, S., Faloutsos, C., Makhija, D., and Kumar, M. (2017). Zoobp: Belief propagation for heterogeneous networks. 10(5).

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.

Gribonval, R. and Nikolova, M. (2020). A characterization of proximity operators. *Journal of Mathematical Imaging and Vision*, 62(6):773–789.

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.

Hamilton, W. L. (2020). *Graph representation learning.* Morgan & Claypool Publishers.

Hinton, G., Srivastava, N., and Swersky, K. (2012). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2.

Hu, Y., You, H., Wang, Z., Wang, Z., Zhou, E., and Gao, Y. (2021). Graph-mlp: Node classification without message passing in graph. *arXiv preprint arXiv:2106.04051*.

Huber, P. J. (1992). Robust estimation of a location parameter. *Breakthroughs in statistics: Methodology and distribution*, pages 492–518.

Ji, S., Pan, S., Cambria, E., Marttinen, P., and Philip, S. Y. (2021). A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*.

Jiang, H., Liu, R., Yan, X., Cai, Z., Wang, M., and Wipf, D. (2023). MuseGNN: Interpretable and convergent graph neural network layers at scale. *arXiv preprint arXiv:2310.12457*.

Kearnes, S., McCloskey, K., Berndl, M., Pande, V., and Riley, P. (2016). Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30:595–608.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Luo, Y., Huang, R., Chen, A., and Zeng, X. (2021). Reslpa: Label propagation with residual learning. In *Proceedings of the 2021 7th International Conference on Computing and Artificial Intelligence*, ICCAI '21.

Ma, Y., Liu, X., Zhao, T., Liu, Y., Tang, J., and Shah, N. (2021). A unified view on graph neural networks as graph signal denoising. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 1202–1211.

Maurya, S. K., Liu, X., and Murata, T. (2022). Simplifying approach to node classification in graph neural networks. *Journal of Computational Science*, 62:101695.

Pan, X., Song, S., and Huang, G. (2020). A unified framework for convolution-based graph neural networks.

Parikh, N. and Boyd, S. (2014). Proximal algorithms. *Found. Trends Optim.*, page 127–239.

Platonov, O., Kuznedelev, D., Babenko, A., and Prokhorenkova, L. (2022). Characterizing graph datasets for node classification: Beyond homophily-heterophily dichotomy. *arXiv preprint arXiv:2209.06177*.

Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A., and Prokhorenkova, L. (2023). A critical look at the evaluation of GNNs under heterophily: Are we really making progress? In *The Eleventh International Conference on Learning Representations*.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17.

Schlichtkrull, M., Kipf, T. N., Bloem, P., van den Berg, R., Titov, I., and Welling, M. (2017). Modeling relational data with graph convolutional networks.

Sinha, A., Malo, P., and Deb, K. (2018). A review on bilevel optimization: From classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295.

Teru, K., Denis, E., and Hamilton, W. (2020). Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning*, pages 9448–9457. PMLR.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

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

Wang, Y., Jin, J., Zhang, W., Yang, Y., Chen, J., Gan, Q., Yu, Y., Zhang, Z., Huang, Z., and Wipf, D. (2022). Why propagate alone? parallel use of labels and features on graphs. In *International Conference on Learning Representations*.

Wang, Z., Ling, Q., and Huang, T. (2016). Learning deep $\ell_0$ encoders. In *AAAI Conference on Artificial Intelligence*, volume 30.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks?

Xue, R., Han, H., Torkamani, M., Pei, J., and Liu, X. (2023). LazyGNN: Large-scale graph neural networks via lazy propagation. In *International Conference on Machine Learning*.

Yamaguchi, Y., Faloutsos, C., and Kitagawa, H. (2016). Camlp: Confidence-aware modulated label propagation. pages 513–521.

Yang, B., Yih, W.-t., He, X., Gao, J., and Deng, L. (2014). Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*.

Yang, Y., Huang, Z., and Wipf, D. (2022a). Transformers from an optimization perspective. In *Advances in Neural Information Processing Systems*.

Yang, Y., Liu, T., Wang, Y., Huang, Z., and Wipf, D. (2022b). Implicit vs unfolded graph neural networks. *arXiv preprint arXiv:2111.06592*.

Yang, Y., Liu, T., Wang, Y., Zhou, J., Gan, Q., Wei, Z., Zhang, Z., Huang, Z., and Wipf, D. (2021). Graph neural networks inspired by classical iterative algorithms. In *International Conference on Machine Learning*, pages 11773–11783. PMLR.

Ying, Z., Bourgeois, D., You, J., Zitnik, M., and Leskovec, J. (2019). Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep sets. *Advances in neural information processing systems*, 30.

Zhang, H., Wang, S., Ioannidis, V. N., Adeshina, S., Zhang, J., Qin, X., Faloutsos, C., Zheng, D., Karypis, G., and Yu, P. S. (2023). Orthoreg: Improving graph-regularized mlps via orthogonality regularization.

Zhang, H., Yan, T., Xie, Z., Xia, Y., and Zhang, Y. (2020). Revisiting graph convolutional network on semi-supervised node classification from an optimization perspective. *arXiv preprint arXiv:2009.11469*.

Zheng, D., Song, X., Ma, C., Tan, Z., Ye, Z., Dong, J., Xiong, H., Zhang, Z., and Karypis, G. (2020). Dgl-ke: Training knowledge graph embeddings at scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 739–748.

Zhou, D., Bousquet, O., Lal, T., Weston, J., and Schölkopf, B. (2003). Learning with local and global consistency. *Advances in neural information processing systems*, 16.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI open*, 1:57–81.

Zhou, K., Huang, X., Zha, D., Chen, R., Li, L., Choi, S.-H., and Hu, X. (2021). Dirichlet energy constrained learning for deep graph neural networks. *Advances in Neural Information Processing Systems*, 34:21834–21846.

Zhu, J., Rossi, R. A., Rao, A., Mai, T., Lipka, N., Ahmed, N. K., and Koutra, D. (2021a). Graph neural networks with heterophily. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35.

Zhu, M., Wang, X., Shi, C., Ji, H., and Cui, P. (2021b). Interpreting and unifying graph neural networks with an optimization framework. *arXiv preprint arXiv:2101.11859*.

Zhu, X. (2005). *Semi-supervised learning with graphs*. PhD Thesis, Carnegie Mellon University.

Zhu, Z., Zhang, Z., Xhonneux, L.-P., and Tang, J. (2021c). Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems*, 34:29476–29490.

# A  Appendix Overview

Appendix content more-or-less follows the order in which it was originally referenced in the main text; we summarize as follows:

- In Section B, we derive the the message passing special case from (4).

- In Section C, we discuss why increasing the expressiveness of BloomGML in meaningful ways is challenging.

- In Sections D and E, we provide the proofs of Propositions 1 and 2, respectively.

- In Section F, we present a general framework for accelerating convergence and related special cases.

- In Section G, we fill in details relating BloomGML to label propagation and graph-regularized MLPs. We also provide supporting empirical results.

- In Section H, we provide model implementation and experiment details related to Section 5.

- In Section I, we explore the connection between NBFNet and BloomGML.

- In Section J, we present additional empirical results/ablations.

# B  Derivation of Message Passing Special Case from (4)

The message passing updates from (4) in Section 2.2 are based on the energy function from (3), which we repeat here for convenience as

$$\ell_{low}(\mathbf{H}; \mathbf{W}, \mathcal{G}) = \sum_{v \in \mathcal{V}} \tfrac{1}{2} \left\| \mathbf{h}_v - \pi(\mathbf{x}_v; \mathbf{W}) \right\|_2^2 + \tfrac{\lambda}{2} \sum_{(u,v) \in \mathcal{E}} \left\| \mathbf{h}_u - \mathbf{h}_v \right\|_2^2. \tag{18}$$

Taking the derivative of this $\ell_{low}$ w.r.t. $\mathbf{h}_v$ we obtain

$$\nabla_{\mathbf{h}_v} \ell_{low}(\mathbf{H}; \mathbf{W}, \mathcal{G}) = \mathbf{h}_v - \pi(\mathbf{x}_v; \mathbf{W}) - \lambda \sum_{u \in \mathcal{N}_v} (\mathbf{h}_u - \mathbf{h}_v). \tag{19}$$

Let $\boldsymbol{\mu}_{(u,r,v)}^{(l)} := \mathbf{h}_u^{(l-1)} - \mathbf{h}_v^{(l-1)}$ and $\mathbf{a}_v^{(l)} := \sum_{u \in \mathcal{N}_v} \boldsymbol{\mu}_{(u,r,v)}^{(l)}$. There will then exist an update function $f_U(\mathbf{h}_v^{(l-1)}, \mathbf{a}_v^{(l)}, \mathbf{x}_v)$ such that we have

$$\mathbf{h}_v^{(l)} = \mathbf{h}_v^{(l-1)} - \gamma \nabla_{\mathbf{h}_v} \ell_{low}(\mathbf{H}; \mathbf{W}, \mathcal{G}) = f_U(\mathbf{h}_v^{(l-1)}, \mathbf{a}_v^{(l)}, \mathbf{x}_v) := (1 - \gamma)\mathbf{h}_v^{(l-1)} + \gamma\lambda \mathbf{a}_v^{(l)} + \gamma\pi(\mathbf{x}_v; \mathbf{W}), \tag{20}$$

which reproduces (4).

# C  Possibilities for Generalizing BloomGML Expressiveness

As alluded to in Section 3.3, increasing the expressiveness of BloomGML in meaningful ways is challenging. At a trivial level, we could always generalize (8) by allowing $f \to f_{(u,r,v)}$ to vary across each triplet; analogously we could allow $\kappa \to \kappa_{(v)}$ and $\eta \to \eta_{(v)}$ or related. But for more interesting enhancements, we might consider flexible expressions of permutation-invariant functions given that the input graph is invariant to the order of nodes and triplets. For example, results from Zaheer et al. (2017) show that, granted certain stipulations on the input domain, permutation invariant functions operating on a set $\mathcal{X}$ can be expressed as $\rho[\sum_{x \in \mathcal{X}} \phi(x)]$ when granted suitable transforms $\rho$ and $\phi$. From this type of result, it is natural to consider, for example, generalizing the first term of (8) via

$$\sum_{(u,r,v) \in \mathcal{E}} f\left(\mathbf{h}_u, \mathbf{h}_v; r\right) \to \rho\left[ \sum_{(u,r,v) \in \mathcal{E}} f\left(\mathbf{h}_u, \mathbf{h}_v; r\right) \right]. \tag{21}$$

The problem here though is that, if we take the gradient of this revised term w.r.t. the node embeddings, by the chain rule the result can depend on all triplets in the graph, which violates the message-passing schema from either Definition 1 or 2.

## D   Proof of Proposition 1

The basic strategy here will be to push components of the original aggregation function $f_A$ into the revised message and update functions $\widetilde{f}_M$ and $\widetilde{f}_U$, respectively. And the underlying goal is to do so without losing any expressiveness in the resulting composite function, even while reducing the revised aggregation function $\widetilde{f}_A$ to simple, parameter-free addition.

To proceed, we rely on a suitable decomposition of permutation-invariant set functions as required by $f_A$ over input messages from $\mathcal{N}_v$, where the domain of each message is $\mathbb{R}^d$. In this regard, a number of permutation-invariant function decompositions have been proposed in Zaheer et al. (2017) with conditions dependent on the domain of the input sets. The strongest result they present provides a useful general form of exact decomposition; however, unfortunately the underpinning guarantee only holds for input sets drawn from a *countable* universe, which $\mathbb{R}^d$ is of course not.

To work around this limitation, we first assume that $|\mathcal{N}_v| = p < n$ for all $v \in \mathcal{V}$, where $p > 0$ is a fixed constant. We may then apply Theorem 9 from Zaheer et al. (2017) which stipulates that any continuous $f_A$ defined on a compact set within $\mathbb{R}^d$ can be approximated arbitrarily closely via a decomposition of the form

$$f_A \left( \{ \boldsymbol{\mu}^{(l)}_{(u,r,v)} : (u,r) \in \mathcal{N}_v \} \right) \approx \rho \left( \sum_{(u,r) \in \mathcal{N}_v} \phi(\boldsymbol{\mu}^{(l)}_{(u,r,v)}) \right), \tag{22}$$

where $\rho : \mathbb{R}^{\tilde{d}} \to \mathbb{R}^d$ and $\phi : \mathbb{R}^d \to \mathbb{R}^{\tilde{d}}$ are suitable transforms.

To extend to the general case where $|\mathcal{N}_v|$ may vary across nodes (since graphs are generally not limited to a fixed node degree), we add $\max_{v \in \mathcal{V}} |\mathcal{N}_v| - |\mathcal{N}_v|$ dummy neighbors $(u', r')$ to $\mathcal{N}_v$ for every $v \in \mathcal{V}$, and form the augmented neighbor sets $\mathcal{N}'_v$, with the restriction that $\phi(\boldsymbol{\mu}_{u',r',v}) = 0$. Note that in the neighborhood of the associated dummy messages $\boldsymbol{\mu}_{(u',r',v)}$, we can no longer guarantee a close approximation to the true function; however, we can cluster such dummy messages in an area of arbitrarily small measure, so their impact is negligible. Given $\mathcal{N}'_v$ so defined, $f_A$ can be approximated arbitrarily closely (except for within a negligible, vanishingly small area of the input domain) via a decomposition of the form

$$f_A(\{ \boldsymbol{\mu}^{(l)}_{(u,r,v)} : (u,r) \in \mathcal{N}_v \}) \approx \rho \left( \sum_{(u,r) \in \mathcal{N}'_v} \phi(\boldsymbol{\mu}^{(l)}_{(u,r,v)}) \right) \equiv \rho \left( \sum_{(u,r) \in \mathcal{N}_v} \phi(\boldsymbol{\mu}^{(l)}_{(u,r,v)}) \right). \tag{23}$$

Now define $\widetilde{f}_M := \phi \circ f_M$, $\quad \widetilde{\boldsymbol{\mu}}^{(l)}_{(u,r,v)} := \widetilde{f}_M(\mathbf{h}^{(l-1)}_u, r, \mathbf{h}^{(l-1)}_v) \in \mathbb{R}^{\tilde{d}}$, $\quad$ and $\widetilde{f}_A(\{ \widetilde{\boldsymbol{\mu}}^{(l)}_{(u,r,v)} : (u,r) \in \mathcal{N}_v \}) = \sum_{(u,r) \in \mathcal{N}_v} \widetilde{\boldsymbol{\mu}}^{(l)}_{(u,r,v)}$. Then we have

$$\rho \circ \widetilde{f}_A \circ \widetilde{f}_M \approx f_A \circ f_M \tag{24}$$

Furthermore, if we define $\widetilde{f}_U := f_U \circ \rho$, we may conclude that

$$\widetilde{f}_U \circ \widetilde{f}_A \circ \widetilde{f}_M \quad \approx \quad f_U \circ f_A \circ f_M \tag{25}$$

in the sense described above.

## E   Proof of Proposition 2

We assume that $\widetilde{f}_M$ satisfies the gradient representation criteria from Definition 3. Now recall that the energy function $\ell_{low}$ is given by

$$\ell_{low}(\mathbf{H}; \mathbf{W}, \mathcal{G}) := \sum_{(u,r,v) \in \mathcal{E}} f(\mathbf{h}_u, \mathbf{h}_v; r) + \sum_{v \in \mathcal{V}} [\kappa(\mathbf{h}_v; \mathbf{x}_v) + \eta(\mathbf{h}_v)]. \tag{26}$$

First, we choose

$$f(\mathbf{h}_u, \mathbf{h}_v; r) \quad := \quad \widetilde{\zeta}(\Phi^\top \mathbf{h}_u, \Phi^\top \mathbf{h}_v; r), \tag{27}$$

where $\Phi$ and $\widetilde{\zeta}$ instantiate the gradient representation criteria for $\widetilde{f}_M$. From this it follows that

$$
\begin{aligned}
\frac{\partial f\left(\mathbf{h}_u, \mathbf{h}_v; r\right)}{\partial \mathbf{h}_v} &= \Phi \widetilde{f}_M(\mathbf{h}_u, r, \mathbf{h}_v) \\
\frac{\partial f\left(\mathbf{h}_u, \mathbf{h}_v; r\right)}{\partial \mathbf{h}_u} &= \Phi \widetilde{f}_M(\mathbf{h}_v, r^{-1}, \mathbf{h}_u).
\end{aligned}
\tag{28}
$$

We may then choose $\boldsymbol{\mu}_{(u,r,v)}^{(l)} := \widetilde{f}_M(\mathbf{h}_u^{(l-1)}, r, \mathbf{h}_v^{(l-1)}) \in \mathbb{R}^{\widetilde{d}}$, and by adding up all gradient terms depending on a given node $v$, we can produce the aggregation function $\boldsymbol{a}_v^{(l)} := \widetilde{f}_A(\{\boldsymbol{\mu}_{(u,r,v)}^{(l)} : (u,r) \in \mathcal{N}_v\}) = \sum_{(u,r) \in \mathcal{N}_v} \boldsymbol{\mu}_{(u,r,v)}^{(l)}$, noting that in certain cases $r = \bar{r}^{-1}$ for some other relation $\bar{r} \in \mathcal{R}$. And by linearity of the aggregation, we can later push $\Phi$ through to the update function (see below).

In this way, the gradient of $\ell_{low}$ w.r.t. $\mathbf{h}_v^{(l-1)}$, excluding the (possibly) non-differentiable $\eta$ term is given by $\Phi \boldsymbol{a}_v^{(l)} + \kappa'(\mathbf{h}_v^{(l-1)}; \mathbf{x}_v)$. Given the Lipschitz continuous gradients of $\widetilde{\zeta}$, it follows that for a suitably small $\gamma' > 0$, there will always exist an upper bound of the form

$$
\ell_{low}(\mathbf{H}; \mathbf{W}, \mathcal{G}) \leq \sum_{v \in \mathcal{V}} \frac{1}{2\gamma\sigma(\boldsymbol{a}_v^{(l)})} \left[ \left\| \mathbf{h}_v - \left( \mathbf{h}_v^{(l-1)} - \gamma\sigma(\boldsymbol{a}_v^{(l)}) \left[ \Phi \boldsymbol{a}_v^{(l)} + \kappa'(\mathbf{h}_v^{(l-1)}; \mathbf{x}_v) \right] \right) \right\|_2^2 + \eta(\mathbf{h}_v) \right] + C
\tag{29}
$$

for all $\gamma \in (0, \gamma']$, where $C$ is an irrelevant constant, $\sigma$ is a preconditioner function, and we achieve equality iff $\mathbf{H} = \mathbf{H}^{(l-1)}$. From this bound, we can construct an update function by taking a proximal step that is guaranteed to reduce or leave unchanged $\ell_{low}$. The form is given by

$$
\hat{f}(\mathbf{h}_v^{(l-1)}, \boldsymbol{a}_v^{(l)}, \mathbf{x}_v) := \mathbf{prox}_{\eta, \gamma\sigma(\boldsymbol{a}_v^{(l)})} \left( \mathbf{h}_v^{(l-1)} - \gamma\sigma(\boldsymbol{a}_v^{(l)}) \left[ \Phi \boldsymbol{a}_v^{(l)} + \kappa'(\mathbf{h}_v^{(l-1)}; \mathbf{x}_v) \right] \right),
\tag{30}
$$

completing the proof.

# F    Accelerating BloomGML Convergence with Momentum/Hidden States

At the end of Section 3.3, we briefly alluded to retaining the expressiveness of (11) and the efficient message passing structure of Definition 2 while accelerating convergence, meaning a smaller value of $L$ will be adequate for approximating a minimum of $\ell_{low}$. We fill in further details here. At a high level, this goal can be accomplished through the use of additional hidden states $\mathbf{S} = \{\mathbf{s}_v\}_{v \in \mathcal{V}} \in \mathbb{R}^{n \times d}$ that instantiate momentum (Polyak, 1964). The latter can mute the amplitude of oscillations in noisy gradients while traversing flatter regions of the loss surface more quickly. A revised version of (11) festooned with momentum is given by

$$
\begin{aligned}
\mathbf{s}^{(l)} &= \beta\mathbf{s}^{(l-1)} + (1 - \beta)\sigma(\mathbf{a}^{(l)}) \left[ \Phi \mathbf{a}^{(l)} + \kappa'(\mathbf{h}^{(l-1)}; \mathbf{x}) \right] \\
\mathbf{h}^{(l)} &= \mathbf{prox}_{\eta, \gamma} \left( \mathbf{h}^{(l-1)} - \gamma\mathbf{s}^{(l)} \right),
\end{aligned}
\tag{31}
$$

where $\beta$ is an additional trade-off parameter.

Beyond the vanilla version of momentum given by (31), for reference we now define a more general formulation that encompasses many well-known acceleration methods including AdaGrad (Duchi et al., 2011), RMSProp (Hinton et al., 2012), and Adam (Kingma and Ba, 2014) as special cases.

To begin, let $\ell_{low}$ be any differentiable energy function expressible as (8) with $\eta = 0$ (we omit handling the more general case for brevity, although it can be derived in a similar way) and $\mathbf{g}_v^{(l)} := \Phi \boldsymbol{a}_v^{(l)} + \kappa'(\mathbf{h}_v^{(l-1)}; \mathbf{x}_v) \in \mathbb{R}^d$. Given auxiliary variables $\mathbf{S} = \{\mathbf{s}_v\}_{v \in \mathcal{V}} \in \mathbb{R}^{n \times d}$ and a coefficient $\beta > 0$, the iterative mapping[2]

$$
\begin{aligned}
\mathbf{s}^{(l)} &= \beta\mathbf{s}^{(l-1)} + (1 - \beta)\xi(\mathbf{g}^{(l-1)}) \\
\mathbf{h}^{(l)} &= \mathbf{h}^{(l-1)} - \gamma\sigma(\mathbf{s}^{(l)})\varphi(\mathbf{g}^{(l-1)})
\end{aligned}
\tag{32}
$$

provides a general framework for accelerating convergence. Here $\xi : \mathbb{R}^d \to \mathbb{R}^d$, and $\varphi : \mathbb{R}^d \to \mathbb{R}^d$ are linear node-wise functions defined on $\mathbf{g}$. $\sigma(\cdot) : \mathbb{R}^d \to \mathbb{R}^+ > 0$ is an arbitrary positive function. Note that when $k > 1$

---

[2]We omit a subscript $v$ in the following context for simplicity.

auxiliary variables are involved, $\varphi$ is extended to $\mathbb{R}^{kd} \to \mathbb{R}^d$. Such an iterative mapping preserves the properties of message-passing functions. To clarify this, let $f_{U'}(\mathbf{h}, \mathbf{a}, \mathbf{s}, \mathbf{x})$ denote the mapping defined in (32). For any $v \in \mathcal{V}$, we have $\mathbf{h}_v^{(l)} = f_{U'}(\mathbf{h}_v^{(l-1)}, \mathbf{a}_v^{(l)}, \mathbf{s}_v^{(l-1)}, \mathbf{x}_v)$, which means one update step only takes information from $\{v \cup \mathcal{N}_v\}$. Also, $\mathbf{a}_v = \sum_{(u,r) \in \mathcal{N}_v} \boldsymbol{\mu}_{(u,r,v)}$ is a permutation-invariant function by construction, which means $f_{U'}$ is also permutation invariant over the sets of 1-hop neighbors.

Turning now to the remainder of this section, we verify that the vanilla momentum from above, as well as AdaGrad, RMSProp, and Adam are all special cases of (32).

**Momentum as Special Case.** To show that vanilla momentum is a message-passing layer formed as a special case of (32), we let $\xi(\mathbf{g}) = \mathbf{g}$, $\sigma(\mathbf{s}) = \mathbf{s}$, and $\varphi = 1$, then the mapping is given by

$$
\begin{aligned}
\mathbf{s}^{(l)} &= \beta \mathbf{s}^{(l-1)} + (1 - \beta) \mathbf{g}^{(l-1)}, \\
\mathbf{h}^{(l)} &= \mathbf{h}^{(l-1)} - \gamma \mathbf{s}^{(l)}.
\end{aligned}
\tag{33}
$$

which reproduces (31).

**AdaGrad.** Here we show AdaGrad (Duchi et al., 2011) is also a special case of (32). Let $\beta = 0$, $\xi(\mathbf{g}) = \text{diag}(\mathbf{g}\mathbf{g}^T)$, $\varphi(\mathbf{g}) = \mathbf{g}$, and $\sigma(\mathbf{s}) = (\mathbf{s}^{(l)} + \epsilon)^{-\frac{1}{2}}$ where $\epsilon$ is a small constant to avoid division by zero. Then the updating rule of AdaGrad is given by

$$
\begin{aligned}
\mathbf{s}^{(l)} &= \text{diag}(\mathbf{g}^{(l-1)}(\mathbf{g}^{(l-1)})^T), \\
\mathbf{h}^{(l)} &= \mathbf{h}^{(l-1)} - \gamma \left(\mathbf{s}^{(l)} + \epsilon\right)^{-\frac{1}{2}} \odot \mathbf{g}^{(l-1)}.
\end{aligned}
\tag{34}
$$

Here the square root, inverse, and $\odot$ are all element-wise operations.

**RMSprop.** RMSprop (Hinton et al., 2012) divides the gradient by a running average of its recent magnitude. To show that it is also a special case of (32), let $\xi(\mathbf{g}) = \mathbf{g}^2$, $\varphi(\mathbf{g}) = \mathbf{g}$, and $\sigma(\mathbf{s}) = (\mathbf{s}^{(l)} + \epsilon)^{-\frac{1}{2}}$. Then the updating rule of RMSprop is

$$
\begin{aligned}
\mathbf{s}^{(l)} &= \beta \mathbf{s}^{(l-1)} + (1 - \beta)(\mathbf{g}^{(l-1)})^2, \\
\mathbf{h}^{(l)} &= \mathbf{h}^{(l-1)} - \left(\mathbf{s}^{(l)} + \epsilon\right)^{-\frac{1}{2}} \odot \mathbf{g}^{(l-1)}.
\end{aligned}
\tag{35}
$$

**Adam.** Adam (Kingma and Ba, 2014) requires two auxiliary variables $\mathbf{s}_1$ and $\mathbf{s}_2$. Let $\xi_1(\mathbf{g}) = \mathbf{g}$, $\xi_2(\mathbf{g}) = \mathbf{g}^2$, $\varphi(\mathbf{g}) = 1$, and $\sigma(\mathbf{s}_1^{(l)}, \mathbf{s}_2^{(l)}) = \left(\sqrt{\frac{\mathbf{s}_1^{(l)}}{1-\beta_1^l}} + \epsilon\right)^{-1} \cdot \frac{\mathbf{s}_1^{(l)}}{1-\beta_2^l}$. The updating rule of Adam is

$$
\begin{aligned}
\mathbf{s}_1^{(l)} &= \beta_1 \mathbf{s}_1^{(l-1)} - (1 - \beta_1)\mathbf{g}^{(l-1)}, \\
\mathbf{s}_2^{(l)} &= \beta_2 \mathbf{s}_2^{(l-1)} + (1 - \beta_2)(\mathbf{g}^{(l-1)})^2, \\
\mathbf{h}^{(l)} &= \mathbf{h}^{(l-1)} - \gamma \left(\sqrt{\frac{\mathbf{s}_1^{(l)}}{1-\beta_1^l}} + \epsilon\right)^{-1} \cdot \frac{\mathbf{s}_1^{(l)}}{1-\beta_2^l}.
\end{aligned}
\tag{36}
$$

# G   More on Label Propagation and Graph-Regularized MLPs

In this section we pick up where we left off in Section 4.3, filling in details at the nexus of label propagation, graph-regularized MLPs, and BloomGML.

## G.1   Label Propagation as a BloomGML Special Case

Consider the lower-level energy given by

$$
\ell_{low}(\mathbf{H}; \mathcal{G}) := \lambda \text{tr}\left(\mathbf{H}^\top \mathbf{L} \mathbf{H}\right) + \left\|\bar{\mathbf{Y}} - \mathbf{H}\right\|_{\mathcal{F}}^2 + \sum_{v \in \mathcal{V}'} \mathcal{I}_\infty[\mathbf{h}_v \neq \bar{\mathbf{y}}_v],
\tag{37}
$$

where $\mathbf{L}$ is the graph Laplacian of $\mathcal{G}$ and $\bar{\mathbf{Y}} \in \mathbb{R}^{n \times d}$ is constructed with $\bar{\mathbf{y}}_v = \mathbf{y}_v$ if $v \in \mathcal{V}'$ and $\|\bar{\mathbf{y}}_v\| = 0$ otherwise. This form is a special case of BloomGML when we assume $m = 1$, $f(\mathbf{h}_u, \mathbf{h}_v) = \lambda \|\mathbf{h}_u - \mathbf{h}_v\|_2^2$ with $\lambda > 0$ a hyperparameter, and $\mathbf{X} = \bar{\mathbf{Y}}$, $\kappa(\mathbf{h}_v; \mathbf{x}_v) = \|\mathbf{h}_v - \mathbf{x}_v\|_2^2 \equiv \|\mathbf{h}_v - \bar{\mathbf{y}}_v\|_2^2$. Furthermore, we generalize $\eta(\mathbf{h}_v) \to \eta(\mathbf{h}_v; \mathbf{x}_v) \equiv \eta(\mathbf{h}_v; \bar{\mathbf{y}}_v)$ and choose

$$\eta(\mathbf{h}_v; \bar{\mathbf{y}}_v) = \mathcal{I}[\|\bar{\mathbf{y}}_v\| \neq 0] \mathcal{I}_\infty[\mathbf{h}_v \neq \bar{\mathbf{y}}_v'], \tag{38}$$

where we are implicitly assuming that $\|\bar{\mathbf{y}}_v\| \neq 0$ for any $v \in \mathcal{V}'$. The first two terms of (37) are smooth, while the last enforces that embeddings for nodes with observable labels must equal those labels. We may minimize using proximal gradient steps of the form

$$
\begin{aligned}
\mathbf{H}^{(l)} &= \mathbf{prox}_{\eta, \gamma} \left[ (1-\gamma)\mathbf{H}^{(l-1)} - \gamma \left( \lambda \mathbf{L} \mathbf{H}^{(l-1)} - \bar{\mathbf{Y}} \right) \right] \\
&= \mathcal{P}_{\bar{Y}} \left[ (1-\gamma)\mathbf{H}^{(l-1)} - \gamma \left( \lambda \mathbf{L} \mathbf{H}^{(l-1)} - \bar{\mathbf{Y}} \right) \right],
\end{aligned} \tag{39}
$$

where $\mathcal{P}_{\bar{Y}} : \mathbb{R}^{n \times c} \to \mathbb{R}^{n \times c}$ is a projection operator that leaves rows $v \notin \mathcal{V}'$ unchanged, while setting rows $v \in \mathcal{V}'$ as $\mathbf{h}_v \to \bar{\mathbf{y}}_v = \mathbf{y}_v$. Several points are worth noting regarding this result:

- For different choices of $\gamma$ and $\lambda$, as well as the possible inclusion of a gradient preconditioner, the update from (39) can exactly reproduce various forms of label propagation (LP) (Zhou et al., 2003; Zhu, 2005), which represents a family of semi-supervised learning algorithms designed to predict unlabeled nodes by propagating observed labels across edges of the graph.

- As a representative LP example, we include a gradient preconditioner $\mathbf{D}^{-1}$ within (39), where $\mathbf{D}$ is the diagonal degree matrix of $\mathcal{G}$, and choose $\lambda \to \infty$, $\gamma \to 0$ such that $\lambda \gamma \to 1$. Then (39) reduces to the standard, interpretable form

$$\mathbf{H}^{(l)} = \mathcal{P}_{\bar{Y}} \left[ \mathbf{D}^{-1} \mathbf{A} \mathbf{H}^{(l-1)} \right], \tag{40}$$

  where $\mathbf{A}$ is the adjacency matrix of $\mathcal{G}$.

- The input argument to $\mathcal{P}_{\bar{Y}}[\ \cdot\ ]$ in (39) exactly reduces to (4) across all $v \in \mathcal{V}$ if we replace the $\pi$ used in (4) with the corresponding labels $\bar{\mathbf{y}}_v$; this follows from the derivations presented in Section B. Moreover, the node-wise projection operator $\mathcal{P}_{\bar{Y}}$ merely introduces a nonlinear activation function which can be merged into an update function $\hat{f}_U$. In this way, executing the LP model propagation steps induces node embeddings analogous to those from a message-passing GNN architecture.

- If we include the $\eta$-based indicator term in (37), then w.l.o.g. we can actually remove the $\kappa$-based term and still reproduce some notable LP variants. However, we include the general form for two reasons: (i) It will facilitate more direct comparisons with graph-regularized MLP models below, and (ii) If we instead remove the $\eta$-based indicator term, then we can nonetheless recoup other flavors of LP without a projection step by retaining the $\kappa$-based term.

- There exist more exotic LP models that are not directly covered by (37), such as ZooBP (Eswaran et al., 2017) and CAMLP (Yamaguchi et al., 2016), which include compatibility matrices to address graphs beyond vanilla homogeneous structure. These can be accommodated by generalizing (37); however, we do not pursue this course further here.

We close by noting that, LP methods traditionally do not have parameters $\mathbf{W}$ to train, and the motivation for including an upper-level loss $\ell_{up}$ analogous to BloomGML may be unclear. However, recently trainable variants have been proposed (Wang et al., 2022) that incorporate randomized sampling to mitigate label leakage. BloomGML can cover many such cases by supplementing (37) with parameters that can be subsequently trained using, for example, a node classification loss for $\ell_{up}$.

## G.2 Graph-Regularized MLPs, BloomGML, and Connections with LP

Graph-regularized MLPs (GR-MLP) represent a class of models typically applied to node classification tasks whereby no explicit message-passing occurs within actual layers of the architecture itself (Ando and Zhang,

2006; Hu et al., 2021; Zhang et al., 2023). Instead, a base model (like an MLP or related) is trained using a typical supervised learning loss combined with an additional regularization factor designed to push together the predictions of node labels sharing an edge. These models are often motivated by their efficiency at inference time, where the graph structure is no longer utilized.

As a simple representative example,[3] consider the loss

$$\ell(\mathbf{W}; \mathcal{G}) := \tfrac{1}{2}\|\bar{\mathbf{Y}} - \mathbf{H}\|_{\mathcal{F}}^2 + \tfrac{\lambda}{2}\mathrm{tr}\left[\mathbf{H}^\top \mathbf{L}\mathbf{H}\right], \quad \text{s.t.} \quad \mathbf{h}_v = \pi(\mathbf{x}_v; \mathbf{W}) \quad \forall v \in \mathcal{V}. \tag{41}$$

Here we are assuming that $\bar{\mathbf{Y}}$ is defined analogously as in Section G.1. In this expression, the first term represents quadratic supervision applied to node labels $\bar{\mathbf{Y}}$, while the second term introduces the eponymous graph-regularization with graph Laplacian $\mathbf{L}$. Within the constraint, $\pi$ is some node-wise trainable model such as an MLP. Hence model training amounts to learning a $\mathbf{W}$ such that the resulting node embeddings closely match the given labels subject to smoothing across the graph. For this purpose, we take gradient steps

$$\mathbf{W}^{(l)} = \mathbf{W}^{(l-1)} - \gamma \left.\frac{\partial \ell(\mathbf{W}; \mathcal{G})}{\partial \mathbf{W}}\right|_{\mathbf{W}=\mathbf{W}^{(l-1)}}, \quad l = 1, \ldots, L. \tag{42}$$

We now show how these steps induce implicit message-passing GNN-like structure in certain cases, and later, connect to label propagation. To see this effect in transparent terms, assume that $\pi(\mathbf{x}; \mathbf{W}) = \mathbf{x}\mathbf{W}$. Then we have

$$\frac{\partial \ell(\mathbf{W}; \mathcal{G})}{\partial \mathbf{W}} = \mathbf{X}^\top \left(\mathbf{H} + \lambda \mathbf{L}\mathbf{H} - \bar{\mathbf{Y}}\right). \tag{43}$$

From here we define $\mathbf{H}^{(l)} := \mathbf{X}\mathbf{W}^{(l)}$ such that the above gradient steps w.r.t. $\mathbf{W}$ produce a corresponding sequence of embeddings given by

$$\mathbf{H}^{(l)} = \mathbf{H}^{(l-1)} - \gamma \mathbf{X}\mathbf{X}^\top \left(\mathbf{H}^{(l-1)} + \lambda \mathbf{L}\mathbf{H}^{(l-1)} - \bar{\mathbf{Y}}\right). \tag{44}$$

Provided that $\mathbf{X}$ is full column rank, we can assume w.l.o.g. that columns of $\mathbf{X}$ are orthonormal, i.e., $\mathbf{X}^\top \mathbf{X} = \mathbf{I}$. (This is possible because we can always convert non-orthonormal columns into orthonormal ones via an invertible linear transformation that can be absorbed into $\mathbf{W}$.) Consequently, we may directly conclude that $\mathbf{X}\mathbf{X}^\top = \mathcal{P}_X$, where $\mathcal{P}_X$ indicates an orthogonal projection to the range of $\mathbf{X}$, denoted range[$\mathbf{X}$]. We then provide the following straightforward interpretations of this process:

- From the perspective of BloomGML, we can convert (41) to an equivalent $\ell_{low}$ given by

$$\ell_{low}(\mathbf{H}; \mathbf{W}, \mathcal{G}) := \tfrac{\lambda}{2} \sum_{(u,v)\in\mathcal{E}} \|\mathbf{h}_u - \mathbf{h}_v\|_2^2 + \sum_{v\in\mathcal{V}} \tfrac{1}{2} \|\mathbf{h}_v - \bar{\mathbf{y}}_v\|_2^2 + \mathcal{I}_\infty\{\mathbf{H} \notin \mathrm{range}[\mathbf{X}]\}, \tag{45}$$

  where we are treating the labels from $\bar{\mathbf{Y}}$ as auxiliary inputs to the $\kappa$ function, and we are trivially generalizing the non-smooth $\eta$ function to apply across all of $\mathbf{H}$ while including a dependency on the original node features from $\mathbf{X}$. Additionally, the proximal gradient update for minimizing (45) is given by

$$\mathbf{H}^{(l)} = \mathcal{P}_X \left[(1 - \gamma)\mathbf{H}^{(l-1)} - \gamma\left(\lambda \mathbf{L}\mathbf{H}^{(l-1)} - \bar{\mathbf{Y}}\right)\right], \tag{46}$$

  which is equivalent to (44) provided $\mathbf{H}^{(0)} \in \mathrm{range}[\mathbf{X}]$. Analogous to the LP derivations from Section G.1, the input argument to $\mathcal{P}_X[\ \cdot\ ]$ in (46) exactly reduces to (4) across all $v \in \mathcal{V}$ if we replace the $\pi$ used in (4) with the corresponding labels $\bar{\mathbf{y}}_v$; this follows from the derivations presented in Section B. In this way, training GR-MLP model weights induces node embeddings analogous to those from a message-passing GNN architecture, the final projection operator notwithstanding.

- If $d = n$, then $\mathcal{P}_X = \mathbf{X}\mathbf{X}^\top = \mathbf{I}$ and we are operating in an overparameterized regime where $\mathbf{H} = \mathbf{W}$. In this special case, the iterations from (46) exactly reproduce LP from Section G.1, excluding the projection step as some LP variants do.

- In the more typical setting with $d < n$, the lingering difference between (46) and LP hinges on the projection operator being invoked, $\mathcal{P}_X$ versus $\mathcal{P}_{\bar{Y}}$ (where $\mathcal{P}_{\bar{Y}}$ is defined in Section G.1; it is *not* equivalent to a projection to range[$\bar{\mathbf{Y}}$]). Which operator is to be preferred depends heavily on the quality of input features. If range[$\mathbf{X}$] closely aligns with range[$\mathbf{Y}$] (the range space of the true label matrix), then $\mathcal{P}_X$ will be preferable. In contrast, when the relationship is closer to range[$\mathbf{X}$] $\perp$ range[$\mathbf{Y}$], then $\mathcal{P}_{\bar{Y}}$ will generally be superior.

---

[3]Admittedly, this example does not cover the full diversity of possible GR-MLP models in the literature. Nonetheless, it remains a useful starting point for analysis purposes.

### G.3 Empirical Illustration of LP/GR-MLP Insights

To illustrate some of the concepts from the previous section, we conduct a simple experiment involving Cora and Citeseer data. We choose $\pi(\mathbf{x}; \mathbf{W}) = \mathbf{x}\mathbf{W}$ and train GR-MLP models based on (46) using (i) original node features, and (ii) overparameterized features whereby $d = n$. We also compare with an LP model, excluding the projection step for a more direct evaluation.

Node classification results from these experiments are shown in Table 6. The first two rows confirm the equivalence between GR-MLP and LP for the reasons detailed in Section G.2. Furthermore, by examining the second and third rows, we are able to observe how the quality of node features influences the efficacy of $\mathcal{P}_X$. Specifically, it has been previously demonstrated that the features and labels of Cora nodes have minimal positive (linear) correlation (Luo et al., 2021). Therefore, from Table 6 we see that the GR-MLP model using the original features performs much worse than the overparameterized version. In contrast, with Citeseer where the input features are more aligned with the labels, the original feature model significantly outperforms the overparameterized one.

Table 6: *Node classification accuracy (%) comparisons.* Results are averaged over 5 trials; error bars (not shown) are negligible.

|  | Cora | Citeseer |
|---|---|---|
| Label Propagation | 70.2 | 50.2 |
| GR-MLP Overparam. | 70.2 | 50.2 |
| GR-MLP Original | 60.4 | 64.1 |

## H Model Implementation and Section 5 Experiment Details

In this section we provide further details regarding the experiments from Section 5.

**Details for Table 1.** Our implementation is based on modifications of the public codebase from Yang et al. (2021). We adopt the hyperparameters they reported for all datasets. For BloomGML, we apply a vectorized version of a Huber penalty in the form

$$\delta(\mathbf{u}) := \sum_{i=1}^{d} \begin{cases} \frac{1}{2}u_i^2, & \text{for } |u_i| < 1 \\ |u_i| - \frac{1}{2}, & \text{otherwise,} \end{cases} \tag{47}$$

where $\{u_i\}_{i=1}^{d}$ are the elements of the input vector $\mathbf{u} \in \mathbb{R}^d$. We then form the energy function

$$\ell_{low}(\mathbf{H}; \mathbf{W}, \mathcal{G}) = \sum_{(u,v)\in\mathcal{E}} f(\mathbf{h}_u, \mathbf{h}_v; r) + \sum_{v\in\mathcal{V}} \kappa(\mathbf{h}_v; \mathbf{x}_v) = \frac{\lambda}{2} \sum_{(u,v)\in\mathcal{E}} \|\mathbf{h}_u - \mathbf{h}_v\|_2^2 + \sum_{v\in\mathcal{V}} \delta\left(\mathbf{h}_v - \pi(\mathbf{x}_v; \mathbf{W})\right), \tag{48}$$

where the dependency of $f$ on $r$ is irrelevant for homogeneous graphs. Note that because (48) is smooth and differentiable (i.e., the Huber loss is differentiable and the $\ell_1$ norm defaults to merely a summation since its argument is non-negative), no $\eta$ term is needed. Hence, we choose $\mathcal{A}$ as gradient descent for producing the BloomGML model $\mathcal{A}[\ell_{low}, \cdot]$. For $\ell_{up}$ we apply a standard node classification loss. We reproduce Table 1 from the main text with error bars obtained from averaging over 10 trials; these results are presented in Table 7.

Table 7: Full version of Table 1 including error bars from averaging over 10 runs. *Performance mitigating spurious input features.* Here the detect ratio is obtained by computing $\|\mathbf{h}_v - \pi(\mathbf{x}_v; \mathbf{W})\|_2$ for all $v \in \mathcal{V}$ and then segmenting out the percentage of corrupted nodes within the largest 20%.

|  |  | Cora | Citeseer | Pubmed | Arxiv |
|---|---|---|---|---|---|
| Accuracy | Base | $54.97 \pm 1.95$ | $38.98 \pm 1.96$ | $45.67 \pm 3.21$ | $47.63 \pm 0.23$ |
|  | BloomGML | $\mathbf{65.83 \pm 2.28}$ | $\mathbf{49.33 \pm 1.95}$ | $\mathbf{72.70 \pm 1.58}$ | $\mathbf{69.11 \pm 0.23}$ |
| Detect Ratio | BloomGML | 94.27 | 87.82 | 96.98 | 100.00 |

**Details for Table 2.** Building on the results from above and the description in Section 5 in the main text, we consider the generalized form

$$\ell_{low}(\mathbf{H}; \mathbf{W}, \mathcal{G}) = \sum_{(u,v) \in \mathcal{E}} f(\mathbf{h}_u, \mathbf{h}_v; r) + \sum_{v \in \mathcal{V}} \kappa(\mathbf{h}_v; \mathbf{x}_v) = \frac{\lambda}{2} \sum_{(u,v) \in \mathcal{E}} \|\mathbf{h}_u \mathbf{C} - \mathbf{h}_v\|_2^2 + \sum_{v \in \mathcal{V}} \delta(\mathbf{h}_v - \pi(\mathbf{x}_v; \mathbf{W})). \quad (49)$$

As before, this expression is differentiable so we can choose $\mathcal{A}$ as gradient descent with step-size parameter $\gamma$. All of the hyperparameters were selected via Bayesian optimization using *Wandb*[4] and we also provide the model hyperparameters search space in Table 9. Note that we also found that it can be effective to simply learn $\lambda$, i.e., absorb it into $\mathbf{W}$, which reduces the hyperparameter search space and can potentially even improve performance. We show such supporting ablations in Section J.1.

In Table 8 we reproduce results from Table 2, including error bars from averaging over 10 runs and an additional ablation. For the latter, we compare $\kappa(\mathbf{h}; \mathbf{x}) = \|\mathbf{h} - \pi(\mathbf{x}; \mathbf{W})\|_2^2$ labeled as BloomGML (w/o Huber) with the $\kappa(\mathbf{h}; \mathbf{x}) = \delta(\mathbf{h}_v - \pi(\mathbf{x}_v; \mathbf{W}))$ implied within (49). From Table 8 we observe that the Huber version works better on 4 of 5 datasets, the exception being Amazon where it is significantly worse. As one candidate explanation for this exception, we note that the Amazon dataset has a significantly higher *adjusted homophily* ratio than the other 4 datasets, where adjusted homophily is defined as in Platonov et al. (2022) (this metric is less sensitive to the number of classes and their balance).

We also trained several standard GNN architectures, e.g., GCN and GAT, for reference purposes; however, we found that their performance was considerably worse than the heterophily baseline architectures from Table 8 (and therefore Table 2 as well). This is in contrast to Platonov et al. (2023), which presents somewhat stronger results associated with such popular architectures. However, this apparent contradiction can be resolved by closer examination of the public implementations from Platonov et al. (2023). Here we observe that additional modifications to the original models have been introduced to the codebase that can significantly alter performance. While certainly valuable to consider, these types of enhancements can be widely applied and lie beyond the scope of our bilevel optimization lens. Indeed the flexibility of BloomGML also readily accommodates further architectural adjustments with the potential to analogously improve performance as well on specific heterophily tasks.

Table 8: Full version of Table 2 including additional ablations and error bars from averaging over 10 runs. *Node classification on heterophily graphs.* Following convention, accuracy (%) is reported for Roman and Amazon, while ROC-AUC (%) is used for Minesweeper, Tolokers, and Questions. The numbers in the upper block are from Platonov et al. (2023).

| | Roman | Amazon | Minesweep | Tolokers | Questions | Avg. |
|---|---|---|---|---|---|---|
| FAGCN | 65.22 ± 0.56 | 44.12 ± 0.30 | 88.17 ± 0.73 | 77.75 ± 1.05 | 77.24 ± 1.26 | 70.50 |
| FSGNN | 79.92 ± 0.56 | 52.74 ± 0.83 | 90.08 ± 0.70 | 82.76 ± 0.61 | **78.86 ± 0.92** | 76.87 |
| GBK-GNN | 74.57 ± 0.47 | 45.98 ± 0.71 | 90.85 ± 0.58 | 81.01 ± 0.67 | 74.47 ± 0.86 | 73.38 |
| JacobiConv | 71.14 ± 0.42 | 43.55 ± 0.48 | 89.66 ± 0.40 | 68.66 ± 0.65 | 73.88 ± 1.16 | 69.38 |
| Base from (3) | 76.63 ± 0.24 | 52.37 ± 0.20 | 88.97 ± 0.05 | 80.91 ± 0.24 | 76.72 ± 0.49 | 75.12 |
| BloomGML (w/o Huber) | 84.45 ± 0.31 | **52.92 ± 0.39** | 91.83 ± 0.28 | 84.84 ± 0.27 | 77.62 ± 0.33 | 78.33 |
| BloomGML (w/ Huber) | **85.26 ± 0.25** | 51.00 ± 0.47 | **93.30 ± 0.16** | **85.92 ± 0.14** | 77.93 ± 0.34 | **78.68** |

Table 9: Model hyperparameters selection search space for Tables 2 and 3. For BloomGML (w/o Huber), $\lambda$ is a learnable parameter that is not searched by BO; see Section J.1 for an ablation on learning $\lambda$.

| Hyperparameters | Range for Table 2 | Range for Table 3 |
|---|---|---|
| $L$ | [2,4,6] | [4, 8, 16] |
| $\lambda$ | [0.1 , 1, 5, 10] | [0.01, 0.1 , 1, 10] |
| $\gamma$ | [0 , 0.5, 1] | [0 , 0.1, 0.5, 1] |
| MLP layers before prop | [0, 1, 2] | [0, 1, 2] |
| Hidden size | [128,256,512] | [16, 64, 128] |

---

[4]https://wandb.ai/site

**Details for Figure 1.** To show the effect of BloomGML to have connected nodes with deviated embeddings, we use a Huber penalty for the edge-dependent regularizer term leading to the energy function

$$\ell_{low}(\mathbf{H}; \mathbf{W}, \mathcal{G}) = \sum_{(u,v)\in\mathcal{E}} f(\mathbf{h}_u, \mathbf{h}_v; r) + \sum_{v\in\mathcal{V}} \kappa(\mathbf{h}_v; \mathbf{x}_v) = \frac{\lambda}{2} \sum_{(u,v)\in\mathcal{E}} \delta(\mathbf{h}_u - \mathbf{h}_v) + \frac{1}{2} \sum_{v\in\mathcal{V}} \|\mathbf{h}_v - \pi(\mathbf{x}_v; \mathbf{W})\|_2^2. \quad (50)$$

For the experiments, we fixed the hidden embedding size to 1024, which gives the model more capacity for handling heterphility, and we run Bayesian optimization with the same range as in the left block of Table 9 for the remaining hyperparameters.

**Details for Table 3.** For these results involving heterogeneous graphs, we modify the codebase from the HALO model (Ahn et al., 2022) and adopt $\kappa(\mathbf{h}; \mathbf{x}) = \delta(\mathbf{h}_v - \pi(\mathbf{x}_v; \mathbf{W}))$ or $\kappa(\mathbf{h}; \mathbf{x}) = \omega(\mathbf{h}_v - \pi(\mathbf{x}_v; \mathbf{W}))$, where $\omega$ is the Log-Cosh loss given by

$$\omega(\mathbf{u}) := \sum_{i=1}^{d} \log(\cosh u_i). \quad (51)$$

The hyperparameters we used are also selected via Bayesian optimization with Wandb; the search range is provided in the right block of Table 9, for both HALO and BloomGML. Note that we observe negligible variance in the test accuracies across different seeds, so we omit error bars from multiple runs.

**Details for Table 4.** We provide specifics directly tied to creating Table 4 here, while a more comprehensive treatment of how NBFNet relates to BloomGML is deferred to Section I. Our implementation is based on modifications of the public codebase from Zhu et al. (2021c), which involves (among other things) two trainable linear transformations. The first involves parameters $\mathbf{W}_r$ and $\mathbf{b}_r$ used to create the relation embeddings for each relation type via $\mathbf{e}_r = \mathbf{W}_r\mathbf{q}_q + \mathbf{b}_r$, where $\mathbf{q}_q$ is the query embedding. The other is $\Phi$ used to transform the output of the aggregation function back to the embedding space. To satisfy the criteria of BloomGML, we modify the code to share these weights across all layers. Performance results for RefactorGNN (Chen et al., 2022) were taken from the original paper due to lack of publicly-available reproducible code for conducting our own comparisons.

**Details for Table 5 and Figure 2.** For Table 5, we modify the codebase from Yang et al. (2021) by using the update step defined in (31). The components are chosen based on (4). The hyperparameters are chosen via grid search over $\lambda$, $\gamma$ and $\beta$. For Figure 2, we use the same set of hyperparameters for both SGD and BloomGML (momentum), with the number of message-passing layers being 200.

## I   Exploring the Connection between NBFNet and BloomGML

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{R}, \mathcal{E})$, let $\mathcal{G}_{cq}$ denote a conditional version with the same node, relation, and edge set, but with node features $\mathbf{X}_{cq} = \{\mathbf{x}_{v,cq}\}_{v\in\mathcal{V}}$ and labels $\mathbf{Y}_{cq} = \{\mathbf{y}_{v,cq}\}_{v\in\mathcal{V}}$ that depend on a fixed source node $c \in \mathcal{V}$ (in this section $c$ should not be confused with the class label dimension used elsewhere) and query relation $q \in \mathcal{R}$. By conditioning in this way, KGC or link prediction queries specific to $c$ and $q$ can be converted to node classification, e.g. $\mathbf{y}_{v,cq} = 1$ if $(c, q, v)$ is a true edge in the original graph. We may then later expand to unrestricted link prediction tasks by replicating across all $c \in \mathcal{V}$ and $q \in \mathcal{R}$. This is the high-level strategy of NBFNet.

In this section, we first show that given a graph $\mathcal{G}_{cq}$ described as above, an NBFNet-like architecture with parameters shared across layers can be induced using BloomGML iterations of the form $\mathcal{A}[\ell_{low}, \cdot]$ for suitable choice of energy $\ell_{low}$ and $\mathcal{A}$. For this purpose, we define the lower-level energy function as

$$\ell_{low}(\mathbf{H}_{cq}; \mathbf{W}, \mathcal{G}_{cq}) := \quad (52)$$
$$\sum_{(u,r,v)\in\mathcal{E}_{cq}} \left(\mathbf{h}_{v,cq}^\top \Phi \mathbf{h}_{u,cq} + \mathbf{h}_{v,cq}^\top \Phi \mathbf{e}_r + \mathbf{h}_{u,cq}^\top \Phi \mathbf{e}_{r^{-1}}\right) + \frac{1}{2} \sum_{v\in\mathcal{V}_{cq}} \left[\|\Psi^{\frac{1}{2}}\mathbf{h}_{v,cq} + \Psi^{-\frac{1}{2}}\Phi\mathbf{x}_{v,cq}\|_2^2 + \eta(\mathbf{h}_{v,cq})\right],$$

where $\mathbf{H}_{cq} = \{\mathbf{h}_{v,cq}\}_{v\in\mathcal{V}}$ are node embeddings and $\mathbf{e}_r := \mathbf{W}_r\mathbf{q} + \mathbf{b}_r$ are relation embeddings, $\mathbf{q}$ is a so-called shared query relation embedding (distinct from the relation index of query $q$), and $\mathbf{W}_r$ and $\mathbf{b}_r$ are trainable parameters that are bundled within $\mathbf{W}$ along with $\Phi, \Psi \in \mathbb{R}^{d\times d}$. Additionally, $\mathbf{x}_{v,cq} := \mathcal{I}(c = v)\mathbf{q}$ for all $v \in \mathcal{V}_{cq}$. To ensure the gradient expressions are symmetric in form across relations and inverse relations, we require that $\Phi$ and $\Psi$ are symmetric matrices. However, it has been shown in Yang et al. (2022b) that in certain circumstances,

asymmetric weights can be reproduced by symmetric ones via an appropriate expansion of the hidden dimensions. Let

$$f(\mathbf{h}_{u,cq}, \mathbf{h}_{v,cq}; r) = \mathbf{h}_{v,cq}^\top \Phi \mathbf{h}_{u,cq} + \mathbf{h}_{v,cq}^\top \Phi \mathbf{e}_r + \mathbf{h}_{u,cq}^\top \Phi \mathbf{e}_{r^{-1}} \tag{53}$$

Then we have

$$\begin{aligned}
\frac{\partial f\left(\mathbf{h}_{u,cq}, \mathbf{h}_{v,cq}; r\right)}{\partial \mathbf{h}_{v,cq}} &= \Phi \mathbf{h}_{u,cq} + \Phi \mathbf{e}_r =: \Phi \widetilde{f}_M(\mathbf{h}_{u,cq}, r, \mathbf{h}_{v,cq}) \\
\frac{\partial f\left(\mathbf{h}_{u,cq}, \mathbf{h}_{v,cq}; r\right)}{\partial \mathbf{h}_{u,cq}} &= \Phi \mathbf{h}_{v,cq} + \Phi \mathbf{e}_{r^{-1}} =: \Phi \widetilde{f}_M(\mathbf{h}_{u,cq}, r^{-1}, \mathbf{h}_{v,cq})
\end{aligned} \tag{54}$$

Thus, the aggregated function is

$$\mathbf{a}_{v,cq} = \sum_{(u,r) \in \mathcal{N}_v} \widetilde{f}_M(\mathbf{h}_{u,cq}, r, \mathbf{h}_{v,cq}) = \sum_{(u,r) \in \mathcal{N}_{v,cq}} (\mathbf{h}_{u,cq} + \mathbf{e}_r) \tag{55}$$

Next, let

$$\kappa(\mathbf{h}_{v,cq}; \mathbf{x}_{v,cq}) = ||\Psi^{\frac{1}{2}} \mathbf{h}_{v,cq} + \Psi^{-\frac{1}{2}} \Phi \mathbf{x}_{v,cq}||_2^2 \tag{56}$$

Then, by reintroducing $\eta$ and adopting proximal gradient descent for $\mathcal{A}$, we produce the message-passing function

$$\begin{aligned}
\boldsymbol{\mu}_{(u,r,v),cq}^{(l)} &:= \mathbf{h}_{u,cq}^{(l-1)} + \mathbf{e}_r^{(l-1)} \\
\mathbf{a}_{v,cq}^{(l)} &:= \sum_{(u,r) \in \mathcal{N}_{v,cq}} \boldsymbol{\mu}_{(u,r,v),cq}^{(l)} \\
\mathbf{h}_{v,cq}^{(l)} &:= \mathbf{prox}_{1,\eta} \left( \Phi \left[ \mathbf{a}_{v,cq}^{(l)} + \mathbf{x}_{v,cq} \right] + \Psi \mathbf{h}_{v,cq}^{(l-1)} \right).
\end{aligned} \tag{57}$$

This expression closely resembles an NBFNet model layer with summation aggregation.

Proceeding further, we extend the scope of our discussion to an expanded graph $\mathcal{G}_{exp}$ with node set $\mathcal{V}_{exp}$ and edge set $\mathcal{E}_{exp}$, where $\mathcal{G}_{exp} = \cup \mathcal{G}_{cq}$, $\forall (c,q,\cdot) \in \mathcal{E}_{exp}$. Analogously, we define an extended energy function over $\mathcal{G}_{exp}$ as

$$\begin{aligned}
\ell_{low}(\mathbf{H}; \mathbf{W}, \mathcal{G}_{exp}) &:= \sum_{(c,q,\cdot) \in \mathcal{E}_{exp}} \ell_{low}(\mathbf{H}_{cq}; \mathbf{W}, \mathcal{G}_{cq}) \\
&= \sum_{(c,q,\cdot) \in \mathcal{E}_{exp}} \left( \sum_{(u,r,v) \in \mathcal{E}_{cq}} \left( \mathbf{h}_{v,cq}^\top \Phi \mathbf{h}_{u,cq} + \mathbf{h}_{v,cq}^\top \Phi \mathbf{e}_r + \mathbf{h}_{u,cq}^\top \Phi \mathbf{e}_{r^{-1}} \right) \right. \\
&\qquad \left. + \frac{1}{2} \sum_{v \in \mathcal{V}_{cq}} \left[ ||\Psi^{\frac{1}{2}} \mathbf{h}_{v,cq} + \Psi^{-\frac{1}{2}} \Phi \mathbf{x}_{v,cq}||_2^2 + \eta(\mathbf{h}_{v,cq}) \right] \right)
\end{aligned} \tag{58}$$

For elements in $\mathbf{Y}_{cq}$, positive instances correspond with true edges in the original graph, while negative instances are obtained by sampling. With $\mathbf{Y}_{exp} := \cup \mathbf{Y}_{cq}$, we define an upper-level energy as

$$\ell_{up}(\mathbf{W}, \mathbf{Y}_{exp}; \Theta) := \sum_{(c,q,\cdot) \in \mathcal{E}_{exp}} \sum_{v \in \mathcal{V}'_{cp}} \mathcal{D} \left[ g \left( \mathbf{h}_{v,cq}^{(L)}(\mathbf{W}); \Theta \right), \mathbf{y}_{v,cq} \right], \tag{59}$$

where $\mathcal{V}'_{cq}$ represents the node set of each $\mathcal{G}_{cq}$ with observed labels. In this way, the resulting bilevel optimization process can be viewed as instantiating a BloomGML node classification task on the expanded graph $\mathcal{G}_{exp}$, with architecture mimicking a specialized NBFNet-like model designed to minimize $\ell_{low}$.

## J  Additional Experiments

In this section we include additional ablations and further demonstration of the versatility of BloomGML.

## J.1 Ablation on Learning $\lambda$

There are two potential advantages to learning differentiable hyperparameters within $\ell_{low}$. First, relative to hyperparameter tuning, it may be possible to boost performance. And secondly, even if performance improvements are stubborn, by learning such hyperparameters, we economize the sweep over any remaining hyperparameters as the effective search space has compressed along one dimension. In this regard, Table 10 shows results on the heterophily datasets (and the same setup from Tables 2 and 8) with and without learning $\lambda$. From these results we observe that learning $\lambda$ produces reliable performance, suggesting that it can be trained instead of tuned (as a hyperparameter) without sacrificing accuracy.

Table 10: *Ablation on learnable $\lambda$*. Accuracy results are obtained by averaging over 10 trials.

|  | Roman | Amazon | Minesweep | Tolokers | Questions | Avg. |
|---|---|---|---|---|---|---|
| w/o Huber | $84.28 \pm 0.63$ | $52.76 \pm 0.27$ | $91.89 \pm 0.25$ | $84.66 \pm 0.33$ | $77.46 \pm 0.30$ | 78.21 |
| w/o Huber (learnable $\lambda$) | $84.45 \pm 0.31$ | $\mathbf{52.92 \pm 0.39}$ | $91.83 \pm 0.28$ | $84.84 \pm 0.27$ | $77.62 \pm 0.33$ | 78.33 |
| w/ Huber | $85.26 \pm 0.25$ | $51.00 \pm 0.47$ | $\mathbf{93.30 \pm 0.16}$ | $\mathbf{85.92 \pm 0.14}$ | $77.93 \pm 0.34$ | **78.68** |
| w/ Huber (learnable $\lambda$) | $\mathbf{85.36 \pm 0.36}$ | $51.32 \pm 0.43$ | $92.70 \pm 0.60$ | $85.86 \pm 0.18$ | $\mathbf{78.01 \pm 0.45}$ | 78.65 |

## J.2 Efficiently Enforcing Layernorm Using $\eta$

If for some reason we prefer to have a model with layer-normalized node embeddings, one option is to introduce a penalty of the form $(\|\mathbf{h}\|_2^2 - 1)^2$, absorb this factor into $\kappa(\mathbf{h}; \mathbf{x})$, and pthen roceed to minimize $\ell_{low}$ using regular gradient descent for $\mathcal{A}$. However, this requires an additional trade-off hyperparameter, and if we want to be arbitrarily close to the unit norm, convergence will become prohibitively slow. Alternatively, we can enforce strict adherence to unit norm by choosing $\eta(\mathbf{h}) = \mathcal{I}_\infty\{\|\mathbf{h}\|_2^2 \neq 1\}$ and then adopt proximal gradient descent for $\mathcal{A}$ to handle the resulting discontinuous loss surface. This involves simply taking gradient steps on all other differentiable terms and then projecting to the unit sphere (i.e., the proximal operator). We compare these two approaches in Figure 3, where proximal gradient descent converges far more rapidly. For the model without proximal gradient descent, we add the penalty $\alpha(\|\mathbf{h}\|_2^2 - 1)^2$ to the energy from (3), where $\alpha$ is the trade-off parameter; we choose $\alpha = 1$ in Figure 3. For the model with proximal gradient descent, we project each node embedding to the surface of a ball satisfying $\|\mathbf{h}\|_2^2 = 1$.
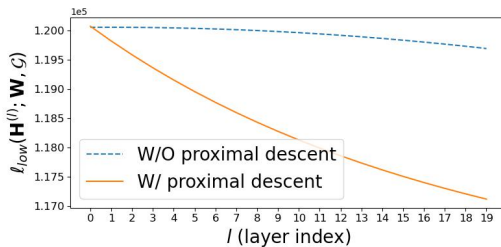


Figure 3: $\ell_{low}$ value versus the number of propagation steps in Roman dataset.