A Theorem 1 Proof

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

Theorem A.1 (Message passing in FMs)**.** *The gradient descent operator* GD *(7) on the node embeddings of a DistMult model (Equation (4)) with the maximum likelihood objective in Equation (3) and a multi-relational graph $\mathcal{T}$ defined over entities $\mathcal{E}$ induces a message-passing operator whose composing functions are:*

$$q_{\mathrm{M}}(\phi[v], r, \phi[w]) = \begin{cases} \phi[w] \odot g(r) & \text{if } (r,w) \in \mathcal{N}_+^1[v], \\ (1 - P_\theta(v|w,r))\phi[w] \odot g(r) & \text{if } (r,w) \in \mathcal{N}_-^1[v]; \end{cases} \tag{14}$$

$$q_{\mathrm{A}}(\{m[v,r,w] \,:\, (r,w) \in \mathcal{N}^1[v]\}) = \sum_{(r,w)\in\mathcal{N}^1[v]} m[v,r,w]; \tag{15}$$

$$q_{\mathrm{U}}(\phi[v], z[v]) = \phi[v] + \alpha z[v] - \beta n[v], \tag{16}$$

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

$$n[v] = \frac{|\mathcal{N}_+^1[v]|}{|\mathcal{T}|} \mathbb{E}_{P_{\mathcal{N}_+^1[v]}} \mathbb{E}_{u\sim P_\theta(\cdot|v,r)} g(r) \odot \phi[u] + \frac{|\mathcal{T}^{-v}|}{|\mathcal{T}|} \mathbb{E}_{P_{\mathcal{T}^{-v}}} P_\theta(v|s,r) g(r) \odot \phi[s], \tag{17}$$

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 $\phi[v]$:

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

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 of these components is given by:

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

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

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

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

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

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

$$\mathrm{GD}(\phi,\mathcal{T})[v] = \phi[v] + \alpha \underbrace{\sum_{\{(\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{N}_+^1[v]\}} \phi[\bar{\mathrm{w}}]\odot g(\bar{\mathrm{r}}) + \sum_{(\bar{\mathrm{r}},\bar{\mathrm{v}})\in\mathcal{N}_-^1[v]} \phi[\bar{\mathrm{v}}]\odot g(\bar{\mathrm{r}})\,(1-P(v|\,\bar{\mathrm{v}},\bar{\mathrm{r}}))}_{v\text{'s neighbourhood}\to v} \tag{18}$$

$$\underbrace{-\alpha \sum_{(\bar{\mathrm{v}},\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{T},\bar{\mathrm{v}}\neq v,\bar{\mathrm{w}}\neq v} P(v|\,\bar{\mathrm{v}},\bar{\mathrm{r}})g(\bar{\mathrm{r}})\odot\phi[\bar{\mathrm{v}}] - \alpha \sum_{(v,\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{T}}\sum_u P(u|v,\bar{\mathrm{r}})g(\bar{\mathrm{r}})\odot\phi[u]}_{\text{beyond neighbourhood}\to v}. \tag{19}$$

Note that the component "v's neighbourhood $\to v$" (highlighted in red) in Equation (18) 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

$$m[v,r,w] = q_{\mathrm{M}}(\phi[v],r,\phi[w]) = \begin{cases}\phi[w]\odot g(r), \text{ if } (r,w)\in\mathcal{N}_+^1[v],\\ \phi[w]\odot g(r)(1-P(v|w,r)), \text{ if } (r,w)\in\mathcal{N}_-^1[v],\end{cases} \tag{20}$$

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}:

$$\begin{aligned} z[v] &= q_{\mathrm{A}}(\{m[v,r,w] \,:\, (r,w)\in\mathcal{N}^1[v]\}) \\ &= \sum_{(\bar{\mathrm{r}},\bar{\mathrm{w}})\in\mathcal{N}_+^1[v]} m^l[v,\bar{\mathrm{r}},\bar{\mathrm{w}}] + \sum_{(\bar{\mathrm{r}},\bar{\mathrm{v}})\in\mathcal{N}_-^1[v]} m^l[\bar{\mathrm{v}},\bar{\mathrm{r}},v] = \sum_{(r,w)\in\mathcal{N}^1[v]} m[v,r,w]. \end{aligned} \tag{21}$$

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:

$$\mathrm{GD}(\phi,\mathcal{T})[v] = q_{\mathrm{U}}(\phi[v],z[v]) = \phi[v] + \alpha z[v] - \beta n[v]. \tag{22}$$

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

$$n[v] = \frac{|\mathcal{N}_+^1[v]|}{|\mathcal{T}|}\mathbb{E}_{(v,\bar{\mathrm{r}},\bar{\mathrm{w}})\sim P_{\mathcal{N}_+^1[v]}}\mathbb{E}_{u\sim P(\cdot|v,\bar{\mathrm{r}})}g(\bar{\mathrm{r}})\odot\phi[u] + \frac{|\mathcal{T}^{-v}|}{|\mathcal{T}|}\mathbb{E}_{(\bar{\mathrm{v}},\bar{\mathrm{r}},\bar{\mathrm{w}})\sim P_{\mathcal{T}^{-v}}}P(v|\,\bar{\mathrm{v}},\bar{\mathrm{r}},)g(\bar{\mathrm{r}})\odot\phi[\bar{\mathrm{v}}] \tag{23}$$

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. □

A.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 [17] and AdaGrad optimiser [4], which we use for our experiments. For N3 regularizer, we add a gradient term induced by the regularised loss:

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

where L_{fit} is the training loss, L_{reg} is the regularisation term, $\mathrm{sign}(\cdot)$ 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:

$$q_{\mathrm{M}}(\phi[v],r,\phi[w]) = \begin{cases}\phi[w]\odot g(r) - \lambda\mathrm{sign}(\phi[w])\phi[w]^2, \text{ if } (r,w)\in\mathcal{N}_+^1[v],\\ \phi[w]\odot g(r)(1-P(v|w,r)) - \lambda\mathrm{sign}(\phi[w])\phi[w]^2, \text{ if } (w,r)\in\mathcal{N}_-^1[v];\end{cases} \tag{24}$$

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:

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

In general, we can interpret any auxiliary variable introduced by the optimizer (e.g. the velocity) as an additional part of the entities and relations representations on which message passing happens. However, the specific equations would depend on the optimizer's dynamics and would be hard to formally generalise.

A.2 Extensions to Other Score Functions e.g. ComplEx

The two main design choices in Theorem A.1 are 1) the score function Γ, and 2) the optimization dynamics over the node embeddings. In the paper, we chose DistMult and GD because of their mathematical simplicity, leading to easier-to-read formulas. We can adapt the theorem to general, smooth scoring functions $\Gamma : .\mathcal{E} \times \mathcal{R} \times \mathcal{E} \to \mathbf{R}$ by replacing occurrences of the gradient of DistMult with a generic $\nabla\Gamma$ (the gradient of DistMult w.r.t. $\phi[v]$ at (v, r, w) is simply $g(r) \odot \phi[w]$). This gives us the following lemma:

Lemma A.2 (Message passing in FMs)**.** *The gradient descent operator* GD *(7) on the node embeddings of a general score function with the maximum likelihood objective in Equation* (3) *and a multi-relational graph $\mathcal{T}$ defined over entities $\mathcal{E}$ induces a message-passing operator whose composing functions are:*

$$q_{\text{M}}(\phi[v], r, \phi[w]) = \begin{cases} \nabla_{\phi[v]}\Gamma(v, r, w) & \text{if } (r, w) \in \mathcal{N}_+^1[v], \\ (1 - P_\theta(v|w, r))\nabla_{\phi[v]}\Gamma(w, r, v) & \text{if } (r, w) \in \mathcal{N}_-^1[v]; \end{cases} \tag{25}$$

$$q_{\text{A}}(\{m[v, r, w] \,:\, (r, w) \in \mathcal{N}^1[v]\}) = \sum_{(r,w)\in\mathcal{N}^1[v]} m[v, r, w]; \tag{26}$$

$$q_{\text{U}}(\phi[v], z[v]) = \phi[v] + \alpha z[v] - \beta n[v], \tag{27}$$

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

$$n[v] = \frac{|\mathcal{N}_+^1[v]|}{|\mathcal{T}|}\mathbb{E}_{P_{\mathcal{N}_+^1[v]}}\mathbb{E}_{u\sim P_\theta(\cdot|v,r)}\nabla_{\phi[v]}\Gamma(v, r, u) + \frac{|\mathcal{T}^{-v}|}{|\mathcal{T}|}\mathbb{E}_{P_{\mathcal{T}^{-v}}}P_\theta(v|s, r)\nabla_{\phi[v]}\Gamma(s, r, v), \tag{28}$$

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

Accordingly, the node representation updating equations in Section 3.1 can be re-written as follows

$$\text{GD}(\phi, \{(v, r, w)\})[v] = \phi[v] + \alpha \left(\underbrace{\nabla_{\phi[v]}\Gamma(v, r, w)}_{w\to v} \underbrace{- \sum_{u\in\mathcal{E}} P_\theta(u|v, r)\nabla_{\phi[v]}\Gamma(v, r, u)}_{u\to v} \right),$$

$$\text{GD}(\phi, \{(v, r, w)\})[w] = \phi[w] + \alpha \underbrace{(1 - P_\theta(w|v, r))\, \nabla_{\phi[w]}\Gamma(v, r, w)}_{v\to w},$$

$$\text{GD}(\phi, \{(v, r, w)\})[u] = \phi[u] + \alpha \left(\underbrace{-P_\theta(u|v, r)\nabla_{\phi[u]}\Gamma(v, r, u)}_{v\to u} \right).$$

$\nabla_{\phi[\cdot]}\Gamma$ can be different for different models. For example, here we offer a specific derivation for ComplEx [39]. Let $d = K/2$ be the hidden size for ComplEx. The ComplEx score function is given as follows

$$\begin{aligned}\Gamma(v, r, w) =& < \psi[r]_{(0:d)}, \phi[v]_{(0:d)}, \phi[w]_{(0:d)} > + < \psi[r]_{(0:d)}, \phi[v]_{(d:)}, \phi[w]_{(d:)} > \\ &+ < \psi[r]_{(d:)}, \phi[v]_{(0:d)}, \phi[w]_{(d:)} > - < \psi[r]_{(d:)}, \phi[v]_{(d:)}, \phi[w]_{(0:d)} >\end{aligned} \tag{29}$$

where $(0:d)$ indicates the real part of the complex vector and $(d:)$ indicates the image part of the complex vector. The gradients of the ComplEx score function with respect to the real/image node representations are given by $\frac{\partial\Gamma(v,r,w)}{\partial\phi[v]_{(0:d)}} = \psi[r]_{(0:d)} \odot \phi[w]_{(0:d)} + \psi[r]_{(d:)} \odot \phi[w]_{(d:)}$, $\frac{\partial\Gamma(v,r,w)}{\partial\phi[v]_{(d:)}} = \psi[r]_{(0:d)} \odot \phi[w]_{(d:)} - \psi[r]_{(d:)} \odot \phi[w]_{(0:d)}$, $\frac{\partial\Gamma(v,r,w)}{\partial\phi[w]_{(0:d)}} = \psi[r]_{(0:d)} \odot \phi[v]_{(0:d)} - \psi[r]_{(d:)} \odot \phi[v]_{(d:)}$, $\frac{\partial\Gamma(v,r,w)}{\partial\phi[w]_{(d:)}} = \psi[r]_{(0:d)} \odot \phi[v]_{(d:)} + \psi[r]_{(d:)} \odot \phi[v]_{(0:d)}$. Concatenating gradients for the real part and the image part, we have the gradients

$$\nabla_{\phi[v]}\Gamma(v, r, w) = \frac{\partial\Gamma(v, r, w)}{\partial\phi[v]_{(0:d)}} \parallel \frac{\partial\Gamma(v, r, w)}{\partial\phi[v]_{(d:)}},$$

$$\nabla_{\phi[w]}\Gamma(v, r, w) = \frac{\partial\Gamma(v, r, w)}{\partial\phi[w]_{(0:d)}} \parallel \frac{\partial\Gamma(v, r, w)}{\partial\phi[w]_{(d:)}}.$$

B 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 [37, 49]. "-" means the numbers are not applicable. Table 4 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_v*i*_ind ($i \in [1, 2, 3, 4]$) most. Future work could consider the impact of textual node features provided by different pre-trained 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,[5] we also consider the *full ranking* evaluation protocol, where the ground-truth subject/object entity is ranked against all the entities. Table 5 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.

C Additional Results on The Impact of Meaningful Node Features

To better understand the impact that 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 3 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.

[5] One implementation for such evaluation can be found in GraIL's codebase.

Depth	3	6	∞
Δ Test MRR	0.060	0.045	0.016

Table 3: The Impact of Meaningful Node Feature on *FB15K237_v1*. Δ Test MRR is computed by `test mrr (textual node features)` $-$ `test mrr (random node features)`. Larger Δ means meaningful node features bring more benefit.

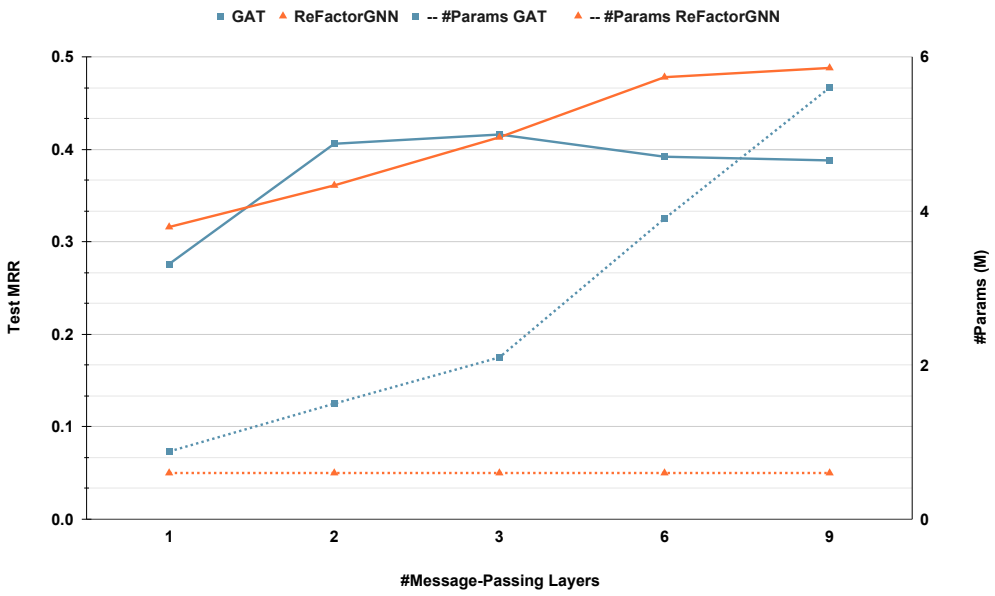

Figure 4: Performance vs Parameter Efficiency as #Layers Increases on *FB15K237_v2*. The left axis is Test MRR while the right axis is #Parameters. The solid lines and dashed lines indicate the changes of Test MRR and the changes of #Parameters.

Table 4: 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 5: 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

D Additional Results on Parameter Efficiency

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

E Discussion on Complexity

We can analyse the scalability of REFACTOR GNNS along three axes, the number of layers L, the embedding size d, and the number of triplets/edges in the graph $|\mathcal{T}|$. For scalability w.r.t. to the number of layers, let L denote the number of message-passing layers. Since REFACTOR GNNS tie the weights across the layers, the parameter complexity of REFACTOR GNNS is $\mathcal{O}(1)$, while it is $\mathcal{O}(L)$ for standard GNNs such as GATs, GCNs, and R-GCNs. Additionally, since REFACTOR GNNS adopt layer-wise training enabled via the external memory for node state caching, the training memory footprint is also $\mathcal{O}(1)$ as opposed to $\mathcal{O}(L)$ for standard GNNs. For scalability w.r.t the embedding size, let d denote the embedding size. REFACTOR GNNS scale linearly with d, as opposed to most GNNs in literature where the parameter and time complexities scale quadratically with d. For scalability w.r.t. the number of triplets/edges in the graph, we denote the entity set as $\mathcal{E}$, the relation set as $\mathcal{R}$, and the triplets as $\mathcal{T}$. NBFNet requires $O(LT^2d + L\mathcal{T}Vd^2)$ inference run-time complexity since the message-passing is done for every source node and query relation – quadratic w.r.t the number of triplets $\mathcal{T}$ while REFACTOR GNNS are of linear complexity w.r.t $\mathcal{T}$. Extending the complexity analysis in NBFNet [49] to all the triplets, we include a detailed table for complexity comparison in Table 6. The inference complexity refers to the cost per forward pass over the entire graph.

	Parameter Complexity	Training Memory Complexity	Inference Memory Complexity	Training Time Complexity	Inference Time Complexity
GAT	$O(Ld^2)$	$O(L\|V\|d)$	$O(L\|V\|d)$	$O(L\|V\|d^2 + L\|T\|d)$	$O(L\|V\|d^2 + L\|T\|d)$
R-GCN	$O(L\|R\|d^2)$	$O(L\|V\|d)$	$O(L\|V\|d)$	$O(L\|T\|d^2 + L\|V\|d^2)$	$O(L\|T\|d^2 + L\|V\|d^2)$
NBFNet	$O(L\|R\|d^2)$	$O(L\|V\|\|T\|d)$	$O(L\|V\|\|T\|d)$	$O(L\|T\|^2d + L\|T\|\|V\|d^2)$	$O(L\|T\|^2d + L\|T\|\|V\|d^2)$
REFACTOR GNNS	$O(\|R\|d)$	$O(\|V\|d)$	$O(L\|V\|d)$	$O(\|T\|\|V\|d)$	$O(L\|T\|\|V\|d)$

Table 6: Complexity Comparison.

F Discussion on Expressiveness of FMs, GNNs and REFACTOR GNNS

We envision one interesting branch of future work would be a unified framework of expressiveness for all three model categories: FMs, GNNs and REFACTOR GNNS. To the best of our knowledge, there are currently two separate notions of expressiveness, one for FMs and the other for GNNs. While these two notions of expressiveness are both widely acclaimed within their own communities, it is unclear how to bridge them and produce a new tool supporting the analysis of the empirical applications (REFACTOR GNNS) that seam the two communities.

Fully Expressiveness for Adjacency Recovery. In the FM community, a FM is said to be *fully expressive* [13] if, for any given graph $\mathcal{T}$ over entities $\mathcal{E}$ and relations $\mathcal{R}$, it can fully reconstruct the input adjacency tensor with a embedding size bounded by $\min(|\mathcal{E}||\mathcal{R}|, |\mathcal{T}| + 1)$. We can generalise this expressiveness analysis to the spectrum of FM-GNN models (REFACTOR GNNS). In the $L \to \infty$ limit, REFACTOR GNNS are as fully expressive as the underlying FMs. In fact, a REFACTOR GNN based on DistMult [45] is not fully expressive (because of its symmetry); however a REFACTOR GNN based, e.g. on ComplEx [39, 17] can reach full expressiveness for $L \to \infty$.

Weisfeiler-Leman Tests for Nodes/Graphs Separation. For GNNs, established results concern the separation power of induced representations in terms of Weisfeiler-Leman (WL) isomorphism tests [42, 6]. However, none of these results is directly applicable to our setting (e.g. they only consider one relationship). Nevertheless, if we consider our REFACTOR GNNS in a one-relationship, simple graph setting, following the formalism of [6], we note that the REFACTOR Layer function cannot be written in Guarded Tensor Language since at each layer it computes a global term $n[v]$. Moreover, REFACTOR GNNS only process information coming from two nodes at one time. These two facts imply that REFACTOR GNNS have a separation power upper bound comparable to the 1-WL test, i.e. comparable to 1-MPNN (not guarded).

We are not aware of explicit connections between the two above notions of expressiveness. We think there is some possibility that we can bridge them, which itself will be a very interesting research direction, but would require a very substantial amount of additional work and presentation space and is thus beyond the scope of this paper.

Alternatively, we can also increase the expressiveness of REFACTOR GNNS by adding more parameters to the message, aggregation and update operators. For example, introducing additional MLPs to transform the input node features or include non-linearity in the GNN update operator. This would be a natural way to increase the expressiveness of REFACTOR GNNS.

Another method for increasing expressive power for link prediction task only is to extend ReFactor GNNs from node-wise to pairwise (Sec 2.2 in our paper) representations like GraIL [37] and NBFNet [49], which is more computationally intensive, but yields more powerful as node representations are not standalone but adapted to a specific query.

G 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 [17].

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 [50] 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.

Our code will be available at ReFactorGNN. We adapted the LADIES subgraph sampler from the GPT-GNN codebase [11] for sampling on knowledge graphs. The datasets we used can be downloaded from the repositories Datasets for Knowledge Graph Completion with Textual Information about Entities and GraIL - Graph Inductive Learning. We implemented REFACTOR GNNS using the *MessagePassing* API in *PyTorch Geometric*. Specially, we used *message_and_aggregate* function to compute the aggregated messages.

H Potential Negative Societal Impact

Our work focus on efficient reasoning over knowledge graphs. A potential negative societal impact is that some people might use the methods for inferring private information using their own collected knowledge graphs. However, this issue is also commonly faced by any other research work on knowledge graph reasoning.