GRAFENNE: Learning on Graphs with Heterogeneous and Dynamic Feature Sets

Shubham Gupta [*1] **Sahil Manchanda** [*1] **Sayan Ranu** [1] **Srikanta Bedathur** [1]

Abstract

Graph neural networks (GNNs), in general, are built on the assumption of a static set of features characterizing each node in a graph. This assumption is often violated in practice. Existing methods partly address this issue through *feature imputation*. However, these techniques *(i)* assume uniformity of feature set across nodes, *(ii)* are transductive by nature, and *(iii)* fail to work when features are added or removed over time. In this work, we address these limitations through a novel GNN framework called GRAFENNE. GRAFENNE performs a novel *allotropic* transformation on the original graph, wherein the nodes and features are decoupled through a bipartite encoding. Through a carefully chosen message passing framework on the allotropic transformation, we make the model parameter size independent of the number of features and thereby inductive to *both* unseen nodes and features. We prove that GRAFENNE is at least as expressive as any of the existing message-passing GNNs in terms of Weisfeiler-Leman tests, and therefore, the additional inductivity to unseen features does not come at the cost of expressivity. In addition, as demonstrated over four real-world graphs, GRAFENNE empowers the underlying GNN with high empirical efficacy and the ability to learn in continual fashion over streaming feature sets.

1. Introduction and Related Work

Graph Neural Networks (GNNs) have witnessed immense popularity in modeling topological data. GNNs have produced state-of-the-art results in molecular property prediction (Ying et al., 2021; Rampášek et al., 2022), protein function prediction (Hamilton et al., 2017; Nishad et al., 2021; You et al., 2019), modeling of physical systems (Bhattoo et al., 2023; Bishnoi et al., 2023; Thangamuthu et al., 2022; Bhattoo et al., 2022), traffic forecasting (Gupta et al., 2023; Jain et al., 2021), material discovery (Bihani et al., 2023), learning combinatorial algorithms (Manchanda et al., 2020; Ranjan et al., 2022; Chakraborty et al., 2023; Manchanda & Ranu, 2023), and graph generative modeling (Goyal et al., 2020; You et al., 2018; Gupta et al., 2022; Vignac et al., 2023). The effectiveness of GNNs is closely associated with the availability of high-quality input node features (Rossi et al., 2021). An inherent assumption in existing GNNs is the availability of *all* features for each node in the graph. In practice, this assumption is often violated producing *heterogeneous* and *dynamic* feature sets. To motivate, we list two commonly occurring scenarios.

• **Applicability of features:** In graphs with heterogeneous features, the feature set characterizing node v may not be relevant for node u. As an example, consider a co-purchase graph over items in an e-commerce database. While the feature CPU clock-speed is relevant for smartphones, it does not apply to smartphone covers. While one may homogenize the features set across all nodes by attributing a special value to denote non-relevant features, it significantly enlarges feature dimensionality leading to inefficiencies in the modeling, computational and storage components.

• **Feature set refinement:** Feature sets may get altered over time (Leskovec & Krevl, 2014; Bai et al., 2018). Consider the evolution of smartphones into the foldable form factor. While a feature characterizing the form-factor is required in today's context, it was not conceivable five years back. Similarly, in a dating network, users might choose to remove/hide personal attributes related to income, education, and occupation at a later stage after initial sign-up. Furthermore, the dating app might ask for covid vaccination status in the post-pandemic era. Since the number of model parameters in GNNs is a function of the input node feature dimension, when the feature set changes, the entire model needs to be retrained from scratch. The ideal solution lies in decoupling the GNNs parameters with feature set size and *continually adapting* the existing model to new features without forgetting the past.

[*]Equal contribution [1]Department of Computer Science and Engineering, IIT Delhi. Correspondence to: Shubham Gupta <shubham.gupta@cse.iitd.ac.in>, Sahil Manchanda <sahil.manchanda@cse.iitd.ac.in>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

1.1. Existing Works

The closest work to ours is topology-aware feature imputation over missing features (Taguchi et al., 2021; Jiang & Zhang, 2020; Rossi et al., 2021)[1]. In feature imputation, the value of a non-existing feature is predicted based on other existing features and the graph topology. Feature imputation, however, is not adequate for the proposed problem.

- **Feature relevance:** Feature imputation assumes that a feature is relevant, but missing. Heterogeneous feature sets surfaces a different problem where a feature itself is not relevant and hence imputation is an irrational task.
- **Transductive modeling:** Topology-aware feature imputation algorithms, such as GCNMF (Taguchi et al., 2021), PAGNN (Jiang & Zhang, 2020) and FP (Rossi et al., 2021), require computation of the Graph Laplacian, rendering them incapable of modeling unseen nodes, i.e., nodes not observed in training.
- **Lacking ability for continual learning:** As discussed above, it is natural for datasets to update feature sets to stay in sync with their evolution. This necessitates developing a *continual learning* framework for a streaming node feature scenario that avoids *catastrophic forgetting* on the portion of the graph that is unaffected in the update. Existing algorithms for topology–aware feature imputation do not support this need due to being transductive. On the other hand, algorithms for continual learning on graphs (Wang et al., 2022; 2020a; Liu et al., 2021) perform continual learning only over nodes with homogeneous feature sets.

1.2. Contributions

In this work, we propose GRAph FEature Neural NEtwork (GRAFENNE) to address the above-highlighted gaps. GRAFENNE is built on the following novel contributions:

- **Continual, assumption-free and GNN-agnostic modeling:** GRAFENNE transforms the input graph into a graph consisting of two disjoint sets of *graph nodes* and *feature nodes*. Through a novel message passing scheme across these nodes, GRAFENNE ensures three key properties. First, the number of model parameters is independent of the number of nodes or features. Hence, it is inductive to both unseen nodes and features. This enables an easy transition to *continual* learning. Second, the flexible transformation and message passing scheme can mimic any of the existing GNN architectures. Third, it bypasses the need to impute features and thereby imbibing the ethos of heterogeneous feature sets.
- **Expressivity:** We prove that given any GNN of k-WL expressivity, they can be embedded into the GRAFENNE framework to retain their full expressive power under k-WL while also imbibing the above mentioned properties.
- **Empirical evaluation:** Extensive experiments on diverse real-world datasets establish that GRAFENNE consistently outperforms baseline methods across various levels of feature scarcity on both homophilic as well as heterophilic graphs. Further, the method is robust to extreme low availability of node features.

[1]Topology-unaware methods (Wu et al., 2021) are not well-suited for the task due to their inability to model node-dependencies (See. Appendix Sec. L)

2. Preliminaries

Definition 1 (Graph). *A graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ over node and edge sets $\mathcal{V}$ and $\mathcal{E} = \{(u, v) \mid u, v \in \mathcal{V}\}$ respectively. $\mathbf{X} \in \mathbb{R}^{|V| \times |F|}$ is a node feature matrix where F is the set of all features in graph $\mathcal{G}$. F is assumed to be available for all nodes $v \in \mathcal{V}$.*

Assuming $\mathbf{x}_v \in \mathbb{R}^{|F|}$ as input feature vector for every node $v \in \mathcal{V}$, the 0th layer embedding of node v is denoted as:

$$\mathbf{h}_v^0 = \mathbf{x}_v \;\; \forall v \in \mathcal{V} \tag{1}$$

To compute the ℓth layer representation of node v, GNNs compute the message from its neighbourhood $\mathcal{N}_v = \{u \mid (u, v) \in \mathcal{E}\}$ and aggregate them as follows:

$$\mathbf{m}_v^\ell(u) = \text{MSG}^\ell(\mathbf{h}_u^{\ell-1}, \mathbf{h}_v^{\ell-1}) \; \forall u \; \in \mathcal{N}_v \tag{2}$$

$$\overline{\mathbf{m}}_v^\ell = \text{AGGREGATE}^l(\{\!\{\mathbf{m}_v^l(u), \forall u \in \mathcal{N}_v\}\!\}) \tag{3}$$

where MSG^l and AGGREGATE^l are either pre-defined functions (Ex: MEANPOOL) or neural networks (GAT (Veličković et al., 2018)). $\{\!\{\ldots\}\!\}$ denotes a multi-set; it is a multi-set since the same message may be received from multiple neighbors.Finally, GNNs compute the lth layer representation of node v as follows:

$$\mathbf{h}_v^\ell = \text{COMBINE}^l(\mathbf{h}_v^{\ell-1}, \overline{\mathbf{m}}_v^\ell) \tag{4}$$

where COMBINE is a neural network. The node representations of the final layer, denoted as $\mathbf{h}_v$, are used for downstream tasks such as node classification, link prediction, etc.

Assumptions: As summarized in the above generic framework, all node representations undergo the same transformation in each layer. While this design allows GNNs to decouple the number of model parameters from the node set size $|\mathcal{V}|$, it forces the parameter size to be a function of the node representation dimension. Moreover, node features $\mathbf{x}_v$ are tightly coupled with node representation $\mathbf{h}_v^\ell$ in each hidden layer ℓ (See Eq. 1 and Eq. 4). Consequently, if the feature set size is heterogeneous, or dynamic due to changes over time, GNNs are either inapplicable, or requires re-training from scratch following each feature set update. Our objective is to remove this assumption and mitigate the resultant shortcomings.

3. Problem Formulation

First, we redefine graph (Def. 1) by relaxing the constraint that all nodes should consist of same set of input features:

Definition 2 (Graph with heterogeneous feature set). *A graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ where $\mathcal{V}$ is a set of nodes and $\mathcal{E} = \{(u, v) \mid u, v \in \mathcal{V}\}$ is a set of edges. $\mathbf{X} = \left[\mathbf{x}_1, \mathbf{x}_2 \ldots \mathbf{x}_{|\mathcal{V}|}\right]$ is a collection of node feature vectors where $\mathbf{x}_v \in \mathbb{R}^{|F_v|}, \forall v \in \mathcal{V}$. F_v is the set of features available at node v and $F = \bigcup_{v \in \mathcal{V}} F_v$ is the union of available features across all nodes in the graph.*

We note that $\mathbf{X}$ is not a matrix since the dimension of each $\mathbf{x}_v$ $\forall v \in \mathcal{V}$ can be different. Moreover, each dimension in $\mathbf{x}_v$ may have a different meaning for every v. Thus, existing GNNs cannot train on these graphs without including missing-value imputation as an intermediate step. Motivated by this, we now define the following novel formulation.

Problem 1 (GNN for graphs with heterogeneous features).

Input: *Given a graph $\mathcal{G}$ (Def. 2), let $Y : \mathcal{V} \rightarrow \mathbb{R}$ be a hidden function that maps a node to a real number*[2]*. $Y(v)$ is known to us only for subset $\mathcal{V}_l \subset \mathcal{V}$ and may model some downstream task such as node classification, or link prediction.*

Goal: *Learn parameters Θ of a graph neural network, denoted as GNN_Θ, that predicts $Y(v)$, $\forall v \in \mathcal{V}_l$ accurately. We focus on inductive learning so that GNN_Θ can predict on unseen graphs (and nodes).*

Prob. 1 is formulated in the context of static graphs with heterogeneous feature sets. It does not consist any temporal characterisation. We further generalize Def. 2 to allow temporal characteristics.

Definition 3 (Dynamic graphs). *A dynamic (or streaming) graph is a sequence of graph (as per Def. 2) snapshots recorded at consecutive timestamps $t = 1, 2 \ldots$ and represented as $\overrightarrow{\mathcal{G}} = (\mathcal{G}_1, \mathcal{G}_2 \ldots)$ where $\mathcal{G}_t = (\mathcal{V}_t, E_t, \mathbf{X}_t)$.*

In Def. 3, nodes/edges/features can be added or existing ones can be deleted in consecutive snapshots, i.e., $\mathcal{V}_{t+1} = \mathcal{V}_t + \Delta\mathcal{V}_t$, $E_{t+1} = E_t + \Delta E_t$ and $\mathbf{X}_{t+1} = \mathbf{X}_t + \Delta\mathbf{X}_t$. We note that different subsets of features can be added/removed for all nodes or a subset of nodes during each update. To ease the notational burden, we overload the notation of $\Delta\mathcal{V}_T$ to denote all nodes that were either deleted, added, or underwent a feature change. Each of the three update types can be distinguished with an appropriate indicator variable.

Following Def. 3, we write $\mathcal{G}_t = \mathcal{G}_{t-1} + \Delta\mathcal{G}_t$ where $\Delta\mathcal{G}_t = (\Delta\mathcal{V}_t, \Delta E_t, \Delta X_t)$. The sequential graph updates $\Delta\mathcal{G}_t$ may contain new patterns, thus making GNN_{Θ_1} trained using $\mathcal{G}_1$, ineffective over $\mathcal{G}_2$, $\mathcal{G}_3$, $\ldots, \mathcal{G}_t$. However, retraining GNN_{Θ_t} from scratch on $\mathcal{G}_t$ is computationally expensive as well. A vanilla solution of online learning resides in updating Θ_{t-1} only on $\Delta\mathcal{G}_t$. This, however, may lead to *catastrophic forgetting* (Parisi et al., 2019) since the learned parameters will be biased towards new patterns and drift away from patterns learned earlier. This motivates us to develop a *continual training* framework where learned parameters perform effectively on $\Delta\mathcal{G}_t$ while retaining information about $\mathcal{G}_t - \Delta\mathcal{G}_t$ as well. This leads us to define the following problem statement.

[2]It is easy to adapt $Y(v)$ for edge or graph level tasks by learning an aggregation function over its constituent node representations.

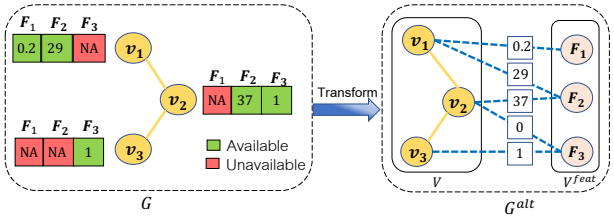


Figure 1: Transformation of input graph $\mathcal{G}$ to its allotropic form $\mathcal{G}^{alt}$. In $\mathcal{G}$, the features marked in green are available for the corresponding node.

Problem 2 (Continual Learning over GNNs). *We extend Prob. 1 for dynamic graphs with the following objectives:*

- **Training accuracy:** *Given $\overrightarrow{\mathcal{G}} = \{\mathcal{G}_1, \ldots, \mathcal{G}_t\}$, learn $\{\Theta_1, \ldots, \Theta_t\}$ such that $\forall v \in \mathcal{V}_{t_l}$, GNN_{Θ_t} is accurate; $\mathcal{V}_{t_l} \subseteq \mathcal{V}_t$ is the set of training nodes.*
- **Computational efficiency:** *Learning Θ_t should be significantly more efficient than retraining from scratch on $\mathcal{V}_{t_l}$.*

We achieve the above objectives by grounding our update mechanism for Θ_t predominantly on $\Delta\mathcal{V}_t \cap \mathcal{V}_{t_l}$. The next section details our methodology.

4. Proposed GNN framework: GRAFENNE

GRAFENNE constitutes of three major components: **(1)** An *allotropic* graph transformation that enables decoupling of model parameter-size from the number of features, **(2)** a novel generic message-passing mechanism on transformed graph that is provably as expressive as performing message passing on the original graph, and **(3)** a continual learning framework to adapt to new nodes, edges and features efficiently and without catastrophic forgetting.

4.1. Graph Transformation

Given graph snapshot (Def. 2) $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, we construct its *allotropic* version $\mathcal{G}^{alt} = \left(\mathcal{V}^{alt}, \mathcal{E}^{alt}\right)$. $\mathcal{V}^{alt} = \mathcal{V} \cup \mathcal{V}^{feat}$, where in addition to the original nodes $\mathcal{V}$, we add a node for each unique feature in $\mathcal{G}$. Formally,

$$\mathcal{V}^{feat} = F = \{f \mid f \in \cup_{\forall v \in \mathcal{V}} F_v\} \quad \text{(Recall Def. 2)}$$

We call $\mathcal{V}^{feat}$ the *feature nodes*. A regular node $v \in \mathcal{V}$ is connected to a feature node $f \in \mathcal{V}^{feat}$ if f characterizes v, i.e., $f \in f_v$. The weight of this edge is the value of feature f in node v. In addition, we also retain all the edges among the original nodes. Thus $\mathcal{E}^{alt} = \mathcal{E} \cup \mathcal{E}^{feat}$, where:

$$\mathcal{E}^{feat} = \{(v, f, \mathbf{x}_v[f]) \mid v \in \mathcal{V}, f \in F_v\}$$

Here, $\mathbf{x}_v[f] \in \mathbb{R}$ refers to the edge weight/feature value between original graph node v and feature $f \in \mathcal{V}^{feat}$. Fig. 1 illustrates the transformation process.

In $\mathcal{G}^{alt}$, the feature nodes only have graph nodes as neighbors. In contrast, graph nodes have both feature nodes as well as other graph nodes in their neighborhood. To distinguish between these two types, we define the notions of *graph neighborhood* and *feature neighborhood*.

Definition 4 (Feature Neighborhood). *The feature neighborhood of a node $v \in \mathcal{V}^{alt}$ is defined as $\mathcal{N}_v^{feat} = \{u \mid (u, v) \in \mathcal{E}^{feat},\ v \in \mathcal{V}^{feat}\}$.*

Definition 5 (Graph Neighborhood). *For a given node $v \in \mathcal{V}^{alt}$, the graph neighbourhood $\mathcal{N}_v^{\mathcal{G}} = \{u \mid (u, v) \in \mathcal{E},\ v \in \mathcal{V}\}$ consists of only graph nodes.*

Note that $\forall v \in \mathcal{V}^{feat}$, $\mathcal{N}_v^{\mathcal{G}} = \emptyset$.

4.2. Message Passing Layer for $\mathcal{G}^{alt}$

Our goal is to perform message passing on $\mathcal{G}^{alt}$ in order to learn rich representations for graph nodes $v \in \mathcal{V}$ such that: **(1)** Attribute information is captured from the feature nodes, **(2)** Topological information is captured from the neighborhood defined over $\mathcal{E}$, **(3)** It decouples the size of model parameters from the number of features, and **(4)** It theoretically guarantees that message passing on the allotropic form does not lead to reduction in expressive power when compared to executing a GNN on $\mathcal{G}$.

To achieve our goals, the message passing scheme is broken down into three phases. A single layer of message passing is completed on the completion of these three phases.

•**Phase 1 – Aggregating feature information:** Messages are sent from feature nodes $u \in \mathcal{V}^{feat}$ to graph nodes $v \in \mathcal{V}$ to aggregate edge weights (carrying the feature value). Formally,

$$\mathbf{m}_v^{\ell}(u) = \textsc{Msg}_{feat}^{\ell}\left(\mathbf{h}_u^{\ell-1}, \mathbf{h}_v^{\ell-1}, e_{uv}\right) \tag{5}$$

$$\overline{\mathbf{m}}_v^{\ell} = \textsc{Aggregate}_{feat}^{\ell}\left(\left\{\!\!\left\{\mathbf{m}_v^{\ell}(u) \mid u \in \mathcal{N}_v^{feat}\right\}\!\!\right\}\right) \tag{6}$$

$$\mathbf{h}_v^{\ell} = \textsc{Combine}_{feat}^{\ell}\left(\mathbf{h}_v^{\ell-1}, \overline{\mathbf{m}}_v^{\ell}\right) \tag{7}$$

Here, $e_{uv} = \mathbf{x}_v[u]$ is edge weight between feature node $u \in \mathcal{V}^{feat}$ and graph node $v \in \mathcal{V}$ in $\mathcal{G}^{alt}$. ℓ denotes the message-passing layer.

• **Phase 2 – Aggregating topological information:** Utilizing the information aggregated in the previous phase, exchange messages between graph nodes $u, v \in \mathcal{V}$ and aggregate them as follows:

$$\mathbf{m}_v^{\ell}(u) = \textsc{Msg}_{\mathcal{G}}^{\ell}\left(\mathbf{h}_u^{\ell}, \mathbf{h}_v^{\ell}\right) \tag{8}$$

$$\overline{\mathbf{m}}_v^{\ell} = \textsc{Aggregate}_{\mathcal{G}}^{\ell}\left(\left\{\!\!\left\{\mathbf{m}_v^{\ell}(u) \mid u \in \mathcal{N}_v^{\mathcal{G}}\right\}\!\!\right\}\right) \tag{9}$$

$$\mathbf{h}_v^{\ell} = \textsc{Combine}_{\mathcal{G}}^{\ell}\left(\mathbf{h}_v^{\ell}, \overline{\mathbf{m}}_v^{\ell}\right) \tag{10}$$

• **Phase 3 – Integrating attribute and topology information:** Messages are exchanged back from graph nodes $v \in \mathcal{V}$ to feature nodes $u \in \mathcal{V}^{feat}$ and aggregated as follows.

$$\mathbf{m}_u^{\ell}(v) = \textsc{Msg}_{\mathcal{G}'}^{\ell}\left(\mathbf{h}_v^{\ell}, \mathbf{h}_u^{\ell-1}, e_{uv}\right) \tag{11}$$

$$\overline{\mathbf{m}}_u^{\ell} = \textsc{Aggregate}_{\mathcal{G}'}^{\ell}\left(\left\{\!\!\left\{\mathbf{m}_u^{\ell}(v) \mid v \in \mathcal{N}_u^{feat}\right\}\!\!\right\}\right) \tag{12}$$

$$\mathbf{h}_u^{\ell} = \textsc{Combine}_{\mathcal{G}'}^{\ell}\left(\mathbf{h}_u^{\ell-1}, \overline{\mathbf{m}}_u^{\ell}\right) \tag{13}$$

Fig. 2 provides a visual depiction of the three phases of message passing in GRAFENNE. $\textsc{Msg}_{feat}^{\ell}$, $\textsc{Msg}_{\mathcal{G}}^{\ell}$, $\textsc{Msg}_{\mathcal{G}'}^{\ell}$, $\textsc{Aggregate}_{feat}^{\ell}$, $\textsc{Aggregate}_{\mathcal{G}}^{\ell}$, $\textsc{Aggregate}_{\mathcal{G}'}^{\ell}$, $\textsc{Combine}_{feat}^{\ell}$, $\textsc{Combine}_{\mathcal{G}}^{\ell}$ and $\textsc{Combine}_{\mathcal{G}'}^{\ell}$ are neural network based functions with semantics defined in preliminaries section. Any existing GNN can be used to implement these three phases since they are essentially exchanging information between nodes. This makes GRAFENNE a general and flexible method. We discuss our specific implementation in §. 4.2.1.

Initialization: We use $\mathbf{0}$ as a feature vector for graph nodes $\mathcal{V}$ in $\mathcal{G}^{alt}$ since they no longer have any node features, i.e.,

$$\mathbf{h}_v^0 = \mathbf{0}\ \forall\ v \in \mathcal{V} \tag{14}$$

For feature nodes $\mathcal{V}^{feat}$, we set them either to a learnable vector initialized randomly or to a latent representation learnt in a pre-processing step. Specifically,

$$\mathbf{h}_u^0 = \boldsymbol{w}_u \in \mathbb{R}^d \quad \forall\ u \in \mathcal{V}^{feat} \tag{15}$$

4.2.1. SPECIFICS

The neural networks in phase 1 and phase 3 are defined as attention-based aggregators. Since phase 2 involves message passing only among graph nodes, we can adopt the message passing scheme of any static GNN (Hamilton et al., 2017; Veličković et al., 2018; Kipf & Welling, 2017; Morris et al., 2019). Some possible options are outlined in App. A. Phases 1 and 3 may also be customized to different neural networks as per needs.

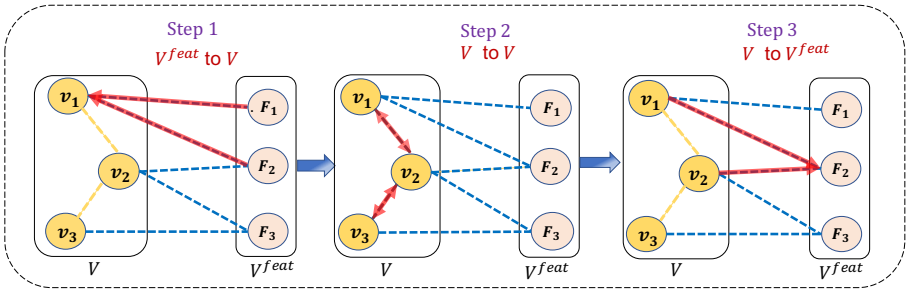


Figure 2: A break-down of the message passing layer of GRAFENNE into its three phases.

Phase 1: $\forall v \in \mathcal{V},\ u \in \mathcal{N}_v^{feat}$

$$\mathbf{m}_v^\ell(u) = \text{LEAKYRELU}\left(\mathbf{W}_1^\ell \mathbf{h}_v^{\ell-1} \Big|\Big| \mathbf{W}_2^\ell \mathbf{h}_u^{\ell-1} \Big|\Big| \mathbf{w}_3^\ell e_{uv}\right) \tag{16}$$

$$\alpha_{vu} = \frac{\exp\left(\mathbf{w}_4^{\ell\,T} \mathbf{m}_v^\ell(u)\right)}{\sum\limits_{u' \in \mathcal{N}_v^{feat}} \exp\left(\mathbf{w}_4^{\ell\,T} \mathbf{m}_v^\ell(u')\right)}, \tag{17}$$

$$\mathbf{h}_v^\ell = \text{MLP}\left(\mathbf{W}_5^\ell \mathbf{h}_v^{l-1} \Big|\Big| \sum_{u \in \mathcal{N}_v^{feat}} \alpha_{vu} \mathbf{W}_6^\ell \mathbf{h}_u^{\ell-1}\right) \tag{18}$$

Phase 3: $\forall u \in \mathcal{V}^{feat},\ v \in \mathcal{N}_u^{\mathcal{G}}$

$$\mathbf{m}_u^\ell(v) = \text{LEAKYRELU}\left(\mathbf{W}_7^\ell \mathbf{h}_u^{\ell-1} \Big|\Big| \mathbf{W}_8^\ell \mathbf{h}_v^{l} \Big|\Big| \mathbf{w}_9^\ell e_{uv}\right)$$

$$\alpha_{vu} = \frac{\exp\left(\mathbf{w}_{10}^{\ell\,T} \mathbf{m}_u^\ell(v)\right)}{\sum\limits_{v' \in \mathcal{N}_u^{\mathcal{G}}} \exp\left(\mathbf{w}_{10}^{\ell\,T} \mathbf{m}_u^\ell(v')\right)}$$

$$\mathbf{h}_u^\ell = \text{MLP}\left(\mathbf{W}_{11}^\ell \mathbf{h}_u^{\ell-1} \Big|\Big| \sum_{v \in \mathcal{N}_u^{\mathcal{G}}} \alpha_{vu} \mathbf{W}_{12}^\ell \mathbf{h}_v^{l}\right)$$

All weights matrices and vectors of the form $\mathbf{W}_i^\ell$ and $\mathbf{w}_i^\ell$ respectively are trainable parameters. $||$ represents the *concatenation* operator. Note that since each edge weight goes through an MLP, the proposed scheme is expressive enough to model scaling and translation factors.

4.3. Theoretical Characterization

With the formalization of our message-passing algorithm, we have a GNN framework for graphs with heterogeneous feature sets. Next, we analyze its inductivity, expressivity, and complexity.

4.3.1. INDUCTIVITY

Proposition 1. GRAFENNE *is inductive to both unseen features and nodes, i.e., once trained, it is capable of producing representations for unseen nodes with unseen features.*

Existing GNNs are not feature-inductive as they must be re-trained if a new feature is added to the input graph. GRAFENNE decouples features from GNNs's parameters by treating them as nodes in $\mathcal{G}^{alt}$. Moreover, GRAFENNE learns aggregation functions over feature nodes to compute the graph node representations in Phase-1. These aggregation functions are independent of the number of features available to the target node. Such design empowers GRAFENNE to detect patterns even if unseen feature nodes are added in the target node. We formally prove this in App. B.

4.3.2. EXPRESSIVITY

Expressivity of GNNs is measured by their ability to discriminate non-isomorphic graph structures (Xu et al., 2019; Morris et al., 2019) in terms of k-*Weisfeiler Leman (WL)* equivalence. As discussed in § 4.2.1, the Phase-2 of the message-passing layer could adopt any existing GNN Ψ's message-passing scheme. We show that executing GRAFENNE on the allotropic form $\mathcal{G}^{alt}$ with Ψ in Phase-2 does not lead to a reduction in expressive power over executing GNN Ψ on $\mathcal{G}$.

Theorem 1 (Expressivity of GRAFENNE). *Let $\Psi_{\mathcal{G}}(v) : \mathcal{V} \rightarrow \mathbb{R}^d$ be a trained L-layered* GNN *on $\mathcal{G}$. When an L-layered* GRAFENNE *is trained on $\mathcal{G}^{alt}$ with Ψ in Phase-2 of message passing to produce $\Psi_{alt}(v) : \mathcal{V} \rightarrow \mathbb{R}^d$, $\forall\, v_1, v_2 \in \mathcal{V}$ if $\Psi_{\mathcal{G}}(v_1) \neq \Psi_{\mathcal{G}}(v_2)$ then representations produced by* GRAFENNE *are different as well i.e. $\Psi_{\mathcal{G}}(v_1) \neq \Psi_{\mathcal{G}}(v_2) \rightarrow \Psi_{alt}(v_1) \neq \Psi_{alt}(v_2)\ \forall\, v_1, v_2 \in \mathcal{V}$.*

Proof: See App. C. □.

Corollary 1. GRAFENNE is as expressive as k-WL.

PROOF: k-GNN (Morris et al., 2019) is as expressive as k-WL on static graphs. Therefore, it follows from Thm. 1, if k-GNN is used in Phase-2, GRAFENNE is also as expressive as k-WL. □.

Next, we establish that the neural architecture of GRAFENNE is expressive enough to recover the node feature vectors of the original space from the allotropic graph representation.

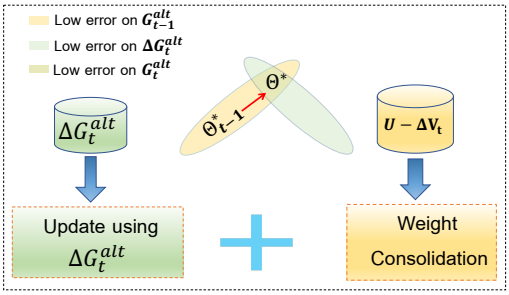


Figure 3: Architecture diagram for updating GRAFENNE at time t in a continual fashion. Θ^*_{t-1} is the optimal parameter space for $\mathcal{G}^{alt}_{t-1}$. Θ^* represents the optimal parameter space for $\mathcal{G}^{alt}_t$. Since our proposed feature streaming scenario is different from task or class incremental learning, therefore the concept of forgetting in our case is not task-specific but is associated to the unaffected portion of the graph at any given time t.

***Theorem** 2. Let v be a node characterized by a d-dimensional feature vector $\mathbf{x}_v = [x_1, x_2, \cdots, x_d]$ in the original graph $\mathcal{G}$. Thus, in the allotropic graph $\mathcal{G}^{alt}$, v is connected to d feature nodes with edge weight x_i when connecting to feature node corresponding to dimension i. We show that* GRAFENNE *can recover the original feature vector $\mathbf{x}_v$ from the allotropic graph $\mathcal{G}^{alt}$ in Phase 1.*

Proof. Refer to App. D. □

This result is important since it shows that for Phase 2, GRAFENNE would have the same level of information that its base GNN would have if operating on the original graph.

4.3.3. COMPLEXITY OF GRAFENNE

In most GNNs such as GRAPHSAGE, GAT and GIN, time and space computational complexity for embedding generation of each node is bounded by $O\left(\prod_{\ell=1}^{L} S_\ell\right)$ (Hamilton et al., 2017) where L is no. of layers in GNN and S_ℓ is no. of sampled neighbors at each level of the computation graph. In practise, $S_\ell \leq K$ is used where K is a small integer.

In GRAFENNE, this bound increases to $O\left(\prod_{\ell=1}^{L} (S_\ell \times \mid F \mid + \mid \mathcal{V} \mid \times \mid F \mid)\right)$ due to the 3-stage message passing layer. In practice, as in GRAPHSAGE, one could sample nodes and features to reduce the complexity. We elaborate on these implementation strategies and optimization to exploit sparsity in feature space in App. E.

4.4. Continual Learning Framework with GRAFENNE

In this section, we discuss the adaptation of GRAFENNE to learn on graphs with streaming updates, i.e., $\overrightarrow{\mathcal{G}}^{alt} = \left(\mathcal{G}_1^{alt}, \mathcal{G}_2^{alt}, \ldots\right)$ corresponding to the input streaming graph $\overrightarrow{\mathcal{G}} = (\mathcal{G}_1, \mathcal{G}_2, \ldots)$ as described in Def. 3. As the graph is updated, the parameters of GRAFENNE also need to be updated to capture the changes in the graph. Our goal is to search for model parameters that fit on the updated portion of the graph while also not forgetting the patterns learned on the unaltered portion. Towards that objective, we aim to learn to adjust the magnitude of the parameter updates at time t on certain model weights based on how important they are to the unaffected graph $\mathcal{G}_t^{alt} - \Delta G_t^{alt}$. We achieve this through *Elastic Weight Consolidation* (EWC) (Wang et al., 2020a; Kirkpatrick et al., 2017). Specifically, GRAFENNE penalizes significant changes to parameters that are important for the unaltered graph. Fig. 3 illustrates *Elastic Weight Consolidation* for our streaming feature scenario. The specifics of this component are discussed in App.F.

5. Experiments

In this section, we examine the effectiveness of GRAFENNE wrt. **(1)** Robustness to different missing feature rates **(2)** Adaptation to different GNN architectures **(3)** Ablation study, and **(4)** Performance on continual learning. Details of the experimental setup in terms of hardware and software framework, train-test splits, default parameter value, etc., are listed in App. G. Our codebase is available at `https://github.com/data-iitd/Grafenne`.

5.1. Datasets

We evaluate GRAFENNE on the real-world graphs listed in Table 1. Among these, Actor is a heterophilic graph, whereas the rest are homophilic. Further details on the semantics of the datasets are provided in App. H.

5.2. Baselines

To deal with missing features, we consider five different feature imputation strategies namely: **(1)** GCNMF (Taguchi et al., 2021), **(2)** PAGNN (Jiang & Zhang, 2020), **(3)** FP (Rossi et al., 2021), **(4)** Marking missing features with a special label, **(5)** Imputing based on the mean of the neighborhood values. Among the above strategies, GCNMF and PAGNN propose their own GNNs. In contrast, the other three algorithms are all pre-processing methods and therefore can be integrated with any GNN of choice. As the base GNN architecture, we consider GRAPHSAGE (Hamilton et al., 2017), GIN (Xu et al., 2019), GAT (Veličković et al., 2018). GCNMF, PAGNN, and FP are all transductive,

Table 1: Dataset statistics

Dataset	# Nodes	# Edges	# Features	#Labels
Cora (Sen et al., 2008)	2708	10556	1433	7
CiteSeer (Yang et al., 2016)	3327	9104	3703	6
Physics (Shchur et al., 2018)	34493	495924	8415	5
Actor (Pei et al., 2020)	7600	33544	931	5

while the other two are inductive. We also compare with FATE (Wu et al., 2021), which is an algorithm for feature adaptation. We show that feature adaptation methods, when adapted for graphs, are not adequate. A detailed differentiation in methodology and empirical comparison is provided in App. L.

For the pre-processing based methods, we use the notations "FP +⟨GNN⟩" and "NM +⟨GNN⟩" for FP and neighborhood mean respectively. If we only use the name of the GNN, then it indicates imputation with a special label for missing value.

5.3. Tasks

GRAFENNE is generic enough to accommodate any of the standard predictive tasks on graphs. We choose two of the most popular tasks of *node classification* and *link prediction* to benchmark GRAFENNE and the baselines. As per standard practice (Hamilton et al., 2017), for node classification, we quantify performance in terms of accuracy, i.e., the percentage of correct predictions, and for link prediction, we use area under the receiver operating curve (AUCROC).

5.4. Empirical Evaluation

First, we evaluate on static graphs with heterogeneous features sets. Next, we evaluate performance on streaming graphs. Finally, we perform ablation studies. Since all of the pre-processing features require a base GNN, we primarily use GRAPHSAGE as the GNN of choice. To ensure a fair comparison, we also use GRAPHSAGE as the message passing scheme in Phase-2 of GRAFENNE. Nonetheless, for the sake of completeness, we also present results when GRAPHSAGE is replaced with GAT and GIN. To measure the impact of missing features, we take datasets with complete features, and randomly delete p portion (ratio) of the features per node. p is varied across various values. This strategy is consistent with evaluation methodology of our baselines PAGNN,GCNMF, and FP.

5.4.1. STATIC GRAPHS

Table 2 show the results on node classification at multiple feature missing rates. A similar table for link prediction is provided in Table 4. We observe that GRAFENNE outperforms baseline methods on a diverse range of missing rates. Especially on higher missing rates, we observe a performance gap of more than 10% between GRAFENNE and the best baseline method. Further, on dataset Actor, we observe that GRAFENNE obtains significantly better accuracy gain of over 5% on all missing rates. Interestingly, we also observe that the performance of FP and mean-neighborhood (NM) methods can be significantly improved when complemented with GRAFENNE's message passing framework, i.e., NM +GRAFENNE and FP +GRAFENNE. We note that FP is a transductive feature imputation method that requires

Table 2: Accuracy of GRAFENNE and baselines on node classification at various missing rates p. Std. dev. values $<$ 0.01 are approximated to 0.

Dataset	Method	$p = 0$	$p = 0.5$	$p = 0.99$
Cora	GRAPHSAGE	83.80 ± 0.48	83.06 ± 0.62	72.58 ± 0.71
	GCNMF	80.07 ± 0.0	67.52 ± 0.0	33.02 ± 0.0
	PAGNN	82.47 ± 0.0	84.68 ± 0.0	67.89 ± 0.0
	NM + GRAPHSAGE	-	83.46 ± 0.36	78.48 ± 0.54
	FP + GRAPHSAGE	-	83.72 ± 0.53	81.25 ± 0.44
	GRAFENNE	$\mathbf{87.6 \pm 0.73}$	84.35 ± 0.27	78.85 ± 0.29
	NM + GRAFENNE	-	85.05 ± 0.36	78.78 ± 0.62
	FP + GRAFENNE	-	$\mathbf{85.46 \pm 0.21}$	$\mathbf{82.91 \pm 0.92}$
CiteSeer	GRAPHSAGE	73.43 ± 0.97	70.85 ± 0.35	57.14 ± 0.96
	GCNMF	71.47 ± 0.0	60.36 ± 0.0	23.12 ± 0.0
	PAGNN	73.57 ± 0.0	72.82 ± 0.0	58.70 ± 0.0
	NM + GRAPHSAGE	-	70.96 ± 0.45	61.80 ± 0.46
	FP + GRAPHSAGE	-	71.02 ± 0.65	$\mathbf{65.25 \pm 1.08}$
	GRAFENNE	$\mathbf{73.90 \pm 0.84}$	72.91 ± 0.95	64.08 ± 0.79
	NM + GRAFENNE	-	72.88 ± 0.55	63.03 ± 0.93
	FP + GRAFENNE	-	$\mathbf{74.20 \pm 0.40}$	64.64 ± 0.6
Actor	GRAPHSAGE	32.90 ± 0.79	30.61 ± 0.86	22.90 ± 0.50
	GCNMF	24.53 ± 0.0	23.75 ± 0.0	21.57 ± 0.0
	PAGNN	23.81 ± 0.0	23.81 ± 0.0	$\mathbf{25.39 \pm 0.0}$
	NM + GRAPHSAGE	-	29.27 ± 0.69	21.73 ± 0.07
	FP + GRAPHSAGE	-	29.15 ± 0.79	23.89 ± 1.03
	GRAFENNE	$\mathbf{38.90 \pm 0.84}$	$\mathbf{35.02 \pm 0.21}$	23.97 ± 0.58
	NM + GRAFENNE	-	32.03 ± 0.70	24.01 ± 0.80
	FP + GRAFENNE	-	32.76 ± 1.06	24.02 ± 0.40

re-training in cases of unseen nodes or new features, making GRAFENNE an attractive alternative in streaming graphs. These results are a direct consequence of the nature of propagation introduced by our proposed method. We also evaluate GRAFENNE on extremely high missing rates such as $p = 0.99999$ on large-scale dataset physics in table 5 where see GRAFENNE performs exceptionally well indicating its application in settings where the nominal amount of data is shared by very few users.

In Tables 3 and G (in appendix), we investigate the impact of the GNN used in Phase-2 of GRAFENNE's message passing scheme. Towards that, GRAPHSAGE is replaced with GAT and GIN. We observe that regardless of the GNN, when empowered within the GRAFENNE framework, an

Table 3: Accuracy of GRAFENNE (GAT) and GRAFENNE (GIN) with benchmark GNNs, GAT and GIN.

Dataset	Method	$p = 0$	$p = 0.5$	$p = 0.99$
Cora	GAT	$\mathbf{86.10 \pm 0.7}$	82.50 ± 0.96	73.72 ± 0.57
	GRAFENNE (GAT)	85.97 ± 0.53	$\mathbf{85.16 \pm 0.63}$	$\mathbf{79.74 \pm 0.51}$
	GIN	85.09 ± 0.92	82.91 ± 0.89	73.28 ± 0.35
	GRAFENNE (GIN)	$\mathbf{85.94 \pm 0.39}$	$\mathbf{84.25 \pm 0.64}$	$\mathbf{82.36 \pm 1.17}$
CiteSeer	GAT	71.59 ± 0.85	69.15 ± 0.91	59.21 ± 0.73
	GRAFENNE (GAT)	$\mathbf{73.21 \pm 0.33}$	$\mathbf{72.64 \pm 0.76}$	$\mathbf{64.29 \pm 0.61}$
	GIN	72.16 ± 0.58	69.84 ± 1.10	60.15 ± 1.31
	GRAFENNE (GIN)	$\mathbf{73.45 \pm 1.04}$	$\mathbf{72.58 \pm 0.59}$	$\mathbf{64.32 \pm 1.15}$
Actor	GAT	26.68 ± 0.92	25.92 ± 0.49	$\mathbf{24.69 \pm 1.55}$
	GRAFENNE (GAT)	$\mathbf{33.86 \pm 0.88}$	$\mathbf{32.07 \pm 0.67}$	24.23 ± 0.4
	GIN	26.93 ± 0.76	26.48 ± 1.36	22.98 ± 0.56
	GRAFENNE (GIN)	$\mathbf{29.13 \pm 0.91}$	$\mathbf{29.15 \pm 1.36}$	$\mathbf{24.03 \pm 0.5}$

improvement is observed. Interestingly, even when almost all features are available ($p = 0$), for the majority of the cases, GRAFENNE outperforms solely using a GNN on the original graph.

Table 4: AUCROC of GRAFENNE and baselines on link prediction task . Note that we have not included GCNMF and PAGNN as their code adapted for the link prediction task is not generalizing on test data

Dataset	Method	$p = 0$	$p = 0.5$	$p = 0.99$
Cora	GRAPHSAGE	0.86 ± 0.002	0.84 ± 0.002	0.7523 ± 0.05
	NM + GRAPHSAGE	-	0.8687 ± 0.0008	0.8297 ± 0.0014
	FP + GRAPHSAGE	-	0.8773 ± 0.0015	0.9137 ± 0.0008
	GRAFENNE	$\mathbf{0.8780 \pm 0.004}$	0.8501 ± 0.005	0.8015 ± 0.006
	NM + GRAFENNE	-	0.9009 ± 0.0026	0.8632 ± 0.0007
	FP + GRAFENNE	-	$\mathbf{0.9344 \pm 0.0020}$	$\mathbf{0.9263 \pm 0.0012}$
CiteSeer	GRAPHSAGE	0.8251 ± 0.005	0.7617 ± 0.001	0.7223 ± 0.002
	NM + GRAPHSAGE	-	0.8213 ± 0.0018	0.8001 ± 0.0019
	FP + GRAPHSAGE	-	0.8506 ± 0.0014	0.8875 ± 0.0011
	GRAFENNE	$\mathbf{0.8681 \pm 0.0010}$	0.8047 ± 0.010	0.7378 ± 0.008
	NM + GRAFENNE	-	0.8944 ± 0.0067	0.8462 ± 0.0024
	FP + GRAFENNE	-	$\mathbf{0.9348 \pm 0.0024}$	$\mathbf{0.9012 \pm 0.0030}$
Actor	GRAPHSAGE	0.6569 ± 0.0013	0.7029 ± 0.0026	0.6969 ± 0.003
	NM + GRAPHSAGE	-	0.7473 ± 0.0004	0.6885 ± 0.0001
	FP + GRAPHSAGE	-	0.7552 ± 0.0003	0.7721 ± 0.0017
	GRAFENNE	$\mathbf{0.7047 \pm 0.0050}$	0.7021 ± 0.005	0.7029 ± 0.007
	NM + GRAFENNE	-	0.7562 ± 0.0032	0.7169 ± 0.0012
	FP + GRAFENNE	-	$\mathbf{0.7801 \pm 0.0026}$	$\mathbf{0.7869 \pm 0.0013}$

5.4.2. CONTINUAL LEARNING WITH GRAFENNE

Setup: To evaluate GRAFENNE for continual learning, we evaluate on dynamic graphs, where features get added/deleted over time and graph structure also changes over time.

- **Feature Addition and Deletion:** A subset of features for a subset of nodes is added or deleted at each timestamp. We first sample nodes with probability p_n. For each selected node, we randomly select features for addition/deletion. The probability of a feature getting added or deleted to a node is p_f^{add} and p_f^{del} respectively.
- **Edges:** A subset of edges in the graph are added/deleted over time. The probability of an edge getting selected for deletion is p_e^{del} and the probability of an edge getting added to the graph is p_e^{add}.

In our experiments, for Cora and CiteSeer we set $p_n = 0.03$, $p_f^{add} = 0.05, p_f^{del} = 0.4, p_e^{del} = p_e^{add} = 0.0005$ and $T = 9$. For Physics dataset we set $p_n = 0.003, p_f^{del} = 0.8, p_f^{add} = 0.0001, p_e^{del} = p_e^{add} = 0.00005$.

Real-world dynamic dataset: Additionally, we extract streaming DBLP dataset (Tang et al., 2008) between 1992 to 1997 where nodes and edges get added over time. This dataset has 3075 nodes, 6368 edges and 5 classes. We set $p_n = 0.05, p_f^{del} = 0.1, p_f^{add} = 0.05$.

We set λ defined in Eq. 23 to 100000. We set $|U| = 300$ for DBLP and $|U| = 25$ for other datasets(§ 4.4).

Baselines: Given a graph stream $\overrightarrow{\mathcal{G}} = (\mathcal{G}_1, \mathcal{G}_2 \ldots, \mathcal{G}_T)$, we compare GRAFENNE for continual learning with:

(1) ORACLE: Retraining from scratch: ORACLE discards existing parameters and retrains GRAFENNE from scratch on $\mathcal{G}_t$ resulting in optimal parameters Θ_t for $\mathcal{G}_t$. This method provides the upper bound of achievable performance.

(2) FT: Fine-tuning: We update the parameters using *only* the affected nodes in $\mathcal{G}_t$, i.e., $\Delta\mathcal{V}_t$.

(3) ER: Experience Replay: Here, in addition to FT, we preserve a small sample of past nodes in memory and replay them when training on $\mathcal{G}_t$ to avoid forgetting of past patterns.

(4) ContGNN(Wang et al., 2020b): Preserves past knowledge through a combination of experience replay and weight regularization.

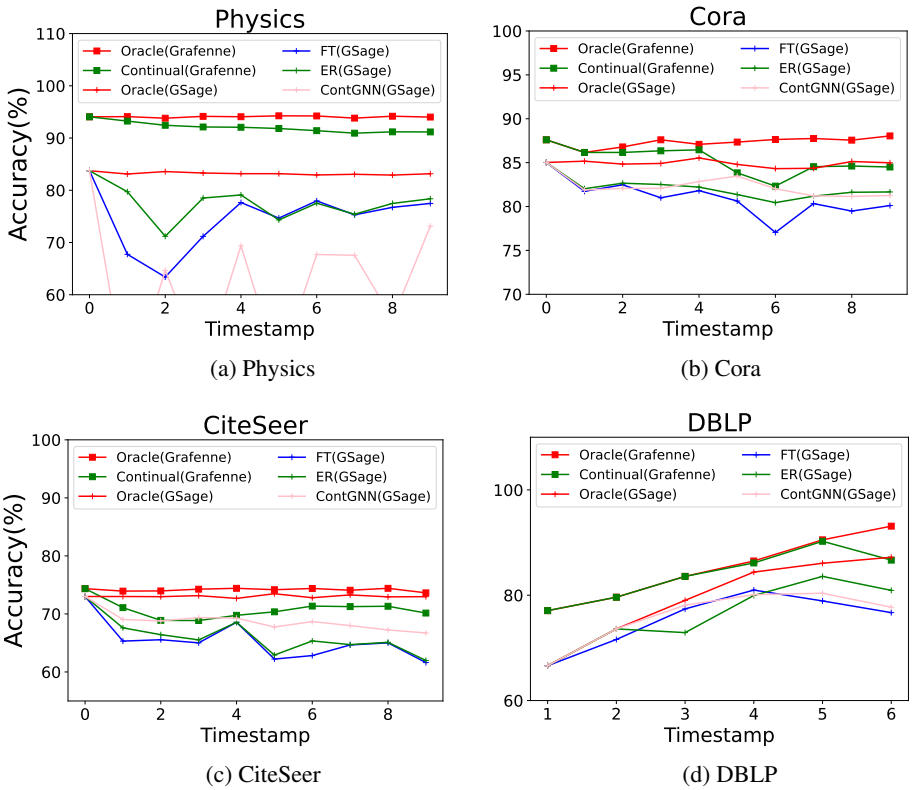
Results on Continual Learning Scenario: In Fig. 4 we compare the performance of proposed continual method for training GRAFENNE along with Oracle, FT, ER and ContGNN(Wang et al., 2020b) methods. We report test performance on the entire graph at each timestamp for every method. In Fig. 4, we observe that the continual method on GRAFENNE can maintain significantly higher accuracy on the entire test-set compared to other methods. The fine-tuning method only updates the affected portion of the graph, hence it suffers from catastrophic forgetting on the unaffected nodes, which is evident from the accuracy metric on the test nodes from the whole graph. Methods such as ER and ContGNN preserve past knowledge by employing a small set of memory for replay. However, they do not cater to the unseen feature scenario, hence overall performance on these methods deteriorates over time. On the other hand, GRAFENNE coupled with elastic weight consolidation on an unaffected portion of graph reduces the extent of catastrophic forgetting, hence achieves superior performance. Additionally, in the case of large-size datasets i.e Physics, we observe that GRAFENNE significantly outperforms existing methods on all timestamps showing better scalability.

6. Conclusion

Graph Neural Networks have shown significant performance gains on graph-structured data. However existing works mostly focused on graphs with an identical set of available node features. Moreover, existing state-of-the-art imputation techniques to tackle scenario of dissimilar node features are transductive in nature. In this work, we proposed a novel inductive method GRAFENNE that can learn on graphs having nodes with heterogeneous features. In addition to this, we also formulated a novel problem of lifelong learning on graphs with streaming features. Further, to solve this problem, we proposed *elastic weight consolidation* based continual learning method for training GRAFENNE on dynamic graphs. Through extensive evaluation on 4 real-world

Table 5: Node classification performance comparison at extreme missing rates p on large-scale Physics dataset. We report the mean classification accuracy (%) along with the standard deviation on five runs. Std. dev. values < 0.01 are approximated to 0. If a baseline produces an error during execution, we denote it as $*$.

Method	$p = 0$	$p = 0.5$	$p = 0.99$	$p = 0.999$	$p = 0.9999$	$p = 0.99999$
GRAPHSAGE	96.91 ± 0.05	96.29 ± 0.17	92.92 ± 0.11	84.11 ± 0.08	61.05 ± 0.27	52.11 ± 0.04
GCNMF	92.52 ± 0.0	81.12 ± 0.0	50.4856 ± 0.0	51.45 ± 0.0	$*$	$*$
PAGNN	94.28 ± 0.0	93.88 ± 0.0	88.09 ± 0.0	75.44 ± 0.0	60.71 ± 0.0	51.35 ± 0.0
NM + GRAPHSAGE	–	96.08 ± 0.11	94.47 ± 0.14	92.35 ± 0.06	82.96 ± 0.07	57.48 ± 0.01
FP + GRAPHSAGE	–	96.41 ± 0.09	95.04 ± 0.0	94.34 ± 0.15	93.07 ± 0.22	78.44 ± 0.71
GRAFENNE	$\mathbf{97.02 \pm 0.05}$	96.23 ± 0.23	94.49 ± 0.18	94.31 ± 0.19	94.11 ± 0.21	89.49 ± 0.15
NM + GRAFENNE	–	95.82 ± 0.12	94.70 ± 0.18	$\mathbf{94.61 \pm 0.0}$	94.57 ± 0.19	89.56 ± 0.06
FP + GRAFENNE	–	$\mathbf{96.36 \pm 0.08}$	$\mathbf{95.02 \pm 0.05}$	94.57 ± 0.15	$\mathbf{95.33 \pm 0.21}$	$\mathbf{93.17 \pm 0.16}$



(a) Physics (b) Cora

(c) CiteSeer (d) DBLP

Figure 4: Continual learning performance on Physics, Cora, CiteSeer and DBLP. The x-axis represents the timestamps of graph updates and the y-axis represents the test accuracy(%) corresponding to each timestamp.

datasets, we established that GRAFENNE achieves superior performance against baseline approaches at various feature missing rates p and is also robust at extremely high missing feature rates e.g. $p = 0.99999$. Furthermore, we highlight the capability of GRAFENNE to integrate with different existing inductive GNN architectures and show significant performance gains. Additionally, GRAFENNE achieves high-quality results in the streaming scenario and hence shows its ability to learn effectively in the lifelong learning setup. In terms of future work, it will be interesting to explore GRAFENNE on graphs having inter-feature relations allowing the creation of feature-feature edges in $\mathcal{G}^{alt}$.

7. Acknowledgement

Shubham Gupta acknowledges Info Edge (India) Limited for supporting his Ph.D. Sahil Manchanda acknowledges Qualcomm for supporting him through Qualcomm Innovation Fellowship and he also acknowledges GP Goyal Alumni Grant of IIT Delhi for supporting this travel. Sayan Ranu acknowledges the Nick McKeown chair position endowment. Srikanta Bedathur was partially supported by DS Chair Professor of AI grant and an IBM AI Horizons Network (AIHN) grant.

References

Bai, T., Nie, J.-Y., Zhao, W. X., Zhu, Y., Du, P., and Wen, J.-R. *An Attribute-Aware Neural Attentive Model for Next Basket Recommendation*, pp. 1201–1204. Association for Computing Machinery, New York, NY, USA, 2018. ISBN 9781450356572. URL https://doi.org/10.1145/3209978.3210129.

Bhattoo, R., Ranu, S., and Krishnan, N. Learning articulated rigid body dynamics with lagrangian graph neural network. *Advances in Neural Information Processing Systems*, 35:29789–29800, 2022.

Bhattoo, R., Ranu, S., and Krishnan, N. A. Learning the dynamics of particle-based systems with lagrangian graph neural networks. *Machine Learning: Science and Technology*, 2023.

Bihani, V., Manchanda, S., Sastry, S., Ranu, S., and Krishnan, N. Stridernet: A graph reinforcement learning approach to optimize atomic structures on rough energy landscapes. In *ICML*, 2023.

Bishnoi, S., Bhattoo, R., Ranu, S., and Krishnan, N. Enhancing the inductive biases of graph neural ode for modeling dynamical systems. *ICLR*, 2023.

Chakraborty, P., Ranu, S., Mantri, K. S. I., and De, A. Learning and maximizing influence in social networks under capacity constraints. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, pp. 733–741, 2023.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Goyal, N., Jain, H. V., and Ranu, S. Graphgen: a scalable approach to domain-agnostic labeled graph generation. In *Proceedings of The Web Conference 2020*, pp. 1253–1263, 2020.

Gupta, M., Kodamana, H., and Ranu, S. FRIGATE: Frugal spatio-temporal forecasting on road networks. In *29th SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023. URL https://openreview.net/forum?id=2cTw2M47L1.

Gupta, S., Manchanda, S., Bedathur, S., and Ranu, S. Tigger: Scalable generative modelling for temporal interaction graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 6819–6828, 2022.

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

Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Jain, J., Bagadia, V., Manchanda, S., and Ranu, S. Neuromlr: Robust & reliable route recommendation on road networks. *Advances in Neural Information Processing Systems*, 34:22070–22082, 2021.

Jiang, B. and Zhang, Z. Incomplete graph representation and learning via partial graph neural networks, 2020. URL https://arxiv.org/abs/2003.10130.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 427–431. Association for Computational Linguistics, April 2017.

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

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Leskovec, J. and Krevl, A. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, 2014. Accessed: 2022-03-18.

Liu, H., Yang, Y., and Wang, X. Overcoming catastrophic forgetting in graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 8653–8661, 2021.

Manchanda, S. and Ranu, S. Lifelong learning for neural powered mixed integer programming. *AAAI*, 2023.

Manchanda, S., Mittal, A., Dhawan, A., Medya, S., Ranu, S., and Singh, A. Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs. *Advances in Neural Information Processing Systems*, 33:20000–20011, 2020.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33014602. URL https://doi.org/10.1609/aaai.v33i01.33014602.

Nishad, S., Agarwal, S., Bhattacharya, A., and Ranu, S. Graphreach: Position-aware graph neural network using reachability estimations. *IJCAI*, 2021.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

Pei, H., Wei, B., Chang, K. C., Lei, Y., and Yang, B. Geom-gcn: Geometric graph convolutional networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=S1e2agrFvS.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. Recipe for a General, Powerful, Scalable Graph Transformer. *Advances in Neural Information Processing Systems*, 35, 2022.

Ranjan, R., Grover, S., Medya, S., Chakaravarthy, V., Sabharwal, Y., and Ranu, S. Greed: A neural framework for learning graph distance functions. In *Advances in Neural Information Processing Systems*, 2022.

Rossi, E., Kenlay, H., Gorinova, M. I., Chamberlain, B. P., Dong, X., and Bronstein, M. On the unreasonable effectiveness of feature propagation in learning on graphs with missing node features, 2021. URL https://arxiv.org/abs/2111.12128.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI Magazine*, 29 (3):93, Sep. 2008. doi: 10.1609/aimag.v29i3.2157. URL https://ojs.aaai.org/index.php/aimagazine/article/view/2157.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Taguchi, H., Liu, X., and Murata, T. Graph convolutional networks for graphs containing missing features. *Future Generation Computer Systems*, 117:155 – 168, 2021.

Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., and Su, Z. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 990–998, 2008.

Thangamuthu, A., Kumar, G., Bishnoi, S., Bhattoo, R., Krishnan, N., and Ranu, S. Unravelling the performance of physics-informed graph neural networks for dynamical systems. In *Advances in Neural Information Processing Systems*, 2022.

Vasile, F., Smirnova, E., and Conneau, A. Meta-prod2vec: Product embeddings using side-information for recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pp. 225–232, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340359. doi: 10.1145/2959100.2959160. URL https://doi.org/10.1145/2959100.2959160.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Vignac, C., Krawczuk, I., Siraudin, A., Wang, B., Cevher, V., and Frossard, P. Digress: Discrete denoising diffusion for graph generation. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=UaAD-Nu86WX.

Wang, C., Qiu, Y., Gao, D., and Scherer, S. Lifelong graph learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13719–13728, 2022.

Wang, J., Song, G., Wu, Y., and Wang, L. Streaming graph neural networks via continual learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, pp. 1515–1524, New York, NY, USA, 2020a. Association for Computing Machinery. ISBN 9781450368599. doi: 10.1145/3340531.3411963. URL https://doi.org/10.1145/3340531.3411963.

Wang, J., Song, G., Wu, Y., and Wang, L. Streaming graph neural networks via continual learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 1515–1524, 2020b.

Wu, Q., Yang, C., and Yan, J. Towards open-world feature extrapolation: An inductive graph learning approach. *Advances in Neural Information Processing Systems*, 34: 19435–19447, 2021.

Xu, D., Ruan, C., Korpeoglu, E., Kumar, S., and Achan, K. Theoretical understandings of product embedding for e-commerce machine learning. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, WSDM '21, pp. 256–264, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450382977. doi: 10.1145/3437963.3441736. URL https://doi.org/10.1145/3437963.3441736.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference*

on Learning Representations, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 40–48. JMLR.org, 2016.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. Do transformers really perform badly for graph representation? In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=OeWooOxFwDa.

You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. Graphrnn: Generating realistic graphs with deep auto-regressive models. In Dy, J. G. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5694–5703. PMLR, 2018. URL http://proceedings.mlr.press/v80/you18a.html.

You, J., Ying, R., and Leskovec, J. Position-aware graph neural networks. In *International conference on machine learning*, pp. 7134–7143. PMLR, 2019.

A. Appendix

A. Phase 2 Specifics of Message Passing

Phase 2 is a message-passing layer among graph nodes $\mathcal{V}$. GRAFENNE adopts the message passing implementation of GRAPHSAGE, which is defined as follows.

GRAFENNE:

$$\mathbf{h}_v^\ell = \sigma\left(\mathbf{W}_{13}^{\ell^T}\left(\mathbf{h}_v^l \Big|\Big| \frac{1}{\mid \mathcal{N}_v^{feat} \mid} \sum_{u \in \mathcal{N}_v^{feat}} \mathbf{h}_u^l\right)\right) \quad \forall v \in \mathcal{V} \tag{19}$$

The message passing layer of GRAFENNE is flexible enough to accommodate other architectures as well such as GAT (Veličković et al., 2018) and GIN (Xu et al., 2019).

GRAFENNE (GAT): $\forall v \in \mathcal{V}$

$$\begin{aligned}
\mathbf{m}_v^\ell(u) =& \text{LEAKYRELU}\left(\mathbf{W}_{13}^\ell \mathbf{h}_v^l \Big|\Big| \mathbf{W}_{14}^\ell \mathbf{h}_u^l\right) \quad \forall u \in \mathcal{N}_v^{\mathcal{G}} \cup v, \\
\alpha_{vu} =& \frac{\exp\left(\mathbf{w}_{15}^{\ell\,T} \mathbf{m}_v^\ell(u)\right)}{\sum_{u' \in \mathcal{N}_v^{\mathcal{G}} \cup v} \exp\left(\mathbf{w}_{15}^{\ell\,T} \mathbf{m}_v^\ell(u')\right)}, \\
\mathbf{h}_v^\ell =& \sum_{u \in \mathcal{N}_v^{\mathcal{G}} \cup v} \alpha_{vu} \mathbf{W}_{16}^\ell \mathbf{h}_u^\ell
\end{aligned} \tag{20}$$

GRAFENNE (GIN):

$$\mathbf{h}_v^\ell = \text{MLP}\left((1+\epsilon)\,\mathbf{h}_v^\ell + \sum_{u \in \mathcal{N}^G(v)} \mathbf{h}_u^\ell\right) \quad \forall v \in \mathcal{V} \tag{21}$$

All weights matrices and vectors of the form $\mathbf{W}_i^\ell$ and $\mathbf{w}_i^\ell$ are trainable parameters; ϵ is a hyper-parameter.

B. Inductive Analysis of GRAFENNE

We first define an input test graph $\mathcal{G}_{test} = (\mathcal{V}_{test}, \mathcal{E}_{test}, \mathbf{X}_{test})$, which is an updated graph of input training graph $\mathcal{G}$ i.e. $\mathcal{G}_{test} = \mathcal{G} + \nabla\mathcal{G}$ where $\nabla\mathcal{G} = (\nabla\mathcal{V}, \nabla\mathcal{E}, \nabla\mathbf{X})$. Similar to def. 2, where we define F as set of available features in graph $\mathcal{G}$, we also define feature set $F_{test} = \bigcup_{v \in \mathcal{V}_{test}} F_v$ on G_{test} where F_v is set of features available at node v. We also assume an unknown feature super-set $\mathcal{F}$ where $F, F_{test} \subseteq \mathcal{F}$. Now, in lieu of proposition 1, we describe and prove the following theorem 3.

Theorem 3 (Inductivity of GRAFENNE). *Let $\Psi_{\mathcal{G}}(v) : \mathcal{V} \to \mathbb{R}^d$ be a trained L-layered* GNN *on a graph $\mathcal{G}$ and $\Psi_{alt}(v) : \mathcal{V} \to \mathbb{R}^d$ be a L-layered* GRAFENNE *trained on $\mathcal{G}^{alt}$ with Ψ in Phase-2 of message passing. Given a test graph $\mathcal{G}_{test}$,* GRAFENNE *Ψ_{alt} can generalize to unseen nodes with unseen features in $\mathcal{G}_{test}^{alt}$ i.e. $\Psi_{alt}(v) \approx Y(v)$ even $if\ \exists f \in F_v \wedge f \notin F,\ \forall v \in \mathcal{V}_{test} - \mathcal{V}$. This holds true given that the following conditions hold.*

1. *Ψ is a node-inductive* GNN *i.e. it is able to generalize to unseen nodes albeit with seen features.*
2. *A feature embedding space $\mathcal{Z} \subseteq \mathbb{R}^d$ which reflects the semantic/statistical relationships among vectors and an embedding function over categorical variables $\Phi(f) : \mathcal{F} \to \mathcal{Z}$ which can map any seen or unseen feature to this embedding space $\mathcal{Z}$ i.e., $\Phi(f) \in \mathcal{Z},\ \forall f \in F_{test} - F$.*

Proof: Condition-1 always holds, as GRAFENNE adopts the node inductive GNN Ψ in Step-2 as seen in eq. 8, 9 and 10 which are independent of no. of graph nodes $|\mathcal{V}|$ in G^{alt}. All graph nodes $\mathcal{V}$ are initialized with **0** and assuming that condition-2 holds true, all feature nodes $\mathcal{V}^{feat}$ are initialized with vectors in embedding space $\mathcal{Z}$. Combining this and the fact that eq. 5, 6, 7, 11, 12 and 13 are independent of both no. of nodes V and no. of features F, makes GRAFENNE both node-inductive and feature-inductive. □.

Remark on condition-2: Node attributes are usually composed of bag-of-words in citation graphs, product categories in e-commerce graphs, and medical diagnoses in case of health-care-related graphs. In such cases, a categorical transformation function Φ can be learned in the pre-processing stage. For eg., word-embedding methods (Joulin et al., 2017) for the bag of words features, language models(Devlin et al., 2018) on the textual description of items categories or healthcare-related

categories. There also exist specialized product category encoding methods (Xu et al., 2021; Vasile et al., 2016) which utilize skip-gram model on product sequences generated from user sessions.

C. Expressive Power of GRAFENNE– Proof of Thm. 1

Similar to other works on analyzing the expressive power of GNNs, we assume that the input feature space $\mathcal{X}$ is countable. Since proposed graph transformation only impacts the features and their information flow during message-passing between graph nodes $\mathcal{V}$ in $\mathcal{G}^{alt}$, it is sufficient to show that GRAFENNE produces distinct feature representation for nodes having different features in Phase-1, i.e., after Phase-1 $\mathbf{h}_v^\ell(v_3) \neq \mathbf{h}_v^\ell(v_4)\ \forall\ \ell \in [1 \ldots L], v_3, v_4 \in \mathcal{V},\ if \exists\ \mathbf{x}_{v_3} \neq \mathbf{x}_{v_4}$. If this holds true, and since Phase-2 GNN utilizes $\Psi_{\mathcal{G}}$ itself, Thm. 1 is proved. We now need to prove the following lemma:

Lemma 1. GRAFENNE Ψ_{alt} produces distinct transformation for graph nodes $\forall v \in \mathcal{V}$ in phase -1 given that their input feature vectors are different, i.e.,

$$\mathbf{h}_v^\ell(v_3) \neq \mathbf{h}_v^\ell(v_4)\ \forall\ \ell \in [1..L], v_3, v_4 \in \mathcal{V}\ if\ \exists\ \mathbf{x}_{v_3} \neq \mathbf{x}_{v_4} \tag{22}$$

This is true only if following conditions hold:

1. MSG^ℓ_{feat}, $\text{COMBINE}^\ell_{feat}$, $\text{MSG}^\ell_{\mathcal{G}'}$ and $\text{COMBINE}^\ell_{\mathcal{G}'}$ $\forall \ell \in [1 \ldots L]$ in equations 5, 7, 11 and 13 are universal function approximators such as MLPs(Hornik et al., 1989).
2. $\text{AGGREGATE}^\ell_{feat}$, $\text{AGGREGATE}^\ell_{\mathcal{G}'}$ should be injective aggregators over MULTISETS i.e. produce different representations of different MULTISETS.

If conditions 1 and 2 hold true and since each feature node is initialized with injective transformation in Eq. 15, for $l = 1$ we can see that $\mathbf{h}_v^1(v_3) \neq \mathbf{h}_v^1(v_4)\ \forall v_3, v_4 \in \mathcal{V}\ if\ \exists\ \mathbf{x}_{v_3} \neq \mathbf{x}_{v_4}$ after Phase-1 of GRAFENNE. For $l > 1$, after Phase-3, we see that $\mathbf{h}_v^\ell\ \forall v \in \mathcal{V}^{feat}$ will be distinct for all feature nodes due to assumptions 1 and 2. This follows to $\mathbf{h}_v^\ell(v_3) \neq \mathbf{h}_v^\ell(v_4)\ \forall v_3, v_4 \in \mathcal{V}\ if\ \exists\ \mathbf{x}_{v_3} \neq \mathbf{x}_{v_4}\ \forall \ell > 1$ after Phase-1. This concludes our analysis. □

D. Proof of Thm 2

Proof. As per Eq. 5, let $\mathbf{h}_i^0 \in \mathbb{R}^d$ the representation of feature node corresponding to dimension i, which is also learnable, be a one-hot encoding where dimension i is 1, and rest are 0. $\mathbf{h}_v^0$ is a zero-vector inconsequential to following analysis, and e_{iv} is the value corresponding to i^{th} dimension of $\mathbf{x}_v$. With these inputs, let us assume MSG^ℓ_{feat} is such that it computes messages of the form $\mathbf{m}_v^1(i) \in \mathbb{R}^{2d}$ where the first d dimensions are $\mathbf{h}_i^0$, the rest of the dimensions have value $\mathbf{x}_v[i]$, i.e., $\forall k : d+1 \leq k \leq 2d, \mathbf{m}_v^1(i)[k] = x_v[i]$. The learning task is, therefore, to learn the $\text{AGGREGATE}^\ell_{feat}$ function $f(\{\!\{\mathbf{m}_v^1(i)\}\!\}) = \mathbf{x}_v$, i.e., recover the original feature vector from the messages received from feature nodes in the allotropic graph. Examining the messages $\mathbf{m}_v^1(i)$ it is clear that, $\mathbf{x}_v[i] = \mathbf{m}_v^1(i)[i] \times \mathbf{m}_v^1(i)[d+i]$. From the universal approximation theorem, an MLP can learn this function; hence, GRAFENNE can recover the original feature space from the allotropic representation. □

E. Complexity Analysis

To reduce this computational burden, we bound the number of features by $S^{\mathcal{V}^{feat}}$ during message aggregation in Phase-1 (Eqs. 5 and 6). Similarly in Phase-3, we bound the number of graph nodes to compute the feature node embedding by $S^{\mathcal{V}}$. This leads to $O\left(\prod_{i=1}^{L}\left(S_i \times S^{\mathcal{V}^{feat}} + S^{\mathcal{V}} \times S^{\mathcal{V}^{feat}}\right)\right)$ bound on time and space complexities for generating embedding for every node. $S^{\mathcal{V}}$ is the no. of graph nodes for computing feature node embeddings and $S^{\mathcal{V}^{feat}}$ is the no. of feature nodes for computing graph node embeddings. In the datasets we have considered for evaluation, all nodes have large dimensional features, but they are highly sparse. For eg., in *Cora* out of 1433 features, on average 18 features have value 1 for all nodes. Similarly, all feature nodes on average have a value of 1 in 34 graph nodes. Thus, connecting graph nodes with only those feature nodes having value 1 and vice-versa results in a low-computational overhead. We perform sampling in case of large scale datasets eg. *Physics*, where each graph node has on average 34 features having value 1 out of 8415 features and each feature node has on average 135 graph nodes.

F. Extension to Continual Learning for Dynamic Graphs

To perform elastic weight consolidation, we randomly sample a small set of training nodes $U \in \mathcal{V}_{t_l}$ from the graph. Then, we compute the importance of model weights on the loss of $U - \Delta\mathcal{V}_{t_l}$ where $\Delta\mathcal{V}_{t_l}$ is the set of affected training nodes at

time t. Specifically,

$$\Omega_w = \mathbb{E}_{(v)\sim\left(U-\Delta\mathcal{V}_{t_l}\right)}\left[\left(\frac{\delta\mathcal{L}(v)}{\delta\Theta_w}\right)^2\right]$$

Here, Θ refers to the model parameters of GRAFENNE, Ω_w refers to importance of the wth weight parameter. The term $\frac{\delta\mathcal{L}(v)}{\delta\Theta_w}$ calculates the gradient of the loss on unaffected nodes with respect to the parameter w. When the parameter update is to take place with respect to the updated data $\Delta\mathcal{G}_t^{alt}$, we penalize updates to the weights that are important for the representative sample of nodes that were not updated in the latest timestamp t using Ω_w calculated above. We accomplish it by the below loss function.

$$\mathcal{L}_{cont} = \sum_{v\in\Delta\mathcal{V}_{t_l}} \mathcal{L}(v) + \sum_w \frac{\lambda}{2}\Omega_w\left(\Theta_t^w - \Theta_{t-1}^w\right)^2 \tag{23}$$

The first term refers to the loss computed on the updated set of training nodes in the current timestamp. The second term is a quadratic penalty term on the difference between the parameters for the new timestamp and the previous timestamp. We also observe that we only need to store the current model parameters and model parameters of previous phase $t-1$, as evident from Eq. 23. λ is a hyper-parameter reflecting how important the unaffected portion of the current graph $\mathcal{G}_t$ is compared to the updated portion of the graph.

G. Experimental Environment

All experiments are performed on an Intel Xeon Gold 6248 processor with 80 cores, 1 Tesla V-100 GPU card with 32GB GPU memory, and 377 GB RAM with Ubuntu 18.04. We perform a $60\%-20\%-20\%$ data split for train-test-validation. These splits are generated at random. In all experiments, we have used 2 layers of message-passing and trained GRAFENNE using the Adam optimizer with a learning rate of 0.0001 and choose the model based on the best validation loss. All experiments have been executed 5 times. We report the mean and standard deviations. Standard deviation below 0.01 have been approximated to 0 in node classification results.

H. Datasets

Cora (Yang et al., 2016), CiteSeer (Yang et al., 2016) and DBLP (Tang et al., 2008) are citation graphs where each node is a paper and the edge manifests a citation. The node labels represent the research category. In Cora and CiteSeer, the node attributes contain a bag of words of the paper text and in DBLP, the node attributes contain bag of words of keywords (Tang et al., 2008). We also use a large scale graph Physics (Shchur et al., 2018) to show the scalability capabilities of GRAFENNE. Physics is a co-authorship graph where each node is an author and edges represent if two nodes co-authored a paper. Node attributes are bag-of-words of authors' papers. The task is to map each author to its corresponding research area. These graphs are homophilic. We also use a heterophilic dataset, Actor(Pei et al., 2020) to evaluate GRAFENNE. The actor is a co-occurrence graph of actor nodes on the same Wikipedia page. Node features are bags of words from Wikipedia pages, and node labels are actor categories from Wikipedia pages.

I. Impact of 3-phase Message Passing Compared to Vanilla Message Passing on Transformed Graph

Table F: Performance of GRAFENNE in node classification task when message passing is performed in standard mode on $\mathcal{G}^{alt}$ on full dataset

Method	Cora	CiteSeer	Actor
Traditional	81.36 ± 0.80	66.72 ± 1.40	36.20 ± 0.29
GRAFENNE	$\mathbf{87.6 \pm 0.73}$	$\mathbf{73.90 \pm 0.84}$	$\mathbf{38.90 \pm 0.84}$

Table F shows the importance of the proposed 3-phased message passing framework. Specifically, we use GRAPHSAGE on the allotropic graph instead of the proposed 3-phased message passing. As visible, there is a significant drop in quality.

J. Additional Results

Table G: AUCROC of GRAFENNE (GAT) and GRAFENNE (GIN) with benchmark GNNs, GAT and GIN on link-prediction task

Dataset	Method	$p = 0$	$p = 0.5$	$p = 0.99$
Cora	GRAPHSAGE	0.86 ± 0.002	0.84 ± 0.002	0.7523 ± 0.05
	GRAFENNE	$\mathbf{0.8780 \pm 0.004}$	$\mathbf{0.8501 \pm 0.005}$	$\mathbf{0.8015 \pm 0.006}$
	GAT	0.8681 ± 0.0027	0.8316 ± 0.0010	0.7468 ± 0.0024
	GRAFENNE (GAT)	$\mathbf{0.8765 \pm 0.002}$	$\mathbf{0.8392 \pm 0.0047}$	$\mathbf{0.7841 \pm 0.005}$
	GIN	0.8552 ± 0.0022	0.8267 ± 0.0037	0.7399 ± 0.0051
	GRAFENNE (GIN)	$\mathbf{0.8591 \pm 0.007}$	$\mathbf{0.8301 \pm 0.008}$	$\mathbf{0.7692 \pm 0.011}$
CiteSeer	GRAPHSAGE	0.8251 ± 0.005	0.7617 ± 0.001	0.7223 ± 0.002
	GRAFENNE	$\mathbf{0.8681 \pm 0.010}$	$\mathbf{0.8047 \pm 0.010}$	$\mathbf{0.7378 \pm 0.008}$
	GAT	0.8123 ± 0.0011	0.7789 ± 0.0008	0.7205 ± 0.005
	GRAFENNE (GAT)	$\mathbf{0.8546 \pm 0.008}$	$\mathbf{0.8042 \pm 0.005}$	$\mathbf{0.7268 \pm 0.010}$
	GIN	0.8319 ± 0.0031	0.7767 ± 0.0018	0.7203 ± 0.007
	GRAFENNE (GIN)	$\mathbf{0.8720 \pm 0.0058}$	$\mathbf{0.8046 \pm 0.0040}$	$\mathbf{0.7228 \pm 0.0003}$
Actor	GRAPHSAGE	$.6569 \pm 0.0$	0.7029 ± 0.0026	0.6969 ± 0.003
	GRAFENNE	$\mathbf{0.7047 \pm 0.005}$	$\mathbf{0.7021 \pm 0.0034}$	$\mathbf{0.7029 \pm 0.0026}$
	GAT	$\mathbf{.7436 \pm 0.0016}$	0.7436 ± 0.0025	0.6721 ± 0.0035
	GRAFENNE (GAT)	0.7142 ± 0.006	$\mathbf{0.7090 \pm 0.006}$	$\mathbf{0.6942 \pm 0.004}$
	GIN	0.7549 ± 0.0	$\mathbf{0.7995 \pm 0.0}$	$\mathbf{0.7856 \pm 0.0}$
	GRAFENNE (GIN)	$\mathbf{0.7982 \pm 0.0}$	0.7905 ± 0.0013	0.7539 ± 0.0

K. Impact of Feature Translation

We translate features in Cora dataset by a factor of 10 in the node classification task. In Table H we observe that the performance of GRAFENNE remain intact. p represents the ratio of features deleted per node as defined in Sec 5.4(Empirical evaluation.).

	$p = 0$	$p = 0.5$	$p = 0.99$
GRAFENNE	87.6 ± 0.73	84.35 ± 0.27	78.85 ± 0.29
GRAFENNE (Translate)	87.7 ± 0.81	84.03 ± 0.31	78.78 ± 0.99

Table H: Impact of feature translation (by a factor of 10) on the Cora dataset.

L. Comparison with FATE (Wu et al., 2021)

As we explain below, FATE tackles a different problem, is significantly different in methodology and consequently, when adapted for our problem, generates substantially inferior results.

Difference in problem formulation: The input to our problem is a graph where nodes are annotated with feature vectors. In FATE, the input does not include a graph. FATE takes as input just a set of feature vectors. In addition, FATE also assumes the feature vectors to be a set of attributes (represented as one-hot encoding). In our problems, the feature vectors may represent either attributes or continuous-valued.

Difference in methodology: For feature adaptation, FATE forms a bipartite graph where the two sets of nodes correspond to original data points and feature values. While we also form a data(node)-feature graph, the methodology is dramatically different.

1. **Structure of graph representation:** GRAFENNE is not a bipartite graph as there are node-node edges in addition to node-feature edges (Recall Fig. 1).
2. **Modeling feature values:** Since FATE assumes attributed feature vectors, which are one-hot encodings, an edge exists in

the bipartite graph from the original data point to a feature-node if the feature (attribute) is present in that data point. This design is not adequate for our problem since features could be continuous-valued. Hence, in our allotropic graph construction, the edges from feature nodes to data nodes are weighted indicating the feature value in the node.

3. **Handling continuous-value data:** FATE does discuss strategies to adapt their methodology to continuous-valued features by discretizing the feature space into bins. This solution is not adequate since:
   (a) It's not clear what should be the bin-width.
   (b) More importantly, binning feature spaces and treating them as discrete values in the form of one-hot encodings distort the notion of similarity in the original feature space. Specifically, let's assume a bin width of 25 on a feature ranging from 0 to 100. A value in bins 0-25 is more similar to a value in 26-50 than to one in 76-100. This semantics gets lost when discretized since FATE treats every two bin values as either being the same or different (as in items drawn from a set).
   (c) Finally, FATE results in $f \times m$ feature-nodes where f is the number of features and m is the number of feature values (bins) per feature on average. In contrast, the proposed work generates f feature nodes. Thus, FATE (Wu et al., 2021) leads to higher storage and computation overheads.
4. **Message-passing scheme:** Since the input data in FATE is not a graph and the input is assumed to be attributed feature vectors, the message-passing scheme's primary objective is to learn co-occurrence correlation across features. The task in our case is significantly more complex. Specifically, we (1) need to learn feature co-occurrence patterns, (2) continuous-valued feature imputation as a function of topology, and (3) the objective function (such as node classification, link prediction, etc) as a joint function of topology and features. Owing to the difference in objectives, while FATE decouples the objective task (Ex. classification) from the feature adaptation task. In contrast, we learn feature adaptation and the end objective in an end-to-end manner. Furthermore, while FATE directly uses message passing mechanism of GNNs on the bipartite graph, we have devised a 3-phased message passing framework on the transformed graph due to the more complex modeling needs. As evident from Table F in Appendix Sec. G, the 3-phased message passing obtains superior performance.

Empirical evaluation: We have added FATE (Wu et al., 2021) as a baseline in the table below. p represents the ratio of features deleted per node as defined in Sec 5.4(Empirical evaluation.). The results are presented for node classification. To adapt FATE for our task where the input is a graph, in addition to the bipartite graph, we added node-node edges as in our construction. Yet, FATE produces significantly inferior results, due to the issues outlined above.

Dataset\Method	GRAFENNE				FATE		
	$p = 0$	$p = 0.5$	$p = 0.99$		$p = 0$	$p = 0.5$	$p = 0.99$
Cora	87.6 ± 0.73	84.35 ± 0.27	78.85 ± 0.29	—	71.11 ± 1.11	52.43 ± 1.21	30.42 ± 0.22
Actor	38.9 ± 0.84	35.02 ± 0.21	23.97 ± 0.58	—	34.42 ± 0.11	32.25 ± 0.93	22.57 ± 0.15
CiteSeer	73.9 ± 0.84	72.91 ± 0.95	64.29 ± 1.02	—	69.7 ± 0.78	61.11 ± 0.06	22.69 ± 0.51
Physics	97.02 ± 0.05	96.23 ± 0.23	94.49 ± 0.18	—	96.34 ± 0.09	92.87 ± 0.22	53.55 ± 0.15

Table I: Comparison of GRAFENNE against FATE at different missing rates on the node classification task.