ReFactor GNNs: Revisiting Factorisation-based Models from a Message-Passing Perspective

Anonymous Author(s)
Anonymous Affiliation
Anonymous Email

Abstract

Factorisation-based Models (FMs), such as DistMult, have enjoyed enduring success for Knowledge Graph Completion (KGC) tasks, often outperforming Graph Neural Networks (GNNs). However, unlike GNNs, FMs struggle to incorporate node features and generalise to unseen nodes in inductive settings. Our work bridges the gap between FMs and GNNs by proposing ReFactor GNNs. This new architecture draws upon *both* modelling paradigms, which previously were largely thought of as disjoint. Concretely, using a message-passing formalism, we show how FMs can be cast as GNNs by reformulating the gradient descent procedure as message-passing operations, which forms the basis of our ReFactor GNNs. Our ReFactor GNNs achieve state-of-the-art inductive performance while using an order of magnitude fewer parameters.

1 Introduction

In recent years, machine learning on graphs has attracted significant attention due to the abundance of graph-structured data and developments in graph learning algorithms. Graph Neural Networks (GNNs) have shown state-of-the-art performance for many graph-related problems, such as node classification [1] and graph classification [2]. Their main advantage is that they can easily be applied in an inductive setting: generalising to new nodes and graphs without re-training. However, despite many attempts at applying GNNs for multi-relational link prediction such as Knowledge Graph Completion [3], there are still few positive results compared to factorisation-based models (FMs) [4, 5]. As it stands, GNNs either – after resolving reproducibility concerns – deliver significantly lower performance [6, 7] or yield negligible performance gains at the cost of highly sophisticated architecture designs [8]. A notable exception is NBFNet [9], but even here the advance comes at the price of a high computational inference cost compared to FMs. Furthermore, it is unclear how NBFNet could incorporate node features, which – as we will see in this work – leads to remarkably lower performance in an inductive setting. On the flip side, FMs, despite being a simpler architecture, have been found to be very accurate for knowledge graph completion when coupled with appropriate training strategies [10] and training objectives [11, 12]. However, they also come with shortcomings in that they, unlike GNNs, can not be applied in an inductive setting.

Given the respective strengths and weaknesses of FMs and GNNs, *can we bridge these two seemingly different model categories?* While exploring this question, we make the following contributions:

- By reformulating the training process using message-passing primitives, we show a practical connection between FMs and GNNs, i.e. FMs can be treated as a special instance of GNNs.
- Based on this connection, we propose a new family of architectures, ReFactor GNNs, that interpolates between FMs and GNNs and allow FMs to be used inductively.
- In an empirical investigation across well-established benchmarks (see the appendix), our ReFactor GNNs achieve state-of-the-art inductive performance across the board and comparable transductive performance to FMs – despite using an order of magnitude fewer parameters.

2 Background

Knowledge Graph Completion [KGC, 13] is a canonical task of multi-relational link prediction. The goal is to predict missing edges given the existing edges in the knowledge graph. Formally, a knowledge graph contains a set of entities (nodes) $\mathcal{E} = \{1, \ldots, |\mathcal{E}|\}$, a set of relation (or edge) types $\mathcal{R} = \{1, \ldots, |\mathcal{R}|\}$, and a set of typed edges between the entities $\mathcal{T} = \{(v_i, r_i, w_i)\}_{i=1}^{|\mathcal{T}|}$, where each triple (v_i, r_i, w_i) indicates a relationship of type $r_i \in \mathcal{R}$ between the *subject* $v_i \in \mathcal{E}$ and the *object* $w_i \in \mathcal{E}$ of the triple. Given a (training) knowledge graph, the KGC task [3] aims at identifying missing links by answering $(v, r, ?)$ queries i.e. predicting the object given the subject and the relation.

Multi-relational link prediction models can be trained via maximum likelihood, by fitting a parameterized conditional categorical distribution $P_\theta(w \mid v, r)$ over the candidate objects of a relation, given the subject v and the relation type r: $P_\theta(w|v, r) = \mathrm{Softmax}(\Gamma_\theta(v, r, \cdot))[w]$, where $\Gamma_\theta : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \to \mathbb{R}$ is a *scoring function* that, given a triple (v, r, w), returns the likelihood that the corresponding edge appears in the graph. In this paper, we illustrate our derivations using DistMult [4] as the score function Γ and defer extensions to general score functions, e.g. ComplEx [5] to the appendix. In DistMult, the score function Γ_θ is defined as the tri-linear dot product of the embeddings of the subject, relation type, and object of the triple: $\Gamma_\theta(v, r, w) = \sum_{i=1}^{K} f_\phi(v)_i f_\phi(w)_i g_\psi(r)_i$, where $f_\phi : \mathcal{E} \to \mathbb{R}^K$ and $g_\psi : \mathcal{R} \to \mathbb{R}^K$ are learnable maps parameterised by ϕ and ψ that encode entities and relation types into K-dimensional representations, and $\theta = (\phi, \psi)$. We will refer to f and g as the entity and relational *encoders*, respectively.

We can learn the model parameters θ by minimising the expected negative log-likelihood $\mathcal{L}(\theta)$ of the ground-truth entities for the queries $(v, r, ?)$ obtained from $\mathcal{T}$:

$$\arg\min_\theta \mathcal{L}(\theta) \quad \text{where} \quad \mathcal{L}(\theta) = -\frac{1}{|\mathcal{T}|} \sum_{(v,r,w)\in\mathcal{T}} \log P_\theta(w|v, r). \tag{1}$$

During inference, we use the distribution P_θ for ranking missing links.

Factorisation-based Models for KGC. In factorisation-based models, which we assume to be DistMult, f_ϕ and g_ψ are simply parameterised as look-up tables, associating each entity and relation with a continuous distributed representation:

$$f_\phi(v) = \phi[v],\ \phi \in \mathbb{R}^{|\mathcal{E}|\times K} \quad \text{and} \quad g_\psi(r) = \psi[r],\ \psi \in \mathbb{R}^{|\mathcal{R}|\times K}. \tag{2}$$

GNN-based Models for KGC. GNNs were originally proposed for node or graph classification tasks [14, 15]. To adapt them to KGC, previous work has explored two different paradigms: *node-wise entity representations* [16] and *pair-wise entity representations* [9, 17]. Though the latter paradigm has shown promising results, it requires computing an embedding representation for any pair of nodes, which can be too computationally expensive for large-scale graphs with millions of entities. Additionally, node-wise representations allow for using a single evaluation of $f_\phi(v)$ for multiple queries involving v. Models based on the first paradigm differ from pure FMs only in the entity encoder and lend themselves well for a fairer comparison with pure FMs. We will therefore focus on this class and leave the investigation of pair-wise representations to future work.

Let $q_\phi : \mathcal{G} \times \mathcal{X} \to \bigcup_{S\in\mathbb{N}^+} \mathbb{R}^{S\times K}$ be a GNN encoder, where $\mathcal{G} = \{G \mid G \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}\}$ is the set of all possible multi-relational graphs defined over $\mathcal{E}$ and $\mathcal{R}$, and $\mathcal{X}$ is the input feature space, respectively. Then we can set $f_\phi(v) = q_\phi(\mathcal{T}, X)[v]$. Following the standard message-passing framework [2, 18] used by the GNNs, we view $q_\phi = q^L \circ \ldots \circ q^1$ as the recursive composition of $L \in \mathbb{N}^+$ layers that compute intermediate representations h^l for $l \in \{1, \ldots, L\}$ (and $h^0 = X$) for all entities in the KG. Each layer is made up of the following three functions:

- A *message function* $q^l_{\mathrm{M}} : \mathbb{R}^K \times \mathcal{R} \times \mathbb{R}^K \to \mathbb{R}^K$ that computes the message along each edge. Given an edge $(v, r, w) \in \mathcal{T}$, q^l_{M} not only makes use of the node states $h^{l-1}[v]$ and $h^{l-1}[w]$ (as in standard GNNs) but also uses the relation r. Denote the message as $m^l[v, r, w] = q^l_{\mathrm{M}}\left(h^{l-1}[v], r, h^{l-1}[w]\right)$;
- An *aggregation function* $q^l_{\mathrm{A}} : \bigcup_{S\in\mathbb{N}} \mathbb{R}^{S\times K} \to \mathbb{R}^K$ that aggregates all messages from the 1-hop neighbourhood of a node; denote the aggregated message as $z^l[v] = q^l_{\mathrm{A}}\left(\{m^l[v, r, w] \mid (r, w) \in \mathcal{N}^1[v]\}\right)$;
- An *update function* $q^l_{\mathrm{U}} : \mathbb{R}^K \times \mathbb{R}^K \to \mathbb{R}^K$ that produces the new node states h^l by combining previous node states h^{l-1} and the aggregated messages z^l: $h^l[v] = q^l_{\mathrm{U}}(h^{l-1}[v], z^l[v])$.

3 Implicit Message-Passing in FMs

The sharp difference in analytical forms might give rise to the misconception that GNNs incorporate message-passing over the neighbourhood of each node (up to L-hops), while FMs do not. In this work, we show that by explicitly considering the training dynamics of FMs, we can uncover and analyse the hidden message-passing mechanism within FMs. In turn, this will lead us to the formulation of a novel class of GNNs well suited for multi-relational link prediction tasks (Section 4). Specifically, we propose to interpret the FMs' optimisation process of their objective (1) as the entity encoder. If we consider, for simplicity, a gradient descent training dynamic, then

$$f_{\phi^t}(v) = \phi^t[v] = \mathrm{GD}^t(\phi^{t-1}, \mathcal{T})[v] = \underbrace{\mathrm{GD}^t \circ \ldots \mathrm{GD}^1}_{t}(\phi^0, \mathcal{T})[v], \tag{3}$$

where ϕ^t is the embedding vector at the t-th step, $t \in \mathbb{N}^+$ is the total number of iterations and ϕ^0 is a random initialisation. GD is the gradient descent operator:

$$\mathrm{GD}(\phi, \mathcal{T}) = \phi - \alpha \nabla_\phi \mathcal{L} = \phi + \alpha \sum_{(v,r,w)\in\mathcal{T}} \frac{\partial \log P(w|v,r)}{\partial \phi}, \tag{4}$$

where $\alpha = \beta\,|\mathcal{T}|^{-1}$, with a $\eta > 0$ learning rate. We now dissect Equation (4) in two different (but equivalent) ways. In the first, which we dub the *edge view*, we separately consider each addend of the gradient $\nabla_\phi \mathcal{L}$. In the second, we aggregate the contributions from all the triples to the update of a particular node. With this latter decomposition, which we call the *node view*, we can explicate the message-passing mechanism at the core of the FMs. While the edge view suits a vectorised implementation better, the node view further exposes the information flow among nodes, allowing us to draw an analogy to message-passing GNNs.

To fully uncover the message-passing mechanism of FMs, we now focus on the gradient descent operation over a single node $v \in \mathcal{E}$, referred to as the *central node* in the GNN literature. Recalling Equation (4), we have:

$$\mathrm{GD}(\phi, \mathcal{T})[v] = \phi[v] + \alpha \sum_{(v,r,w)\in\mathcal{T}} \frac{\partial \log P(\bar{\mathrm{w}} \mid \bar{\mathrm{v}}, \bar{\mathrm{r}})}{\partial \phi[v]}, \tag{5}$$

which aggregates the information stemming from the updates presented in the edge view. The next theorem describes how this total information flow to a particular node can be recast as an instance of message passing (cf. Section 2). We defer the proof to the appendix.

Theorem 3.1 (Message passing in FMs)**.** *The gradient descent operator* GD *(Equation* (5)*) on the node embeddings of a DistMult model (Equation* (2)*) with the maximum likelihood objective in Equation* (1) *and a multi-relational graph $\mathcal{T}$ defined over entities $\mathcal{E}$ induces a message-passing operator whose composing functions are:*

$$q_{\mathrm{M}}(\phi[v], r, \phi[w]) = \begin{cases} \phi[w] \odot g(r) & \text{if } (r,w) \in \mathcal{N}^1_+[v], \\ (1 - P_\theta(v|w,r))\phi[w] \odot g(r) & \text{if } (r,w) \in \mathcal{N}^1_-[v]; \end{cases} \tag{6}$$

$$q_{\mathrm{A}}(\{m[v,r,w] \,:\, (r,w) \in \mathcal{N}^1[v]\}) = \sum_{(r,w)\in\mathcal{N}^1[v]} m[v,r,w]; \tag{7}$$

$$q_{\mathrm{U}}(\phi[v], z[v]) = \phi[v] + \alpha z[v] - \beta n[v], \tag{8}$$

where, defining the sets of triples $\mathcal{T}^{-v} = \{(s,r,o) \in \mathcal{T} \,:\, s \neq v \wedge o \neq v\}$,

$$n[v] = \frac{|\mathcal{N}^1_+[v]|}{|\mathcal{T}|} \mathbb{E}_{P_{\mathcal{N}^1_+[v]}} \mathbb{E}_{u\sim P_\theta(\cdot|v,r)} g(r) \odot \phi[u] + \frac{|\mathcal{T}^{-v}|}{|\mathcal{T}|} \mathbb{E}_{P_{\mathcal{T}^{-v}}} P_\theta(v|s,r) g(r) \odot \phi[s], \tag{9}$$

where $P_{\mathcal{N}^1_+[v]}$ and $P_{\mathcal{T}^{-v}}$ are the empirical probability distributions associated to the respective sets.

What emerges from the equations is that each gradient step contains an explicit information flow from the neighbourhood of each node, which is then aggregated with a simple summation. Through this direct information path, t steps of gradient descent cover t-hop neighbourhood of v. As t goes towards infinity – or in practice – as training converges, FMs capture the global graph structure. The update function (8) somewhat deviates from classic message-passing as $n[v]$ of Equation (9) involves global information. However, we note that we can interpret this mechanism under the framework of augmented message passing [19] and, in particular, as an instance of *graph rewiring*.

Based on Theorem 3.1 and Equation (3), we can now view ϕ as the transient node states h (cf. Section 2) and GD on node embeddings as a message-passing layer. This dualism sits at the core of the ReFactor GNN model, which we describe next.

4 ReFactor GNNs

FMs are trained by minimising the objective (1), initialising both sets of parameters (ϕ and ψ) and performing GD until approximate convergence (or until early stopping terminates the training). The implications are twofold: i) the initial value of the entity lookup table ϕ does not play any major role in the final model after convergence; and ii) if we introduce a new set of entities, the conventional wisdom is to retrain[1] the model on the expanded knowledge graph. This is computationally rather expensive compared to the "inductive" models that require no additional training and can leverage node features like entity descriptions. However, as we have just seen in Theorem 3.1, the training procedure of FMs may be naturally recast as a message-passing operation, which suggests that it is possible to use FMs for inductive learning tasks. In fact, we envision that there is an entire novel spectrum of model architectures interpolating between pure FMs and (various instantiations of) GNNs. Here we propose one simple implementation of such an architecture which we dub ReFactor GNNs. Figure 1 gives an overview of ReFactor GNNs.

The ReFactor Layer. A ReFactor GNN contains L ReFactor layers, that we derive from Theorem 3.1. Aligning with the GNN notations we introduced in Section 2, given a KG $\mathcal{T}$ and entity representations $h^{l-1} \in \mathbb{R}^{|\mathcal{E}| \times K}$, the ReFactor layer computes the representation of a node v as follows:

$$h^l[v] = q^l(\mathcal{T}, h^{l-1})[v] = h^{l-1}[v] - \beta n^l[v] + \alpha \sum_{(r,w) \in \mathcal{N}^1[v]} q^l_M(h^{l-1}[v], r, h^{l-1}[w]), \tag{10}$$

where the terms n^l and q^l_{M} derive from Equation (9) and Equation (6), respectively. Differing from the R-GCN, the first GNN on multi-relational graphs, where the incoming and outgoing neighbourhoods are treated equally [16], ReFactor GNNs treat incoming and outgoing neighbourhoods differently. As we will show in the experiments, this allows ReFactor GNNs to achieve good performances also on datasets containing non-symmetric relationships. In fact, the ReFactor layer is built upon DistMult, which, despite being a symmetric operator, induces asymmetry into the final representation.

Equation (10) describes the full batch setting, which can be expensive if the KG contains many edges. Therefore, in practice, whenever the graph is big, we adopt a stochastic evaluation of the ReFactor layer by decomposing the evaluation into several mini-batches. We partition $\mathcal{T}$ into a set of computationally tractable mini-batches. For each of them, we restrict the neighbourhoods to the subparagraph induced by it and readjust the computation of $n^l[v]$ to include only entities and edges present in it. We leave the investigation of other stochastic strategies (e.g. by taking Monte Carlo estimations of the expectations in Equation (9)) to future work. Finally, we cascade the mini-batch evaluation to produce one full layer evaluation.

Training. The learnable parameters of ReFactor GNNs are the relation embeddings ψ. Inspired by [20], we learn ψ by layer-wise (stochastic) gradient descent. This is in contrast to conventional GNN training, where we need to backpropagate through all the layers. A (full-batch) GD training dynamic for ψ can be written as $\psi_{t+1} = \psi_t - \eta \nabla \mathcal{L}_t(\psi_t)$, where $\mathcal{L}_t(\psi_t) = -|\mathcal{T}|^{-1} \sum_{\mathcal{T}} \log P_{\psi_t}(w|v,r)$, with:

$$P_{\psi_t}(w|v,r) = \mathrm{Softmax}(\Gamma(v,r,\cdot))[w], \qquad \Gamma(v,r,w) = \langle h^t[v], h^t[w], g_{\psi_t}(r) \rangle$$

and the node state update as

$$h^t = \begin{cases} X & \text{if } t \bmod L = 0 \\ q^{t \bmod L}(\mathcal{T}, h^{t-1}) & \text{otherwise} \end{cases} \tag{11}$$

Implementation-wise, such a training dynamic equals to using an external memory for storing historical node states h^{t-1} akin to the procedure introduced in [21]. The memory can then be queried to compute h^t using Equation (10). Under this perspective, we periodically clear *the node state cache* every L full batches to force the model to predict based on on-the-fly L-layer message-passing. After training, we obtain ψ^* and do the inference by running L-layer message-passing with ψ^*.

Due to page limits, we leave the empirical study over the proposed ReFactor GNNs in the appendix. In general, we observe ReFactor GNNs to achieve state-of-the-art inductive performance.

[1]Typically until convergence, possibly by partially warm-starting θ.

References

[1] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 1, 11

[2] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017. 1, 2, 11

[3] Maximilian Nickel, Lorenzo Rosasco, and Tomaso A. Poggio. Holographic embeddings of knowledge graphs. In *AAAI*, pages 1955–1961. AAAI Press, 2016. 1, 2, 10

[4] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR (Poster)*, 2015. 1, 2

[5] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Eric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2071–2080, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/trouillon16.html. 1, 2, 10

[6] Deepak Nathani, Jatin Chauhan, Charu Sharma, and Manohar Kaul. Learning attention-based embeddings for relation prediction in knowledge graphs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4710–4723, 2019. 1

[7] Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha Talukdar, and Yiming Yang. A re-evaluation of knowledge graph completion methods. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5516–5522, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.489. URL https://aclanthology.org/2020.acl-main.489. 1

[8] Xiaoran Xu, Wei Feng, Yunsheng Jiang, Xiaohui Xie, Zhiqing Sun, and Zhi-Hong Deng. Dynamically pruned message passing networks for large-scale knowledge graph reasoning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkeuAhVKvB. 1

[9] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal A. C. Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. *CoRR*, abs/2106.06935, 2021. URL https://arxiv.org/abs/2106.06935. 1, 2, 8, 9, 10, 13

[10] Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You can teach an old dog new tricks! on training knowledge graph embeddings. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BkxSmlBFvr. 1, 10

[11] Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 2869–2878. PMLR, 2018. 1, 10, 13, 16

[12] Yihong Chen, Pasquale Minervini, Sebastian Riedel, and Pontus Stenetorp. Relation prediction as an auxiliary training objective for improving multi-relational graph representations. In *3rd Conference on Automated Knowledge Base Construction*, 2021. URL https://openreview.net/forum?id=Qa3uS3H7-Le. 1, 10

[13] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proc. IEEE*, 104(1):11–33, 2016. 2

[14] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, 2:729–734 vol. 2, 2005. 2

[15] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009. 2

[16] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018. 2, 4, 10

[17] Komal K. Teru, Etienne G. Denis, and William L. Hamilton. Inductive relation prediction by subgraph reasoning. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 9448–9457. PMLR, 2020. 2, 7, 8, 13

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

[19] Petar Veličković. Message passing all the way up, 2022. URL https://arxiv.org/abs/2202.11097. 3, 9, 11

[20] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. L2-gcn: Layer-wise and learned efficient training of graph convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2127–2135, 2020. 4

[21] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Jure Leskovec. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In *ICML*, 2021. 4

[22] Charles Kemp, Joshua B. Tenenbaum, Thomas L. Griffiths, Takeshi Yamada, and Naonori Ueda. Learning systems of concepts with an infinite relational model. In *AAAI*, pages 381–388. AAAI Press, 2006. 7

[23] Tara Safavi and Danai Koutra. Codex: A comprehensive knowledge graph completion benchmark. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8328–8350, 2020. 7

[24] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd workshop on continuous vector space models and their compositionality*, pages 57–66, 2015. 7

[25] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017. 8

[26] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Few-shot representation learning for out-of-vocabulary words. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2019. 8, 11, 16

[27] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. GraphSAINT: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJe8pkHFwS. 8, 11

[28] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. 8, 11

[29] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL https://arxiv.org/abs/1908.10084. 8, 15

[30] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, pages 809–816. Omnipress, 2011. 10

[31] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI*, pages 1811–1818. AAAI Press, 2018.

[32] Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Q. Phung. A novel embedding model for knowledge base completion based on convolutional neural network. In *NAACL-HLT (2)*, pages 327–333. Association for Computational Linguistics, 2018. 10

[33] Xiaoran Xu, Wei Feng, Yunsheng Jiang, Xiaohui Xie, Zhiqing Sun, and Zhi-Hong Deng. Dynamically pruned message passing networks for large-scale knowledge graph reasoning. In *ICLR*. OpenReview.net, 2020. 10

[34] Zhao Zhang, Fuzhen Zhuang, Hengshu Zhu, Zhi-Ping Shi, Hui Xiong, and Qing He. Relational graph neural network with hierarchical attention for knowledge graph completion. In *AAAI*, pages 9612–9619. AAAI Press, 2020.

[35] Ren Li, Yanan Cao, Qiannan Zhu, Guanqun Bi, Fang Fang, Yi Liu, and Qian Li. How does knowledge graph embedding extrapolate to unseen data: a semantic evidence view. *CoRR*, abs/2109.11800, 2021. 10

[36] Prachi Jain, Sushant Rathi, Mausam, and Soumen Chakrabarti. Knowledge base completion: Baseline strikes back (again). *CoRR*, abs/2005.00804, 2020. URL https://arxiv.org/abs/2005.00804. 10

[37] Balasubramaniam Srinivasan and Bruno Ribeiro. On the equivalence between positional node embeddings and structural graph representations. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJxzFySKwH. 10

[38] Yifei Shen, Yongji Wu, Yao Zhang, Caihua Shan, Jun Zhang, Khaled B Letaief, and Dongsheng Li. How powerful is graph convolution for recommendation? *arXiv preprint arXiv:2108.07567*, 2021. 10

[39] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003. 11

[40] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In *ICLR (Poster)*, 2016. 11

[41] Léon Bottou. *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35289-8. doi: 10.1007/978-3-642-35289-8_25. URL https://doi.org/10.1007/978-3-642-35289-8_25. 11

[42] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011. URL http://jmlr.org/papers/v12/duchi11a.html. 13

A Architecture

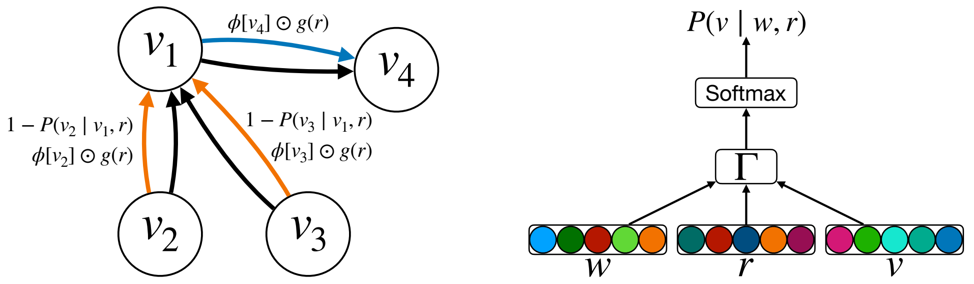

Figure 1: ReFactor GNN architecture – the left figure describes the messages (coloured edges) used to update the representation of node v_1, which depend on the type of relationship between the sender nodes and v_1 in the graph $G = \{(v_2, r_1, v_1), (v_3, r_2, v_1), (v_1, r_3, v_4)\}$; the right figure describes the computation graph for calculating $P(v \mid w, r)$, where $v, w \in \mathcal{E}$ and $r \in \mathcal{R}$: the embedding representations of w, r, and v are used to score the edge (w, r, v) via the scoring function Γ, which is then normalised via the Softmax function.

Figure 1 shows the architecture of REFACTOR GNNS.

B Experiments

We perform experiments to answer the following questions regarding REFACTOR GNNS:

- **Q1.** REFACTOR GNNS are derived from a message-passing reformulation of FMs: do they also inherit their predictive accuracy in *transductive* KGC tasks? Appendix B.1
- **Q2.** Are REFACTOR GNNS more statistically accurate than other GNN baselines in *inductive* KBC tasks? Appendix B.2
- **Q3.** Can we simplify REFACTOR GNNS by removing the term $n[v]$, which involves nodes not in the 1-hop neighbourhood? Appendix B.3

For transductive experiments, we used three well-established KGC datasets: *UMLS* [22], *CoDEx-S* [23], and *FB15K237* [24]. For inductive experiments, we used the inductive KGC benchmarks introduced by GraIL [17], which include 12 datasets, or rather 12 pairs of knowledge graphs:

Entity Encoder	UMLS	CoDEx-S	FB15K237
Lookup (FM, specif. DistMult)	0.90	0.43	0.30
REFACTOR GNNS ($L = \infty$)	0.93	0.44	0.33

Table 1: Test MRR for transductive KGC tasks.

(*FB15K237_vi*, *FB15K237_vi_ind*), (*WN18RR_vi*, *WN18RR_vi_ind*), and (*NELL_vi*, *NELL_vi_ind*), where $i \in [1, 2, 3, 4]$, and (*_vi*, *_vi_ind*) represents a pair of graphs with a shared relation vocabulary and non-overlapping entities. We follow the standard KGC evaluation protocol by fully ranking all the candidate entities and computing two metrics using the ranks of the ground-truth entities: Mean Reciprocal Ranking (MRR) and Hit Ratios at Top K (Hits@K) with $K \in [1, 3, 10]$. For the inductive KGC, we additionally consider the partial-ranking evaluation protocol used by GraIL for fair comparison. Empirically, we find full ranking more difficult than partial ranking, and thus more suitable for reflecting the differences among models on GraIL datasets – we would like to call for future work on GraIL datasets to also adopt full ranking protocol on these datasets.

We grid-searched over the hyper-parameters, and selected the best configuration based on validation MRR. Since training deep GNNs with full-graph message passing might be slow for large knowledge graphs, we follow the literature [25–27] to sample sub-graphs for training GNNs. Considering that sampling on-the-fly often prevents high utilisation of GPUs, we resort to a two-stage process: we first sampled and serialised sub-graphs around the target edges in the mini-batches; we then trained the GNNs with the serialised sub-graphs. To ensure we have sufficient sub-graphs for training the models, we sampled for 20 epochs for each knowledge graph, i.e. 20 full-passes over the full graph. The sub-graph sampler we currently used is LADIES [26].

B.1 REFACTOR GNNS for Transductive Learning (Q1)

REFACTOR GNNS are derived from the message-passing reformulation of FMs. We expect them to have roughly the same performance as FMs for transductive KGC tasks. To verify this, we run experiments on the datatsets UMLS, CoDEx-S, and FB15K237. For fair comparison, we use **??** as the decoder and consider i) lookup embedding table as the entity encoder, which forms the FM when combined with the decoder (Section 2), and ii) REFACTOR GNNS as the entity encoder. REFACTOR GNNS are trained with $L = \infty$, i.e. we never clear the node state cache. Since transductive KGC tasks do not involve new entities, the node state cache in REFACTOR GNNS can be directly used for link prediction. Table 1 summarises the result. We observe that REFACTOR GNNS achieve a similar or slightly better performance compared to the FM. This shows that REFACTOR GNNS are able to capture the essence of FMs and thus maintain strong at transductive KGC.

B.2 REFACTOR GNNS for Inductive Learning (Q2)

Despite FMs' good empirical performance on transductive KGC tasks, they fail to be inductive as GNNs. According to our reformulation, this is due to the infinite message-passing layers hidden in FMs' optimisation. Discarding infinite message-passing layers, REFACTOR GNNS enable FMs to perform inductive reasoning tasks by learning to use a finite set of message-passing layers for prediction similarly to GNNs.

Here we present experiments to verify REFACTOR GNNS's capability for inductive reasoning. Specifically, we study the task of inductive KGC and investigate whether REFACTOR GNNS can generalise to unseen entities. Following [17], on GraIL datasets, we trained models on the original graph, and run 0-shot link prediction on the *_ind* test graph. Similar as the transductive experiments, we use **??** as the decoder and vary the entity encoder. We denote three-layer REFACTOR GNNS as `REFACTOR GNNs (3)` and six-layer REFACTOR GNNS as `REFACTOR GNNs (6)`. We consider several baseline entity encoders: i) `no-pretrain`, models without any pretraining on the original graph; ii) `GAT(3)`, three-layer graph attention network [28]; iii) *GAT(6)*, six-layer graph attention network; iv) `GraIL`, a sub-graph-based relational GNN [17]; v) `NBFNet`, a path-based GNN [9], current SoTA on GraIL datasets. In addition to randomly initialised vectors as the node features, we also use as node features RoBERTa Encodings of the entity descriptions, which are produced by SentenceBERT [29]. Due to space reason, we present the results on (*FB15K237_v1*, *FB15K237_v1_ind*) in Figure 2. Results on other datasets are similar and can be found in the appendix. We can see that without RoBERTa

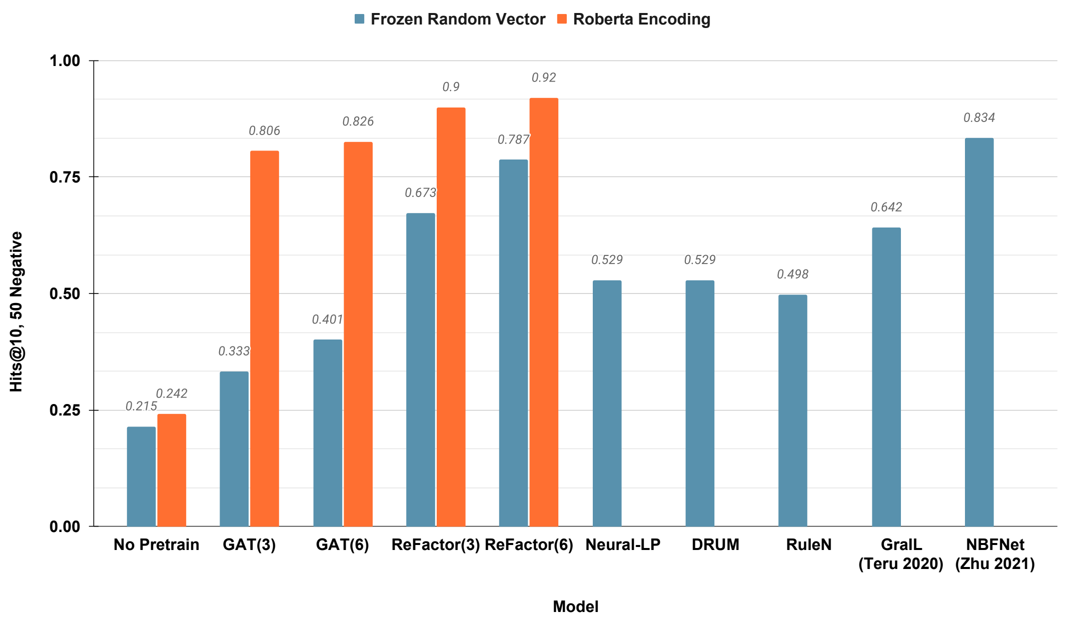

Figure 2: Inductive KGC Performance. Models are trained on the KG *FB15K237_v1* and tested on another KG *FB15K237_v1_ind*, where the entities are completely new. The results of GraIL and NBFNet are taken from [9]. It is unclear how to incorporate node features in GraIL and NBFNet.

Embeddings as node features, ReFactor GNNs perform better than GraIL (+23%); with RoBERTa Embeddings as node features, ReFactor GNNs outperform both GraIL (+43%) and NBFNet (+10%), achieving new SoTA results on inductive KGC tasks.

Performance vs Parameter Efficiency as #Message-Passing Layers Increases. Usually, as the number of message-passing layers increases in GNNs, the over-smoothing issue occurs while the computational cost also increases exponentially. ReFactor GNNs avoid this by layer-wise training and sharing the weights across layers. Here we compare ReFactor GNNs with $\{1, 3, 6, 9\}$ message-passing layer(s) with same-depth GATs – results are summarised in Figure 3. We observe that increasing the number of message-passing layers in GATs does not necessarily improve the predictive accuracy – the best results were obtained with 3 message-passing layers on *FB15K237_v1* while using 6 and 9 layers leads to performance degradation. On the other hand, ReFactor GNNs obtain consistent improvements when increasing #Layers from 1 to 3, 6, and 9. ReFactor GNNs $(6, 6)$ and $(9, 9)$ clearly outperform their GAT counterparts. Most importantly, ReFactor GNNs are more parameter-efficient than GATs, with a constant #Parameters as #Layers increases.

B.3 Beyond Message-Passing (Q3)

As shown by Theorem 3.1, ReFactor GNNs contain not only terms capturing information flow from the 1-hop neighbourhood, which falls into the classic message-passing framework, but also a term $n[v]$ that involve nodes outside the 1-hop neighbourhood. The term $n[v]$ can be treated as *augmented message-passing* on a dynamically rewired graph [19]. Here we perform ablation experiments to measure the impact of the $n[v]$ term. Table 2 summarises the ablation results: we can see that, without the term $n[v]$, ReFactor GNNs with random vectors as node features yield a 2% lower MRR, while ReFactor GNNs with RoBERTa encodings as node features produce a 7% lower MRR. This suggests that augmented message-passing also plays a significant role in ReFactor GNNs' generalisation properties in downstream link prediction tasks. Future work might gain more insights by further dissecting the $n[v]$ term.

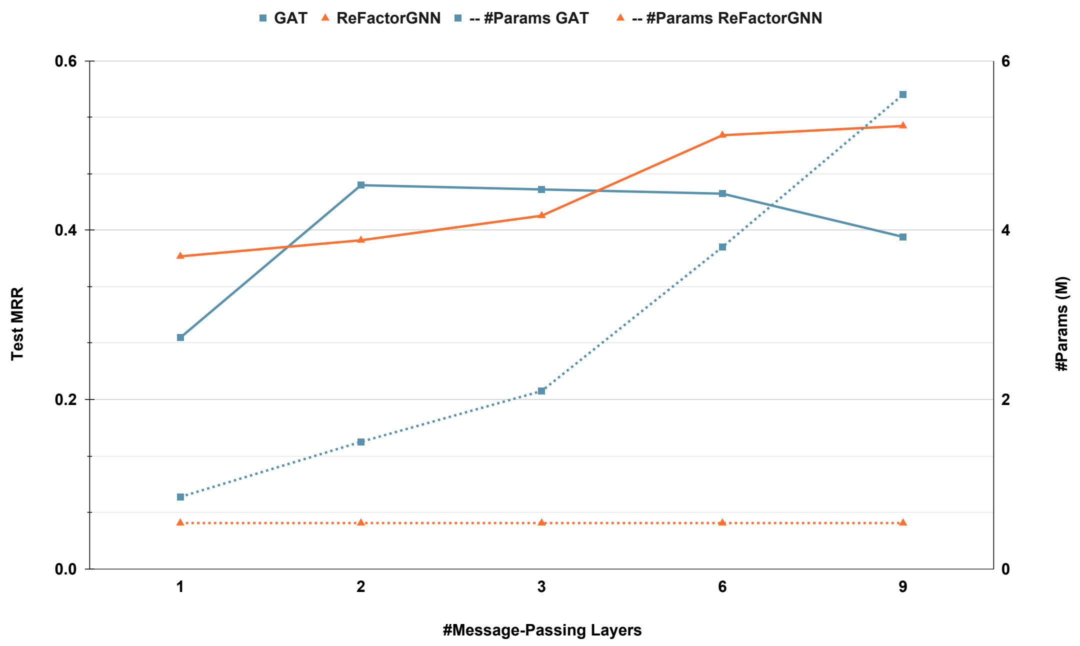

Figure 3: Performance vs Parameter Efficiency as #Layers Increases on *FB15K237_v1*. The left axis is Test MRR while the right axis is #Parameters. The solid lines and dashed lines indicate the changes of Test MRR and the changes of #Parameters.

Test MRR	Frozen Random Representations	RoBERTa Encodings
with $n[v]$	0.425	0.486
without $n[v]$	0.418	0.452

Table 2: Ablation on $n[v]$ for ReFactor GNNs (6) trained on *FB15K237_v1*.

C Related Work

Multi-Relational Graph Representation Learning. Previous work on multi-relational graph representation learning focused either on FMs [3–5, 11, 12, 30–32] or on GNN-based models [16, 33–35]. Recently, FMs were found to be significantly more accurate than GNNs in KGC tasks, when coupled with specific training strategies [10, 11, 36]. While more advanced GNNs [9] for KBC are showing promise at the cost of extra algorithm complexity, little effort has been devoted to establish the links between plain GNNs and FMs, which are strong multi-relational link predictors despite their simplicity. Our work aims to *align* GNNs with FMs so that we can combine the strengths from both families of models.

Relationships between FMs and GNNs. We would like to clarify our scope, by highlighting that our "FM" refers to factorisation-based models used for KGC, different from matrix factorisation, where there are no relational parameters. Similarly, our "GNN" refers to GNNs developed for KGC, which incorporate (positional) node features as elaborated in Section 2. We recognise that a very recent work [37] builds a theoretical link between structural GNNs and node (positional) embeddings, where the second model category encompasses not only FMs but also many practical GNNs. Both our FMs and GNNs fall into the second model category. *Therefore, we consider our work building a more fine-grained connection between positional node embeddings produced by FMs and positional node embeddings produced by GNNs, while at the same time focusing on KGC.* Beyond FMs in KGC, using graph signal processing theory, [38] show that matrix factorisation (MF) based recommender models correspond to ideal low-pass graph convolutional filters. Coincidentally, they also find infinite neighbourhood coverage in MF although using a completely different approach and focusing on a different domain in contrast to our work.

Message Passing in GNNs. Message passing allows to recursively decompose a global function into simple local, parallelisable computations [39]. Recently, [2] provided a unified message-passing reformulation for various GNN architectures, including Graph Attention Networks [28], Gated Graph Neural Networks [40], and Graph Convolutional Networks [1]. In this work, we show that FMs can also be cast as a special type of GNNs, by considering SGD updates [41] over node embeddings as message-passing operations between nodes. To the best of our knowledge, our work is the first to provide such connections between FMs and GNNs. In our work, we show that FMs can be seen as instances of GNNs, with a characteristic feature being in the nodes being considered during the message-passing process: our ReFactor GNNs can be seen as using an *Augmented Message-Passing* process on a dynamically re-wired graph [19].

D Conclusion & Future Work

Our work establishes a link between FMs and GNNs on the task of multi-relational link prediction. The reformulation of FMs as GNNs addresses the question why FMs are stronger multi-relational link predictors compared to plain GNNs. Guided by the reformulation, we further propose a new variant of GNNs, ReFactor GNNs, which combines the strengths of both FMs and classic GNNs. Empirical experiments show that ReFactor GNNs produce significantly more accurate results than our GNN baselines on link prediction tasks.

Since we adopt a two-stage (sub-graph serialisation and model training) approach instead of online sampling, there can be side effects from the low sub-graph diversity. In our experiments, we only used LADIES [26] for sub-graph sampling. We plan to experiment with different sub-graph sampling algorithms, such as GraphSaint [27], and see how this affects the downstream link prediction results. Furthermore, it would be interesting to analyse decoders other than DistMult, as well as additional optimisation schemes beyond SGD and AdaGrad.

E Theorem 1 Proof

In this section, we prove Theorem 1, which we restate here for convenience.

Theorem E.1 (Message passing in FMs)**.** *The gradient descent operator* GD *(5) on the node embeddings of a DistMult model (Equation (2)) with the maximum likelihood objective in Equation (1) and a multi-relational graph $\mathcal{T}$ defined over entities $\mathcal{E}$ induces a message-passing operator whose composing functions are:*

$$q_{\mathrm{M}}(\phi[v], r, \phi[w]) = \begin{cases} \phi[w] \odot g(r) & \text{if } (r,w) \in \mathcal{N}^1_+[v], \\ (1 - P_\theta(v|w,r))\phi[w] \odot g(r) & \text{if } (r,w) \in \mathcal{N}^1_-[v]; \end{cases} \tag{12}$$

$$q_{\mathrm{A}}(\{m[v,r,w] \,:\, (r,w) \in \mathcal{N}^1[v]\}) = \sum_{(r,w)\in\mathcal{N}^1[v]} m[v,r,w]; \tag{13}$$

$$q_{\mathrm{U}}(\phi[v], z[v]) = \phi[v] + \alpha z[v] - \beta n[v], \tag{14}$$

where, defining the sets of triplets $\mathcal{T}^{-v} = \{(s,r,o) \in \mathcal{T} \,:\, s \neq v \wedge o \neq v\}$,

$$n[v] = \frac{|\mathcal{N}^1_+[v]|}{|\mathcal{T}|} \mathbb{E}_{P_{\mathcal{N}^1_+[v]}} \mathbb{E}_{u \sim P_\theta(\cdot|v,r)} g(r) \odot \phi[u] + \frac{|\mathcal{T}^{-v}|}{|\mathcal{T}|} \mathbb{E}_{P_{\mathcal{T}^{-v}}} P_\theta(v|s,r) g(r) \odot \phi[s], \tag{15}$$

where $P_{\mathcal{N}^1_+[v]}$ and $P_{\mathcal{T}^{-v}}$ are the empirical probability distributions associated to the respective sets.

Proof. Remember that we assume that there are no triplets where the source and the target node are the same (i.e. (v,r,v), with $v \in \mathcal{E}$ and $r \in \mathcal{R}$), and let $v \in \mathcal{E}$ be a node in $\mathcal{E}$. First, let us consider the gradient descent operator GD over v's node embedding $\phi[v]$:

$$\mathrm{GD}(\phi, \mathcal{T})[v] = \phi[v] + \alpha \sum_{(\bar{\mathrm{v}},\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{T}} \frac{\partial \log P(\bar{\mathrm{w}} \mid \bar{\mathrm{v}}, \bar{\mathrm{r}})}{\partial \phi[v]}.$$

The gradient is a sum over components associated with the triplets $(\bar{\mathrm{v}},\bar{\mathrm{r}},\bar{\mathrm{w}}) \in \mathcal{T}$; based on whether the corresponding triplet involves v in the subject or object position, or does not involve v at all, these components can be grouped into three categories:

1. Components corresponding to the triplets where $\bar{\mathrm{v}} = v \wedge \bar{\mathrm{w}} \neq v$. The sum over these components is given by:

$$\sum_{(v,\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{T}} \frac{\partial \log P(\bar{\mathrm{w}}\,|v,\bar{\mathrm{r}})}{\partial\phi[v]} = \sum_{(v,\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{T}} \left[\frac{\partial\Gamma(v,\bar{\mathrm{r}},\bar{\mathrm{w}})}{\partial\phi[v]} - \sum_u P(u|v,\bar{\mathrm{r}})\frac{\partial\Gamma(v,\bar{\mathrm{r}},u)}{\partial\phi[v]}\right]$$
$$= \sum_{(\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{N}^1_+[v]} \phi[\bar{\mathrm{w}}]\odot g(\bar{\mathrm{r}}) - \sum_{(v,\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{T}}\sum_u P(u|v,\bar{\mathrm{r}})g(\bar{\mathrm{r}})\odot\phi[u].$$

2. Components corresponding to the triplets where $\bar{\mathrm{v}} \neq v \wedge \bar{\mathrm{w}} = v$. The sum over these components is given by:

$$\sum_{(\bar{\mathrm{v}},\bar{\mathrm{r}},v)\in\mathcal{T}} \frac{\partial \log P(v|\,\bar{\mathrm{v}},\bar{\mathrm{r}})}{\partial\phi[v]} = \sum_{(\bar{\mathrm{v}},\bar{\mathrm{r}},v)\in\mathcal{T}} \left[\frac{\partial\Gamma(\bar{\mathrm{v}},\bar{\mathrm{r}},v)}{\partial\phi[v]} - \sum_u P(u|\,\bar{\mathrm{v}},\bar{\mathrm{r}})\frac{\partial\Gamma(\bar{\mathrm{v}},\bar{\mathrm{r}},u)}{\partial\phi[v]}\right]$$
$$= \sum_{(\bar{\mathrm{v}},\bar{\mathrm{r}})\in\mathcal{N}^1_-[v]} g(\bar{\mathrm{r}})\odot\phi[\bar{\mathrm{v}}]\,(1-P(v|\,\bar{\mathrm{v}},\bar{\mathrm{r}})).$$

3. Components corresponding to the triplets where $\bar{\mathrm{v}} \neq v \wedge \bar{\mathrm{w}} \neq v$. The sum over these components is given by:

$$\sum_{(\bar{\mathrm{v}},\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{T}} \frac{\partial \log P(\bar{\mathrm{w}}\,|\,\bar{\mathrm{v}},\bar{\mathrm{r}})}{\partial\phi[v]} = \sum_{(\bar{\mathrm{v}},\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{T}} \left[0 - \sum_u P(u|\,\bar{\mathrm{v}},\bar{\mathrm{r}})\frac{\partial\Gamma(\bar{\mathrm{v}},\bar{\mathrm{r}},u)}{\partial\phi[v]}\right]$$
$$= \sum_{(\bar{\mathrm{v}},\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{T}} -P(v|\,\bar{\mathrm{v}},\bar{\mathrm{r}})\frac{\partial\Gamma(\bar{\mathrm{v}},\bar{\mathrm{r}},v)}{\partial\phi[v]}.$$
$$= \sum_{(\bar{\mathrm{v}},\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{T}} -P(v|\,\bar{\mathrm{v}},\bar{\mathrm{r}})g(\bar{\mathrm{r}})\odot\phi[\bar{\mathrm{v}}].$$

Collecting these three categories, the GD operator over $\phi[v]$, or rather the node representation update in DistMult, can be rewritten as:

$$\mathrm{GD}(\phi,\mathcal{T})[v] = \phi[v] + \alpha \underbrace{\sum_{\{(\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{N}^1_+[v]\}} \phi[\bar{\mathrm{w}}]\odot g(\bar{\mathrm{r}}) + \sum_{(\bar{\mathrm{r}},\bar{\mathrm{v}})\in\mathcal{N}^1_-[v]} \phi[\bar{\mathrm{v}}]\odot g(\bar{\mathrm{r}})\,(1-P(v|\,\bar{\mathrm{v}},\bar{\mathrm{r}}))}_{v\text{'s neighbourhood}\to v} \quad (16)$$

$$\underbrace{-\alpha \sum_{(\bar{\mathrm{v}},\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{T},\bar{\mathrm{v}}\neq v,\bar{\mathrm{w}}\neq v} P(v|\,\bar{\mathrm{v}},\bar{\mathrm{r}})g(\bar{\mathrm{r}})\odot\phi[\bar{\mathrm{v}}] - \alpha\sum_{(v,\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{T}}\sum_u P(u|v,\bar{\mathrm{r}})g(\bar{\mathrm{r}})\odot\phi[u].}_{\text{beyond neighbourhood}\to v} \quad (17)$$

Note that the component "v's neighbourhood $\to v$" (highlighted in red) in Equation (16) is a sum over v's neighbourhood – gathering information from positive neighbours $\phi[\bar{\mathrm{w}}], (\cdot,\bar{\mathrm{w}})\in\mathcal{N}^1_+[v]$ and negative neighbours $\phi[\bar{\mathrm{v}}], (\cdot,\bar{\mathrm{v}})\in\mathcal{N}^1_-[v]$. Hence, each atomic term of the sum can be seen as a message vector between v and v's neighbouring node. Formally, letting w be v's neighbouring node, the message vector can be written as follows

$$m[v,r,w] = q_{\mathrm{M}}(\phi[v],r,\phi[w]) = \begin{cases}\phi[w]\odot g(r), \text{ if } (r,w)\in\mathcal{N}^1_+[v],\\ \phi[w]\odot g(r)(1-P(v|w,r)), \text{ if } (r,w)\in\mathcal{N}^1_-[v],\end{cases} \quad (18)$$

which induces a bi-directional message function q_M. On the other hand, the summation over these atomic terms (message vectors) induces the aggregate function q_{A}:

$$z[v] = q_{\mathrm{A}}(\{m[v,r,w]\ :\ (r,w)\in\mathcal{N}^1[v]\})$$
$$= \sum_{(\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{N}^1_+[v]} m^l[v,\bar{\mathrm{r}},\bar{\mathrm{w}}] + \sum_{(\bar{\mathrm{r}},\bar{\mathrm{v}})\in\mathcal{N}^1_-[v]} m^l[\bar{\mathrm{v}},\bar{\mathrm{r}},v] = \sum_{(r,w)\in\mathcal{N}^1[v]} m[v,r,w]. \quad (19)$$

Finally, the component "beyond neighbourhood $\to v$" (highlighted in blue) is a term that contains dynamic information flow from global nodes to v. If we define:

$$n[v] = \frac{1}{|\mathcal{T}|} \sum_{(v,\bar{\mathrm{r}},\bar{\mathrm{w}}) \in \mathcal{T}} \sum_{u} P(u|v,\bar{\mathrm{r}}) g(\bar{\mathrm{r}}) \odot \phi[u] + \frac{1}{|\mathcal{T}|} \sum_{(\bar{\mathrm{v}},\bar{\mathrm{r}},\bar{\mathrm{w}}) \in \mathcal{T}, \bar{\mathrm{v}} \neq v, \bar{\mathrm{w}} \neq v} P(v|\,\bar{\mathrm{v}},\bar{\mathrm{r}}) g(\bar{\mathrm{r}}) \odot \phi[\bar{\mathrm{v}}],$$

the GD operator over $\phi[v]$ then boils down to an update function which utilises previous node state $\phi[v]$, aggregated message $z[v]$ and a global term $n[v]$ to produce the new node state:

$$\mathrm{GD}(\phi, \mathcal{T})[v] = q_{\mathrm{U}}(\phi[v], z[v]) = \phi[v] + \alpha z[v] - \beta n[v]. \tag{20}$$

Furthermore, $n[v]$ can be seen as a weighted sum of expectations by recasting the summations over triplets as expectations:

$$n[v] = \frac{|\mathcal{N}_{+}^{1}[v]|}{|\mathcal{T}|} \mathbb{E}_{(v,\bar{\mathrm{r}},\bar{\mathrm{w}}) \sim P_{\mathcal{N}_{+}^{1}[v]}} \mathbb{E}_{u \sim P(\cdot|v,\bar{\mathrm{r}})} g(\bar{\mathrm{r}}) \odot \phi[u] + \frac{|\mathcal{T}^{-v}|}{|\mathcal{T}|} \mathbb{E}_{(\bar{\mathrm{v}},\bar{\mathrm{r}},\bar{\mathrm{w}}) \sim P_{\mathcal{T}^{-v}}} P(v|\,\bar{\mathrm{v}},\bar{\mathrm{r}},) g(\bar{\mathrm{r}}) \odot \phi[\bar{\mathrm{v}}] \tag{21}$$

where $\mathcal{T}^{-v} = \{(\bar{\mathrm{v}},\bar{\mathrm{r}},\bar{\mathrm{v}}') \in \mathcal{T} |\, \bar{\mathrm{v}} \neq v \wedge \bar{\mathrm{v}}' \neq v\}$ is the set of triplets that do not contain v. □

E.1 Extension to AdaGrad and N3 Regularisation

State-of-the-art FMs are often trained with training strategies adapted for each model category. For example, using an N3 regularizer [11] and AdaGrad optimiser [42], which we use for our experiments. For N3 regularizer, we add a gradient term induced by the regularised loss:

$$\frac{\partial L}{\partial \phi[v]} = \frac{\partial L_{\text{fit}}}{\partial \phi[v]} + \lambda \frac{\partial L_{\text{reg}}}{\partial \phi[v]} = \frac{\partial L_{\text{fit}}}{\partial \phi[v]} + \lambda \text{sign}(\phi[v])\phi[v]^2$$

where L_{fit} is the training loss, L_{reg} is the regularisation term, sign$(\cdot)$ is a element-wise sign function, and $\lambda \in \mathbb{R}_{+}$ is a hyper-parameter specifying the regularisation strength. The added component relative to this regularizer fits into the message function as follows:

$$q_{\mathrm{M}}(\phi[v], r, \phi[w]) = \begin{cases} \phi[w] \odot g(r) - \lambda \text{sign}(\phi[w])\phi[w]^2, \text{ if } (r,w) \in \mathcal{N}_{+}^{1}[v], \\ \phi[w] \odot g(r)(1 - P(v|w,r)) - \lambda \text{sign}(\phi[w])\phi[w]^2, \text{ if } (w,r) \in \mathcal{N}_{-}^{1}[v]; \end{cases} \tag{22}$$

Our derivation in Section 3 focuses on (stochastic) gradient descent as the optimiser for training FMs. Going beyond this, complex gradient-based optimisers like AdaGrad use running statistics of the gradients. For example, for an AdaGrad optimiser, the gradient is element-wisely re-scaled by $\frac{1}{\sqrt{s_v + \epsilon}} \nabla_{\phi[v]} L$ where s is the running sum of squared gradients and $\epsilon > 0$ is a hyper-parameter added to the denominator to improve numerical stability. Such re-scaling can be absorbed into the update equation:

$$\text{AdaGrad}(\phi, \mathcal{T})[v] = \phi[v] + (\alpha z[v] - \beta n[v]) * \frac{1}{\sqrt{s[v] + \epsilon}}.$$

F Additional Results on Inductive KGC Tasks

In this paper, we describe the results on FB15K237_v1_ind under some random seed. To confirm the significance and sensitivity, we further experiment with additional 5 random seeds. Due to our computational budget, for this experiment, we resorted to a coarse grid when performing the hyper-parameters sweeps. Following standard evaluation protocols, we report the mean values and standard deviations of the filtered Hits@10 over 5 random seeds. Numbers for Neural-LP, DRUM, RuleN, GraIL, and NBFNet are taken from the literature [9, 17]. "-" means the numbers are not applicable. Table 3 summarises the results. REFACTOR GNNS are able to make use of both types of input features, while textual features benefit both GAT and REFACTOR GNNS for most datasets. Increasing depth benefits WN18RR_vi_ind ($i \in [1, 2, 3, 4]$) most. Future work could consider the impact of textual node features provided by different pretrained language models. Another interesting direction is to investigate the impact of depth on GNNs for datasets like WN18RR, where many kinds of hierarchies are observed in the data.

In addition to the *partial ranking* evaluation protocol, where the ground-truth subject/object entity is ranked against 50 sampled entities,[2] we also consider the *full ranking* evaluation protocol, where the ground-truth subject/object entity is ranked against all the entities. Table 4 summarises the results. Empirically, we observe that *full ranking* is more suitable for reflecting the differences between models than *partial ranking*. It also has less variance than *partial ranking*, since it requires no sampling from the candidate entities. Hence, we believe there is good reason to recommend the community to use *full ranking* for these datasets in the future.

[2]One implementation for such evaluation can be found in `https://github.com/kkteru/grail/blob/master/test_ranking.py#L448`.

Table 3: Hits@10 with Partial Ranking against 50 Negative Samples. "[T]" indicates using textual encodings of entity descriptions [29] as input (positional) node features; "[R]" indicates using frozen random vectors as input (positional) node feature.

	WN18RR				FB15k-237				NELL-995			
	v1	v2	v3	v4	v1	v2	v3	v4	v1	v2	v3	v4
No Pretrain [R]	0.220±0.048	0.226±0.013	0.244±0.020	0.218±0.050	0.215±0.019	0.207±0.008	0.211±0.002	0.205±0.008	0.543±0.022	0.207±0.008	0.216±0.004	0.198±0.006
No Pretrain [T]	0.267±0.020	0.236±0.020	0.292±0.025	0.253±0.022	0.242±0.018	0.227±0.007	0.240±0.011	0.244±0.003	0.538±0.079	0.234±0.017	0.242±0.020	0.191±0.036
Neural-LP	0.744	0.689	0.462	0.671	0.529	0.589	0.529	0.559	0.408	0.787	0.827	0.806
DRUM	0.744	0.689	0.462	0.671	0.529	0.587	0.529	0.559	0.194	0.786	0.827	0.806
RuleN	0.809	0.782	0.534	0.716	0.498	0.778	0.877	0.856	0.535	0.818	0.773	0.614
GAT(3) [R]	0.583±0.022	0.797±0.002	0.560±0.005	0.660±0.015	0.333±0.042	0.312±0.036	0.407±0.072	0.363±0.050	0.906±0.004	0.303±0.031	0.351±0.009	0.187±0.098
GAT(6) [R]	0.850±0.014	0.841±0.001	0.631±0.020	0.802±0.004	0.401±0.020	0.445±0.018	0.461±0.048	0.406±0.143	0.811±0.039	0.670±0.055	0.341±0.042	0.301±0.002
GAT(3) [T]	**0.970±0.002**	0.980±0.001	0.897±0.005	0.960±0.001	0.806±0.003	0.942±0.001	0.941±0.002	0.954±0.001	0.938±0.005	0.839±0.001	0.962±0.001	0.354±0.002
GAT(6) [T]	0.965±0.002	0.986±0.001	0.920±0.002	0.970±0.003	0.826±0.004	0.943±0.001	0.927±0.003	0.927±0.001	0.904±0.000	0.811±0.001	0.880±0.001	0.297±0.003
GraIL	0.825	0.787	0.584	0.734	0.642	0.818	0.828	0.893	0.595	0.933	0.914	0.732
NBFNet	0.948	0.905	0.893	0.890	0.834	0.949	0.951	0.960	-	-	-	-
ReFactorGNN(3) [R]	0.899±0.003	0.842±0.004	0.605±0.000	0.801±0.002	0.673±0.000	0.812±0.002	0.833±0.003	0.877±0.002	0.913±0.000	0.913±0.011	0.893±0.000	0.838±0.002
ReFactorGNN(6) [R]	0.885±0.000	0.854±0.003	0.738±0.006	0.817±0.004	0.787±0.007	0.903±0.003	0.903±0.002	0.920±0.002	0.971±0.007	0.957±0.003	0.935±0.003	0.927±0.001
ReFactorGNN(3) [T]	0.918±0.002	0.973±0.001	0.910±0.003	0.934±0.001	0.900±0.004	0.959±0.001	0.952±0.002	0.968±0.001	**0.955±0.004**	0.931±0.001	0.978±0.001	0.929±0.001
ReFactorGNN(6) [T]	**0.970±0.002**	**0.988±0.001**	**0.944±0.002**	**0.987±0.000**	**0.920±0.001**	**0.963±0.001**	**0.962±0.002**	**0.970±0.002**	0.949±0.011	**0.963±0.001**	**0.994±0.000**	**0.955±0.002**

Table 4: Hits@10 with Full Ranking against All Candidate Entities. "[T]" indicates using textual encodings of entity descriptions [29] as input (positional) node features; "[R]" indicates using frozen random vectors as input (positional) node feature.

	WN18RR				FB15k-237				NELL-995			
	v1	v2	v3	v4	v1	v2	v3	v4	v1	v2	v3	v4
No Pretrain [R]	0.020±0.006	0.004±0.001	0.004±0.003	0.003±0.001	0.013±0.003	0.012±0.001	0.004±0.001	0.002±0.001	0.255±0.021	0.004±0.001	0.001±0.001	0.003±0.001
No Pretrain [T]	0.027±0.009	0.007±0.003	0.006±0.001	0.005±0.001	0.014±0.001	0.010±0.001	0.007±0.001	0.006±0.001	0.262±0.031	0.006±0.002	0.006±0.002	0.003±0.001
GAT(3) [R]	0.171±0.008	0.504±0.026	0.260±0.022	0.089±0.017	0.074±0.003	0.050±0.014	0.051±0.019	0.023±0.012	0.806±0.019	0.003±0.002	0.008±0.007	0.008±0.004
GAT(6) [R]	0.575±0.005	0.698±0.003	0.312±0.000	0.606±0.002	0.048±0.004	0.028±0.004	0.033±0.018	0.015±0.026	0.491±0.112	0.110±0.048	0.031±0.010	0.031±0.002
GAT(3) [T]	0.794±0.000	0.826±0.000	0.468±0.000	0.705±0.000	0.331±0.000	0.585±0.000	0.505±0.000	0.449±0.000	0.856±0.000	0.245±0.000	0.345±0.000	0.078±0.000
GAT(6) [T]	0.815±0.000	0.808±0.000	0.469±0.000	0.701±0.000	0.416±0.000	0.483±0.000	0.391±0.000	0.388±0.000	0.851±0.000	0.189±0.000	0.137±0.000	0.023±0.000
ReFactorGNN(3) [R]	0.826±0.000	0.758±0.002	0.374±0.004	0.707±0.000	0.455±0.010	0.603±0.008	0.556±0.003	0.587±0.003	0.907±0.004	0.700±0.001	0.630±0.001	0.511±0.001
ReFactorGNN(6) [R]	0.826±0.001	0.769±0.005	0.440±0.001	0.731±0.000	0.558±0.007	0.694±0.006	0.639±0.006	0.640±0.000	**0.967±0.005**	**0.764±0.009**	0.697±0.005	**0.703±0.001**
ReFactorGNN(3) [T]	0.805±0.000	0.796±0.003	0.483±0.000	0.682±0.000	0.589±0.001	0.672±0.001	0.610±0.001	0.611±0.001	0.918±0.000	0.629±0.001	0.634±0.000	0.305±0.000
ReFactorGNN(6) [T]	**0.844±0.004**	**0.848±0.003**	**0.522±0.001**	**0.781±0.001**	**0.619±0.000**	**0.721±0.001**	**0.663±0.000**	**0.660±0.000**	0.913±0.000	0.733±0.000	**0.711±0.000**	0.417±0.000

Depth	3	6	∞
Δ Test MRR	0.060	0.045	0.016

Table 5: The Impact of Meaningful Node Feature on *FB15K237_v1*. Δ Test MRR is computed by `test mrr (Roberta Encodings as node features) − test mrr (Random vectors as node features)`. Larger Δ means meaningful node features bring more benefit.

G Additional Results on The Impact of Meaningful Node Features

To better understand the impact meaningful node features have on ReFactor GNNs for the task of knowledge graph completion, we compare ReFactor GNNs trained with Roberta Encodings (one example of meaningful node features) and ReFactor GNNs trained with Random Vectors (not meaningful node features). We perform experiments on *FB15K237_v1* and vary the number of message-passing layers: $L \in \{3, 6, \infty\}$. Table 5 summarises the differences. We can see that meaningful node features are highly beneficial if ReFactor GNNs are only provided with a small number of message-passing layers. As more message-passing layers are allowed, the benefit of ReFactor GNNs diminishes. The extreme case would be $L = \infty$, where the benefit of meaningful node features becomes negligible. We hypothesise that this might be why meaningful node features haven not been found to be useful for transductive knowledge graph completion.

H Additional Results on Parameter Efficiency

Figure 4 shows the parameter efficiency on the dataset *FB15K237_v2*.

I Experimental Details: Setup, Hyper-Parameters, and Implementation

As we stated in the experiments section, we used a two-stage training process. In stage one, we sample subgraphs around query links and serialise them. In stage two, we load the serialised subgraphs and train the GNNs. For transductive knowledge graph completion, we test the model on the same graph (but different splits). For inductive knowledge graph completion, we test the model on the new graph, where the relation vocabulary is shared with the training graph, while the entities are novel. We use the validation split for selecting the best hyper-parameter configuration and report the corresponding test performance. We include reciprocal triplets into the training triplets following standard practice [11].

For subgraph serialisation, we first sample a mini-batch of triplets and then use these nodes as seed nodes for sampling subgraphs. We also randomly draw a node globally and add it to the seed nodes. The training batch size is 256 while the valid/test batch size is 8. We use the LADIES algorithm [26] and sample subgraphs with depths in $[1, 2, 3, 6, 9]$ and a width of 256. For each graph, we keep sampling for 20 epochs, i.e. roughly 20 full passes over the graph.

For general model training, we consider hyper-parameters including learning rates in $[0.01, 0.001]$, weight decay values in $[0, 0.1, 0.01]$, and dropout values in $[0, 0.5]$. For GATs, we use 768 as the hidden size and 8 as the number of attention heads. We train GATs with 3 layers and 6 layers. We also consider whether or not to combine the outputs from all the layers. For ReFactor GNNs, we use the same hidden size as GAT. We consider whether the ReFactor Layer is induced by a SGD operator or by a AdaGrad operator. Within a ReFactor Layer, we also consider the N3 regulariser strength values $[0, 0.005, 0.0005]$, the α values $[0.1, 0.01]$, and the option of removing the $n[v]$, where the message-passing layer only involves information flow within 1-hop neighbourhood as most the classic message-passing GNNs do.

We use grid search to find the best hyper-parameter configuration based on the validation MRR. Each training run is done using two Tesla V100 (16GB) GPUs with, where data parallelism was implemented via the *DistributedDataParallel* component of *pytorch-lightning*. For inductive learning experiments, inference for all the validation and test queries on small datasets like FB15K237_v1 takes about 1-5 seconds, while on medium datasets it takes approximately 20 seconds, and on big datasets like WN18RR_v4 it requires approximately 60 seconds. For most training runs, the memory footprint is less than 40% (13GB). The training time for 20 full passes over the graph is about 1, 7, and 21 minutes respectively for small, medium, and large datasets.

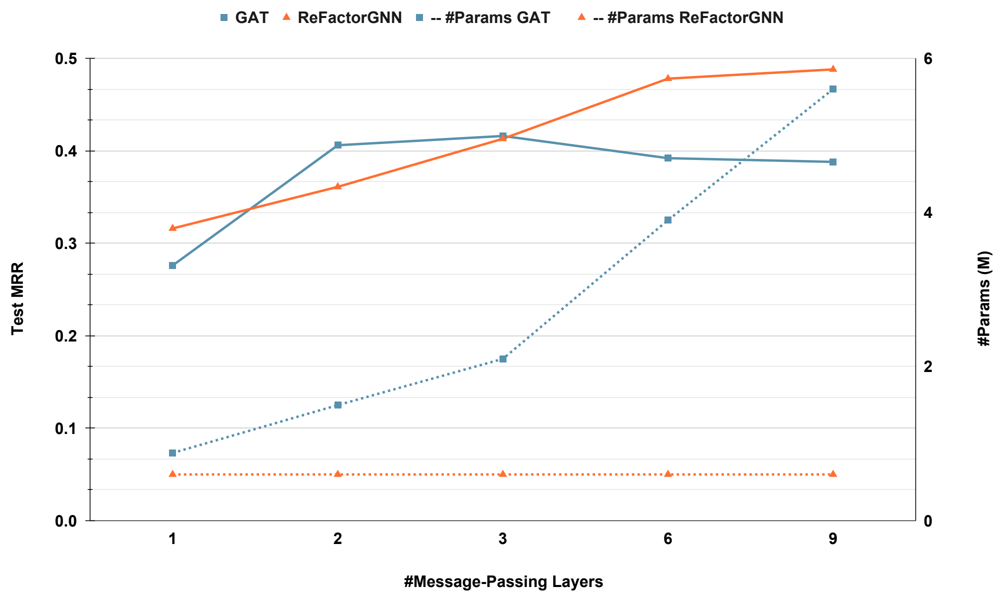

Figure 4: Performance vs Parameter Efficiency as #Layers Increases *FB15K237_v2*. The left axis is Test MRR while the right axis is #Parameters. The solid lines and dashed lines indicate the changes of Test MRR and the changes of #Parameters.

We adapt the LADIES code base for sampling on knowledge graphs[3]. The datasets we use can be downloaded from `https://github.com/villmow/datasets_knowledge_embedding` and `https://github.com/kkteru/grail`. We implement ReFactor GNNs using the `MessagePassing` API[4] in PyG. Specially, we consider using `message_and_aggregate` function[5] to compute the aggregated messages.

[3]`https://github.com/acbull/LADIES`
[4]`https://pytorch-geometric.readthedocs.io/en/latest/notes/create_gnn.html`
[5]`https://pytorch-geometric.readthedocs.io/en/latest/notes/sparse_tensor.html`