Neural Graph Databases

Anonymous Author(s)
Anonymous Affiliation
Anonymous Email

Abstract

Graph databases (GDBs) enable processing and analysis of unstructured, complex, rich, and usually vast graph datasets. Despite the large significance of GDBs in both academia and industry, little effort has been made into integrating them with the predictive power of graph neural networks (GNNs). In this work, we show how to seamlessly combine nearly any GNN model with the computational capabilities of GDBs. For this, we observe that the majority of these systems are based on, or support, a graph data model called the Labeled Property Graph (LPG), where vertices and edges can have arbitrarily complex sets of labels and properties. We then develop LPG2vec, an encoder that transforms an arbitrary LPG dataset into a representation that can be directly used with a broad class of GNNs, including convolutional, attentional, message-passing, and even higher-order or spectral models. In our evaluation, we show that the rich information represented as LPG labels and properties is properly preserved by LPG2vec, and it increases the accuracy of predictions regardless of the targeted learning task or the used GNN model, by up to 34% compared to graphs with no LPG labels/properties. In general, LPG2vec enables combining predictive power of the most powerful GNNs with the full scope of information encoded in the LPG model, paving the way for neural graph databases, a class of systems where the vast complexity of maintained data will benefit from modern and future graph machine learning methods.

1 Introduction

Graph databases are a class of systems that enable storing, processing, analyzing, and the overall management of large and rich graph datasets [22]. They are heavily used in computational biology and chemistry, medicine, social network analysis, recommendation and online purchase infrastructure, and many others [22]. A plethora of such systems exist, for example Neo4j [79] (a leading industry graph database)[1], TigerGraph [3, 108], JanusGraph [106], Azure Cosmos DB [78], Amazon Neptune [6], Virtuoso [88], ArangoDB [11–13], OrientDB [1, 31], and others [2, 25, 32, 34, 38, 41, 47, 86, 87, 89, 95, 103]. Graph databases differ from other classes of graph-related systems and workloads such as graph streaming frameworks [17] in that they deal with transactional support, persistence, physical/logical data independence, data integrity, consistency, and complex graph data models where both vertices and edges may be of different classes and may be associated with arbitrary properties.

An established data model used in the majority of graph databases is called the *Labeled Property Graph* (LPG) [22]. It is the model of choice for the leading industry Neo4j graph database system. LPG has several advantages over other graph data models, such as heterogeneous graphs [116, 123, 126] or the Resource Description Framework (RDF) [69] graphs, often referred to as knowledge graphs (see Figure 1). First, while heterogeneous graphs support different classes of vertices and edges, LPGs offer *arbitrary sets of labels as well as key-value property pairs* that can be attached to vertices and edges. This facilitates modeling very rich and highly complex data. For example, when modeling publications with graph vertices, one can use labels to model an arbitrarily complex hierarchy of types of publications (journal, conference, workshop papers; best papers, best student papers, best paper runner-ups, etc.). We discuss this example further in Section 2. Second, LPG explicitly stores the neighborhood structure of the graph, very often in the form of adjacency lists [22]. Hence, it enables very fast accesses to vertex neighborhoods and consequently fast and scalable graph algorithms and graph queries. This may be more difficult to achieve in data representations such as sets of triples. First, *any possible relation between any two entities in a graph* (i.e., edges, vertices,

[1] According to the DB engines ranking (https://db-engines.com/en/ranking/graph+dbms)

Submitted to the First Learning on Graphs Conference (LoG 2022). Do not distribute.

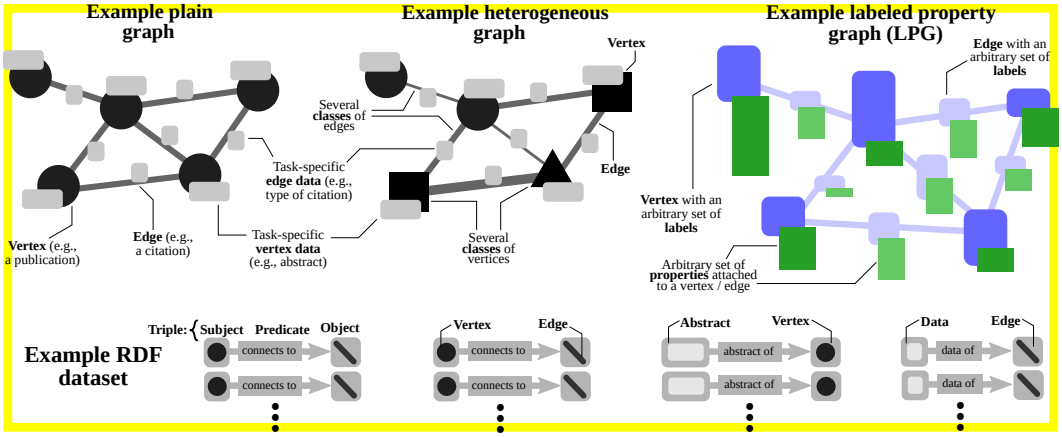

Figure 1: Overview of the Labeled Property Graph (LPG) data model used in graph databases, vs. plain and heterogeneous graphs used in broad graph processing and graph machine learning, and RDF triples.

and *any* other data) is explicitly maintained as a separate triple (see Figure 1). Second, *any entity is fundamentally the same "resource"*, where "vertex" or "edge" are just roles assigned to a given resource; these roles can differ in different triples (i.e., one resource can be both a vertex and an edge, depending on a specific triple). Hence, RDF graphs may need more storage, and they may require more complex indexing structures for vertex neighborhoods, than in the corresponding LPGs.

Graph neural networks (GNNs) have recently become an established part of the machine learning (ML) landscape [18, 20, 30, 35, 50, 52, 53, 102, 122, 132, 134]. Example applications are node, link, or graph classification or regression in social sciences, bioinformatics, chemistry, medicine, cybersecurity, linguistics, transportation, and others. GNNs have been successfully used to provide cost-effective and fast placement of chips [80], improve the accuracy of protein folding prediction [59], simulate complex physics [91, 100], or guide mathematical discoveries [39]. The versatility of GNNs brings a promise of enhanced analytics capabilities in the graph database landscape.

Recently, Neo4j Inc., Amazon, and others have started to investigate harnessing graph ML capabilities into their graph database architectures. However, current efforts only enable limited learning functionalities that do not take advantage of the full richness of data enabled by LPG. For example, Neo4j's Graph Data Science module [56] supports obtaining embeddings and using them for node or graph classification. However, these embeddings are based on the graph structure, with limited support for taking advantage of the full scope of information provided by LPG labels and properties.

Combining LPG-based graph databases with GNNs could facilitate reaching new frontiers in analyzing complex unstructured datasets, and it could also illustrate the potential of GNNs for broad industry. In this work, we first broadly investigate both the graph database setting and GNNs to find the best approach for combining these two. As a result, we develop **LPG2vec**, an encoder that enables harnessing the predictive power of GNNs for LPG graph databases. In LPG2vec, we treat labels and properties attached to a vertex v as an additional source of information that should be integrated with v's input feature vectors. For this, we show how to encode different forms of data provided in such labels/properties. This data is transformed into embeddings that can seamlessly be used with different GNN models. LPG2vec is orthogonal to the software design and can also be used with any GNN framework or graph database implementation.

We combine LPG2vec with three established GNN models (GCN [67], GIN [124], and GAT [111]), and we show that the information preserved by LPG2vec consistently enhances the accuracy of graph ML tasks, i.e., classification or regression of nodes and edges, by up to 34%. Moreover, LPG2vec supports the completion of missing labels and properties in often noisy LPGs. Overall, it enables **Neural Graph Databases**: the first learning architecture that enables harnessing both the structure and rich data (labels, properties) of LPG for highly accurate predictions in graph databases.

2 Background

We first introduce fundamental concepts and notation for the LPG model and GNNs.

2.1 Labeled Property Graph Data Model

Labeled Property Graph Model (LPG) [22] (also called the property graph [7]) is a primary established data model used in graph databases. We focus on LPG because it is supported by the majority of systems, and is a model of choice in many leading ones [22] (see Section 1).

At its core, LPG is based on the plain graph model $G = (V, E)$, where V is a set of vertices and $E \subseteq V \times V$ is a set of edges; $|V| = n$ and $|E| = m$. An edge $e = (u, v) \in E$ is a tuple of the out-vertex u (origin) and the in-vertex v (target). If G is undirected, then an edge $e = \{u, v\} \in E$ is a set of u and v. N_i and d_i denote the neighbors and the degree of a given vertex i ($N_i \subset V$); d is G's maximum degree. LPG then adds arbitrary *labels* and *properties* to vertices and edges. An LPG is formally modeled as a tuple (V, E, L, l, K, W, p). L is a set of labels. $l : V \cup E \mapsto \mathcal{P}(L)$ is a labeling function, mapping – respectively – each vertex and each edge to a subset of labels, where $\mathcal{P}(L)$ is the power set of L, containing all possible subsets of L. In addition to labels, each vertex and edge can have arbitrarily many *properties* (sometimes referenced as attributes). A property is a $(key, value)$ pair, with key being an identifier and $value$ being a corresponding value. Here, K is a set with all possible keys and W is a set with all possible values. For any property, we have $key \in K$ and $value \in W$. Then, $p : (V \cup E) \times K \mapsto W$ is a mapping function from vertices/edges to property values. Specifically, $p(u, key)$ and $p(e, key)$ assign – respectively – a value to a property indexed with a key key, of a vertex u of an edge e. Note that one can assign multiple properties with the same key to vertices and edges (i.e., only the pair $(key, value)$ must be unique).

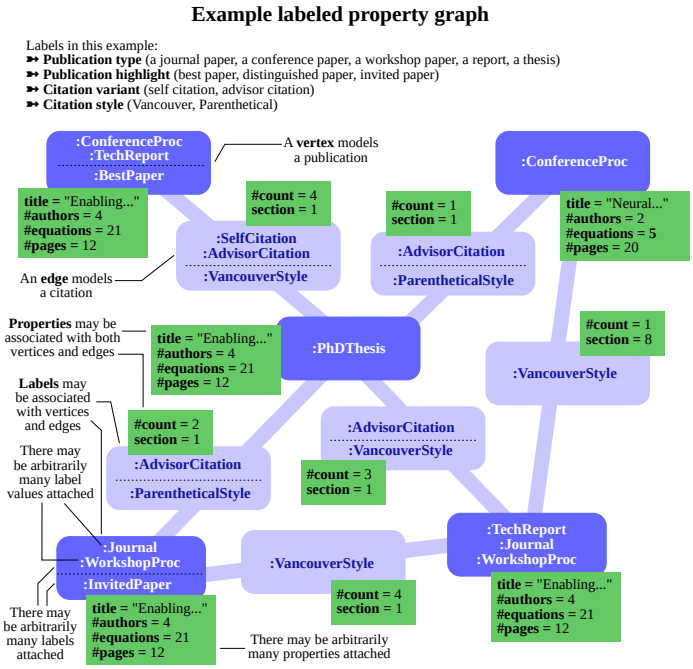

Figure 2: An example of an LPG graph modeling publications and citations between them.

We illustrate an example of an LPG graph in Figure 2.

2.2 Graph Neural Networks

Graph neural networks (GNNs) are a class of neural networks that enable learning over irregular graph datasets [102]. Each vertex (and often each edge) of the input graph usually comes with an *input feature vector* that encodes the semantics of a given task. For example, when vertices and edges model publications and citations between these papers, then a vertex input feature vector is a encoding of the publication abstract (e.g., a one-hot bag-of-words encoding specifying which words are present). Input feature vectors are transformed in a series of GNN layers. In this process, intermediate hidden latent vectors are created. The last GNN layer produces output feature vectors, which are then used for the *downstream ML tasks* such as node classification or graph classification.

Many GNN models exist [20, 33, 36, 101, 107, 120, 122, 130, 132, 134]. Most of such models consist of a series of GNN layers, and a single layer has two stages: (1) the aggregation stage that – for each vertex – combines the features of the neighbors of that vertex, and (2) the neural stage that combines the results of the aggregation with the vertex score from the previous layer into a new score. One may also explicitly distinguish stage (3), a non-linear activation over feature vectors (e.g., ReLU [67]) and/or normalization. We illustrate a simplified view of a GNN layer in Figure 3.

The input, output, and hidden feature vector of a vertex i are denoted with, respectively, $\mathbf{x}_i, \mathbf{y}_i, \mathbf{h}_i$. We have $\mathbf{x}_i \in \mathbb{R}^k$ and $\mathbf{y}_i, \mathbf{h}_i \in \mathbb{R}^{O(k)}$, k is the dimensionality of vertex input feature vectors. "(l)" denotes the l-th GNN layer; $\mathbf{h}_i^{(l)}$ are latent features in layer l.

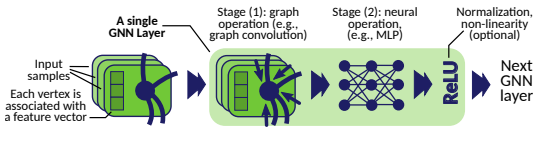

Figure 3: Overview of a single GNN layer.

Formally, the graph aggregation stage of a GNN layer can be described using two functions, ψ and $\bigoplus$. First, the feature vector of each neighbor of i is transformed by a function ψ. Then, the resulting neighbor feature vectors are aggregated using a function $\bigoplus$, such as sum or max. The outcome of $\bigoplus$ is then processed using a third function, φ, that models the neural operation and non-linearity. This gives the latent feature vector $\mathbf{h}_i$ in the next GNN layer. Combined, we have $\mathbf{h}_i^{(l+1)} = \varphi\left(\mathbf{h}_i^{(l)}, \bigoplus_{j \in N(i)} \psi\left(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}\right)\right)$. This is a generic form of GNNs, which can be used to define three major classes of GNNs [29]: *convolutional* GNNs (C-GNNs; examples are GCN [67], GraphSAGE [54], GIN [124], and CommNet [104]), *attentional* GNNs (A-GNNs; examples are MoNet [81], GAT [111], and AGNN [107]), and the most generic *message-passing* GNNs (MP-GNNs; examples are G-GCN [28], EdgeConv [118], MPNN [51], and GraphNets [14]).

To avoid confusion, we always use a term "label" to denote an LPG label, while a term "class" indicates a prediction target in classification tasks.

3 Marrying Graph Databases and Graph Neural Networks

We first investigate how LPG-based graph databases and GNNs can be combined to reach new frontiers of complex graph data analytics.

3.1 How to Use GNNs with GDBs?

It is not immediately clear on how to use GNNs in combination with the LPG data model. Specifically, labels and properties of a vertex v are often seen as additional "vertices" attached to v [84, 93]. From this perspective, it would seem natural to use them during the aggregation phase of a GNN computation, together with the neighbors of v. Similarly, one could consider incorporating attentional GNNs [29], by attending to individual labels and properties. In general, there can be many different approaches for integrating GNNs and LPGs.

Here, we first extensively investigated both the graph database (GDB) and the GNN settings. The goal was to determine the best approach for using GNNs with GDBs in order to benefit the maximum number of different GDB workloads while ensuring a seamless integration with as many GNN models as possible. We consider all major classes of GDB workloads: online transactional, analytical, and serving processing (respectively, OLTP, OLAP, OLSP) [22], and the fundamental GNN model classes (e.g., C-GNN, A-GNN, MP-GNN) [29], for a total of more than 280 analyzed publications or reports.

Our analysis indicated that the most versatile approach for extracting the information from LPG labels and properties is based on encoding labels and properties directly into the input feature vectors, and subsequently feeding such vectors into a selected GNN model. First, this approach only requires modifications to the input feature vectors, which makes LPG2vec fully compatible with any C-GNN, A-GNN, or MP-GNN model (and many others). Second, this approach is very similar in its workflow to schemes such as positional encodings: is is based on preprocessing and feeding additional information into input feature vectors. Hence, it is straightforward to integrate into existing GNN infrastructures.

3.2 Use Cases and Advantages

The first advantage of combining graph databases with GNNs is enhancing the accuracy of traditional GNN tasks: classification and regression of nodes, edges, and graphs (note that tasks such as clustering or link prediction can be expressed as node/edge classification/regression). This is because LPG labels and properties, when incorporated into input feature vectors, carry additional information. This is similar to how different classes of vertices/edges in heterogeneous graphs enhance prediction tasks [123, 126]. However, the challenge is how to incorporate the full rich set of LPG information, i.e., multiple labels and properties, into the learning workflow, while achieving high accuracy and without exacerbating running times or memory pressure.

GNNs can also be used to deliver novel prediction tasks suited for LPG, namely **label prediction** and **property prediction**. In the former, one is interested in assessing whether a given vertex or edge potentially has a specified label, i.e., whether $label \in l(v)$ or $label \in l(e)$, where $label \in L, v \in V, e \in E$ are – respectively – a label, a vertex, and an edge of interest, and l is a labeling function. In the latter, one analogously asks whether a given vertex or edge potentially has a specified property, and – if yes – what its value is, i.e., whether $property \in p(v)$ or $property \in p(e)$, where $p = (k, w), k \in K, w \in W, v \in V, e \in E$ are – respectively – a property, a vertex, and an edge of interest, and p maps v and e to their corresponding properties.

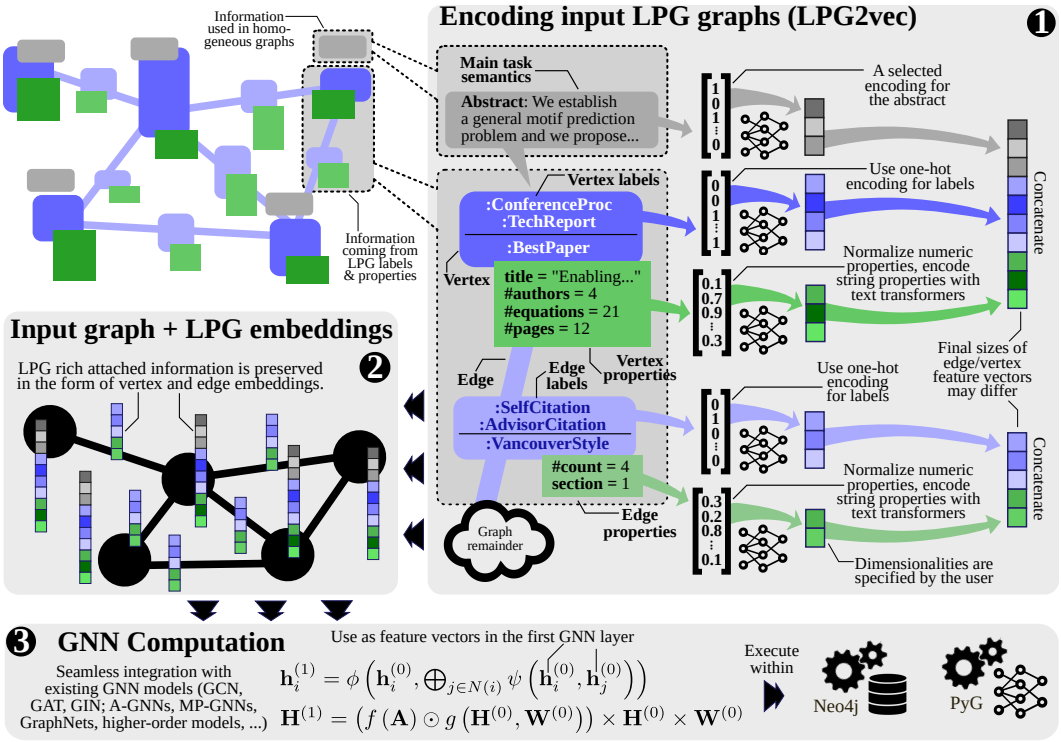

Figure 4: An overview of LPG2vec in the context of processing LPG graphs with GNNs. First ("❶"), the graph data is loaded from disk and encoded using LPG2vec. Here, we differentiate the additional data usually used with homogeneous graphs, that determines the task semantics (in this case, publication abstracts), from the LPG-related additional data (labels, properties). Note that, in practice, encoding the abstract could be just implemented as encoding an additional property. The encoding process gives a graph dataset ("❷") that is ready for the actual GNN computation that can be executed in a dedicated module of a graph database (e.g., Neo Graph Data Science) or in a dedicated ML framework (e.g., PyG). Importantly, LPG2vec preserves all the rich LPG information in the form of vertex and edge embeddings. Thus, the actual input to the GNN computation is a homogeneous graph structure together with the embeddings. This makes the integration with existing GNN models straightforward ("❸"). The computation itself is conducted in a dedicated module of a graph database (e.g., Neo4j's Graph Data Science), but - thanks to the seamless LPG2vec design (i.e., the fact that the output of LPG2vec is a homogeneous graph with enhanced feature vectors) - it can also be conducted in a standalone GNN framework (e.g., PyG).

Here, we observe that predicting new labels can be seamlessly resolved with node/edge classification, with the target learned label being l. Similarly, property prediction is effectively node/edge regression, where w is the learned value. Thus, it means that one can easily use existing GNN models for *LPG graph completion* tasks, i.e., finding missing labels or properties in the often noisy datasets.

3.3 LPG2vec + GNN: Towards A Neural Graph Database

Our architecture for neural graph databases can be seen as an encoder combined with a selected GNN model. An overview is provided in Figure 4.

In the first step to construct an embedding of an LPG, we apply one-hot encoding for labels and properties of each vertex and edge. For labels, the encoding is $\{0,1\}^{|L|}$, where "1" indicates that a given ith label is attached to a given vertex/edge. For properties, the encoding details depend on the property type: If a property can have *discretely many (C) values*, then we encode it using a plain one-hot vector with C entries. A *continuous scalar* property is normalized to $[0;1]$ or, alternatively, discretized and encoded as a one-hot vector. Importantly, one must use the same norm or discretization for all property instances for a given property type. A *numerical vector* is standardized and normalized. Finally, for properties that contain a *string of text*, we use Sentence Transformers, based on sentence-BERT [96], to embed such a property. String embeddings are usually much longer than other numerical properties to preserve most information in strings.

After encoding, labels and properties are concatenated into input feature vectors for each vertex and for each edge. Importantly, the concatenation is done after ordering the elements of a set Labels $\cup$ Property keys (i.e., $L \cup K$) and applying the same ordering for each vertex and for each edge. This ensures that the embeddings of labels/properties follow the same order in each feature vector and that the lengths of feature vectors for, respectively, vertices and edges, are the same.

Cmds

3.4 Seamless Integration with GNN Models and Encodings

Cmds

Both vertex and edge information is straightforwardly harnessed by LPG2vec by first encoding the input vertex or edge labels/properties within LPG2vec. Then, we feed such vertex and edge encodings, as input feature vectors $\mathbf{x}_i$ (for any vertex i) and $\mathbf{x}_{ij}$ (for any edge (i, j)), into a selected GNN model. Here, LPG2vec enables seamless integration with virtually any GNN model, encoding, or architecture. This is due to the simplicity of our solution: all LPG2vec does is providing "enriched" input feature vectors $\mathbf{x}_i$ and $\mathbf{x}_{ij}$. Vectors $\mathbf{x}_i$ can be directly fed to any convolutional, attentional, or message-passing GNN model, as the input vertex feature vectors. Vectors $\mathbf{x}_{ij}$ are fed into any model that also incorporates edge feature vectors.

Cmds

Moreover, LPG2vec also enables easy integration with encoding schemes such as MPGNNs-LSPE [43]. This can be achieved by, for example, concatenating the LPG2vec vectors with any additional encodings, and then using the resulting feature vectors with the selected GNN model.

Cmds

4 Evaluation

Our main goal in the evaluation is to show that LPG2vec successfully harnesses the label and property information from the LPG graph datasets to offer more accurate predictions in graph ML tasks. Our analysis comes with a large evaluation space. Thus, we show selected representative results; full data is in the appendix due to space constraints.

4.1 Experimental Setup

An important part of the experimental setup is finding the **appropriate graph datasets** that *have many labels and properties*. First, we use the Microsoft Academic Knowledge Graph (MAKG) [45]. The original graph is in the RDF format. We extracted data from RDF triples describing consecutive vertices, and we built LPG vertex entries containing the gathered data; single triples containing edges were parsed directly into the LPG format. Due to the huge size of MAKG, we extracted two subgraphs. For this, we consider the following LPG labels of vertices: *:Paper, :Author, :Affiliation, :ConferenceSeries, :ConferenceInstance, :FieldOfStudy*, as well as the links between them. Then, we additionally limit the number of the considered research areas (and thus vertices) in the *:FieldOfStudy* field (four for a small MAKG dataset, 25 for a large MAKG dataset); they form classes to be predicted. For diversified analysis, we make sure that these two datasets differ in their degree distributions, implying different connectivity structure. Second, we use example LPG graphs provided by Neo4j[2]; While these datasets are small, they are original excerpts from industry LPG databases. Most importantly, we use a "citations" network (modeling publications and citations between them), a "Twitter trolls" network (modeling anonymized Twitter trolls and the interaction of retweets), and a network modeling crime investigations. The details of datasets are in Table 1; the appendix provides a full specification of the associated labels/properties in selected datasets, as well as additional results.

n77k

While there are many heterogeneous graphs available online, they have usually single labels (often called types) per vertex or edge. We considered some of these graphs; we first convert them appropriately into the LPG model by transforming certain information from the graph structure into labels and properties. Note that we do *not* compete with heterogeneous representations, datasets, and the associated heterogeneous GNN models (they are outside the scope of this work); instead, we focus on LPG because this is the main established graph data model in graph databases.

Dataset	#vertices	#edges	#labels	#properties	size	Prediction target & ML task details
[MAKG] (small)	3.06M	12.3M	20	28	1.2 GB	Publication area (node classification, 4 classes)
[MAKG] (large)	50.7M	190M	20	28	19.5 GB	Publication area (node classification, 25 classes)
[Neo4j] citations	132k	221k	5	6	51 MB	Citation count (node regression)
[Neo4j] Twitter trolls	281k	493k	13	14	79 MB	Retweet count (node regression)
[Neo4j] crime investigations	61.5k	106k	28	29	17 MB	Crime type (node classification)

Table 1: Considered LPG datasets & ML tasks. **[Neo4j]**: provided by the Neo4j online repository, **[MAKG]**: extracted from MS Academic Knowledge Graph. Additional results for all the datasets are provided in the appendix.

We consider different established GNN models: GCN [67] (a seminal convolutional GNN model), GAT [111] (a seminal attentional GNN model), and GIN [124] (a seminal model having more expressive power than GCN or GAT). We test these models with and without the LPG2vec encoding scheme. Then, when considering models enhanced with LPG2vec, we test variants that harness the additional LPG information coming from only labels, only properties, and from both labels and properties. Our goal is to investigate how exactly the rich additional LPG information influences the

[2]Available at https://github.com/neo4j-graph-examples

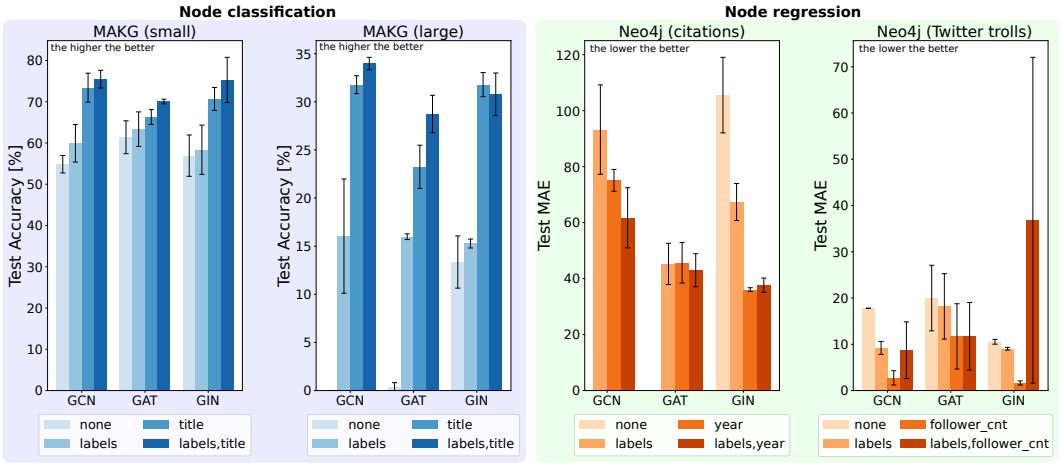

Figure 5: Advantages of preserving the information encoded in LPG labels and properties, for node classification (the MAKG datasets in the left panel; 4 classes for small and 25 classes for large) and node regression (the Neo4j datasets in the right panel).

accuracy of the established graph ML tasks, focusing on node classification (assigning each vertex to one of a given number of classes) and node regression (predicting a real value for each vertex) [122].

We split the datasets into train, val, and test by the ratio of $[0.8, 0.1, 0.1]$. We set the mini-batch size to 32, use the Adam optimizer [64], the learning rate of 0.01 augmented with the cosine annealing decay, and we train for 100 epochs. The node mini-batch sampling is conducted using the GraphSAINT established scheme [129]. We use the cross-entropy and MSE loss functions for classification and regression, respectively. In the design of used GNN models (GCN, GAT, GIN), following the established practice [127], we incorporate one preprocessing MLP layer, followed by two actual GNN layers, and then one additional post-processing MLP layer. We use the PReLU non-linearity.

n77k

Our implementation is integrated into PyG [46]. We use GraphGym [127] as well as Weights & Biases [24] for managing experiments.

4.2 Improving the Accuracy with LPG Labels and Properties

We first analyze how LPG2vec appropriately harnesses the rich information from LPG labels/properties, enabling accuracy improvements for different GNN models. Example results are in Figure 5, showing both node classification and node regression, with MAKG and Neo4j datasets. We plot the the final test accuracy (with the standard deviation) for classification and the mean absolute error (MAE) for node regression. The task is to predict the research area of the publication (for MAKG) and the citation count of a paper (for Neo4j citations). In the results, the baseline with no LPG labels/properties (i.e., only the neighborhood structure) consistently delivers the lowest accuracy (MAKG), or – in some cases such as for the GCN/GAT models and Neo4j citations dataset – is unable to converge. Then, for MAKG, including, respectively, labels (describing *paper types*), a property (*paper title*), and both the labels and the title property, steadily improves the accuracy, reaching nearly 35% for GCN. The trend is similar across all the studied models, and they achieve similarly high accuracy, which indicates that harnessing the appropriate labels/properties is very relevant and - when this information is present - different GNN models will perform similarly well. Neo4j citations and Twitter trolls are similar (note the different metric as this is a node regression task). The main difference is that, for GIN and the Twitter dataset, combining the labels and the *follower count* leads to worse results than only using one of these two individually. This illustrates that certain combinations of LPG information might not always enhance the accuracy; we study this in more detail in Sec. 4.3. Another interesting effect takes place when considering the bare graph structure on the small MAKG. Here, GCN performs worst, GAT is somewhat better, while GIN delivers much higher accuracy than GAT. We conjecture this is because GIN is provably highly expressive in the Weisfeiler-Lehman sense (when considering the bare graph structure) [124]. Overall, the results show the importance of including both the labels and properties when analyzing LPG graphs.

bJap

n77k

n77k

4.3 Selecting the Right Labels and Properties

In some experiments, we observed that selecting certain properties was not improving the accuracy. Moreover, in certain cases, the accuracy was actually diminishing. We analyze this effect in more detail in Figure 6 for the node classification and regression on MAKG small and Neo4j Twitter trolls, with the GIN model, plotting both train and test accuracy. The plots show the impact of using each of the many available properties on the final prediction accuracy. For example, on the small MAKG,

bJap

using the *title* property significantly improves the accuracy, and the majority of other properties also increase it, although by much smaller (often negligible) factor. Still, using the *publication date* property in many cases decreases the accuracy (see the bottom-left plot). We further analyze this effect with heatmaps, by considering each possible *pair of properties*, and how using this pair impacts the results. The accuracy is almost always enhanced, when using *title* together with nearly any other property. Some properties, such as *entity id*, have no effect. Many pair combinations result in slight accuracy improvements. However, in the Neo4j Twitter case (the bottom panel), in the test accuracy, while using many individual properties significantly enhances the accuracy, most combinations of property pairs decrease it. Interestingly, this only happens for the GIN model; *the GCN models and GAT models are able to extract useful knowledge from most property pairs* (these results are provided in the appendix, see Figures 12 and 13). This illustrates that it is important to understand the data and select the right encoded LPG information *and* the model for a given selected graph ML task.

5 Related Work and Discussion

Our work touches on many areas. We now briefly discuss related works. We do not compare LPG2vec to non-GNN baselines because our main goal is to illustrate how to integrate GNN capabilities into GDBs, and *not* to argue that neural methods outperform those of traditional non-GNN baselines. Hence, we do not focus on experiments with traditional GDB non-neural tasks such as BFS or Connected Components.

Cmds

Graph Neural Networks and Graph Machine Learning Graph neural networks (GNNs) emerged as a highly successful part of the graph machine learning field [53]. Numerous GNN models have been developed [18, 20, 30, 35, 50, 53, 102, 122, 132, 134], including convolutional [54, 67, 104, 119, 124], attentional [81, 107, 111], message-passing [14, 28, 51, 100, 118], or – more recently – higher-order ones [4, 5, 15, 26, 82, 98, 99]. Moreover, a large number of software frameworks [46, 57, 58, 70, 73, 110, 112–115, 117, 121, 131, 133, 135], and even hardware accelerators [49, 65, 66, 71, 125] for processing GNNs have been introduced over the last years. LPG2vec enables using all these designs together with the LPG graphs and consequently with LPG-based graph databases. This is because of the fact that the information within LPG labels and properties is encoded into the input features vectors, which can then be seamlessly used with essentially any GNN model or framework of choice.

Graph Databases (GDBs) [22] are systems used to manage, process, analyze, and store vast amounts of rich and complex graph datasets. GDBs have a long history of development and focus in both academia and in the industry, and there has been significant work on them [8, 9, 40, 48, 55, 61, 68]. A lot of research has been dedicated to graph query languages [7, 7, 27], GDB management [27, 60, 79, 90, 92], compression in GDBs and data models [16, 19, 21, 23, 72, 74, 83], execution in novel environments such as the serverless setting [37, 75, 109], and others. Many GDBs exist [1–3, 6, 10–13, 25, 31, 32, 34, 38, 41, 42, 44, 47, 62, 63, 76–78, 85–89, 94, 95, 97, 103, 105, 106, 108, 128, 136, 137]. We enhance the learning capabilities of graph databases by illustrating how to harness all the information encoded in Labeled Property Graph (LPG), a data model underlying the majority of graph databases, and use it for graph ML tasks such as node classification.

6 Conclusion

Graph databases (GDBs), despite being an important part of the graph analytics landscape, have still not embraced the full predictive capabilities of graph neural networks (GNNs). To address this, we first observe that the majority of graph databases use, or support, the Labeled Property Graph (LPG) as their data model. In LPG, the graph structure, stored explicitly in the compressed-sparse row format, is combined with labels and key-value properties that can be attached, in any configuration, to vertices and edges. To integrate GDBs with graph machine learning capabilities, we develop LPG2vec, an encoder that converts LPG labels and properties into input vertex and edge embeddings. This enables seamless integration of any GDB with any GNN model of interest.

Our evaluation shows that incorporating labels and properties into GNN models consistently improves accuracy. For example, GCN, GAT, and GIN models achieve even up to 34% better accuracy in node classification for the LPG representation of the Microsoft Academic Knowledge Graph, compared to a setting without LPG labels and properties. We conclude that LPG2vec will facilitate the development of neural graph databases, a learning architecture that harnesses both the structure and rich data (labels, properties) of LPG for highly accurate predictions in graph databases. It will lead to the wider adoption of GNNs in the broad graph database industry setting.

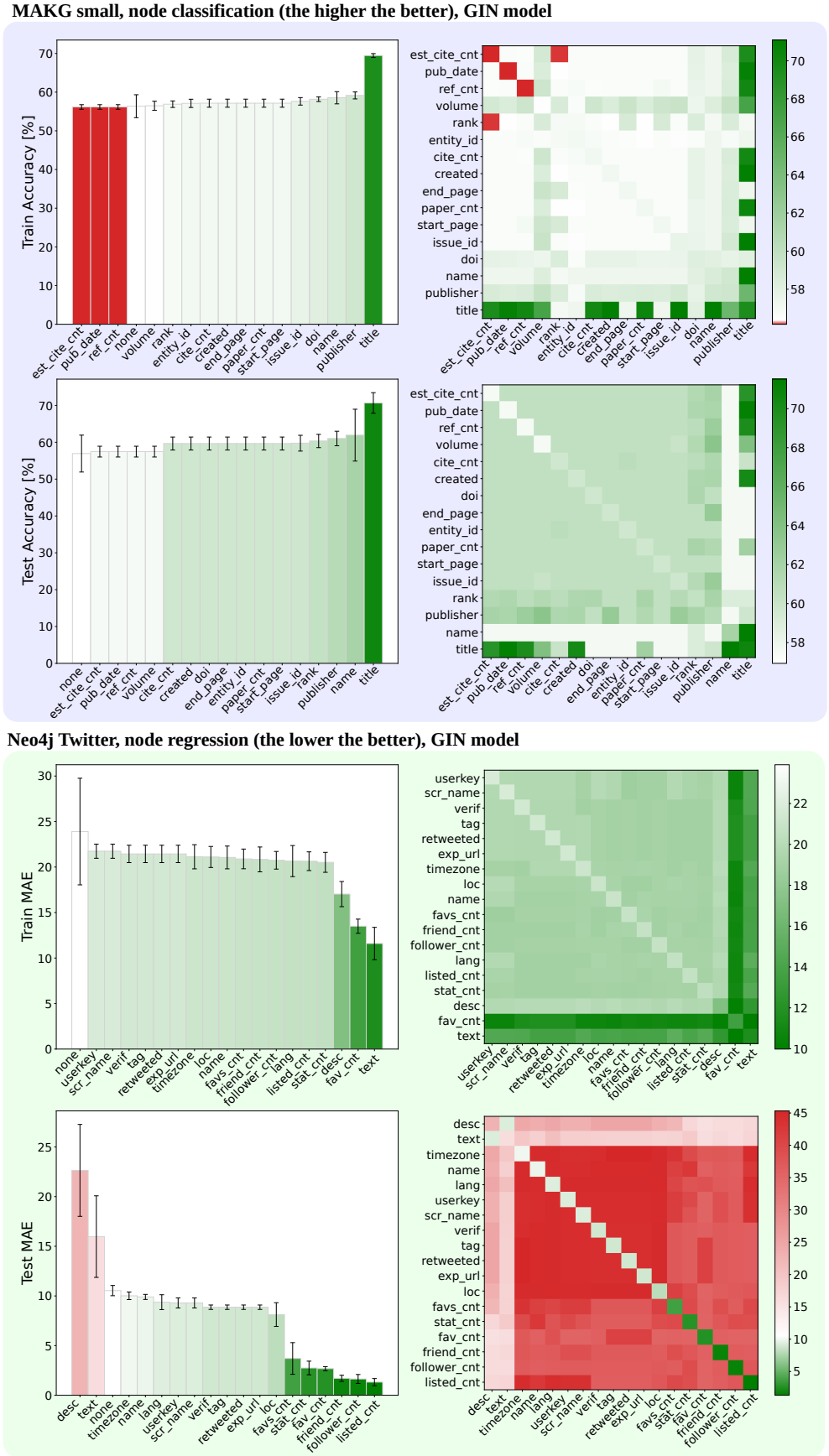

Figure 6: Analysis of the impact from different properties on the accuracy, considering both individual properties and the combinations of property pairs, for the MAKG and Neo4j, with the node classification and regression tasks, for the GIN model. The green color indicates that a given result is better than for a graph with no labels/properties. The red color indicates the results are worse than for a graph with no labels/properties.

References

[1] 2013. *Getting Started with OrientDB*. Packt Publishing Ltd. 1, 8

[2] 2018. Weaver. Available at http://weaver.systems/. 1

[3] 2022. *Using the Linked Data Benchmark Council Social Network Benchmark Methodology to Evaluate TigerGraph at 36 Terabytes*. White Paper. TigerGraph, Inc. 1, 8

[4] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. *arXiv:1905.00067 [cs, stat]* (June 2019). arXiv:cs, stat/1905.00067 8

[5] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*. PMLR, 21–29. 8

[6] Amazon. 2018. Amazon Neptune. Available at https://aws.amazon.com/neptune/. 1, 8

[7] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. 2017. Foundations of Modern Query Languages for Graph Databases. *in ACM Comput. Surv.* 50, 5, Article 68 (2017), 40 pages. https://doi.org/10.1145/3104031 2, 8

[8] Renzo Angles and Claudio Gutierrez. 2008. Survey of Graph Database Models. *in ACM Comput. Surv.* 40, 1, Article 1 (2008), 39 pages. https://doi.org/10.1145/1322432.1322433 8

[9] Renzo Angles and Claudio Gutierrez. 2018. An Introduction to Graph Data Management. In *Graph Data Management, Fundamental Issues and Recent Developments*. 1–32. 8

[10] Apache. 2018. Apache Mormotta. Available at http://marmotta.apache.org/. 8

[11] ArangoDB Inc. 2018. ArangoDB. Available at https://docs.arangodb.com/3.3/Manual/DataModeling/Concepts.html. 1

[12] ArangoDB Inc. 2018. ArangoDB: Index Free Adjacency or Hybrid Indexes for Graph Databases. Available at https://www.arangodb.com/2016/04/index-free-adjacency-hybrid-indexes-graph-databases/.

[13] ArangoDB Inc. 2018. ArangoDB Starter Tool. Available at https://docs.arangodb.com/devel/Manual/Tutorials/Starter/. 1, 8

[14] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018). 4, 8

[15] Austin R Benson et al. 2018. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences* 115, 48 (2018), E11221–E11230. 8

[16] Maciej Besta et al. 2019. Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics. , Article 35 (2019), 25 pages. https://doi.org/10.1145/3295500.3356182 8

[17] Maciej Besta et al. 2022. Practice of Streaming Processing of Dynamic Graphs: Concepts, Models, and Systems. *IEEE TPDS* (2022). 1

[18] Maciej Besta, Raphael Grob, Cesare Miglioli, Nicola Bernold, Grzegorz Kwasniewski, Gabriel Gjini, Raghavendra Kanakagiri, Saleh Ashkboos, Lukas Gianinazzi, Nikoli Dryden, et al. 2022. Motif Prediction with Graph Neural Networks, In ACM KDD. *arXiv preprint arXiv:2106.00761*. 2, 8

[19] Maciej Besta and Torsten Hoefler. 2018. Survey and taxonomy of lossless graph compression and space-efficient graph representations. *arXiv preprint arXiv:1806.01799* (2018). 8

[20] Maciej Besta and Torsten Hoefler. 2022. Parallel and Distributed Graph Neural Networks: An In-Depth Concurrency Analysis. *arXiv preprint arXiv:2205.09702* (2022). 2, 3, 8

[21] Maciej Besta, Cesare Miglioli, Paolo Sylos Labini, Jakub Tětek, Patrick Iff, Raghavendra Kanakagiri, Saleh Ashkboos, Kacper Janda, Michal Podstawski, Grzegorz Kwasniewski, et al. 2022. ProbGraph: High-Performance and High-Accuracy Graph Mining with Probabilistic Set Representations. *arXiv preprint arXiv:2208.11469* (2022). 8

[22] Maciej Besta, Emanuel Peter, Robert Gerstenberger, Marc Fischer, Michał Podstawski, Claude Barthels, Gustavo Alonso, and Torsten Hoefler. 2019. Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries. *arXiv preprint arXiv:1910.09017* (2019). 1, 2, 4, 8

[23] Maciej Besta, Dimitri Stanojevic, Tijana Zivic, Jagpreet Singh, Maurice Hoerold, and Torsten Hoefler. 2018. Log (graph): a near-optimal high-performance graph representation.. In *PACT* (Limassol, Cyprus). ACM, Article 7, 13 pages. https://doi.org/10.1145/3243176.3243198 8

[24] Lukas Biewald. 2020. Experiment Tracking with Weights and Biases. https://www.wandb.com/ Software available from wandb.com. 7

[25] Blazegraph. 2018. BlazeGraph DB. Available at https://www.blazegraph.com/. 1, 8

[26] Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F Montufar, Pietro Lio, and Michael Bronstein. 2021. Weisfeiler and lehman go topological: Message passing simplicial networks. In *International Conference on Machine Learning*. PMLR, 1026–1037. 8

[27] Angela Bonifati, George Fletcher, Hannes Voigt, and Nikolay Yakovets. 2018. Querying graphs. *Synthesis Lectures on Data Management* 10, 3 (2018), 1–184. 8

[28] Xavier Bresson and Thomas Laurent. 2017. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553* (2017). 4, 8

[29] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. 2021. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478* (2021). 4

[30] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42. 2, 8

[31] Callidus Software Inc. 2018. OrientDB: Lightweight Edges. Available at https://orientdb.com/docs/3.0.x/java/Lightweight-Edges.html. 1, 8

[32] Cambridge Semantics. 2018. AnzoGraph. Available at https://www.cambridgesemantics.com/product/anzograph/. 1, 8

[33] Wenming Cao, Zhiyue Yan, Zhiquan He, and Zhihai He. 2020. A comprehensive survey on geometric deep learning. *IEEE Access* 8 (2020), 35929–35949. 3

[34] Cayley. 2018. CayleyGraph. Available at https://cayley.io/ and https://github.com/cayleygraph/cayley. 1, 8

[35] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. 2020. Machine learning on graphs: A model and comprehensive taxonomy. *arXiv preprint arXiv:2005.03675* (2020). 2, 8

[36] Zhiqian Chen et al. 2020. Bridging the gap between spatial and spectral domains: A survey on graph neural networks. *arXiv preprint arXiv:2002.11867* (2020). 3

[37] Marcin Copik, Grzegorz Kwasniewski, Maciej Besta, Michal Podstawski, and Torsten Hoefler. 2020. SeBS: A Serverless Benchmark Suite for Function-as-a-Service Computing. *arXiv preprint arXiv:2012.14132* (2020). 8

[38] DataStax, Inc. 2018. DSE Graph (DataStax). Available at https://www.datastax.com/. 1, 8

[39] Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, et al. 2021. Advancing mathematics by guiding human intuition with AI. *Nature* 600, 7887 (2021), 70–74. 2

[40] Ali Davoudian, Liu Chen, and Mengchi Liu. 2018. A survey on NoSQL stores. *ACM Computing Surveys (CSUR)* 51, 2, Article 40 (2018), 43 pages. https://doi.org/10.1145/3158661 8

[41] Dgraph Labs, Inc. 2018. DGraph. Available at https://dgraph.io/, https://docs.dgraph.io/design-concepts. 1, 8

[42] Ayush Dubey, Greg D Hill, Robert Escriva, and Emin Gün Sirer. 2016. Weaver: a high-performance, transactional graph database based on refinable timestamps. *Proceedings of the VLDB Endowment* 9, 11 (2016), 852–863. 8

[43] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2021. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875* (2021). 6

[44] FactNexus. 2018. GraphBase. Available at https://graphbase.ai/. 8

[45] Michael Färber. 2019. The Microsoft Academic Knowledge Graph: A Linked Data Source with 8 Billion Triples of Scholarly Data. In *Proceedings of the 18th International Semantic Web Conference* (Auckland, New Zealand) *(ISWC'19)*. 113–129. https://doi.org/10.1007/978-3-030-30796-7_8 6

[46] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019). 7, 8

[47] Franz Inc. 2018. AllegroGraph. Available at https://franz.com/agraph/allegrograph/. 1, 8

[48] Santhosh Kumar Gajendran. 2012. A survey on NoSQL databases. *University of Illinois* (2012). 8

[49] Tong Geng, Ang Li, Runbin Shi, Chunshu Wu, Tianqi Wang, Yanfei Li, Pouya Haghi, Antonino Tumeo, Shuai Che, Steve Reinhardt, et al. 2020. AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing. In *IEEE/ACM MICRO*. 8

[50] Lukas Gianinazzi, Maximilian Fries, Nikoli Dryden, Tal Ben-Nun, and Torsten Hoefler. 2021. Learning Combinatorial Node Labeling Algorithms. *arXiv preprint arXiv:2106.03594* (2021). 2, 8

[51] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*. PMLR, 1263–1272. 4, 8

[52] William L Hamilton. 2020. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning* 14, 3 (2020), 1–159. 2

[53] William L Hamilton et al. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017). 2, 8

[54] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 4, 8

[55] Jing Han, E Haihong, Guan Le, and Jian Du. 2011. Survey on NoSQL database. In *2011 6th international conference on pervasive computing and applications*. IEEE, 363–366. 8

[56] Amy E Hodler and Mark Needham. 2022. Graph Data Science Using Neo4j. In *Massive Graph Analytics*. Chapman and Hall/CRC, 433–457. 2

[57] Yuwei Hu et al. 2020. Featgraph: A flexible and efficient backend for graph neural network systems. *arXiv preprint arXiv:2008.11359* (2020). 8

[58] Zhihao Jia et al. 2020. Improving the accuracy, scalability, and performance of graph neural networks with roc. *MLSys* (2020). 8

[59] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596, 7873 (2021), 583–589. 2

[60] Martin Junghanns, André Petermann, Martin Neumann, and Erhard Rahm. 2017. Management and analysis of big graph data: current systems and open challenges. In *Handbook of Big Data Technologies*. Springer, 457–505. 8

[61] R. Kumar Kaliyar. 2015. Graph databases: A survey. In *ICCCA*. 785–790. 8

[62] U. Kang, Hanghang Tong, Jimeng Sun, Ching-Yung Lin, and Christos Faloutsos. 2012. Gbase: An Efficient Analysis Platform for Large Graphs. *In PVLDB* 21, 5 (2012), 637–650. https://doi.org/10.1007/s00778-012-0283-9 8

[63] Chathura Kankanamge, Siddhartha Sahu, Amine Mhedbhi, Jeremy Chen, et al. 2017. Graphflow: An Active Graph Database. In *ACM SIGMOD* (Chicago, Illinois, USA). 1695–1698. https://doi.org/10.1145/3035918.3056445 8

[64] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014). 7

[65] Kevin Kiningham, Philip Levis, and Christopher Ré. 2020. GReTA: Hardware Optimized Graph Processing for GNNs. In *ReCoML*. 8

[66] Kevin Kiningham, Christopher Re, and Philip Levis. 2020. GRIP: a graph neural network accelerator architecture. *arXiv preprint arXiv:2007.13828* (2020). 8

[67] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016). 2, 3, 4, 6, 8

[68] Vijay Kumar and Anjan Babu. 2015. Domain Suitable Graph Database Selection: A Preliminary Report. In *3rd International Conference on Advances in Engineering Sciences & Applied Mathematics, London, UK*. 26–29. 8

[69] Ora Lassila, Ralph R Swick, et al. 1998. Resource description framework (RDF) model and syntax specification. (1998). 1

[70] Shen Li et al. 2020. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704* (2020). 8

[71] Shengwen Liang, Ying Wang, Cheng Liu, Lei He, LI Huawei, Dawen Xu, and Xiaowei Li. 2020. Engn: A high-throughput and energy-efficient accelerator for large graph neural networks. *IEEE TOC* (2020). 8

[72] Bingqing Lyu, Lu Qin, Xuemin Lin, Lijun Chang, and Jeffrey Xu Yu. 2016. Scalable supergraph search in large graph databases. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 157–168. 8

[73] Lingxiao Ma, Zhi Yang, Youshan Miao, Jilong Xue, Ming Wu, Lidong Zhou, and Yafei Dai. 2019. Neugraph: parallel deep neural network computation on large graphs. In *USENIX ATC*. 8

[74] Shuai Ma, Jia Li, Chunming Hu, Xuelian Lin, and Jinpeng Huai. 2016. Big graph search: challenges and techniques. *Frontiers of Computer Science* 10, 3 (2016), 387–398. 8

[75] Zhitao Mao, Ruoyu Wang, Haoran Li, Yixin Huang, Qiang Zhang, Xiaoping Liao, and Hongwu Ma. 2022. ERMer: a serverless platform for navigating, analyzing, and visualizing Escherichia coli regulatory landscape through graph database. *Nucleic Acids Research* (2022). 8

[76] Norbert Martínez-Bazan, Victor Muntés-Mulero, Sergio Gómez-Villamor, M.Ángel Águila Lorente, David Dominguez-Sal, and Josep-L. Larriba-Pey. 2012. Efficient Graph Management Based On Bitmap Indices. *In IDEAS* (2012), 110–119. https://doi.org/10.1145/2351476.2351489 8

[77] Memgraph Ltd. 2018. Memgraph. Available at https://memgraph.com/.

[78] Microsoft. 2018. Azure Cosmos DB. Available at https://azure.microsoft.com/en-us/services/cosmos-db/. 1, 8

[79] Justin J Miller. 2013. Graph Database Applications and Concepts with Neo4j. In *Proceedings of the Southern Association for Information Systems Conference*, Vol. 2324. 1, 8

[80] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. 2021. A graph placement methodology for fast chip design. *Nature* 594, 7862 (2021), 207–212. 2

[81] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In *IEEE CVPR*. 4, 8

[82] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 4602–4609. 8

[83] Chemseddine Nabti and Hamida Seba. 2017. Querying massive graph data: A compress and search approach. *Future Generation Computer Systems* 74 (2017), 63–75. 8

[84] Neo4j, Inc. 2018. Neo4j (3.0 Release).
Available at https://neo4j.com/blog/neo4j-3-0-massive-scale-developer-productivity/. 4

[85] Networked Planet Limited. 2018. BrightstarDB. Available at http://brightstardb.com/. 8

[86] Objectivity Inc. 2018. InfiniteGraph. Available at https://www.objectivity.com/products/infinitegraph/. 1

[87] Ontotext. 2018. GraphDB. Available at https://www.ontotext.com/products/graphdb/. 1

[88] OpenLink. 2018. Virtuoso. Available at https://virtuoso.openlinksw.com/. 1

[89] Oracle. 2018. Oracle Spatial and Graph. Available at https://www.oracle.com/database/technologies/spatialandgraph.html. 1, 8

[90] N.S. Patil, P Kiran, N.P. Kavya, and K.M. Naresh Patel. 2018. A Survey on Graph Database Management Techniques for Huge Unstructured Data. *International Journal of Electrical and Computer Engineering* 81, 2 (2018), 1140–1149. 8

[91] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. 2020. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409* (2020). 2

[92] Jaroslav Pokorny. 2015. Graph databases: their power and limitations. In *IFIP International Conference on Computer Information Systems and Industrial Management*. Springer, 58–69. 8

[93] Irene Polikoff. 2018. Knowledge Graphs vs. Property Graphs - Part I.
Available at https://tdan.com/knowledge-graphs-vs-property-graphs-part-1/27140. 4

[94] Profium. 2018. Profium Sense. Available at https://www.profium.com/en/. 8

[95] Redis Labs. 2018. RedisGraph. Available at https://oss.redislabs.com/redisgraph/. 1, 8

[96] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019). 5

[97] Christopher D. Rickett, Utz-Uwe Haus, James Maltby, and Kristyn J. Maschhoff. 2018. Loading and Querying a Trillion RDF triples with Cray Graph Engine on the Cray XC. In *CUG*. Cray Users Group. 8

[98] Ryan A. Rossi, Nesreen K. Ahmed, and Eunyee Koh. 2018. Higher-Order Network Representation Learning. In *Companion Proceedings of the The Web Conference 2018 (WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 3–4. https://doi.org/10.1145/3184558.3186900 8

[99] Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, Sungchul Kim, Anup Rao, and Yasin Abbasi Yadkori. 2018. HONE: Higher-Order Network Embeddings. *arXiv:1801.09303 [cs, stat]* (May 2018). arXiv:cs, stat/1801.09303 8

[100] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. 2020. Learning to simulate complex physics with graph networks. In *ICML*. 2, 8

[101] Ryoma Sato. 2020. A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078* (2020). 3

[102] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks* 20, 1 (2008), 61–80. 2, 3, 8

[103] Stardog Union. 2018. Stardog. Available at https://www.stardog.com/. 1, 8

[104] Sainbayar Sukhbaatar, Rob Fergus, et al. 2016. Learning multiagent communication with backpropagation. *NeurIPS* (2016). 4, 8

[105] The Apache Software Foundation. 2021. Apache Jena TBD. Available at https://jena.apache.org/documentation/tdb/index.html. 8

[106] The Linux Foundation. 2018. JanusGraph. Available at http://janusgraph.org/. 1, 8

[107] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. 2018. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735* (2018). 3, 4, 8

[108] TigerGraph. 2018. TigerGraph. Available at https://www.tigergraph.com/. 1, 8

[109] Lucian Toader, Alexandru Uta, Ahmed Musaafir, and Alexandru Iosup. 2019. Graphless: Toward serverless graph processing. In *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE, 66–73. 8

[110] Alok Tripathy, Katherine Yelick, and Aydın Buluç. 2020. Reducing communication in graph neural network training. In *ACM/IEEE Supercomputing*. 8

[111] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017). 2, 4, 6, 8

[112] Roger Waleffe, Jason Mohoney, Theodoros Rekatsinas, and Shivaram Venkataraman. 2022. Marius++: Large-Scale Training of Graph Neural Networks on a Single Machine. *arXiv preprint arXiv:2202.02365* (2022). 8

[113] Cheng Wan, Youjie Li, Ang Li, Nam Sung Kim, and Yingyan Lin. 2022. BNS-GCN: Efficient Full-Graph Training of Graph Convolutional Networks with Partition-Parallelism and Random Boundary Node Sampling Sampling. *MLSys* (2022).

[114] Cheng Wan, Youjie Li, Cameron R Wolfe, Anastasios Kyrillidis, Nam Sung Kim, and Yingyan Lin. 2022. PipeGCN: Efficient full-graph training of graph convolutional networks with pipelined feature communication. *arXiv preprint arXiv:2203.10428* (2022).

[115] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. 2019. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv:1909.01315* (2019). 8

[116] Xiao Wang, Deyu Bo, Chuan Shi, Shaohua Fan, Yanfang Ye, and Philip S Yu. 2020. A survey on heterogeneous graph embedding: methods, techniques, applications and sources. *arXiv:2011.14867* (2020). 1

[117] Yuke Wang, Boyuan Feng, Gushu Li, Shuangchen Li, Lei Deng, Yuan Xie, and Yufei Ding. 2020. GNNAdvisor: An Efficient Runtime System for GNN Acceleration on GPUs. *arXiv preprint arXiv:2006.06608* (2020). 8

[118] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. 2019. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)* 38, 5 (2019), 1–12. 4, 8

[119] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871. 8

[120] Shiwen Wu, Fei Sun, Wentao Zhang, and Bin Cui. 2020. Graph neural networks in recommender systems: a survey. *arXiv preprint arXiv:2011.02260* (2020). 3

[121] Yidi Wu, Kaihao Ma, Zhenkun Cai, Tatiana Jin, Boyang Li, Chenguang Zheng, James Cheng, and Fan Yu. 2021. Seastar: vertex-centric programming for graph neural networks. In *EuroSys*. 8

[122] Zonghan Wu et al. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020). 2, 3, 7, 8

[123] Yu Xie, Bin Yu, Shengze Lv, Chen Zhang, Guodong Wang, and Maoguo Gong. 2021. A survey on heterogeneous network representation learning. *Pattern Recognition* 116 (2021), 107936. 1, 4

[124] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018). 2, 4, 6, 7, 8

[125] Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. 2020. Hygcn: A gcn accelerator with hybrid architecture. In *IEEE HPCA*. IEEE, 15–29. 8

[126] Carl Yang, Yuxin Xiao, Yu Zhang, Yizhou Sun, and Jiawei Han. 2020. Heterogeneous network representation learning: A unified framework with survey and benchmark. *IEEE TKDE* (2020). 1, 4

[127] Jiaxuan You, Zhitao Ying, and Jure Leskovec. 2020. Design space for graph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 17009–17021. 7

[128] Pingpeng Yuan, Pu Liu, Buwen Wu, Hai Jin, Wenya Zhang, and Ling Liu. 2013. TripleBit: a fast and compact system for large scale RDF data. *Proceedings of the VLDB Endowment* 6, 7 (2013), 517–528. https://doi.org/10.14778/2536349.2536352 8

[129] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931* (2019). 7

[130] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *KDD*. 793–803. 3

[131] Dalong Zhang et al. 2020. Agl: a scalable system for industrial-purpose graph machine learning. *arXiv preprint arXiv:2003.02454* (2020). 8

[132] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2020. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering* (2020). 2, 3, 8

[133] Da Zheng, Xiang Song, Chengru Yang, Dominique LaSalle, Qidong Su, Minjie Wang, Chao Ma, and George Karypis. 2021. Distributed Hybrid CPU and GPU training for Graph Neural Networks on Billion-Scale Graphs. *arXiv:2112.15345* (2021). 8

[134] Jie Zhou et al. 2020. Graph neural networks: A review of methods and applications. *AI Open* 1 (2020), 57–81. 2, 3, 8

[135] Rong Zhu et al. 2019. Aligraph: A comprehensive graph neural network platform. *arXiv preprint arXiv:1902.08730* (2019). 8

[136] Xiaowei Zhu et al. 2020. LiveGraph: A Transactional Graph Storage System with Purely Sequential Adjacency List Scans. *VLDB* 13, 7 (2020), 1020–1034. https://doi.org/10.14778/3384345.3384351 8

[137] Lei Zou, M. Tamer Özsu, Lei Chen, Xuchuan Shen, Ruizhe Huang, and Dongyan Zhao. 2014. GStore: A Graph-Based SPARQL Query Engine. *VLDB Journal* 23, 4 (2014), 565–590. https://doi.org/10.1007/s00778-013-0337-7 8

Appendix

A Dataset Specification

We present the details about the used datasets.

A.1 MAKG

The dataset *MAKG* (small) consists of $3'066'782$ vertices and $12'314'398$ edges. Each vertex is labeled with either *author* (55%), *paper* (44%), *affiliation* ($< 1\%$), *conferenceseries* ($< 1\%$), *conferenceinstance* ($< 1\%$), *fieldofstudy* ($< 1\%$), or *journal* ($< 1\%$). Vertices with the label *paper* are further subdivided into *book*, *bookchapter*, *conferencepaper*, *journalpaper*, *patentdocument* or *others*. Vertices labeled with *affiliation*, *author*, *conferenceseries*, *conferenceinstance*, *fieldofstudy* and *journal* do all have the properties *rank*, *name*, *papercount*, *citationcount*, and *created*. Some of them have additional properties, e.g., *homepage*. All vertices with the label *paper* have the properties *rank*, *citationcount*, *created*, *title*, *publicationdate*, *referencecount*, and *estimatedcitationcount*. Some of them have the additional properties *publisher*, *volume*, *issueidentifier*, *startingpage*, *endingpage*, or *doi*. Edges do not have properties but each edge has a label which is either *cites* (40%), *creator* (36%), *hasdiscipline* (11%), *apreasinjournal* (6%), *memberof* (5.7%), *appearsinconferenceinstance* ($< 1\%$), *appearsinconferenceseries* ($< 1\%$), or *ispartof* ($< 1\%$).

A.2 Citations

The *citations* dataset contains $132'259$ vertices and $221'237$ edges; we use it mostly for debugging purposes. Each vertex has one label which is either *author* (61%), *article* (39%), or *venue* ($< 1\%$). All *article*-vertices have the properties *index* (a 32-digit HEX number), *title* and *year* (the year in which the article was published). 85% of the articles have the property *abstract* and 72% of them have the property *ncitations* (the article's citation count). Vertices labeled with *author* or *venue* have only one property called *name*. Edges do not have properties but each edge has a label which is either *author* (64%), *venue* (23%), or *cited* (13%).

A.3 Twitter

The dataset *TwitterTrolls* contains $281'136$ vertices and $493'160$ edges. Vertices are labeled with *tweet* (82%), *url* (8%), *hashtag* (5%), *user* (5%), *trolluser* ($< 1\%$), or *source* ($< 1\%$). Vertices wit labels *hashtag*, *source*, *user*, and *url* have a single property each, namely *tag*, *name*, *userkey*, and *expandedurl* respectively. Vertices labeled with *trolluser* do all have the properties *sourcename* and *userkey*. Most of them ($> 80\%$) have additional properties *lang* (language), *verified* (true or false), *name*, *description*, *location*, *timezone*, *createdat*, *favoritescount*, *followerscount*, *friendscount*, *listedcount*, and *statusescount*. Most vertices labeled with *tweet* ($> 80\%$) have properties *createdat*, *createdstr* and *text*. About 25% of them have additional properties *favoritecount*, *retweetcount*, and *retweeted*. Edge do not have properties but each edge has a label which can be *posted* (41%), *hastag* (22%), *postedvia* (12%), *mentions* (11%), *retweeted* (8%), *haslink* (6%), or *inreplyto* ($< 1\%$).

A.4 Differences to Traditional GNN Datasets

Cmds

The main difference between LPG graphs and traditional GNN datasets such as Citeseer or Cora is that the latter usually do not have extensive sets of labels. Instead, these datasets often have vertices from different classes, which may be interpreted as a single label (that would encode such different classes). Moreover, these datasets often do not have rich sets of attached *different* properties. Instead, they may come with extensive feature vectors that encode a single large additional piece of information, for example a whole abstract. Finally, in the graph database setting, it is less common to process graphs such as PROTEINS, where the dataset consists of a very large number of relatively small graph. Instead, it is more common to focus on one large graph dataset.

B Results for Additional Labels, Properties, and Datasets

Figures 7–18 illustrate the impact of using each of the many available properties, and pairs of properties, on the final prediction accuracy. We show results separately for each GNN model and also aggregated for all thee models, for the completeness of the analysis. To facilitate comparing the data, we also replot the results for the GIN model for MAKG small and Neo4j Twitter analyses from Section 4. Finally, Figure 19 shows results for an additional Neo4j dataset modeling crime investigations.

x3Ya

Interestingly, the largest accuracy increase for MAKG is consistently obtained when including the title property. This is the case for all the considered GNN models. Similarly, when detecting trolls, including the counts of friends or followers was crucial in consistent accuracy improvements. This indicates that it is more important to appropriately understand the data and include the right information in the input feature vectors, and once this is achieved, different GNN models would be similarly able to extract this information for more accurate outcomes.

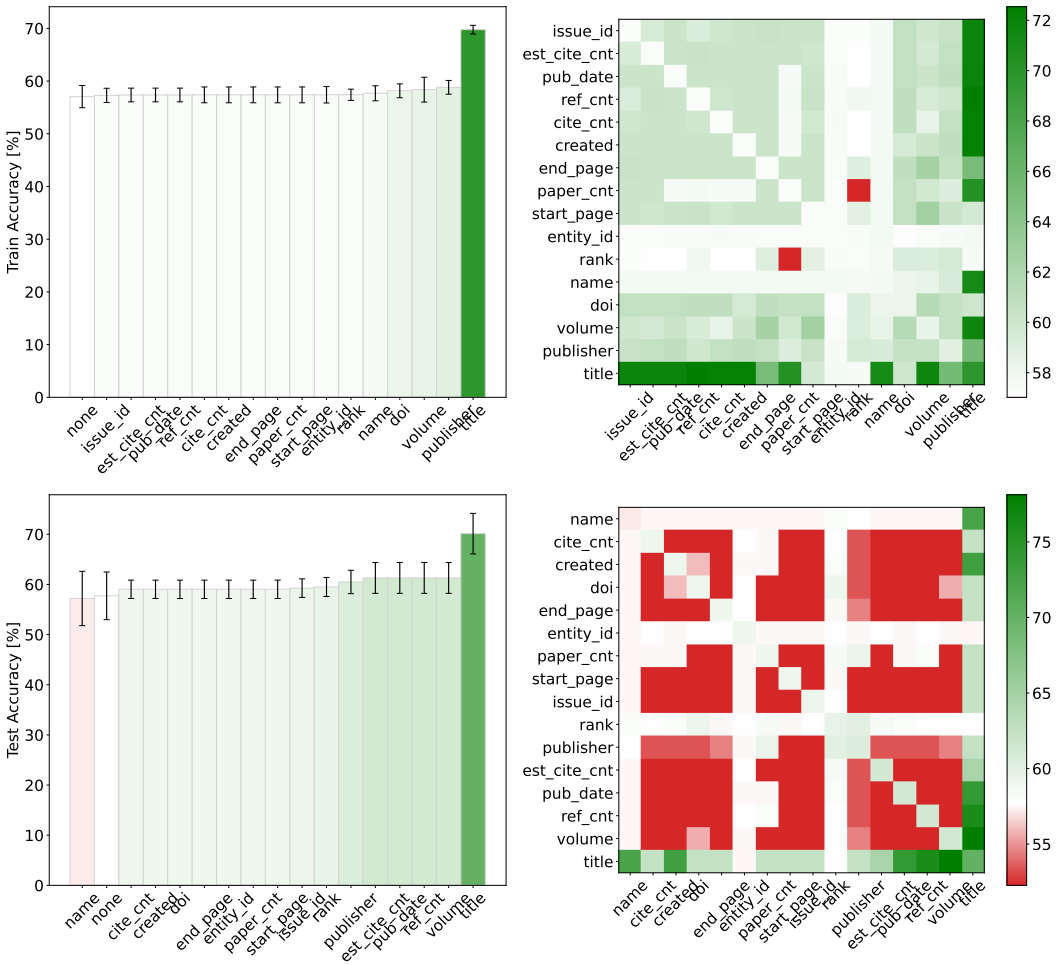

Figure 7: MAKG small (node classification, 4 classes, results aggregated over all three models). Impact from different properties and their combinations on the accuracy. Green: accuracy is better than that of a graph with no labels/properties; red: the accuracy is worse than that of a graph with no labels/properties.

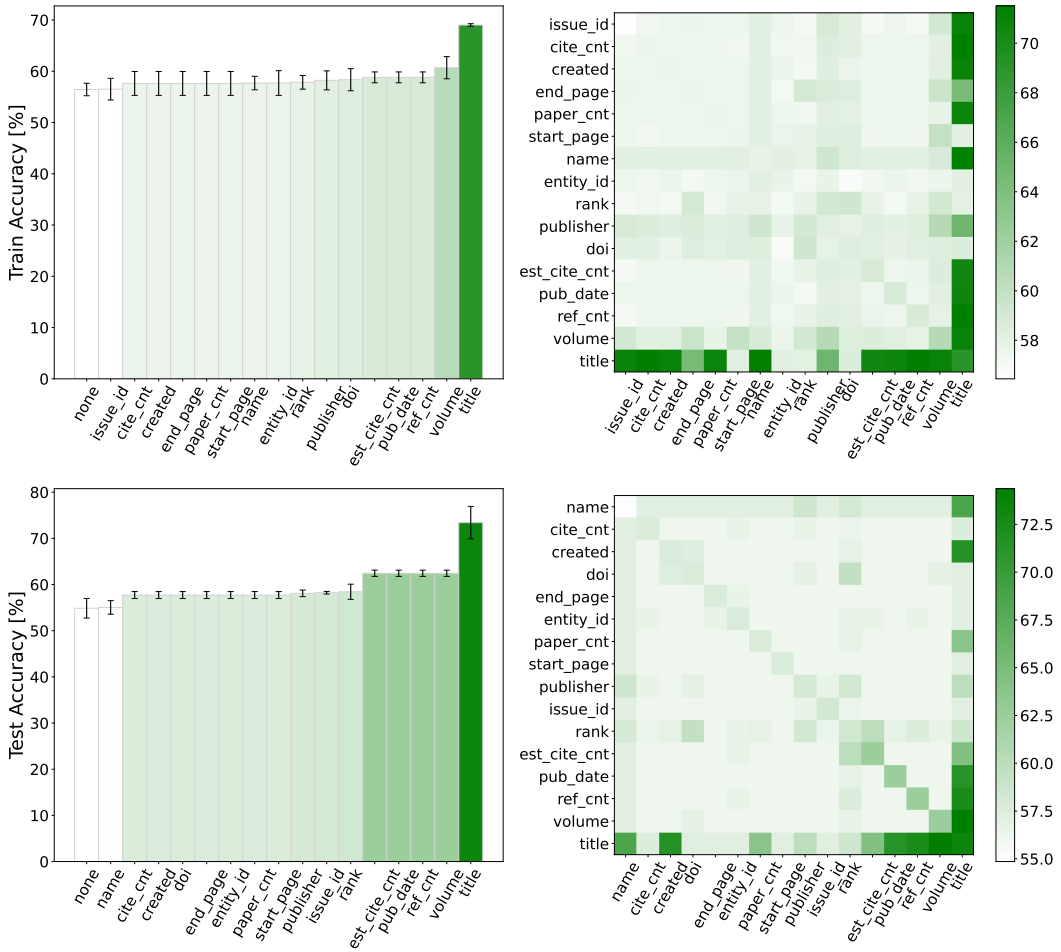

Figure 8: MAKG small (node classification, 4 classes, GCN-only results). Impact from different properties and their combinations on the accuracy. Green: the accuracy is better than that of a graph with no labels/properties; red: the accuracy is worse than that of a graph with no labels/properties.

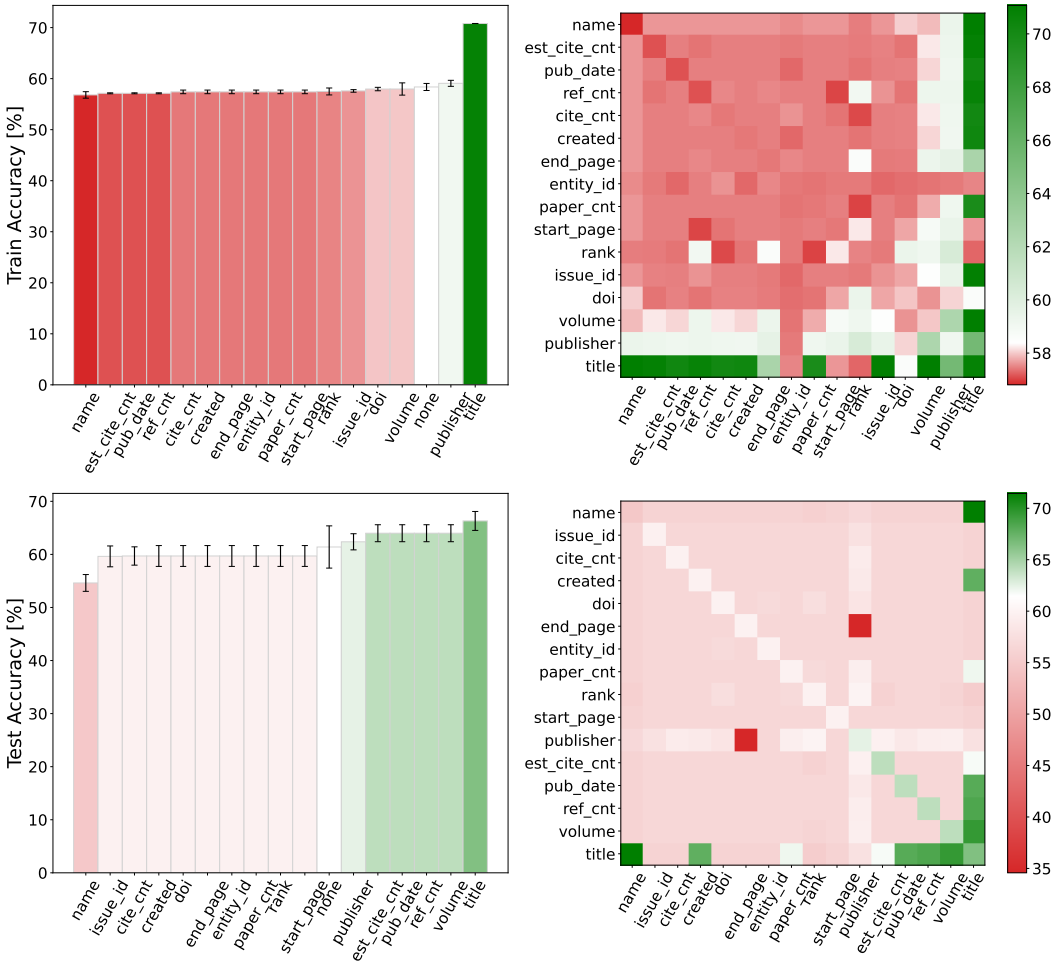

Figure 9: MAKG small (node classification, 4 classes, GAT-only results). Impact from different properties and their combinations on the accuracy. Green: accuracy is better than that of a graph with no labels/properties; red: the accuracy is worse than that of a graph with no labels/properties.

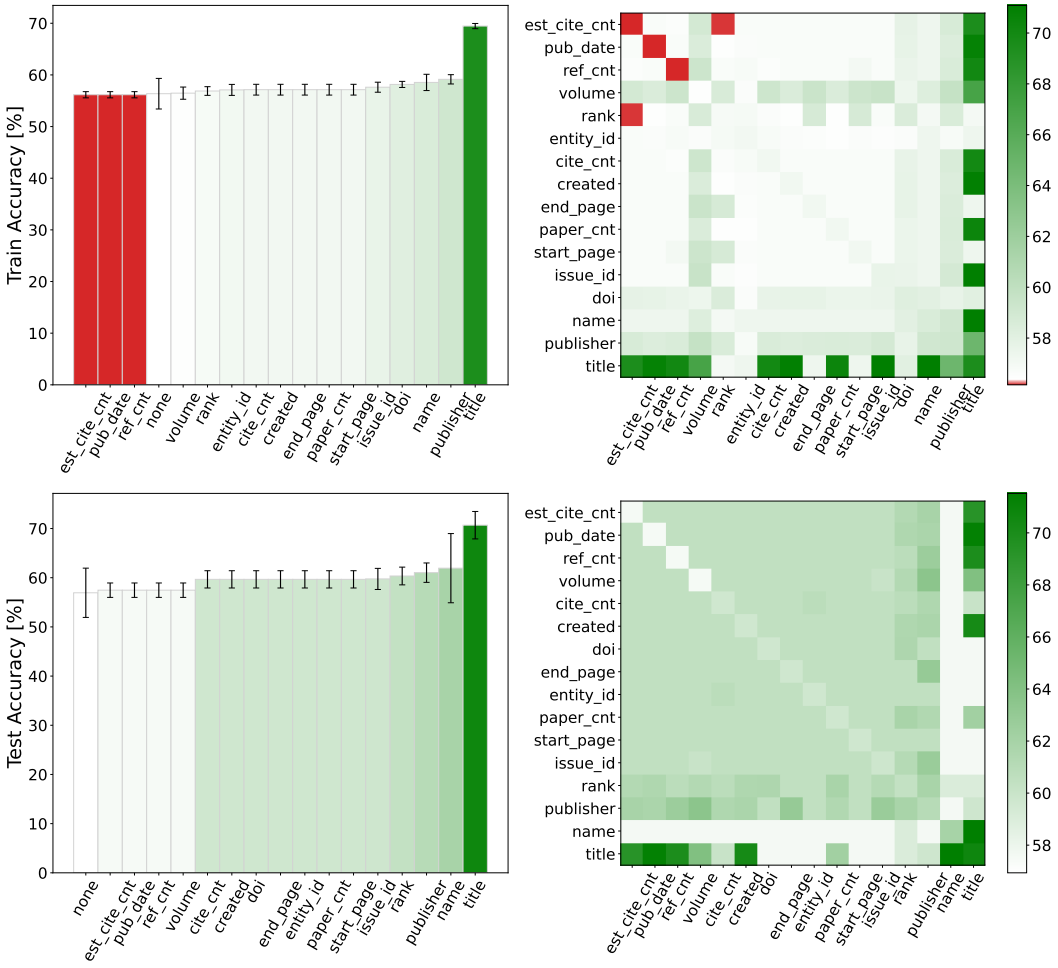

Figure 10: MAKG small (node classification, 4 classes, GIN-only results). Impact from different properties and their combinations on the accuracy. Green: accuracy is better than that of a graph with no labels/properties; red: the accuracy is worse than that of a graph with no labels/properties.

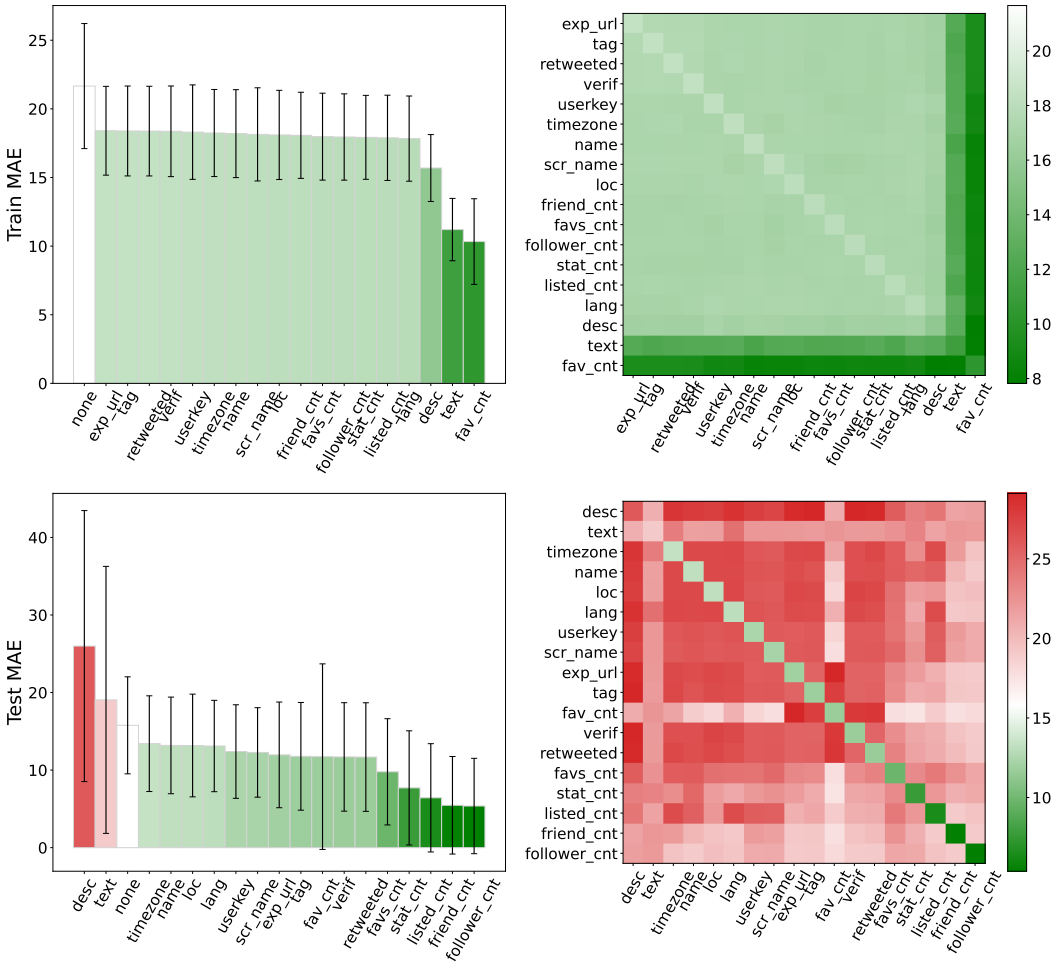

Figure 11: Neo4j Twitter trolls (node regression, results aggregated over all three models). Impact from different properties and their combinations on the MAE. Green: MAE is better than that of a graph with no labels/properties; red: the MAE is worse than that of a graph with no labels/properties.

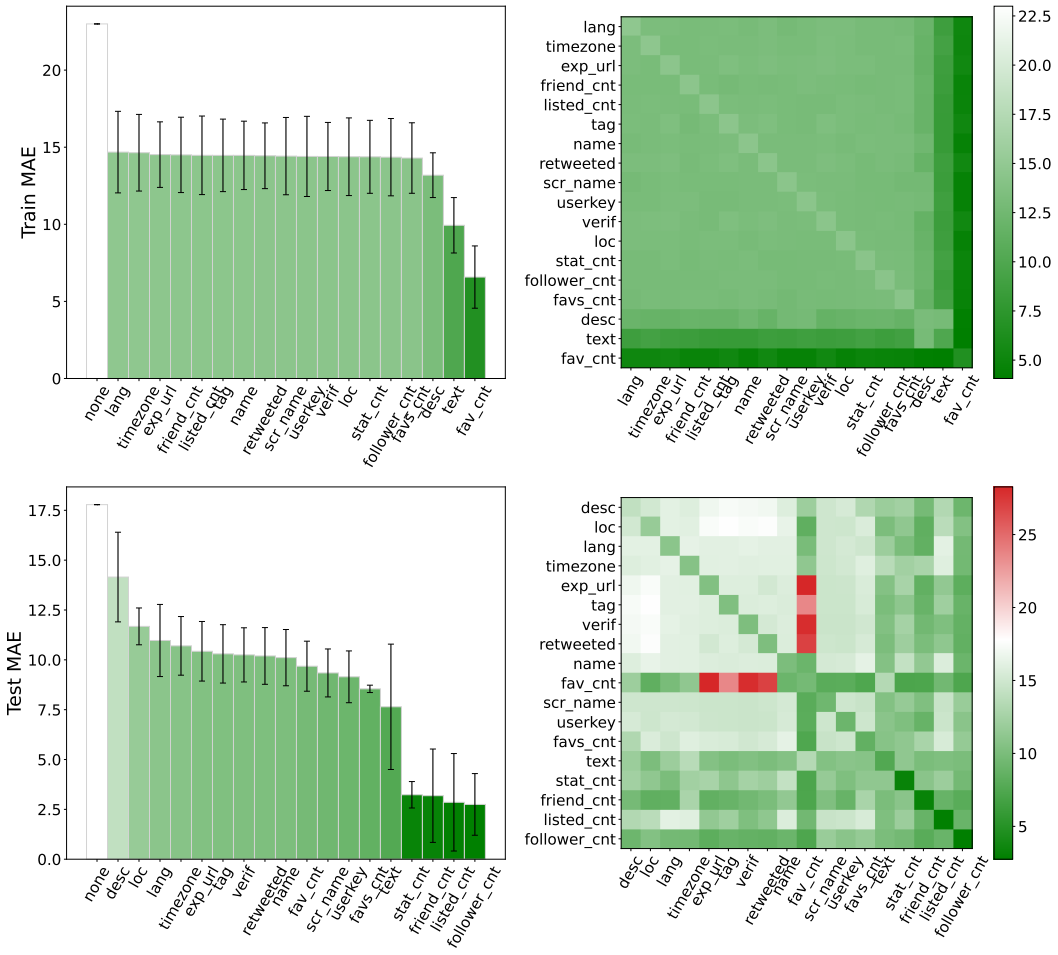

Figure 12: Neo4j Twitter trolls (node regression, GCN-only results). Impact from different properties and their combinations on the MAE. Green: MAE is better than that of a graph with no labels/properties; red: the MAE is worse than that of a graph with no labels/properties.

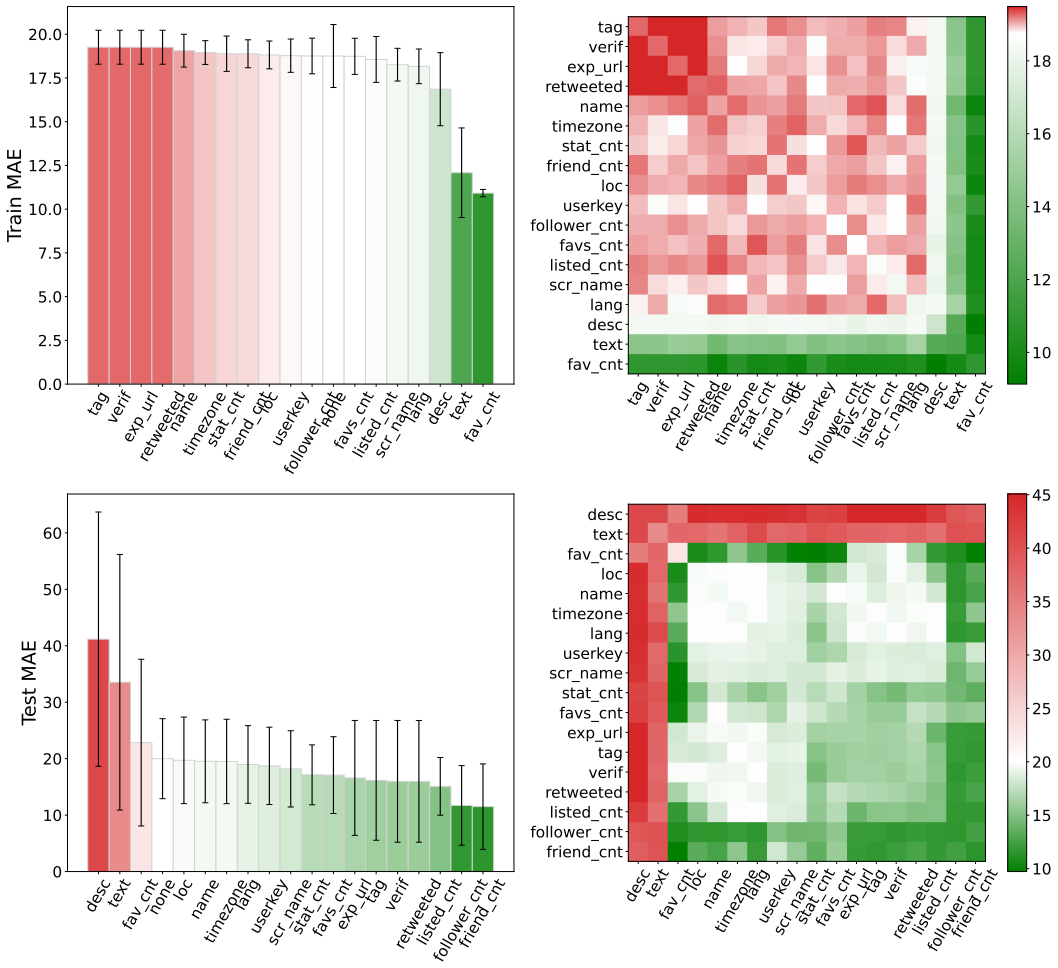

Figure 13: Neo4j Twitter trolls (node regression, GAT-only results). Impact from different properties and their combinations on the MAE. Green: MAE is better than that of a graph with no labels/properties; red: the MAE is worse than that of a graph with no labels/properties.

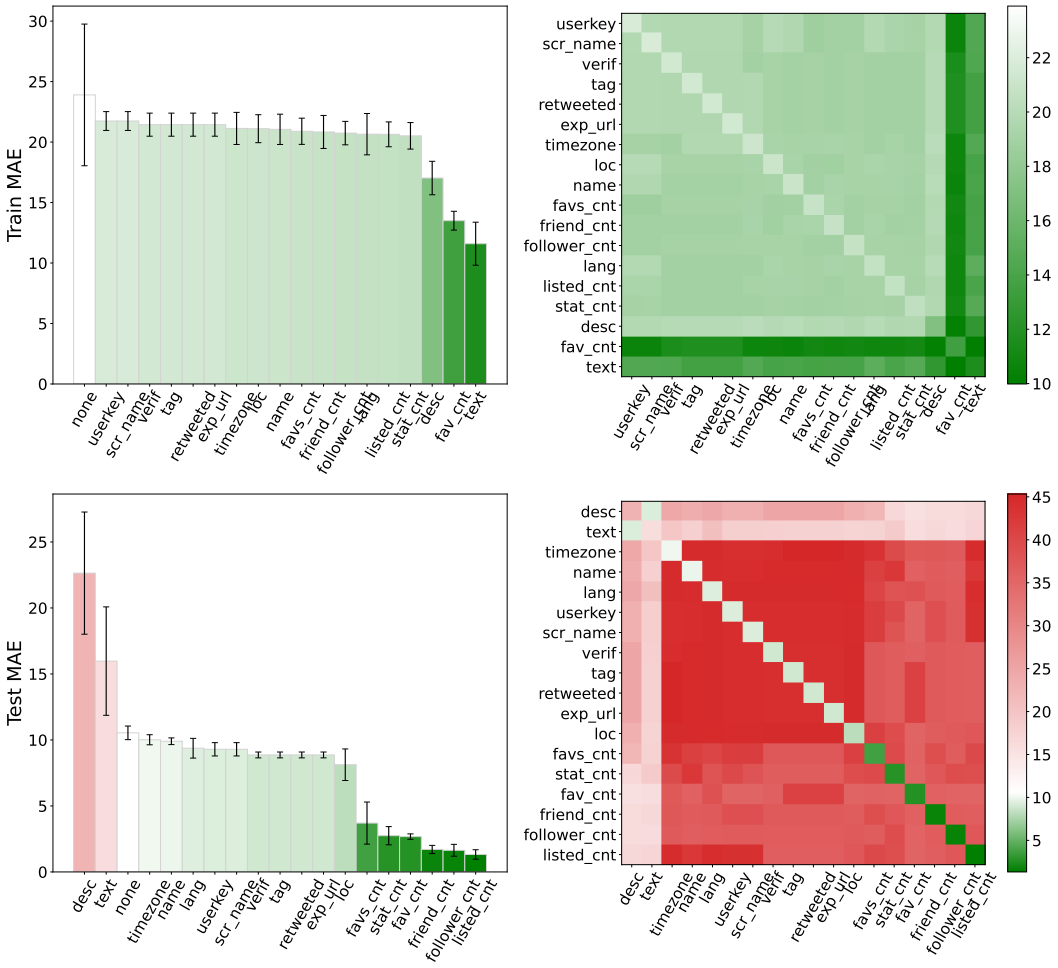

Figure 14: Neo4j Twitter trolls (node regression, GIN-only results). Impact from different properties and their combinations on the MAE. Green: MAE is better than that of a graph with no labels/properties; red: the MAE is worse than that of a graph with no labels/properties.

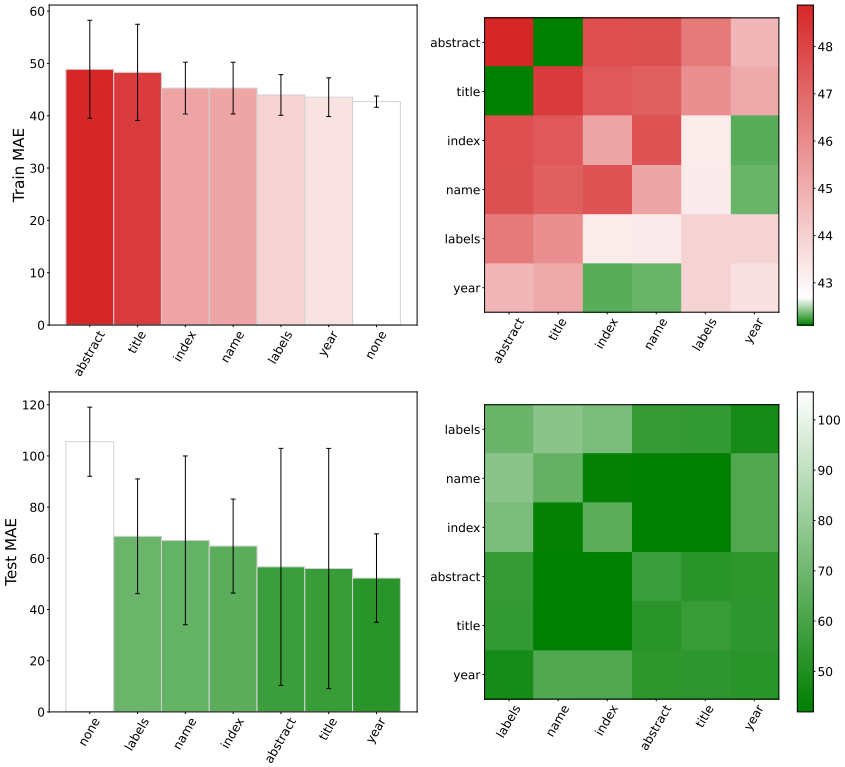

Figure 15: Neo4j citations (node regression, results aggregated over all three models). Impact from different properties and their combinations on the MAE. Green: MAE is better than that of a graph with no labels/properties; red: the MAE is worse than that of a graph with no labels/properties.

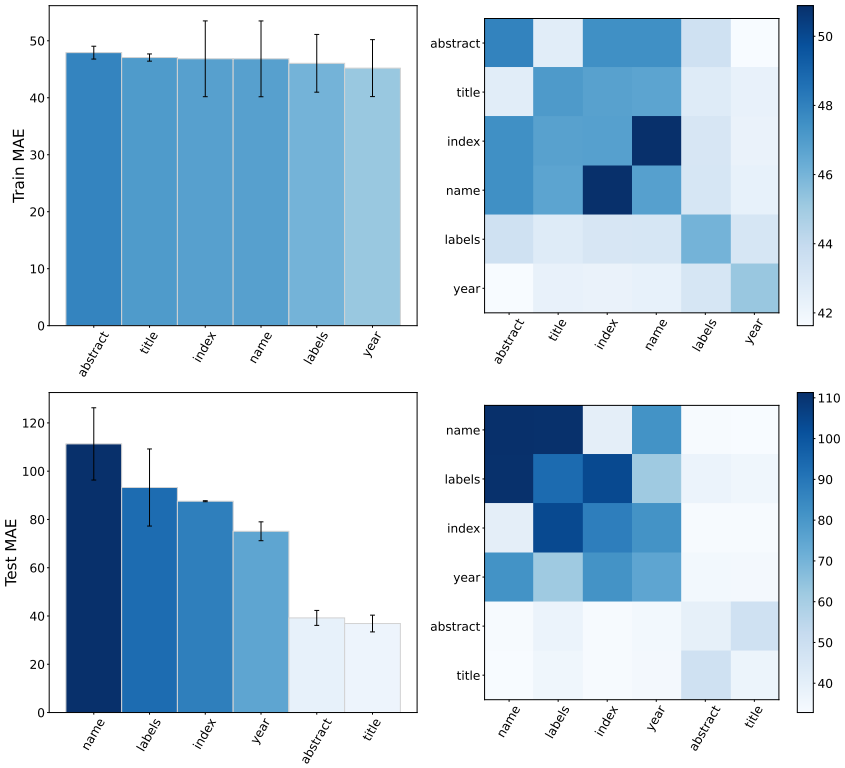

Figure 16: Neo4j citations (node regression, GCN-only results). Impact from different properties and their combinations on the MAE. Here, we do not use green/red colors, because the baselines with no labels/properties could not converge. Instead, we use only one-color (blue) shades to indicate relative improvements.

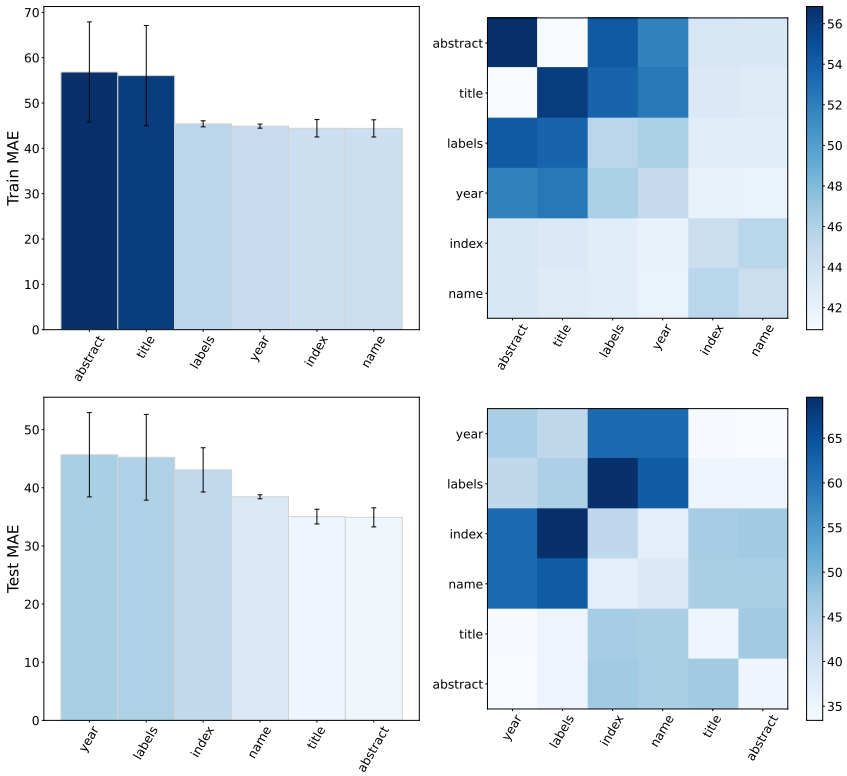

Figure 17: Neo4j citations (node regression, GAT-only results). Impact from different properties and their combinations on the MAE. Here, we do not use green/red colors, because the baselines with no labels/properties could not converge. Instead, we use only one-color (blue) shades to indicate relative improvements.

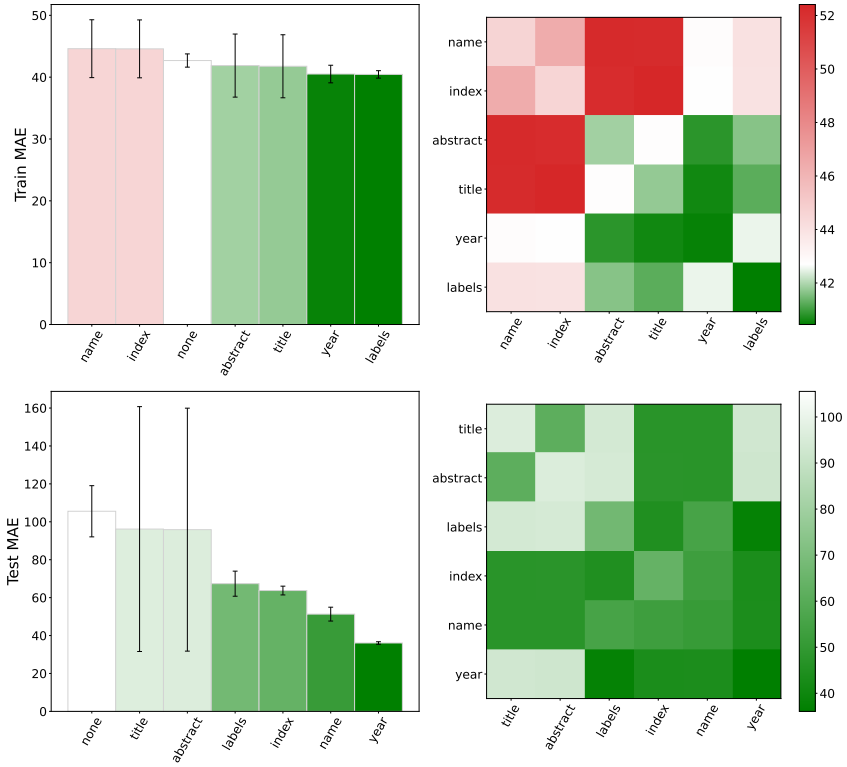

Figure 18: Neo4j citations (node regression, GIN-only results). Impact from different properties and their combinations on the MAE. Green: MAE is better than that of a graph with no labels/properties; red: the MAE is worse than that of a graph with no labels/properties.

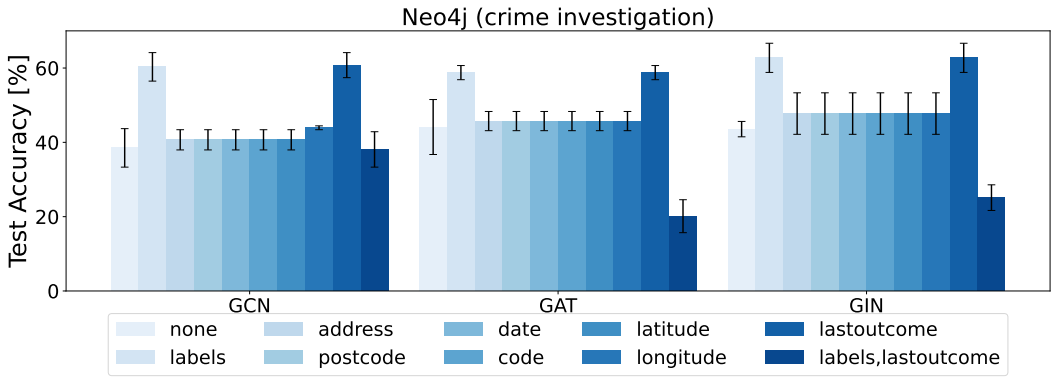

Figure 19: Advantages of preserving the information encoded in LPG labels and properties, for node classification in the Neo4j crime investigation dataset.

C Details of Embedding Construction

n77k

We provide formal specifications of the computed LPG2vec encodings for any vertex i ($\mathbf{x}_i$) and for any edge (i, j) ($\mathbf{x}_{ij}$). The specific fields are as follows: one-hot encoding of the x-th label (l_x) where $x \in \{1, ..., L\}$, one-hot encoding of the y-th property that has C_y potential values ($p_{y,1}, p_{y,2}, ..., p_{y,C_y}$) where $y \in \{1, ..., P\}$, and a string encoding (e.g., BERT) of the z-th text feature that has T_z potential fields ($f_{z,1}, f_{z,2}, ..., f_{z,T_z}$) where $z \in \{1, ..., F\}$. This formal description assumes that all the properties are appropriately discretized and - if needed - normalized. The encoding for edges is fully analogous (for simplicity, we assume that the set of labels and properties $L \cup P$ is common for vertices and edges).

n77k

$$\mathbf{x}_i = \begin{pmatrix} l_1 \\ l_2 \\ \vdots \\ l_L \\ p_{1,1} \\ p_{1,2} \\ \vdots \\ p_{1,C_1} \\ p_{2,1} \\ p_{2,2} \\ \vdots \\ p_{2,C_2} \\ \vdots \\ p_{P,1} \\ p_{P,2} \\ \vdots \\ p_{P,C_P} \\ f_{1,1} \\ f_{1,2} \\ \vdots \\ f_{1,T_1} \\ f_{2,1} \\ f_{2,2} \\ \vdots \\ f_{2,T_2} \\ \vdots \\ f_{F,1} \\ f_{F,2} \\ \vdots \\ f_{F,T_F} \end{pmatrix}$$

$$\mathbf{e}_{i,j} = \begin{pmatrix} l_1 \\ l_2 \\ \vdots \\ l_L \\ p_{1,1} \\ p_{1,2} \\ \vdots \\ p_{1,C_1} \\ p_{2,1} \\ p_{2,2} \\ \vdots \\ p_{2,C_2} \\ \vdots \\ p_{P,1} \\ p_{P,2} \\ \vdots \\ p_{P,C_P} \\ f_{1,1} \\ f_{1,2} \\ \vdots \\ f_{1,F_1} \\ f_{2,1} \\ f_{2,2} \\ \vdots \\ f_{2,F_2} \\ \vdots \\ f_{F,1} \\ f_{F,2} \\ \vdots \\ f_{F,C_F} \end{pmatrix}$$

D Results for Additional Hyperparameters and Models

Cmds n77k x3Ya

We also investigate different training split ratios as well as the counts of convolution layers, see Figures 20 and 21. Adding node features generally improves the accuracy across different GNN models and splits. Differences in the training split ratio for the MAKG dataset have little effect on the accuracy. However, in the citations dataset, the accuracy gets worse when it uses more training data. It indicates that, in this dataset and task, the initial 80% split ratio for the training nodes is too high.

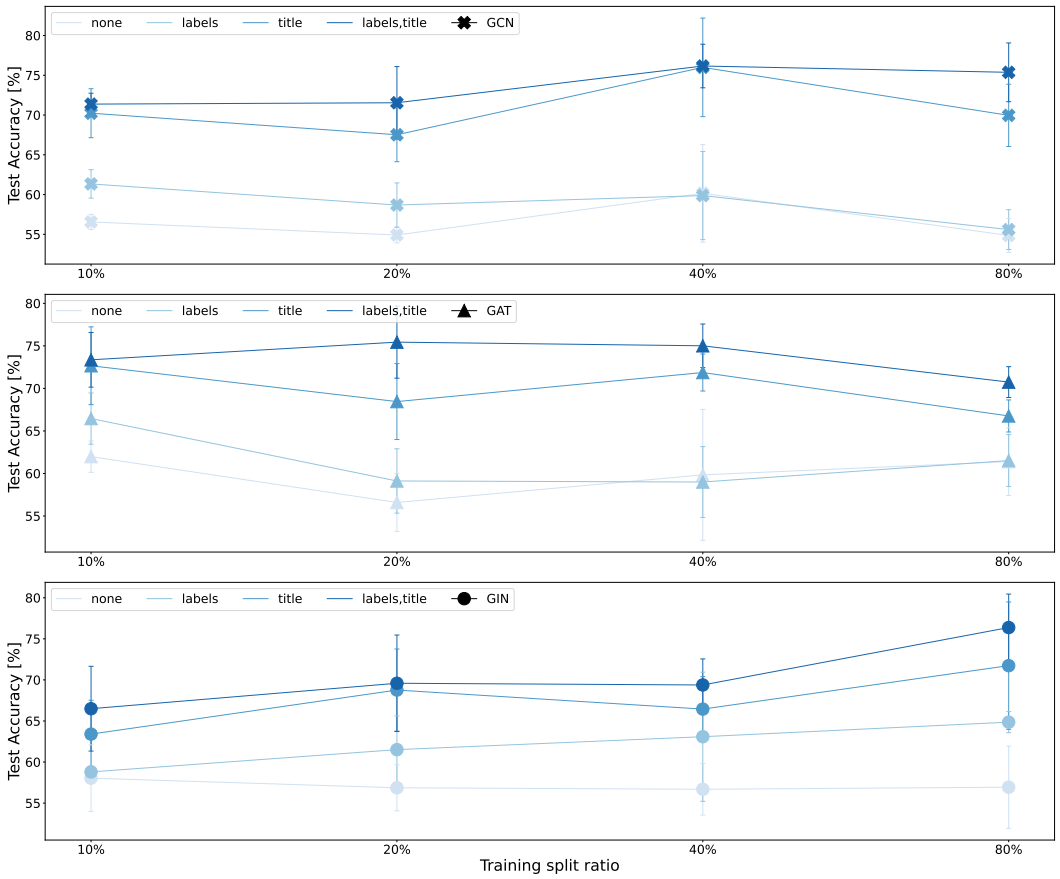

Figure 20: MAKG small (node classification, 4 classes). Impact from different split ratios (the higher the better).

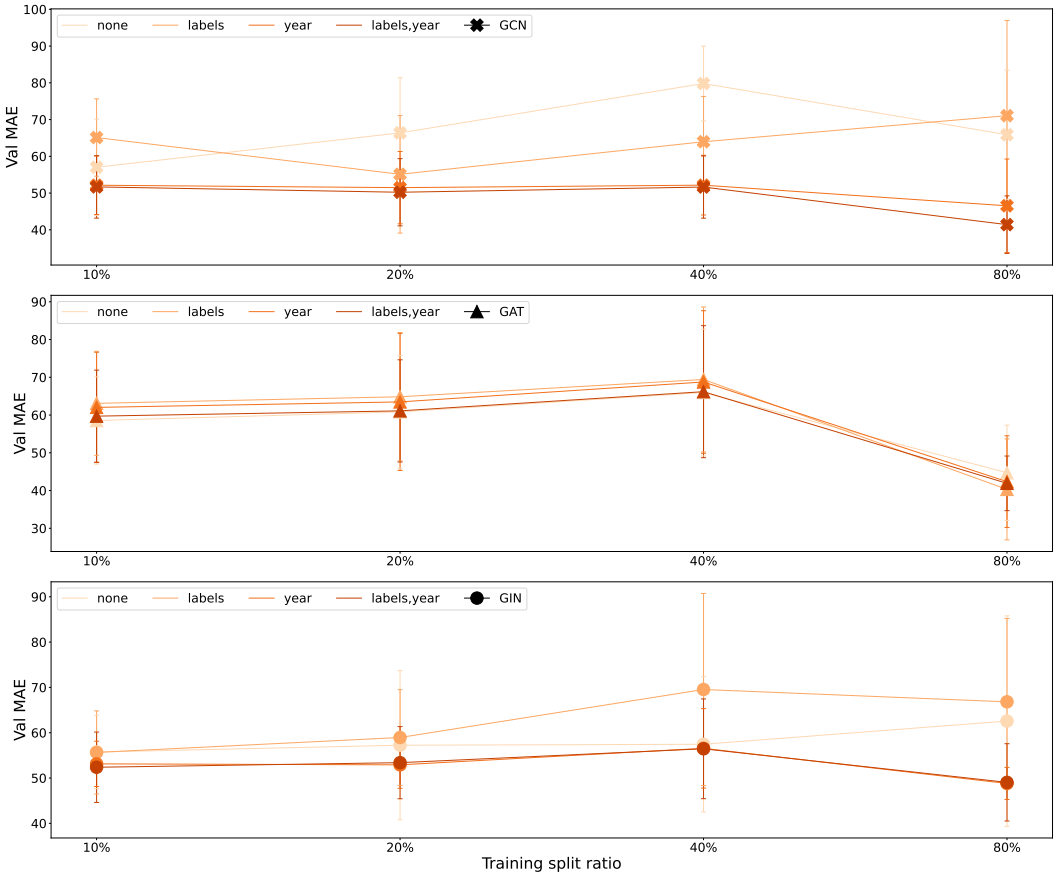

Figure 21: Neo4j citations (node regression). Impact from different split ratios (the lower the better).

We also vary the number convolution layers, see Figure 22. Adding more layers on its own does not bring consistent improvements. This is because the structure of the considered graph datasets usually has a lot of locality and is highly clustered. However, importantly, adding the information from labels and from properties enhances the accuracy consistency across all tried layer counts.

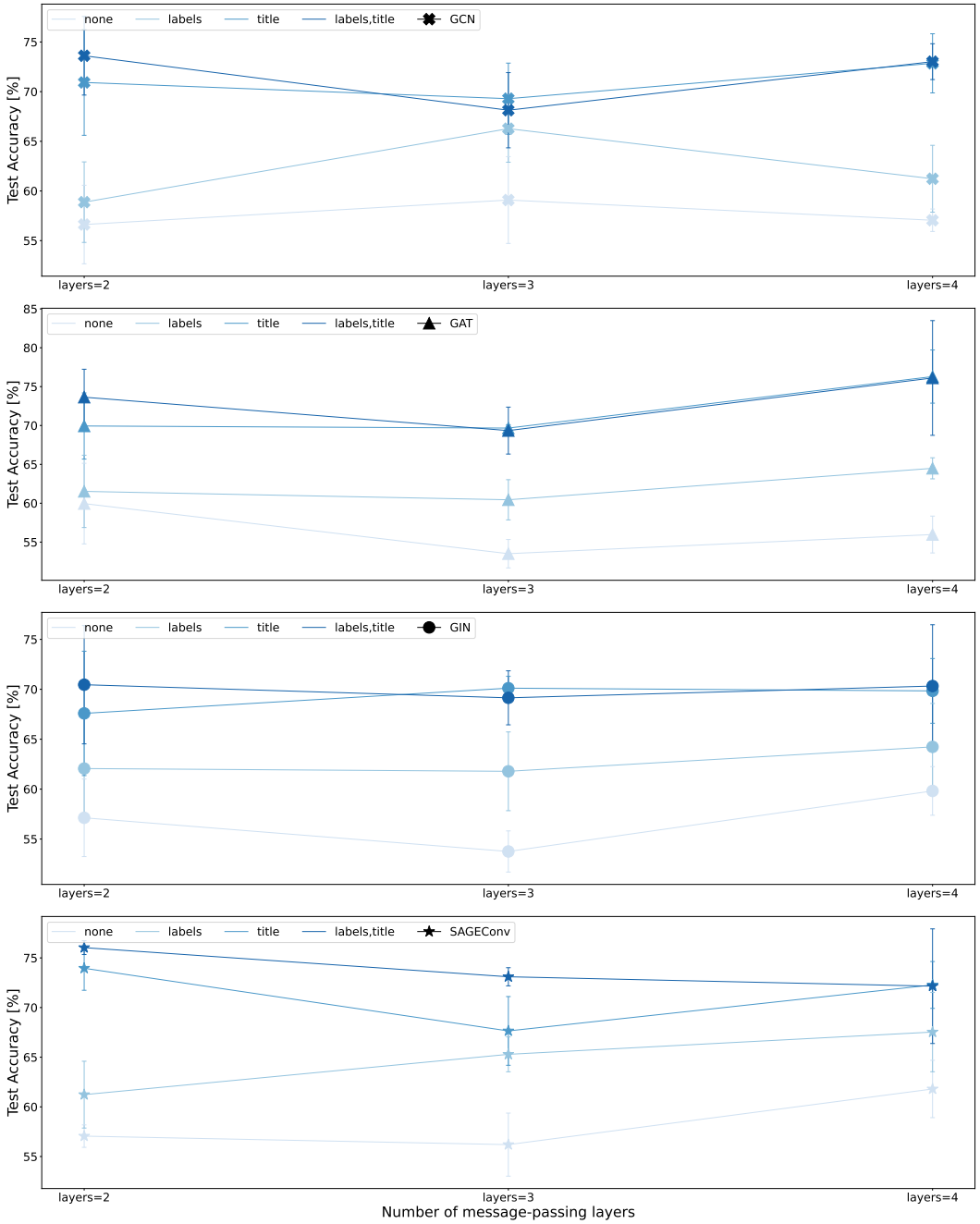

Figure 22: **MAKG small (node classification, 4 classes)**. Impact from different counts of convolution layers (the higher the better).

We also investigated different hyperparameters for LPG2vec embeddings. For example, we experimented with the dimensions of the constructed embeddings. For this, we tried to use an additional MLP to reduce the dimensions of the high dimensional LPG2vec feature vectors, while keeping the information within the features intact. We use two linear layers combined with a dropout layer and the Leaky Relu activation. The dimensions were reduced by different rations, between 20 and 5×. This approach on one hand resulted in much smaller input feature vectors, which could visibly reduce the memory storage overheads for particularly large graphs. However, we also observed consistent accuracy losses across all tried datasets and GNN models. We left more extensive experiments into this direction for future work.

n77k

Finally, we also investigate additional models, GraphSAGE (Figure 23) and plain MLP (Figure 24). As with GCN, GIN, and GAT, adding more labels and more properties enhances the accuracy. MLP comes with much lower accuracy than GraphSAGE for most tried settings (i.e., with most of labels and properties tried). However, interestingly, it becomes only slightly less powerful than GraphSAGE when including the title property. This further shows the importance of harnessing LPG data - when the right data is included into the initial embeddings, it may offer very high accuracy even without considering the graph structure.

Cmds n77k x3Ya

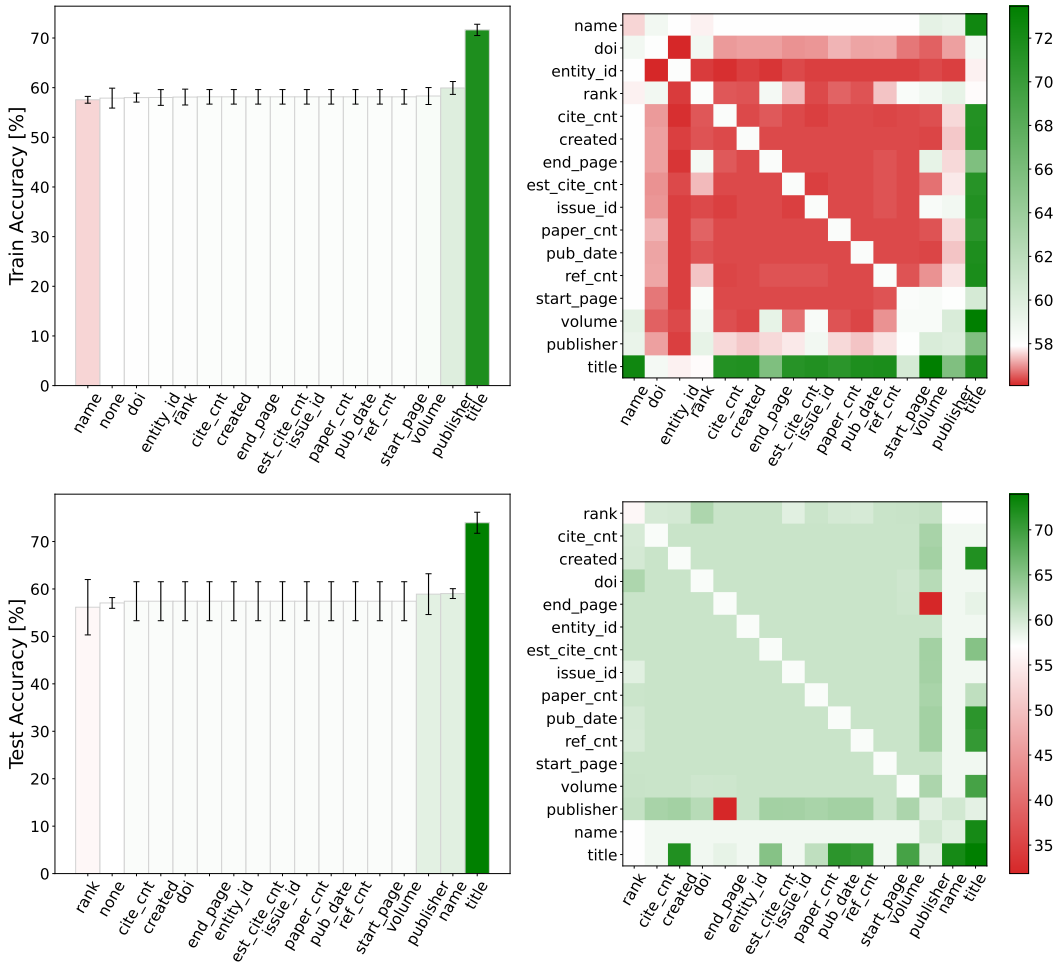

Figure 23: MAKG small (node classification, 4 classes, GraphSAGE-only results). Impact from different properties and their combinations on the accuracy. Green: the accuracy is better than that of a graph with no labels/properties; red: the accuracy is worse than that of a graph with no labels/properties.

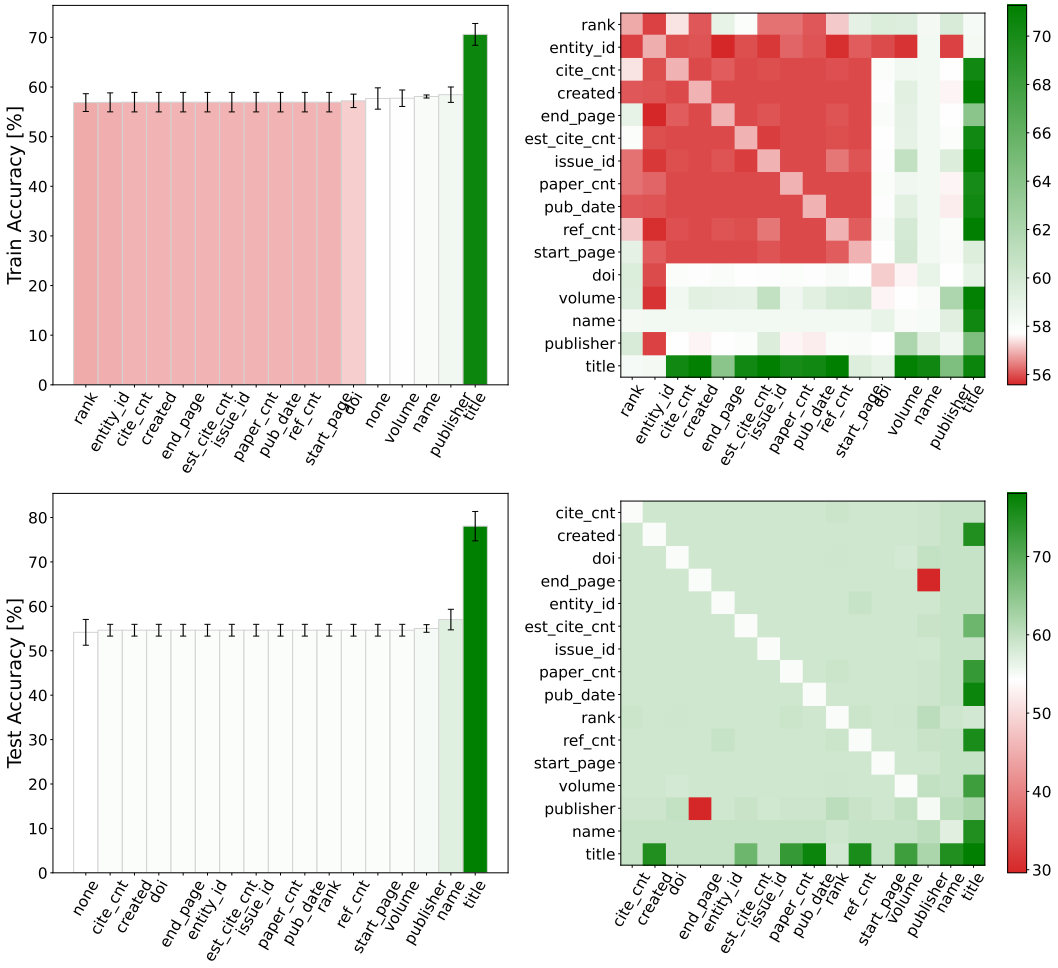

Figure 24: MAKG small (node classification, 4 classes, MLP-only results). Impact from different properties and their combinations on the accuracy. Green: the accuracy is better than that of a graph with no labels/properties; red: the accuracy is worse than that of a graph with no labels/properties.