gHAWK: Structural Encoding for Scalable Training of Graph Neural Networks on Knowledge Graphs

Humera Sabir* **Fatima Farooq*** **Ashraf Aboulnaga**
Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, TX, USA
{humera.sabir,fatima.farooq,ashraf.aboulnaga}@uta.edu

Abstract

Knowledge Graphs (KGs) are a rich source of structured data, and graph neural networks (GNNs) are the dominant tool for learning over them. However, existing message-passing GNNs struggle to scale to large KGs because they rely on the iterative message passing process to learn graph structure. This process is inefficient, especially under mini-batch training, where a node sees only a partial view of its neighborhood. In this paper, we address this problem and present gHAWK, a novel and scalable GNN training framework for large KGs. The key idea is to precompute structural features for each node that capture its local and global structure before GNN training even begins. Specifically, gHAWK introduces a preprocessing step that computes: (a) Bloom filters to compactly encode local neighborhood structure, and (b) TransE embeddings to represent each node's global position in the graph. These features are then fused with any domain-specific features (e.g., text embeddings), producing a node feature vector that can be used with any GNN backbone. Equipped with these structural priors, the GNN no longer needs to rediscover graph structure, significantly improving convergence and model accuracy with little memory overhead. Extensive experiments on large datasets from the Open Graph Benchmark demonstrate that gHAWK improves accuracy on both node property prediction and link prediction tasks. Remarkably, gHAWK enables simple GNNs to match or exceed more complex relation-aware ones, suggesting that careful preprocessing can substitute for architectural complexity in KG learning.

1 Introduction

Knowledge graphs (KGs) represent real-world entities and the relations among them as (head, relation, tail) triples. They support applications such as question answering [19, 30, 24], recommendation systems [17], information retrieval [41, 8], and biomedical reasoning [2]. To leverage the rich, heterogeneous structure in a KG, a first step in many applications is to train a machine learning model that learns representations. Graph neural networks (GNNs) have emerged as a powerful class of models for this step, which learn vector representations of entities, also known as embeddings. At each layer, information from each entity's neighborhood is aggregated through *message passing* [15, 34]: every node updates its embedding by combining messages from its neighbors [32, 36, 7]. By aggregating this structural information across multiple layers, GNNs improve predictive accuracy on downstream tasks such as node property prediction and link prediction.

While GNNs have demonstrated remarkable predictive accuracy, training them on large-scale KGs poses serious scalability challenges, and three properties of KGs make these challenges particularly acute. First, KGs are heterogeneous with high structural diversity: a single graph may contain hundreds of distinct relation types [37], each carrying different semantics, which relation-aware GNNs encode through per-relation parameters [32, 36]. The parameter count grows linearly in the number of relations, increasing memory cost and slowing convergence. Second, full-batch training is infeasible at the scale of millions of entities, so training relies on mini-batches that sample only a

*Equal contribution.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

fraction of each node's neighborhood per step [18, 5, 45]. Sampling discards exactly the structural information the GNN is trying to learn. Third, message passing is local: at each layer, every node can only access information from its immediate neighbors, so capturing structure beyond one hop requires multiple message-passing iterations, and each iteration adds memory and compute cost. Taken together, these three factors mean that **GNNs must reconstruct KG structure during training, one hop at a time, even though that structure is already explicit in the input graph.**

We argue that GNN training on large KGs requires *precomputed structural priors*: features computed once over the input graph that encode each entity's structural context before GNN training starts. Our insight is to **shift the complexity of capturing KG structure from the message-passing stage to a lightweight preprocessing step.** The design question becomes: *what structural information should this prior carry?* We show that two complementary priors are sufficient. The *local* feature vector captures an entity's 1-hop neighborhood, recovering information that mini-batch sampling discards. The *global* feature vector captures the entity's position in graph, providing relational context that message passing would otherwise need many layers to propagate. Together, these priors give each entity a representation of both its local and distant structure, available before the first training step.

We instantiate this design in gHAWK[2] (*Hybrid Aggregation of local and global structure for Web-scale KGs*), a framework for KG learning that combines local and global structural priors with any GNN architecture and any downstream prediction task (Fig. 1). For the local feature, gHAWK uses a *Bloom filter* [1] that compactly encodes each entity's 1-hop neighbors. For the global feature, gHAWK uses a pre-trained knowledge graph embedding (KGE) model, specifically *TransE* [3], that captures each entity's global relational position. gHAWK fuses these two structural vectors with a learned entity embedding, and with any domain-specific node features (e.g., text embeddings in bibliographic KGs), through a learned gating mechanism. The resulting dense vector serves as input either to a GNN backbone or directly to a task-specific decoder. The framework can accommodate alternative local priors (relation-distribution histograms, MinHash) or global priors (DistMult, ComplEx) and we show that they yield comparable performance. We adopt Bloom + TransE as default because both are simple to compute and provide a compact local signature and an interpretable positional encoding. While precomputed node features are not new, we are not aware of any prior work that combines local and global structural priors in this way to scale GNN training on large heterogeneous KGs.

We show that gHAWK improves accuracy on multiple benchmarks for two tasks: node property prediction and link prediction. In summary, our contributions are:

A scalable framework that decomposes structure into local and global priors. gHAWK computes two complementary fixed-size structural features for each entity, instantiated by default with a Bloom filter and a pre-trained TransE embedding. A learned gating mechanism fuses these frozen priors with the entity embedding, allowing the model to balance the precomputed structural signal against task-specific learning during training.

Structural priors substitute for deep message passing across tasks and backbones. Across both node property prediction and link prediction, simple architectures equipped with gHAWK features match or surpass message-passing alternatives. On node property prediction over OGB-MAG and MAG240M [20, 21], gHAWK consistently improves accuracy across six GNN backbones spanning relation-aware, relation-agnostic, sampling-based, and attention-based architectures. On link prediction over OGB-WikiKG2, gHAWK features with a RotatE decoder–using no message passing at all–improve test MRR by 26 percentage points over the strongest embedding-only baseline and exceed specialized query-conditioned methods such as A*Net [50]. On OGB-BioKG, a biomedical KG with type-constrained evaluation, gHAWK improves over RotatE by up to 9.6 MRR points on clinically important relation types (drug–disease, drug–drug, drug–side-effect), where the base model is weakest. Stratified analysis traces both gains to gHAWK's effectiveness at capturing the long-tail: rare entities on WikiKG2 and difficult relations on BioKG. Where learned embeddings provide weak signal, structural priors restore predictive capacity.

A theoretical understanding of the structural-prior paradigm. Beyond gHAWK's empirical results, our analysis reveals a fundamental property of query-independent KG representations: certain entity pairs are structurally indistinguishable from any feature defined over neighborhoods alone, regardless of how the feature is constructed. We call these doppelgangers, and they create a ceiling for the accuracy of query-independent methods, which we analyze in Sec. 5.4 and App. F.

[2] `https://github.com/lids-lab/gHAWK`

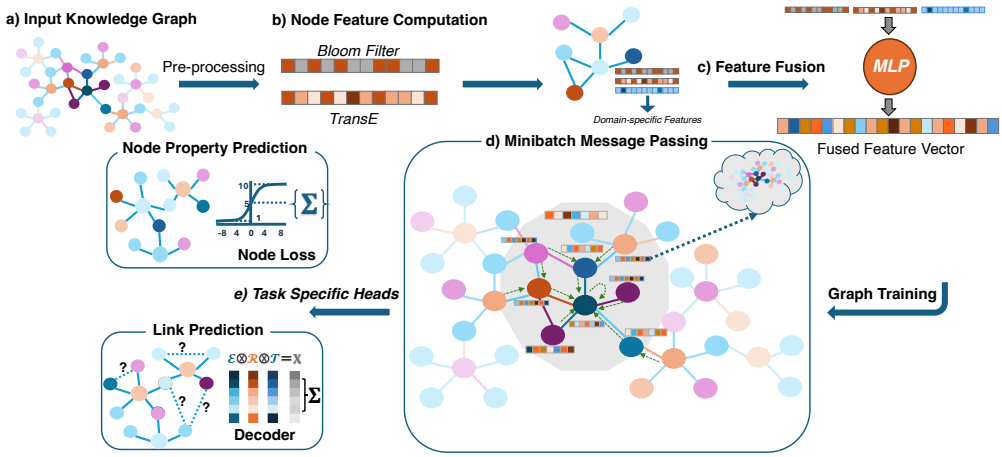


Figure 1: gHAWK pipeline. (a) Input knowledge graph. (b) Node feature computation: in a preprocessing step, gHAWK constructs a Bloom filter encoding the 1-hop neighbors of each node and trains a TransE embedding model to capture the global graph structure. (c) Feature fusion: for every node, the Bloom filter, TransE embedding, and any domain-specific feature vector are fused via an MLP trained with task supervision, creating a dense fused feature vector. (d) Mini-batch message passing: the fused vectors initialize a message-passing GNN that is trained on sampled mini-batch subgraphs of the knowledge graph. (e) Task-specific heads: the resulting node embeddings are fed into separate heads for node property prediction and for link prediction.

2 Background

A knowledge graph (KG) is $G = (\mathcal{E}, \mathcal{R}, \mathcal{T})$ with entities $\mathcal{E}$, relation types $\mathcal{R}$, and triples $\mathcal{T} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$. Each triple (h, r, t) is a directed edge from a head entity h to a tail entity t labeled with a relation type r. For an entity e, we write $\mathcal{N}(e) = \{(r, t) : (e, r, t) \in \mathcal{T}\} \cup \{(r, h) : (h, r, e) \in \mathcal{T}\}$ for its 1-hop neighborhood combining outgoing and incoming edges.

Set signatures. gHAWK encodes each entity's 1-hop neighborhood with a compact, fixed-size set signature, which lets every entity carry its full neighborhood through training as a constant-size vector regardless of how many neighbors it has. We consider two well-established structures. Our default is a *Bloom filter* [1], an m-bit array with k independent hash functions; inserting an element sets k bits, yielding a fixed-size signature with no false negatives and a tunable false-positive rate. We also test *MinHash* [4], which maintains k minimum-valued hashes of inserted elements, preserving Jaccard similarity at $O(k)$ comparison cost. Both produce fixed-size summaries of arbitrary-sized sets, which is what we need to represent neighborhoods of widely varying degree in a single-sized feature vector.

Knowledge graph embeddings. KGE models learn a vector for each entity and each relation by optimizing a scoring function over $\mathcal{T}$. **TransE** [3] models relations as translations: $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$. **DistMult** [44] uses a diagonal bilinear scoring that is symmetric in $\mathbf{h}$ and $\mathbf{t}$. **ComplEx** [35] extends DistMult with complex-valued embeddings to capture asymmetric relations. **RotatE** [33] models relations as rotations in complex space and is widely used as a link-prediction decoder. KGE models train efficiently at scale and produce vectors that summarize each entity's relational role in the graph.

3 The gHAWK Framework

Two design decisions distinguish the gHAWK pipeline in Fig. 1 from prior preprocessing-based KG methods. First, gHAWK *decomposes* structure into a local feature (the 1-hop neighborhood) and a global feature (the entity's position in the relational graph), because no single feature simultaneously captures both kinds of information at a fixed size. Second, gHAWK fuses these features with the learned embedding through a single scalar gate shared across all entities, which forces the model to learn a global balance between structural priors and learned refinement rather than per-entity weights.

For each entity $e \in \mathcal{E}$, gHAWK computes three inputs: a local feature $f_L(e)$, a global feature $f_G(e)$, and a learned embedding $E(e)$ updated during task training. The two structural inputs f_L and f_G are computed once, in preprocessing, from the training triples $\mathcal{T}$ alone, a point we return to in Sec. 3.3. We describe the three options we try for f_L (Sec. 3.1), the three options for f_G (Sec. 3.2), and the fusion mechanism that combines them into a single representation $z(e)$ (Sec. 3.3). Sec. 4 then describes how $z(e)$ enters training, including the choice of backbone and task-specific decoder.

3.1 Local Structural Prior: Neighborhood Encoding

The local structural prior summarizes each entity's immediate relational context. Encoding the 1-hop neighborhood $\mathcal{N}(e)$ as a fixed-size vector requires choosing what information to preserve and what to compress, and gHAWK considers three encodings that span this trade-off. The *Bloom filter signature* hashes each $(r, x) \in \mathcal{N}(e)$ as the string "r_x" into an m-bit array using k hash functions, preserving the relation r as well as the neighbor identity x. App. B.1 gives the full construction procedure. gHAWK defaults to the Bloom filter because it is simple and informative, but we also report results for MinHash and RelDist to show that the framework can accommodate other local features.

3.2 Global Structural Prior: Positional Encoding

The global feature situates each entity in a continuous vector space whose geometry reflects the full graph's relational structure. gHAWK computes this feature with a pre-trained KGE model: the chosen KGE is trained on the full set of training triples $\mathcal{T}$ before gHAWK's downstream task training begins, and its entity vectors are then frozen. Any KGE that produces fixed-size entity vectors fits the framework, but our default is TransE [3] since it fits the global feature's purpose.

The downstream model needs each entity's coordinate in a space that reflects the graph's relational structure as a whole, not a standalone scorer for triples.

TransE's translational geometry fits this role naturally: each entity receives a single vector whose distance from other entities reflects relational proximity, and these distances are interpretable as inputs to the fusion mechanism. More expressive KGEs, such as DistMult and ComplEx, model relations as relation-specific transformations (diagonal matrices and complex rotations, respectively), so their entity vectors are interpreted only in combination with those transformations. This makes them less natural as standalone position vectors, though we evaluate them as alternatives to test gHAWK's robustness.

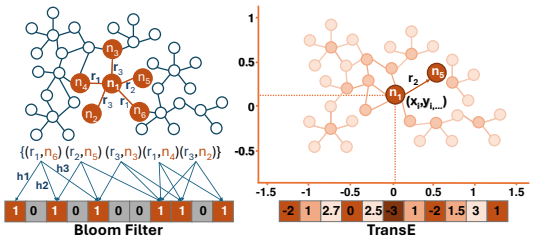


Figure 2: Local and global priors in gHAWK. (Left) A Bloom filter hashes each (r, n) pair of an entity's 1-hop neighborhood into k bits. (Right) TransE places each entity at a point whose position reflects global relational structure.

3.3 Gated Feature Fusion

gHAWK combines the local feature, global feature, learned embedding, and any domain-specific feature in two stages. In the first stage, each input feature is projected into a common d-dimensional latent space by a dedicated two-layer MLP with ReLU activation and dropout, then the projected vectors are concatenated and passed through a fusion MLP to produce a single structural representation:

$$h_{\text{struct}}(e) = \text{MLP}_{\text{fuse}}\Big(\text{MLP}_L(f_L(e)) \,\|\, \text{MLP}_G(f_G(e)) \,\|\, \text{MLP}_X(x(e))\Big), \tag{1}$$

where $\|$ denotes concatenation and $x(e)$ is an optional domain-specific feature. The output $h_{\text{struct}}(e) \in \mathbb{R}^d$ has the same dimension as the learned embedding $E(e)$. In the second stage, the structural representation is combined with $E(e)$ through a single scalar gate $g \in [0, 1]$:

$$z(e) = g \cdot E(e) + (1 - g) \cdot h_{\text{struct}}(e). \tag{2}$$

The gate is learned end-to-end from task supervision and is shared across all entities. A per-entity gate, while more flexible, would let the model memorize a different balance for each entity, defeating the purpose of structural priors as a generalizing signal. A single shared scalar instead forces the model to learn one balance that applies uniformly across the graph.

During downstream training, $f_L(e)$, $f_G(e)$, and $x(e)$ remain frozen. Only the learned embedding $E(e)$, the projection MLPs, the fusion MLP, the gate, and the task-specific model parameters are updated. This separation keeps the structural priors reusable across tasks and backbones, and it bounds training-time memory by the active mini-batch rather than the full graph.

Data hygiene. Every structural prior is computed during preprocessing from the training triples $\mathcal{T}$ alone. The local signatures (Sec. 3.1) are built in a single pass over $\mathcal{T}$ (App. B.1), the global KGE (Sec. 3.2) is trained on $\mathcal{T}$ and then frozen. Negatives are drawn uniformly at random. No held-out triple is ever inserted, scored, or sampled. Validation triples serve only early stopping and model

selection, and test triples only the final evaluation under the official splits. The two tasks differ only in what is held out: for link prediction it is a set of edges, none seen during feature construction; for node property prediction it is node labels, while the graph structure is shared across splits as it is for every baseline, and no label enters a structural feature. This boundary is what makes the reported gains attributable to generalization rather than leakage.

4 Training

The fused representation $z(e)$ produced by gHAWK is task-agnostic: the same representation drives both node property prediction and link prediction, and task-specific behavior is introduced only through the choice of backbone and decoder.

4.1 Computation and storage

The structural priors (Bloom filters, KGE vectors, domain-specific features) are stored as fixed-size tables in CPU memory and transferred to the GPU only for entities in the current mini-batch; no gradients flow to the frozen tables. GPU memory is therefore bounded by the active mini-batch size, enabling training at MAG240M scale (244M entities).

4.2 Backbone GNN Configurations

Relation-aware GNNs (R-GCN, CompGCN) maintain per-relation parameters and model relational semantics with high fidelity. Their parameter count grows as $\mathcal{O}(|\mathcal{R}|d^2)$, which scales poorly when $|\mathcal{R}|$ is large, and gHAWK reduces this burden by providing a structurally informative input representation that allows the GNN to reach strong accuracy with shallower or narrower architectures. *Relation-agnostic and attention-based GNNs* (GraphSAGE, GAT, HGT) share parameters across relations and scale well, but discard edge-type information during message passing. gHAWK's global feature, derived from a relation-aware KGE, injects relational semantics into these architectures through the input representation, partly compensating for the lack of per-relation parameters. The *decoder-only* configuration passes $z(e)$ directly to the task decoder with no GNN layers between fusion and decoder, so only the fusion parameters and the decoder are trained; this configuration avoids message-passing entirely and tests whether the precomputed structural features carry sufficient signal on their own.

4.3 Decoders and Losses

For node property prediction, a two-layer MLP classification head maps the entity representation to logits over the label set. Training minimizes cross-entropy loss over the labeled entities $\mathcal{E}_L$, and we report classification accuracy.

For link prediction, we use RotatE [33] as the scoring function: each relation is represented as a phase vector in complex space, and a triple is scored as the negative distance between the rotated head and the tail. Entity representations are split along the feature dimension into real and imaginary parts to obtain complex-valued inputs to RotatE. We train with the self-adversarial loss of [33], where each negative is weighted by its difficulty score within the batch, with negatives drawn uniformly at random, and we report filtered MRR and Hits@k on both head and tail prediction.

5 Experiments

5.1 Experimental Setup

Our experiments use Open Graph Benchmark datasets [20, 21], specifically OGB-MAG and MAG240M for node property prediction and OGB-WikiKG2 and OGB-BioKG for link prediction. We also use datasets from the Heterogeneous Graph Benchmark [27] in Sec. 5.2.1. All experiments use the official train/validation/test splits provided by the benchmarks. Hardware is a single NVIDIA L40S GPU (48 GB) on a server with an AMD Genoa CPU and 768 GB RAM (App. A.1). Training uses AdamW with learning rate 10^{-3}. For all GNN backbones we use two layers, hidden dimension 256, dropout 0.5, fanout $[25, 20]$ for node prediction; batch size 1024, margin $\gamma = 12$, $K = 32$ negatives, and self-adversarial sampling for link prediction, except that R-GCN is trained full-batch on OGB-MAG and with smaller settings on WikiKG2. Since each backbone uses identical hyperparameters with and without gHAWK, differences in performance can be attributed solely to node features. gHAWK's Bloom filter and KGE parameters are chosen per dataset (App. B), and full training configurations are in App. D.1 and E.1.

The experiments answer four questions for both tasks: whether gHAWK improves accuracy across diverse GNN backbones (Sections 5.2–5.3), whether the framework is robust to the specific encoding chosen for each prior (Sec. 5.4), and whether the benefit generalizes across schemas and domains (Sec. 5.2.1). End-to-end efficiency and resource usage are reported separately in Sec. 5.5.

Table 1: Node property prediction accuracy (%, mean ± std over five seeds). *Default* denotes the domain-specific features provided with each benchmark: Word2Vec for OGB-MAG and RoBERTa for MAG240M. *Local* adds the Bloom filter. *gHAWK* adds both Bloom and TransE. Bold: best accuracy on each dataset.

Model	OGB-MAG (default = Word2Vec)			MAG240M (default = RoBERTa)		
	Default	Local	gHAWK	Default	Local	gHAWK
R-GCN	37.67 ± 0.48	44.51 ± 0.48	47.98 ± 0.34	68.00 ± 0.07	69.99 ± 0.05	**75.04** ± 0.03
GraphSAGE	31.66 ± 0.23	37.11 ± 0.34	43.37 ± 0.31	66.83 ± 0.06	68.76 ± 0.09	71.87 ± 0.04
GraphSAINT	46.81 ± 0.48	50.26 ± 0.37	**57.37** ± 0.26	64.99 ± 0.23	68.30 ± 0.25	72.81 ± 0.08
ClusterGCN	36.03 ± 0.26	41.25 ± 0.53	45.56 ± 0.19	61.13 ± 0.12	57.37 ± 0.11	72.58 ± 0.11
GAT	31.87 ± 0.49	39.36 ± 0.28	49.10 ± 0.29	66.82 ± 0.04	70.09 ± 0.09	71.95 ± 0.05
HGT	43.43 ± 0.52	44.79 ± 0.61	53.39 ± 0.65	57.76 ± 0.06	65.67 ± 0.07	74.52 ± 0.11

5.2 Node Property Prediction

Node property prediction assigns a label to each entity using graph structure, labeled neighbors, and any available domain-specific features. We evaluate gHAWK across six GNN backbones spanning relation-aware, relation-agnostic, sampling-based, and attention-based paradigms (R-GCN, GraphSAGE, GraphSAINT, ClusterGCN, GAT, HGT) on two variants of the Microsoft Academic Graph, which represents bibliographic information: **OGB-MAG** ($\sim$1.9M nodes, predicting venue across 349 classes) and **MAG240M** ($\sim$244M nodes, predicting one of 153 subject areas for 1.4M arXiv papers). The two datasets differ in two ways that matter for this experiment: MAG240M is $126\times$ larger than OGB-MAG, and each dataset comes with its own domain-specific features, namely, text embeddings of each paper's title and abstract (128-dimensional Word2Vec [28] for OGB-MAG and 768-dimensional RoBERTa [25] for MAG240M). Word2Vec and RoBERTa both summarize text, but RoBERTa is more powerful and more expensive to compute, store, and feed through the first GNN layer. We refer to these as *default features*.

Table 2: Micro- / Macro-F1 (%, mean ± std over five seeds) for gHAWK on three heterogeneous graph benchmarks with GraphSAGE. SOTA: PHGT as reported in [26]. Bold: best per row.

	w/o gHAWK	gHAWK	SOTA
DBLP			
Micro-F1	82.50 ± 0.31	**97.52** ± 0.66	95.33
Macro-F1	81.70 ± 0.36	**97.04** ± 0.79	94.96
ACM			
Micro-F1	91.08 ± 0.34	**95.56** ± 0.74	93.72
Macro-F1	91.20 ± 0.39	93.16 ± 0.70	**93.79**
IMDb			
Micro-F1	65.28 ± 0.76	**69.14** ± 1.20	68.81
Macro-F1	61.89 ± 0.88	**66.14** ± 1.49	65.91

The most direct test of gHAWK is whether it improves a backbone trained with the dataset's own default features. Tab. 1 reports test accuracy for each backbone under three conditions: *Default* (the dataset's default features alone), *Local* (default features plus the Bloom filter), and *gHAWK* (default features plus both Bloom and TransE). The results are consistent across datasets and backbones: adding gHAWK on top of the default features always improves accuracy, by far more than the seed standard deviation. On OGB-MAG, the gain ranges from $+9.5$ percentage points on ClusterGCN to $+17.2$ on GAT. On MAG240M, it ranges from $+5.0$ on GraphSAGE to $+16.8$ on HGT. The gains hold across relation-aware, relation-agnostic, sampling-based, and attention-based architectures, so the benefit is not tied to any particular paradigm. The best accuracy is achieved by GraphSAINT + gHAWK on OGB-MAG (57.37%) and by R-GCN + gHAWK on MAG240M (75.04%). The *Local* column represents an ablation showing that adding the Bloom filter alone, without TransE, achieves a substantial portion of the full gHAWK gain on most backbones, indicating that the local neighborhood signal carries information the default features do not. The exception is ClusterGCN on MAG240M, where the Bloom filter alone lowers accuracy and the global prior is needed.

Another interesting test of gHAWK is whether it can *replace* the default features not just complement them. The default features are informative but expensive, and we show in App. D.3 that training with gHAWK on the graph structure only without default features outperforms training with the default features but without gHAWK.

5.2.1 Generalization Across Heterogeneous Domains

To test whether gHAWK generalizes beyond the OGB datasets, we evaluate on three node classification tasks on heterogeneous graph datasets from HGB [27], spanning different schemas and domains: DBLP and ACM (academic bibliography) and IMDb (movie-genre prediction). Tab. 2

Table 3: WikiKG2 link prediction (mean ± std over five seeds). * indicates results reported in [14, 50].

Method	Val. MRR	Test MRR
NodePiece + AutoSF*	0.5806	0.5703
NBFNet	OOM	OOM
A*Net*	0.6851	0.6767
RotatE	0.4352 ± 0.0004	0.4341 ± 0.0009
RotatE + gHAWK	**0.7032** ± 0.0014	**0.6941** ± 0.0016
GraphSAGE	0.5720 ± 0.0060	0.5689 ± 0.0040
GraphSAGE + gHAWK	0.6216 ± 0.0061	0.6129 ± 0.0024
R-GCN	0.3717 ± 0.0233	0.3851 ± 0.0225
R-GCN + gHAWK	0.5953 ± 0.0031	0.5957 ± 0.0039
CompGCN	0.6275 ± 0.0033	0.6256 ± 0.0039
CompGCN + gHAWK	0.6661 ± 0.0016	0.6562 ± 0.0010

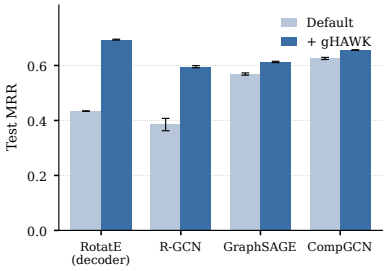


Figure 3: Test MRR on WikiKG2 with and without gHAWK (mean ± std over five seeds). Gains are largest for RotatE (+26.0) and R-GCN (+21.1), which capture the least structure on their own.

Table 4: Head MRR on WikiKG2 (decoder-only) stratified by training frequency. Single run result.

Train. triples	% queries	RotatE	gHAWK	Δ
1–5	20.12	0.0109	**0.3497**	+0.3388
6–20	71.77	0.0714	**0.4658**	+0.3944
21–100	7.17	0.3747	**0.7147**	+0.3400
101–1000	0.72	0.6355	**0.7272**	+0.0917
1000+	0.21	**0.8989**	0.7380	−0.1609
All	100.00	0.0868	**0.4627**	+0.3759

Table 5: Head MRR on WikiKG2 (R-GCN) stratified by training frequency. Single run result.

Train. triples	% queries	R-GCN	gHAWK	Δ
1–5	20.12	0.3852	**0.5432**	+0.1580
6–20	71.77	0.4822	**0.5762**	+0.0940
21–100	7.17	0.5091	**0.5805**	+0.0713
101–1000	0.72	0.4598	**0.5513**	+0.0915
1000+	0.21	0.4300	**0.5295**	+0.0995
All	100.00	0.4644	**0.5696**	+0.1052

reports Micro-F1 and Macro-F1 for GraphSAGE, the simplest of our backbones, with and without gHAWK (Bloom + TransE). gHAWK improves both metrics on every dataset, with the largest gains on DBLP (+15.0 Micro-F1) where citation neighborhood structure is highly discriminative.

These numbers position gHAWK favorably against the recent SOTA PHGT [26] (also shown in the table), a heterogeneous graph transformer specifically engineered for these benchmarks. The contribution here is threefold: *generality*, since the gains hold across both academic bibliography and entertainment domains; *efficiency*, since the same gHAWK recipe works on every benchmark with no dataset-specific tuning, while PHGT uses a heavyweight transformer trained per-dataset; and *strength of priors*, since precomputed structural features fed into a simple backbone match or exceed SOTA accuracy on graphs whose leaderboards are otherwise dominated by purpose-built architectures.

5.3 Link Prediction

Link prediction ranks candidate entities for a specific $(h, r, ?)$ or $(?, r, t)$ query. We evaluate on two OGB benchmarks [20] that span a $25\times$ range in scale: **OGB-WikiKG2** (2.5M entities, 535 relations, 16.1M training triples, 500 sampled negatives per query under filtered MRR), and **OGB-BioKG** (94K entities across five biomedical types, with type-restricted candidates that eliminate trivial cross-type distinctions). WikiKG2 stresses the long-tail behavior of structural priors at scale: over 80% of its entities appear in fewer than 10 training triples, and the long tail concentrates on the head side because tail entities (e.g., countries, organizations) tend to be shared across many heads. More detailed dataset-level statistics appear in App. A.

5.3.1 WikiKG2: Structural Priors Across Architectures

Tab. 3 reports MRR on WikiKG2 across four configurations: a RotatE decoder fed only gHAWK's structural features (no message passing), and three GNN backbones trained with and without gHAWK features. The backbones span the heterogeneous-GNN design space: R-GCN is fully relation-aware, CompGCN treats relations as compositional operators, and GraphSAGE shares parameters across all relations. For competitive context, we compare with the reported results of NodePiece + AutoSF [14], NBFNet [49] (which exhausts GPU memory, as also reported in [50]), and A*Net [50].

In the RotatE decoder-only pipeline, gHAWK improves test MRR from 0.4341 to 0.6941, a $+26$ percentage point gain achieved without any message passing. Adding gHAWK features to message-passing backbones also improves accuracy in every case: R-GCN gains $+21.1$ points, GraphSAGE gains $+4.4$ points, and CompGCN $+3.1$. Across architectures, structural priors provide a consistent

benefit. With structural priors, even a decoder-only architecture like RotatE with no aggregation matches or exceeds full message-passing alternatives, supporting our claim that on large KGs, the quality of input representations matters more than the depth of aggregation.

The mechanism behind these gains is that gHAWK provides the backbone with an accurate representation of the long-tail of the entity-relation distribution. In any embedding-based model, learned vectors only carry signal where training provides enough gradient updates to move the embedding away from initialization. Since a large fraction of WikiKG2 entities appear in fewer than five training triples, their 250-dimensional embeddings receive too few updates and are essentially uninformative. If we separate the test set into queries where the head is masked $(?, r, t)$ and ones where the tail is masked $(h, r, ?)$, we find that uninformative representations are starkest on the head side. Tab. 4 quantifies this. The table groups the head entities into buckets based on how many training triples they appear in. For each bucket, the table shows the fraction of queries for heads in this bucket, the MRR achieved by RotatE alone, the MRR achieved by RotatE + gHAWK, and the gain that gHAWK adds. The first two rows show that RotatE alone has very low MRR for heads that appear in few training triples (20 or fewer). gHAWK adds a gain of over $+30$ percentage points in these cases. These two rows are noteworthy since they account for more than 90% of the queries. The gain shrinks above 100 training triples and turns negative in the 1000+ bucket. Comparing Tab. 3 and 4, we see that RotatE has low accuracy for head queries (near-random against 500 negatives), and the 0.4341 test MRR of RotatE comes almost entirely from the tail side. gHAWK has good accuracy for head queries and overall. A Bloom filter deterministically encodes each entity's neighborhood from the training graph regardless of training frequency, and a pre-trained global embedding propagates signal through shared relations and neighbors even to rare entities. Structural features restore predictive capacity exactly where learned embeddings have none.

5.3.2 Generalization to Biomedical Link Prediction

We test whether gHAWK's benefits generalize beyond WikiKG2 by evaluating on OGB-BioKG, a biomedical knowledge graph that differs from WikiKG2 in three ways. First, BioKG uses *type-restricted evaluation*: when ranking the tail of a query like (Drug X, treats, ?), candidates are restricted to entities of the correct type (Disease) rather than the full entity set, eliminating trivial cross-type distinctions. Second, BioKG has a denser degree distribution (median degree 22) so most entities are well-represented in training. Third, its dominant relations (drug $\rightarrow$ drug, protein $\rightarrow$ protein) are symmetric, unlike most KGs where asymmetry is the norm. We therefore use ComplEx as the KGE for the global feature for BioKG, since it naturally encodes symmetric and asymmetric relations alike.

Table 6: MRR on OGB-BioKG. Top rows: aggregate validation and test performance. Bottom rows: per-relation breakdown (test).

	RotatE	**gHAWK**	Δ
Aggregate			
Validation	0.7997	**0.8380**	+0.038
Test	0.7989	**0.8378**	+0.039
Per-relation (test)			
drug $\rightarrow$ disease	0.531	**0.628**	+0.096
drug $\rightarrow$ drug	0.617	**0.704**	+0.088
disease $\rightarrow$ protein	0.587	**0.636**	+0.049
drug $\rightarrow$ side-effect	0.204	**0.239**	+0.035
drug $\rightarrow$ protein	0.804	**0.832**	+0.028
protein $\rightarrow$ function	0.844	**0.867**	+0.023
function $\rightarrow$ function	0.942	**0.962**	+0.020
protein $\rightarrow$ protein	0.969	**0.973**	+0.004

gHAWK improves test MRR from 0.7989 to 0.8378, a $+3.9$-point gain that closes nearly 20% of the headroom remaining above RotatE. The per-relation breakdown in Tab. 6 shows that this gain concentrates on the clinically important predictions where the base model has the lowest accuracy. Large gains appear on drug $\rightarrow$ disease, drug $\rightarrow$ drug, and drug $\rightarrow$ side-effect. This mirrors what we observed on WikiKG2, where gHAWK's gain similarly concentrated on entities the base model handled poorly.

5.4 Robustness to the Choice of Prior

gHAWK treats Bloom + TransE as the default local/global prior combination, but the framework itself can be used with other choices. To verify this, we evaluate combinations of three local priors (Bloom, RelDist, MinHash) and three global priors (TransE, DistMult, ComplEx) on both tasks, using the strongest backbone for each: GraphSAINT on OGB-MAG for node property prediction and the RotatE decoder-only configuration on WikiKG2 for link prediction.

The two tasks tell the same story. On node property prediction (Tab. 7a), the spread across the entire grid is under three percentage points, and the default Bloom + TransE cell has the best accuracy. Even the weakest combination (MinHash + ComplEx at 55.33%) remains well above the Default

Table 7: Robustness of gHAWK to the choice of local/global prior. Left: node property prediction on OGB-MAG (test accuracy %, GraphSAINT, default Word2Vec features). Right: link prediction on WikiKG2 (MRR, RotatE). Single run result.

(a) Node property prediction.

Local ↓ / Global →	TransE	DistMult	ComplEx
Bloom	**57.97**	57.14	57.38
RelDist	56.65	56.36	56.59
MinHash	56.06	55.41	55.33

(b) Link prediction (Test MRR / Head MRR).

Local ↓ / Global →	TransE	DistMult	ComplEx
Bloom	**.693**/.463	**.693**/.463	.692/.462
RelDist	.686/.458	.677/.451	.681/.461
MinHash	.680/.453	.675/.439	.685/.453

(46.81%) and Default + Local (50.26%) baselines from Tab. 1. On link prediction (Tab. 7b), every combination converges to an overall MRR of about 0.69 and a head MRR of about 0.46, with the default Bloom + TransE having the highest MRR. Across both tasks, the benefit of gHAWK comes from the presence of both a local and a global structural prior, not from the specific encoding chosen (App. E.3 isolates the contribution of each prior), and we adopt Bloom + TransE as the default because it is the simplest to construct, the cheapest to store, and one of the most efficient to pre-train. The full costs of preprocessing for different priors appear in App. B.5.

We observe that every choice of structural prior lifts head MRR from 0.087 for RotatE alone (Tab. 4) to about 0.46 (Tab. 7b), because the priors represent the long-tail of the head-entity distribution. Head MRR never exceeds this level, however, even though ample headroom remains, and the plateau is not the result of a weak choice of prior. Proposition A (App. F) shows why: any query-independent feature, whether hashed (Bloom), counted (RelDist), pre-trained (TransE, DistMult, ComplEx), or jointly learned, produces representations of two entities that grow arbitrarily close as their training neighborhoods converge (for trained features, under the local assumptions of App. F), and so do their decoder scores. Empirically, 93% of gHAWK's rank-1 wrong predictions are *doppelgangers* of the correct answer, and an oracle that resolves doppelganger confusion (App. E.6) attains 0.934 head MRR. The gap between the plateau and the oracle is therefore the cost of doppelganger confusion, and closing it requires conditioning on the query (App. F.3).

5.5 Efficiency and Resource Usage

Because the priors are frozen and transferred to GPU by mini-batch (Sec. 4), the cost gHAWK adds is modest. Preprocessing is a one-time expense shared by every backbone, seed, and ablation (about 25 min on OGB-MAG, 2 h on WikiKG2, and 6.5 h on MAG240M; App. B.5). On a GNN backbone the fusion module adds under 0.1% of parameters. It changes total training time on OGB-MAG by between −9% and +19%, and raises per-epoch time on WikiKG2 by 5–46%. The decoder-only pipeline, where the priors replace the backbone, costs +22% per epoch for its +0.26 MRR gain. Peak GPU memory is dominated by the backbone and entity table rather than the priors (+4.6% on WikiKG2), which is what allows training on MAG240M's 244M entities on a single GPU. Per-backbone measurements are in App. C.

6 Related Work

gHAWK sits at the intersection of six lines of work. Sampling-based GNNs (GraphSAGE [18], FastGCN [5], ClusterGCN [7], GraphSAINT [45]) make large-graph training feasible by restricting each step to a graph subset; gHAWK is *complementary*, shifting structural capture to preprocessing so that Bloom and TransE features remain fully informative regardless of sampling. Precomputed structural features (DeepWalk [31], node2vec [16], NodePiece [14], SIGN [13], Bloom signatures [47]) augment GNN inputs with fixed-size summaries; gHAWK extends this to heterogeneous KGs by combining a frozen local prior with a global KGE through a gated fusion that operates on top of the learned embeddings. Methods that combine local and global information in other domains [9, 10, 12, 6, 43] learn both representations jointly with the task, whereas gHAWK's priors are computed once and reused across tasks and backbones (App. G). Heterogeneous graph transformers (HGT [22], PHGT [26]) are purpose-built for multi-type graphs and achieve SOTA on benchmarks like DBLP and IMDb through dataset-specific architectures; gHAWK matches or exceeds them with a simple GraphSAGE backbone and a single Bloom + TransE recipe applied without per-dataset tuning (Sec. 5.2.1). Query-conditioned methods (NBFNet [49], A*Net [50]) compute query-specific representations to escape the limit on any query-independent feature, but at a per-query cost: at WikiKG2 scale, NBFNet exhausts memory and A*Net's accuracy falls below gHAWK's decoder-only configuration (Sec. 5.3.1); gHAWK trades this expressiveness for the scalability of precomputed query-independent features. Long-tail analysis [29] documents that embedding models degrade for

rare entities, and existing approaches address this indirectly through training-time interventions such as adjusted negative sampling or loss reweighting; gHAWK addresses it directly through features that remain informative regardless of the frequency of training triples.

7 Limitations and Future Work

Three design choices give gHAWK its scalability and generality: its priors are query-independent, they are combined with the learned embedding through a single gate shared across all entities, and they are computed once, in preprocessing, over a static training graph. Each choice also bounds what the method can do, and we discuss the three in turn.

Because the priors are query-independent, they cannot separate doppelganger entities (Proposition A, App. F), which caps head MRR on WikiKG2 well below what an oracle reaches (Sec. 5.4). The same limit applies to message-passing GNNs bounded by 1-WL (App. F.3); for gradient-trained features, the bound assumes local strong convexity around the trained solution (Lemma 3), which KGE objectives do not guarantee globally. The ceiling is the link-prediction form of the local isomorphism problem, which labeling tricks [46] resolve by conditioning the representation on the query. Applied to every candidate, however, query conditioning reintroduces the per-query cost that gHAWK is built to avoid (Sec. 6). Our error analysis points to a cheaper route: 93% of gHAWK's rank-1 errors are doppelgangers of the correct answer (App. E.6), so the errors that remain arise among near-identical candidates competing for the top rank. This motivates query-aware reranking as a general direction.

The shared gate (Sec. 3.3) learns one balance between structural priors and learned embeddings for the whole graph. That balance suits the long tail, where most queries lie and learned embeddings are weakest, but it cannot adapt to the few entities whose embeddings are already strong. For the 0.21% of WikiKG2 queries whose head appears in more than 1000 training triples, RotatE + gHAWK falls below RotatE alone (0.738 versus 0.899 head MRR, Tab. 4). This regression does not occur with R-GCN, whose head MRR in this bucket is only 0.430: with no strong embedding for the priors to dilute, R-GCN + gHAWK improves on R-GCN in every bucket (Tab. 5). A degree-conditioned gate, whose balance varies with an entity's training frequency, is a possible solution for the RotatE case.

Computing the priors once, in preprocessing, adds a one-time cost amortized across every backbone, seed, and ablation (Sec. 5.5), and requires that preprocessing examine only training triples, as ours does (Sec. 3.3); a pipeline adapted to a new dataset that admitted held-out edges would silently inflate link-prediction results. It also ties gHAWK to static-graph training. The local prior updates inductively, with two $\mathcal{O}(k)$ Bloom insertions per new edge, so an unseen entity receives its local prior as soon as its edges appear, while the global KGE prior is refreshed periodically, at a cost of about two hours on WikiKG2. Evaluating gHAWK under genuine graph dynamics, with an inductive KGE as the global prior, is left to future work.

8 Conclusion

We presented gHAWK, a framework for scalable GNN training on KGs that shifts the burden of capturing graph structure from iterative message passing to a one-time preprocessing step. Each entity is represented by a local feature (a Bloom filter encoding its neighborhood) and a global feature (a pre-trained TransE embedding), fused with the learned embedding through a gated mechanism. Our experiments show that these structural priors provide substantial benefit. For node property prediction, gHAWK consistently improves accuracy across six GNN backbones, spanning relation-aware, relation-agnostic, sampling-based, and attention-based architectures. For link prediction, gHAWK enables a RotatE decoder-only pipeline with no message passing to improve test MRR on WikiKG2 from 0.43 to 0.69, a 26 percentage point gain, and to outperform message-passing GNNs. This gain comes from accurately representing the long-tail of the head entity-relation distribution. The same effect transfers to BioKG, where gHAWK improves most on the clinically important drug relations. Beyond these results, we identified a structural property of query-independent representations, which we call doppelganger confusion: entities whose training neighborhoods nearly coincide receive nearly identical representations, and therefore nearly identical scores, whether the features are hashed, counted, or learned (Proposition A). This creates a ceiling that no choice of local or global prior escapes in our experiments: head MRR on WikiKG2 stays at or below 0.46 for all combinations of priors that we tested, while an oracle that resolves the confusion reaches 0.93. This characterization delineates the scope of the structural-prior paradigm, separating the regime where better encodings improve predictions from the regime that requires query conditioning. Such query conditioning is a natural extension of gHAWK. More broadly, our results suggest that on large KGs, the quality of input representations matters more than the depth of aggregation.

Acknowledgments and Disclosure of Funding

This work was funded by the University of Texas at Arlington, the University of Texas System through a Science and Technology Acquisition and Retention (STARs) grant, and the Texas Advanced Computing Center (TACC) through a computing allocation. Humera Sabir acknowledges support from a Fulbright Program grant sponsored by the Bureau of Educational and Cultural Affairs of the U.S. Department of State and the United States Educational Foundation in Pakistan (USEFP), and administered by the Institute of International Education (IIE).

The authors declare no competing interests.

References

[1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 1970.

[2] S. Bonner, I. P. Barrett, C. Ye, R. Swiers, O. Engkvist, A. Bender, C. T. Hoyt, and W. L. Hamilton. A review of biomedical datasets relating to drug discovery: A knowledge graph perspective. *Briefings in Bioinformatics*, 2022.

[3] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *NeurIPS*, volume 26, 2013.

[4] A. Z. Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997*, pages 21–29. IEEE, 1997.

[5] J. Chen, T. Ma, and C. Xiao. FastGCN: fast learning with graph convolutional networks via importance sampling. In *ICLR*, 2018.

[6] S. Chen, H. Yi, Z. You, X. Shang, Y.-A. Huang, L. Wang, and Z. Wang. Local–global structure-aware geometric equivariant graph representation learning for predicting protein–ligand binding affinity. *IEEE Transactions on Neural Networks and Learning Systems*, 36(8):15181–15193, 2025.

[7] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *Int. Conf. on Knowl. Discovery and Data Mining (KDD)*, pages 257–266, 2019.

[8] L. Dietz, A. Kotov, and E. Meij. Utilizing knowledge graphs for text-centric information retrieval. In *ACM SIGIR Conf. on Research & Development in inf. Retrieval*, pages 1387–1390, 2018.

[9] F. Ding, C. Xu, H. Liu, C. Lyu, G. Yang, H. Xiong, and H. Zhou. Global-local coherency contrastive learning for context-aware time series forecasting. *Knowledge-Based Systems*, page 114745, 2025.

[10] J. Du, J. Jin, J. Zhuang, and C. Zhang. Hierarchical graph contrastive learning of local and global presentation for multimodal sentiment analysis. *Scientific Reports*, 14(1):5335, 2024.

[11] V. P. Dwivedi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson. Graph neural networks with learnable structural and positional representations. In *ICLR*, 2022.

[12] M. Eliasof and E. Treister. Global-local graph neural networks for node-classification. *Pattern Recognition Letters*, 184:103–110, 2024.

[13] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti. SIGN: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

[14] M. Galkin, E. Denis, J. Wu, and W. L. Hamilton. NodePiece: Compositional and parameter-efficient representations of large knowledge graphs. In *ICLR*, 2022.

[15] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, pages 1263–1272, 2017.

[16] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Int. Conf. on Knowl. Discovery and Data Mining (KDD)*, 2016.

[17] Q. Guo, F. Zhuang, C. Qin, H. Zhu, X. Xie, H. Xiong, and Q. He. A survey on knowledge graph-based recommender systems. *IEEE Trans. Knowl. Data Eng.*, 34(8):3549–3568, 2022.

[18] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, volume 30, 2017.

[19] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, et al. Knowledge graphs. *ACM Computing Surveys*, 54(4):1–37, 2021.

[20] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*, volume 33, pages 22118–22133, 2020.

[21] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec. OGB-LSC: A large-scale challenge for machine learning on graphs. In *NeurIPS*, 2021.

[22] Z. Hu, Y. Dong, K. Wang, and Y. Sun. Heterogeneous graph transformer. In *Proc. of the Web Conf.*, pages 2704–2710, 2020.

[23] Y. Liang, W. Zhang, Z. Sheng, L. Yang, J. Jiang, Y. Tong, and B. Cui. HGAMLP: Heterogeneous graph attention MLP with de-redundancy mechanism. In *ICDE*, 2024.

[24] X. V. Lin, R. Socher, and C. Xiong. Multi-hop knowledge graph reasoning with reward shaping. In *EMNLP*, pages 3243–3253, 2018.

[25] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[26] Z. Lu, Y. Fang, C. Yang, and C. Shi. Heterogeneous graph transformer with poly-tokenization. In *IJCAI*, 2024.

[27] Q. Lv, M. Ding, Q. Liu, Y. Chen, W. Feng, S. He, C. Zhou, J. Jiang, Y. Dong, and J. Tang. Are we really making much progress? Revisiting, benchmarking and refining heterogeneous graph neural networks. In *SIGKDD*, 2021.

[28] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[29] A. Mohamed, S. P. Parambath, Z. Kaoudi, and A. Aboulnaga. Popularity agnostic evaluation of knowledge graph embeddings. In *Proc. Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 1059–1068, 2020.

[30] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proc. IEEE*, 104(1):11–33, 2015.

[31] B. Perozzi, R. Al-Rfou, and S. Skiena. DeepWalk: Online learning of social representations. In *Int. Conf. on Knowl. Discovery and Data Mining (KDD)*, 2014.

[32] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web: Int. Conf. ESWC*, pages 593–607, 2018.

[33] Z. Sun, Z.-H. Deng, J.-Y. Nie, and J. Tang. RotatE: Knowledge graph embedding by relational rotation in complex space. In *ICLR*, 2019.

[34] T. Thanapalasingam, L. van Berkel, P. Bloem, and P. Groth. Relational graph convolutional networks: A closer look. *PeerJ Computer Science*, 8:e1073, 2022.

[35] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard. Complex embeddings for simple link prediction. In *Int. Conf. on Machine Learning*, pages 2071–2080, 2016.

[36] S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar. Composition-based multi-relational graph convolutional networks. In *ICLR*, 2020.

[37] D. Vrandečić and M. Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 2014.

[38] K. Wang, Y. Xu, and S. Luo. TIGER: Training inductive graph neural network for large-scale knowledge graph reasoning. *PVLDB*, 17(10):2459–2472, 2024.

[39] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

[40] Q. Wu, W. Zhao, C. Yang, H. Zhang, F. Nie, H. Jiang, Y. Bian, and J. Yan. SGFormer: Simplifying and empowering transformers for large-graph representations. In *NeurIPS*, 2023.

[41] C. Xiong, R. Power, and J. Callan. Explicit semantic ranking for academic search via knowledge graph embedding. In *Proc. Int. Conf. on World Wide Web (WWW)*, pages 1271–1279, 2017.

[42] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *ICLR*, 2019.

[43] M. Xu, H. Wang, B. Ni, H. Guo, and J. Tang. Self-supervised graph-level representation learning with local and global structure. In *International conference on machine learning*, pages 11548–11558. PMLR, 2021.

[44] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*, 2015.

[45] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna. GraphSAINT: Graph sampling based inductive learning method. In *ICLR*, 2020.

[46] M. Zhang, P. Li, Y. Xia, K. Wang, and L. Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. In *NeurIPS*, volume 34, pages 9061–9073, 2021.

[47] T. Zhang, H. Yin, R. Wei, P. Li, and A. Shrivastava. Learning scalable structural representations for link prediction with Bloom signatures. In *Proceedings of the ACM Web Conference 2024*, pages 980–991, 2024.

[48] W. Zhou, H. Huang, R. Shi, and B. Liu. Scalable heterogeneous graph learning via heterogeneous-aware orthogonal prototype experts. *arXiv preprint arXiv:2601.05537*, 2026.

[49] Z. Zhu, Z. Zhang, L.-P. Xhonneux, and J. Tang. Neural Bellman-Ford networks: A general graph neural network framework for link prediction. In *NeurIPS*, volume 34, pages 29476–29490, 2021.

[50] Z. Zhu, X. Yuan, M. Galkin, L.-P. Xhonneux, M. Zhang, M. Gazeau, and J. Tang. A*Net: A scalable path-based reasoning approach for knowledge graphs. In *NeurIPS*, volume 36, pages 59323–59336, 2023.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state three contributions: (1) a preprocessing-based framework that combines local and global structural features through gated fusion, (2) consistent improvements across two tasks and four OGB datasets with the headline +26 MRR gain on WikiKG2, and (3) a structural characterization of the query-independent paradigm via Proposition A and the doppelganger phenomenon. Each claim is supported by experiments in Section 5 and theoretical analysis in Appendix F.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 7 discusses limitations (query-independent ceiling, shared gate, static-graph preprocessing). Section 5.4 explicitly characterizes the structural ceiling of the query-independent paradigm: an oracle that resolves doppelganger confusion attains 0.93 head MRR, and we identify query-conditioning as the natural extension required to close this gap. The smaller absolute gain on BioKG is acknowledged in Section 5.3.2 as a consequence of the dataset's denser degree distribution.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Proposition A is stated with its full assumptions covering three feature classes (hash-based, histogram-based, gradient-trained). The complete proof appears in Appendix F, structured as three lemmas, a combination-under-concatenation argument, and a connection to 1-Weisfeiler-Leman equivalence. The convexity assumption for gradient-trained features is explicitly noted with a remark on local minima.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The framework is described fully in Section 3 and Section 4. Bloom filter construction, KGE pre-training configuration, and complete hyperparameter tables appear in Appendix B.1, Appendix B.4, and Appendix E.1 respectively. Code is available at `https://github.com/lids-lab/gHAWK`. All datasets are public (OGB-MAG, MAG240M, OGB-WikiKG2, OGB-BioKG, plus DBLP/ACM/IMDb) and use their official train/validation/test splits.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code repository at `https://github.com/lids-lab/gHAWK`. The repository includes the Bloom filter construction, KGE pre-training, fusion module, and training scripts referenced in Appendix B.1, Appendix B.4, and Appendix E.1. Datasets used are public (OGB-MAG, MAG240M, OGB-WikiKG2, OGB-BioKG, plus DBLP/ACM/IMDb) and accessed through their official sources.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5.1 reports the experimental setup including hardware, optimizer (AdamW, learning rate 10^{-3}), backbone hyperparameters held fixed across configurations to isolate feature-quality effects, Bloom filter parameters per dataset, and KGE pre-training configurations. The complete hyperparameter table for WikiKG2 link prediction appears in Appendix E.1.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The main results (node property prediction in Tab. 1 and 2, WikiKG2 link prediction in Tab. 3 and 20) report mean $\pm$ standard deviation over five random seeds, capturing variability from initialization and sampling. The gains are far larger than the seed standard deviations. Ablations and sensitivity analyses in the appendix are single runs and are marked as such in their captions.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 5.1 reports the hardware (single NVIDIA L40S GPU with 48 GB, AMD Genoa CPU, 768 GB RAM). Appendix B.5 provides per-component preprocessing wall-clock times and peak RAM for all datasets, including the largest case (MAG240M: $\sim$6.5 h for TransE pre-training and $\sim$3 min for Bloom filter construction).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.

- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses only publicly available benchmark datasets and presents methodological contributions to graph neural network training. No human subjects, sensitive data, or potentially harmful applications are involved.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work presents methodological contributions to scalable graph neural network training on knowledge graphs. The framework is general-purpose and not tied to any particular application that would create direct societal risks. Applications of knowledge graph completion include biomedical research and recommendation systems; we do not see direct paths to negative applications beyond those generally associated with machine learning research.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release pretrained models with high misuse risk. The framework operates on existing public knowledge graph benchmarks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All datasets are properly cited: OGB-MAG, MAG240M, OGB-WikiKG2, and OGB-BioKG [20, 21], plus the heterogeneous benchmarks (DBLP, ACM, IMDb) used for cross-domain evaluation. Baseline methods are cited at point of use. All datasets are used under their official licenses and standard evaluation protocols.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: The paper does not introduce new datasets or models requiring asset documentation. Code is provided at `https://github.com/lids-lab/gHAWK`.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.

- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or human subjects research.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects research.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs are not used as a component of the methodology. The paper's framework operates on knowledge graph structure and uses traditional KGE models (TransE, DistMult, ComplEx) as global features.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.

Appendix: Table of Contents

A. **Hardware and Dataset Statistics** 21
B. **Structural Priors: Implementation and Preprocessing Costs** 22
C. **Training and Inference Cost** 24
D. **Node Property Prediction: Extended Results** 25
E. **Link Prediction: Extended Results** 28
F. **Theoretical Analysis of the Doppelganger Effect** 32
G. **Extended Related Work** 34

A Hardware and Dataset Statistics

A.1 Hardware

All experiments used a single server with an AMD Genoa CPU, 768 GB RAM, and four NVIDIA L40S GPUs (48 GB each); each run used one GPU.

A.2 Node Property Prediction Datasets

Tab. 8 reports statistics for both OGB and HGB node property prediction benchmarks.

Table 8: Node property prediction dataset statistics.

Dataset	Nodes	Edges	Entity types	Rel. types	Text feature (*Default*)
OGB-MAG	1.94M	21M	4	4	Word2Vec (128-d)
MAG240M	244M	1.3B	3	3	RoBERTa (768-d)
DBLP	26K	239K	4	6	BoW (334-d)
ACM	10K	547K	4	8	BoW (1902-d)
IMDb	21K	86K	4	6	Metadata vector (3489-d)

OGB-MAG and MAG240M represent the Microsoft Academic Graph at different scales. The task is paper venue prediction on OGB-MAG (349 classes) and paper subject area prediction on MAG240M (153 classes). Word2Vec features encode paper titles. RoBERTa features encode paper abstracts. DBLP, ACM, and IMDb are drawn from the Heterogeneous Graph Benchmark [27] and serve as generalization targets outside the OGB family.

A.3 Link Prediction Datasets

Tab. 9 reports statistics and evaluation protocols for both link prediction benchmarks.

Table 9: Link prediction dataset statistics.

Dataset	Entities	Relations	Train triples	Eval. neg.	Protocol
WikiKG2	2,500,604	535	16,109,182	500	filtered MRR
BioKG	93,773	51	4,762,677	500	same-type filtered MRR

WikiKG2. WikiKG2 is a large general-purpose knowledge graph derived from Wikidata. Its entities span people, places, organizations, and abstract concepts, with 535 relation types covering biographical, geographical, and categorical predicates. Each test query is evaluated against 500 sampled candidate entities under the standard filtered MRR protocol, where triples present in the

training, validation, or test set are removed from the candidate ranking. The dataset's degree distribution is heavily skewed: over 80% of entities appear in fewer than 10 training triples, with the long tail concentrated on the head side because tail entities (e.g., countries, organizations, common categories) tend to be shared across many heads.

BioKG. BioKG is a biomedical knowledge graph organized around five entity categories: drugs, diseases, proteins, functions, and side-effects. The graph is denser than WikiKG2 in absolute terms, with a median entity degree of 22, and most entities are well-represented in training. BioKG uses *type-restricted evaluation*: candidates for a query $(?, r, t)$ are restricted to entities of the type expected by relation r. For example, candidates for the head of (?, treats, X) are restricted to drugs. This restriction eliminates trivial cross-type distinctions and makes prediction comparable across relation types, but it also narrows the negative set to a more challenging within-type candidate pool.

B Structural Priors: Implementation and Preprocessing Costs

The first stage of the gHAWK pipeline is a preprocessing step that constructs local and global structural priors, both from the training triples only. The two priors are then frozen, and their fusion with the learned embedding and any domain-specific features (Sec. 3.3) is trained end-to-end with the downstream task. Sec. 5.4 investigated the effect of the choice of prior on accuracy and justified our decision to use Bloom filters and TransE by default. Here, we provide more details on the implementation of the priors and examine the preprocessing cost required to obtain them.

B.1 Bloom Filter: Construction and Parameter Selection

Rationale. A KG node can learn about its 1-hop neighbors in a single round of message passing under full-batch training, since the node receives messages from *all* its neighbors. Full-batch training is infeasible for large KGs, however, so mini-batch training randomly samples neighbors, providing only a partial neighborhood view.

To mitigate this loss, we represent the set of 1-hop neighbors with a Bloom filter: Bloom filters are fixed-length, tunable, accurate, cheap to compute, and require no explicit adjacency storage. Each node thus carries a lightweight approximate summary of its full neighborhood, independent of sampling during mini-batch formation. The Bloom filter provides a compact structural prior that the downstream GNN must still interpret, and we show it is effective at recovering the information that sampling discards.

Construction. We assign each node $i \in \mathcal{E}$ an m-bit Bloom filter $B[i] \in \{0,1\}^m$ encoding its 1-hop neighborhood. The full set $\{B[i]\}_{i\in\mathcal{E}}$ is constructed in a single pass over the training triples $\mathcal{T}$ (Algorithm 1).

To accurately reflect node i's neighborhood, $B[i]$ must encode both outgoing edges (where i is the head in (i, r, t)) and incoming edges (where i is the tail in (h, r, i)). It must also record the relation type, not only the neighbor's identity. For outgoing edges we encode the pair as the string `"r_t"`; for incoming edges as `"r_h"`. For every triple (h, r, t) the algorithm therefore inserts `"r_t"` into $B[h]$ and `"r_h"` into $B[t]$. Inserting a string sets k bit positions given by k independent hash functions applied to that string. After processing all triples, $B[i]$ holds a compact approximate encoding of node i's relational neighborhood without storing any adjacency list explicitly (see Fig. 2 for an example).

Parameter selection. The key dataset-dependent parameter is n, the expected number of elements inserted per filter. A naive choice is the maximum node degree, but degree distributions in KGs are highly skewed: a few outlier nodes (e.g., papers with unusually large citation counts in MAG240M) would force an unnecessarily large filter for all others. We instead take the sum of the 95th-percentile out-degree and the 95th-percentile in-degree as a stable estimate of the typical neighborhood size. Given n and a target false-positive rate

Algorithm 1: BUILDBLOOMFILTERS

Input: $\mathcal{T}$ (triple list), m (number of bits), k hash functions
Output: $\{B[i]\}_{i\in\mathcal{E}}$ (Bloom filters for all nodes)

```
foreach i ∈ E do
    initialize B[i] ← 0_m
end
foreach (h, r, t) ∈ T do
    B[h].INSERT("r_t")
    B[t].INSERT("r_h")
end
return {B[i]}
```

ε, the required bit width and hash count follow standard formulas [1]:

$$m = -\frac{n \ln \varepsilon}{(\ln 2)^2}, \qquad k = \frac{m}{n} \ln 2.$$

A Bloom filter also becomes uninformative as its fill ratio approaches 1. The expected fill ratio for a node with n edges is

$$\rho(n;\, m, k) = 1 - e^{-kn/m}.$$

We verify that $\rho(n_{\text{med}};\, m, k) \leq \rho^* = 0.5$, where n_{med} is the median node degree, to ensure typical nodes remain discriminative.

In practice, we round m up to the nearest convenient number satisfying both criteria. This yields $m = 128$ for OGB-MAG (median degree ≈ 5), $m = 500$ for WikiKG2 (median degree ≈ 6), and $m = 8{,}000$ for BioKG (median degree ≈ 22). Doubling or halving m around these values changes test accuracy by < 0.5 points on OGB-MAG and test MRR by < 0.005 on WikiKG2. The training configuration is further discussed in Sec. E.1.

When $m \leq 1024$, each filter is stored as a `uint32` array. For $m > 1024$ we apply run-length encoding and store all filters in a single memory-mapped file, paging in only the rows needed per mini-batch. Storage footprints: OGB-MAG ≈ 30 MB; WikiKG2 ≈ 156 MB; BioKG ≈ 94 MB—all comfortably within CPU RAM.

B.2 MinHash: Construction and Parameter Selection

The *MinHash signature* represents the same set of pairs through k minimum-valued hashes, so two entities with similar neighborhoods produce signatures whose Jaccard similarity is preserved with low variance, which provides a different similarity summary than Bloom filters. For each entity e, we use k independent hash functions and compute

$$f_{\text{MinHash}}(e)_i = \min_{\sigma \in \widetilde{\mathcal{N}}(e)} h_i(\sigma).$$

The empirical match rate over k coordinates is an unbiased estimator of the Jaccard similarity of two neighborhoods with standard deviation $\sqrt{J(1-J)/k}$. We use $k = 256$, giving $\sigma \approx 0.031$ at $J = 0.5$. In our ablations (Tab. 7), Bloom and MinHash yield comparable downstream accuracy.

B.3 Relation Distribution (RelDist)

The *relation-distribution histogram* (RelDist) records for each relation $r \in \mathcal{R}$ the count of triples in which each entity e participates with r, yielding a fixed-size $|\mathcal{R}|$-dimensional histogram. For each entity e, form an $|\mathcal{R}|$-dimensional vector where the r-th component counts e's participations with relation r in both head and tail roles. Construction time is $\mathcal{O}(|\mathcal{T}|)$. The vector is sparse and is projected to d dimensions via an MLP before feature fusion. RelDist preserves exact relational role counts but discards neighbor identity, limiting its ability to distinguish entities that play the same relational roles with different partners. The global feature partly compensates for this.

Complexity summary of local priors:

- Bloom filter: m bits/entity, $\mathcal{O}(k|\mathcal{T}|)$ time.
- MinHash: k integers/entity, $\mathcal{O}(k|\mathcal{T}|)$ time.
- RelDist: $|\mathcal{R}|$ counts/entity (sparse), $\mathcal{O}(|\mathcal{T}|)$ time.

B.4 Global Prior: KGE Pre-Training Configuration

The global prior consists of frozen entity vectors from a KGE model trained on the full training set $\mathcal{T}$ before any downstream training begins. TransE is the default since its translational geometry produces entity coordinates whose distances directly reflect relational proximity, a natural input for the fusion gate. ComplEx is used for BioKG to handle symmetric relations. The pre-training configuration for the KGEs is given in Tab. 10.

Table 10: KGE pre-training configuration for different datasets.

Dataset	Dim	Epochs	Model
OGB-MAG	128	30	TransE
MAG240M	128	30	TransE
WikiKG2	250	30	TransE
BioKG	1000	30	ComplEx

B.5 Preprocessing Costs

Both preprocessing steps—computing the local and the global priors—are one-time costs amortized across all downstream training runs, seeds, and backbones. The same Bloom filters and KGE embeddings are reused in all experiments. Tab. 11 shows the preprocessing time and memory requirements for the OGB datasets. Preprocessing time is proportional to dataset size, and most of it is spent on KGE embeddings ($\sim$2 h for WikiKG2 and $\sim$6.5 h for MAG240M, on one GPU). This is a modest one-time overhead given that GNN training itself runs for many more hours.

Table 11: gHAWK preprocessing costs on OGB datasets.

Dataset	Component	Wall-clock time	Peak RAM
OGB-MAG (1.94M / 21M)	Bloom (128 b)	14.8 s	6.1 GiB
	TransE (128-d)	24 min 46 s	5.1 GiB
MAG240M (244M / 1.3B)	Bloom (128 b)	$\sim$3 min	$\sim$24 GiB
	TransE (128-d)	$\sim$6.5 h	$\sim$48 GiB
WikiKG2 (2.5M / 16.1M)	Bloom (500 b)	38 s	7.2 GiB
	TransE (250-d)	$\sim$2 h	8.4 GiB
BioKG (94K / 4.8M)	Bloom (8000 b)	12 s	0.6 GiB
	ComplEx (1000-d)	$\sim$18 min	1.2 GiB

For DBLP, ACM, and IMDb, total preprocessing is under five minutes per dataset on one GPU, with TransE pre-training dominating the one-time cost. Tab. 12 shows the details.

Table 12: gHAWK preprocessing costs on HGB datasets.

Dataset	Component	Wall-clock time	Peak RAM
DBLP (26K / 239K)	Bloom (256 b)	1.4 s	1.38 GiB
	TransE (512-d)	2 min 19 s	1.51 GiB
ACM (10K / 547K)	Bloom (256 b)	3.2 s	1.17 GiB
	TransE (512-d)	4 min 31 s	1.30 GiB
IMDb (21K / 86K)	Bloom (256 b)	0.5 s	1.21 GiB
	TransE (512-d)	51.1 s	1.31 GiB

C Training and Inference Cost

Sec. 5.5 summarizes gHAWK's cost; here we give the per-backbone measurements. For each backbone, the gHAWK priors are toggled with the decoder, optimizer, batch size, and negative sampling held fixed, so the reported differences reflect the priors alone. Tab. 13 reports node property prediction cost on OGB-MAG and Tab. 14 reports link prediction cost on WikiKG2.

On OGB-MAG, total training time barely moves and is lower with gHAWK on some backbones, since the priors are read from CPU and disk rather than recomputed. On WikiKG2, per-epoch time rises by 5–46% on GNN backbones and by 22% in the decoder-only pipeline (Tab. 14).

Peak GPU memory is determined by the backbone rather than the priors, because the priors are moved to the GPU only for the entities being processed: the sampled mini-batch for most backbones,

Table 13: Node property prediction cost on OGB-MAG, toggling the gHAWK priors with default Word2Vec features held fixed. *R-GCN is trained full-batch, which accounts for its short training time; the other backbones use mini-batch sampling.

Backbone	Total train time		Peak GPU
	Default	gHAWK	(GiB)
R-GCN*	9.2 min	9.7 min	27.1
GraphSAGE	7.94 h	9.48 h	23.7
GraphSAINT	1.52 h	1.38 h	11.3
ClusterGCN	1.03 h	1.18 h	22.6
GAT	23.89 h	25.16 h	38.8
HGT	21.82 h	21.62 h	8.2

Table 14: Link prediction cost on WikiKG2, toggling the gHAWK priors on each matched backbone. Per-epoch time is mean wall-clock training time over all epochs under the protocol of Tab. 19; overhead is the gHAWK/Default ratio. Test MRR is repeated from Tab. 3. *R-GCN uses batch size 512 (Tab. 19), so each epoch has twice as many steps.

Backbone	Time (min/epoch)			Test MRR	
	Default	gHAWK	Overhead	Default	gHAWK
RotatE (decoder-only)	43.6	53.4	1.22×	0.4341	**0.6941**
GraphSAGE	41.5	60.7	1.46×	0.5689	**0.6129**
R-GCN*	326.6	343.2	1.05×	0.3851	**0.5957**
CompGCN	33.5	35.9	1.07×	0.6256	**0.6562**

and the whole graph for full-batch R-GCN on OGB-MAG, where the priors are still small next to the backbone's own memory footprint. The node property prediction values are shown in Tab. 13. For link prediction on WikiKG2, peak GPU memory is 29.7 GiB with gHAWK versus 28.4 GiB without for RotatE ($+4.6\%$).

On a GNN backbone the fusion module adds 0.03%–0.07% of parameters. In the decoder-only pipeline it reaches 2.5%. The structural features are stored on disk and paged per mini-batch (2.66 GB on WikiKG2), so they never occupy GPU memory, and inference tracks the backbone's own forward pass because the fused representation is frozen after training and materialized once.

D Node Property Prediction: Extended Results

D.1 Training Configuration

The GNNs are trained according to the following configurations:

OGB-MAG / MAG240M. Two GNN layers; hidden dimension 256; dropout 0.5; fanout $[25, 20]$; AdamW with learning rate 10^{-3}; early stopping on validation accuracy. R-GCN on OGB-MAG is trained full-batch; all other runs use mini-batch sampling.

DBLP / ACM / IMDb. GraphSAGE, two layers, hidden dimension 256, dropout 0.5, fanout $[20, 20]$, AdamW with cosine schedule, batch size 512, learning rate 10^{-3}, weight decay 10^{-4}, 80 epochs evaluated every 5 epochs, 5 seeds. Bloom uses 256 bits, $k = 5$ hash functions; MinHash uses 128 permutations; KGE methods use dimension 512 and 30 epochs; Node2Vec uses dimension 64, walk length 40, context size 10, 10 walks per node, 5 negative samples.

D.2 Extended Backbones on OGB-MAG

Beyond the six backbones in the main paper, we evaluated four additional architectures on OGB-MAG: a decoder-only MLP (no message passing at all), SGC [39] (a linear graph convolution), SIGN [13] (precomputed spectral propagation), and SGFormer [40] (a global-attention transformer). Tab. 15 reports test accuracy with the default Word2Vec features and with gHAWK added on top.

The most striking row is the MLP. A decoder-only MLP equipped with gHAWK features—no message passing whatsoever—reaches 35.61% on OGB-MAG, exceeding the Text-only GraphSAGE

Table 15: Extended GNN backbones on OGB-MAG (test accuracy %, default Word2Vec features).

Model	Default	gHAWK	Δ
MLP	31.95	**35.61**	+3.66
SGC	38.42	**40.84**	+2.42
SIGN	37.54	**40.46**	+2.92
SGFormer	51.35	**56.46**	+5.11

baseline in Tab. 1 (31.66%) by +4.0 points. SGC, SIGN, and SGFormer all benefit consistently from gHAWK, with SGFormer gaining the most (+5.11 points) because its global-attention mechanism lacks an explicit local inductive bias and therefore benefits more from the Bloom filter's neighborhood signature.

D.3 Ablation on Default Text Embedding Features

The default features used in Tab. 1 are informative but expensive. RoBERTa, in particular, produces 768-dimensional vectors that dominate first-layer GNN memory and require running an external language model over every paper in the graph. A natural follow-up question is whether gHAWK's structural priors can substitute for these features, not just complement them. Tab. 16 reports the same six backbones without using the default text embedding features of the two dataset. *No Text* is initialized using random vectors so the model relies only on graph structure, *Local* adds the Bloom filter, and *gHAWK* adds both Bloom and TransE.

Table 16: Node property prediction accuracy (%) without using the domain-specific features. *No Text* uses random vectors as input; *Local* adds the Bloom filter; *gHAWK* means both Bloom and TransE features.

	OGB-MAG			MAG240M		
Model	No Text	Local	gHAWK	No Text	Local	gHAWK
R-GCN	23.37	42.88	**43.71**	21.66	49.32	**59.23**
GraphSAGE	16.14	37.42	**40.69**	23.82	52.88	**61.52**
GraphSAINT	27.41	48.57	**54.55**	27.34	60.90	**60.98**
ClusterGCN	22.12	39.01	**43.89**	25.19	54.71	**59.82**
GAT	24.12	32.83	**45.34**	26.15	54.31	**62.15**
HGT	25.65	44.14	**48.94**	28.45	49.81	**57.55**

Without any features, all six backbones produce less than 30% accuracy on both datasets, confirming that mini-batch training on graph structure alone loses too much information for accurate prediction even when the backbone explicitly models relation types. Adding the Bloom filter alone lifts every backbone by 9–21 percentage points on OGB-MAG and by 21–34 points on MAG240M, recovering most of the gap with the default-features baselines. Adding TransE on top further improves accuracy on every backbone. This demonstrates the importance of both local and global priors in gHAWK.

The most striking observation is what happens when we compare across Tab. 1 and Tab. 16. On OGB-MAG, the structural-only *gHAWK* configuration in Tab. 16 beats the *Default* (Word2Vec-only) column in Tab. 1 for every single backbone, e.g., GraphSAINT reaches 54.55% with no default features versus 46.81% with Word2Vec alone. On MAG240M, where the default features are RoBERTa, which are more expressive than Word2Vec albeit more expensive, structural-only gHAWK recovers roughly 90% of RoBERTa-only accuracy on most backbones while using a fraction of the dimensionality.

Comparing across Tab. 1 and Tab. 15, we see that a decoder-only MLP equipped with gHAWK features and no message passing reaches 35.61% on OGB-MAG (Tab. 15), exceeding the 31.66% of GraphSAGE with the default features only (Tab. 1). A model with no message passing and no text embedding features but with structural priors outperforms a full message-passing GNN that has text embedding features but no structural priors.

As we argued in the introduction, when every node carries an explicit representation of graph structure, training can compensate for the information loss that mini-batch sampling causes on large KGs, and gHAWK is not adding noise on top of strong features but providing real structural information that the GNN would otherwise need to reconstruct from scratch.

D.4 Convergence Time of GNN Backbones

gHAWK also affects how well the GNN converges. Fig. 4 plots validation error versus training time on MAG240M for every backbone, comparing the default RoBERTa features (dotted) to RoBERTa + gHAWK (solid). gHAWK reaches a lower final validation error than the RoBERTa-only baseline on every backbone, and stays below it for most of training. Part of the early-training advantage comes from the fact that gHAWK projects the 768-dimensional RoBERTa vector to 384 dimensions before fusion, which reduces the model's parameter count by 2–12% across backbones (e.g., GAT $3.1 \rightarrow 2.8$M; GraphSAGE/GraphSAINT/ClusterGCN $4.9 \rightarrow 4.4$M; HGT $31.4 \rightarrow 30.7$M; R-GCN $34.2 \rightarrow 30.1$M), so each gradient step is cheaper. The persistent gap in final validation error reflects structural information that the RoBERTa baseline does not recover, even after extended training.

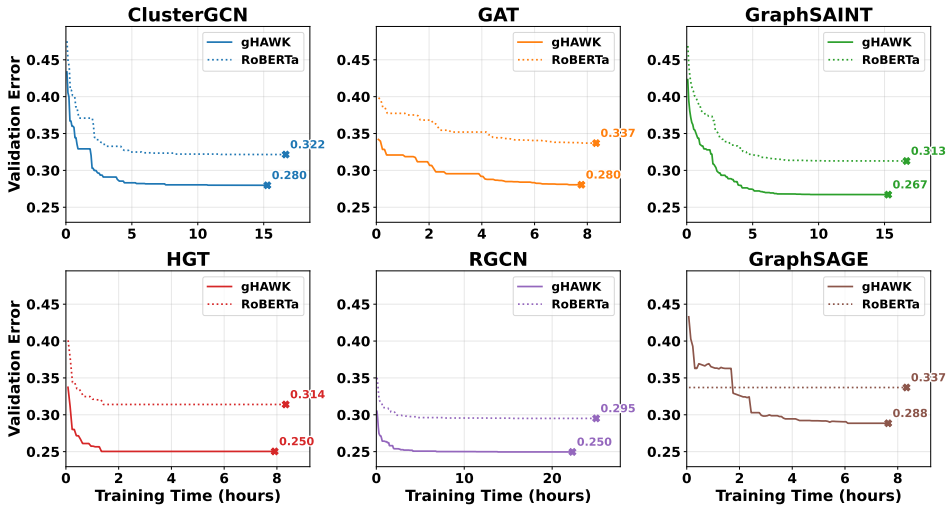


Figure 4: Model convergence on MAG240M (single run per configuration). The plots show validation error versus training time (hours) for each GNN backbone. Dotted lines represent RoBERTa features only, and solid lines represent RoBERTa+gHAWK. X markers denote the final validation error, with the numeric label showing its value.

D.5 Sensitivity to gHAWK Feature Size

Tab. 17 reports test accuracy on OGB-MAG as a function of the number of bits in the Bloom filter and the number of dimensions in TransE, using GraphSAGE with the default Word2Vec features.

Table 17: OGB-MAG test accuracy (%, GraphSAGE + Word2Vec) for different sizes of the Bloom filter and TransE embedding. Single run result.

TransE dim \ Bloom bits	128	256	512
128	43.04	41.47	49.08
256	44.15	42.41	41.84
512	44.68	43.58	42.96

Accuracy generally increases with TransE dimension for 128- and 256-bit filters. The Bloom filter shows a non-monotonic pattern under GraphSAGE: 128 bits outperforms 256 and 512, except that 512 bits paired with 128-d TransE gives the best accuracy in this single run. Under mini-batch sampling, the smaller filter may introduce a stochastic regularization effect that benefits GraphSAGE, while larger deterministic filters cannot be fully exploited because each mini-batch accesses only a partial neighborhood. The default 128-bit / 128-d configuration is the most efficient.

D.6 Feature Fusion Strategies

Tab. 18 compares strategies for feature fusion: simple concatenation (**C**), concatenation with element-wise (Hadamard) products (**CH**), and the learned scalar gating with residual connection (**GR**) used by default.

Table 18: Fusion strategy comparison on OGB-MAG (test accuracy %). C: Concatenation; CH: Concat-Hadamard; GR: Gated Residual. Single run result.

Backbone	C	CH	GR	Best
ClusterGCN	43.18	44.89	**45.70**	GR
GraphSAINT	47.31	48.49	**57.97**	GR
GraphSAGE	31.43	37.16	**43.04**	GR
MLP	**35.61**	29.93	29.39	C
SGC	38.99	37.84	**40.84**	GR
SGFormer	**56.46**	53.71	54.41	C

Gated Residual is the consistent winner for message-passing backbones. The gain over concatenation is largest on GraphSAINT (47.31 $\rightarrow$ 57.97), where the gate learns to suppress structural channels that are noisy for a given subgraph partition. For non-message-passing (MLP) and global-attention (SGFormer) backbones, simple concatenation wins; these architectures already learn implicit attention over their inputs and an additional gate adds redundant capacity.

E Link Prediction: Extended Results

E.1 Training Configuration

Tab. 19 reports the training configuration for the link prediction experiments on WikiKG2.

All four configurations (RotatE and three GNN backbones) share these hyperparameters, with one exception. R-GCN, whose parameter count grows with the 535 relation types (Sec. 4.2), uses a batch size of 512 and a hidden dimension of 100 so that training fits in the memory of a single 48 GB NVIDIA L40S GPU. Both R-GCN runs, with and without gHAWK, use these settings. Within each backbone, the runs with and without gHAWK use identical settings, so the differences in Tab. 3 reflect the node features rather than tuning.

BioKG's configuration inherits most of these values, with two exceptions: the global feature is ComplEx, which encodes the symmetric relations that dominate BioKG (Sec. 5.3.2), and the Bloom filter uses more bits. BioKG's median entity degree is 22, with a heavy upper tail: a non-trivial fraction of evaluation queries involve entities with degree > 100. Holding the saturation budget of the Bloom filter $\rho^* = 0.5$ at the median is insufficient for these high-degree entities. We size the filter at $8{,}000$ bits, which keeps saturation below budget at the 90th percentile of the degree distribution. Total storage for the Bloom filters is approximately 94 MB, and downstream test MRR is insensitive to perturbations within $\pm 50\%$ of this size.

E.2 Detailed gHAWK Link Prediction Results

Tab. 20 and 21 report filtered MRR, Hits@3, and Hits@10 on validation and test sets for all gHAWK configurations. The advantage provided by gHAWK is evident in both tables and across all metrics.

E.3 Single-Prior Ablation

The robustness grid in Sec. 5.4 shows that all combinations of local and global priors converge to similar MRR values. Tab. 22 extends this analysis with single-prior baselines on WikiKG2 (RotatE decoder-only configuration), showing that each prior contributes substantially on its own and the two carry complementary information.

Bloom alone takes test MRR from 0.4345 to 0.6731 ($+23.9$ points) and head MRR from 0.087 to 0.371 ($+28.4$ points). TransE alone takes test MRR to 0.6641 ($+23.0$ points) and head MRR to 0.379 ($+29.2$ points). The combination (Bloom + TransE) reaches test MRR 0.6930 and head MRR

Table 19: Training configuration for WikiKG2 link prediction. All backbones use these values except where R-GCN is noted.

Hyperparameter	Value
Optimization	
Batch size	1024 (R-GCN: 512)
Optimizer	AdamW
Learning rate	$1e{-3}$
Weight decay	$1e{-5}$
Dropout	0.1
Training steps	up to 100 epochs, early stopped
Early stopping	validation MRR
Loss and negative sampling	
Loss	self-adversarial [33]
Self-adv. temperature α	1.0
Margin γ	12
Negatives per positive K	32 (R-GCN: 6)
Negative sampler	uniform
Embeddings and structural features	
Entity / KGE dimension	250
Bloom filter bits m	500
Bloom hash functions k	3
Global feature	TransE
Fusion module	
Projection MLPs	2 layers, 256-dim hidden
Fusion MLP	2 layers, 512-dim output
GNN backbones	
GNN layers	2
GNN hidden dim	256 (R-GCN: 100)
Sampling fanout	[25, 20] (R-GCN: [10, 5])

Table 20: Detailed link prediction results on WikiKG2: mean $\pm$ standard deviation over five seeds. Bold: best per column.

	Validation			Test		
Method	MRR	H@3	H@10	MRR	H@3	H@10
RotatE	$0.4352 \pm .0004$	$0.4472 \pm .0006$	$0.5009 \pm .0005$	$0.4341 \pm .0009$	$0.4411 \pm .0006$	$0.4907 \pm .0011$
+ gHAWK	$\mathbf{0.7032} \pm .0014$	$\mathbf{0.7425} \pm .0011$	$\mathbf{0.8441} \pm .0017$	$\mathbf{0.6941} \pm .0016$	$\mathbf{0.7328} \pm .0019$	$\mathbf{0.8443} \pm .0023$
GraphSAGE	$0.5720 \pm .0060$	$0.6169 \pm .0066$	$0.7528 \pm .0076$	$0.5689 \pm .0040$	$0.6025 \pm .0041$	$0.7391 \pm .0035$
+ gHAWK	$0.6216 \pm .0061$	$0.6654 \pm .0042$	$0.8060 \pm .0028$	$0.6129 \pm .0024$	$0.6453 \pm .0028$	$0.7884 \pm .0021$
R-GCN	$0.3717 \pm .0233$	$0.4136 \pm .0282$	$0.5702 \pm .0255$	$0.3851 \pm .0225$	$0.4309 \pm .0250$	$0.6091 \pm .0240$
+ gHAWK	$0.5953 \pm .0031$	$0.6373 \pm .0049$	$0.7879 \pm .0069$	$0.5957 \pm .0039$	$0.6310 \pm .0036$	$0.7853 \pm .0046$
CompGCN	$0.6275 \pm .0033$	$0.6740 \pm .0036$	$0.8199 \pm .0022$	$0.6256 \pm .0039$	$0.6665 \pm .0035$	$0.8142 \pm .0035$
+ gHAWK	$0.6661 \pm .0016$	$0.7060 \pm .0014$	$0.8352 \pm .0014$	$0.6562 \pm .0010$	$0.6950 \pm .0012$	$0.8336 \pm .0018$

0.4627. Neither prior subsumes the other: the local feature captures neighborhood composition, the global feature captures relational position, and the two carry complementary information that adds beyond either alone.

E.4 Using TransE as a Negative Sampler

gHAWK samples negatives uniformly and relies on the self-adversarial loss [33] to reweight them by difficulty. Because gHAWK's preprocessing already trains a KGE for the global prior, the same KGE could also mine *hard* negatives, a role that usually requires training a separate scoring model. We test whether this second role adds accuracy.

Tab. 23 shows the effect of using the KGE as a global prior and as a negative sampler, in the decoder-only RotatE configuration. The first column uses gHAWK's default uniform negative sampler. The

Table 21: Detailed link prediction results on BioKG.

Method	Validation MRR	Validation H@3	Validation H@10	Test MRR	Test H@3	Test H@10
RotatE	0.7997	0.8577	0.9399	0.7989	0.8588	0.9405
RotatE + gHAWK	**0.8380**	**0.8860**	**0.9496**	**0.8378**	**0.8849**	**0.9487**

Table 22: Single-prior ablation on WikiKG2 (RotatE decoder-only). Each structural prior alone contributes substantially over the RotatE baseline, and the combination yields the full gain. Single run result

Local	Global	Test MRR	Head MRR
–	–	0.4345	0.0868
Bloom	–	0.6731	0.3713
–	TransE	0.6641	0.3791
Bloom	TransE	**0.6930**	**0.4627**

second column uses TransE to score the candidate negatives and trains the model on the negatives that TransE scores as hardest. In each column, the structural prior is either Bloom + TransE or Bloom alone. Thus, the table shows the four possible settings of using (or not using) TransE as a prior and as a negative sampler.

Comparing the two columns of the table, we see that switching the sampler changes test MRR by only $+0.002$ with Bloom + TransE, lowers it with Bloom alone, and lowers validation MRR in both settings, likely because the self-adversarial loss already focuses training on hard negatives. In contrast, comparing the rows of the table, we see that removing TransE as a global prior costs 0.020 to 0.028 under either sampler. Thus, using the KGE as a global prior provides a clear gain, while using it as a negative sampler does not. These results justify gHAWK's design decision to use the KGE only as a global prior.

Table 23: Using the KGE as a global prior and as a negative sampler on WikiKG2 (decoder-only RotatE, single run; validation / test MRR). Test MRR in the uniform column matches Tab. 22.

	Uniform negatives	TransE-mined negatives
Bloom + TransE	0.7026 / 0.6930	0.7017 / 0.6954
Bloom only	0.6851 / 0.6731	0.6762 / 0.6676

E.5 Per-Entity vs. Per-Query Distributions

Tab. 3 shows that gHAWK produces an MRR gain of $+26$ percentage points on WikiKG2. This large gain reflects a distinction between two ways of looking at the long tail. While 80% of WikiKG2's entities appear in fewer than 10 training triples, the distribution of test *queries* is more concentrated: 92% of test queries involve entities with between 1 and 20 training triples. These are the queries that benefit the most from the Bloom filter's ability to capture the long tail of the data distribution. Since reporting MRR uniformly weights queries rather than entities, the gain reflects the query distribution rather than the entity distribution. The majority of the queries are in a regime where gradient-based embeddings cannot produce useful signal, and these queries benefit greatly from gHAWK's Bloom filter, so the overall gain in MRR is high.

A subtle exception is worth flagging. In the bucket of highest-frequency entities ($1000+$ training triples, 0.21% of queries), gHAWK underperforms RotatE (0.7380 vs. 0.8989 head MRR). These entities receive enough gradient signal to learn strong embeddings on their own, so fusing in lower-capacity structural features dilutes them. Because the bucket is so small, the effect on aggregate head MRR is negligible (about 0.0003), and it does not arise with R-GCN, which improves in every frequency bucket, including $1000+$ (Tab. 5). This is consistent with our framework's premise: structural priors help most where learned embeddings cannot reach, which is the long tail that dominates the query distribution.

E.6 Doppelganger Confusion and Scoring with a Doppelganger Oracle

Empirical scope of doppelganger confusion. In Sec. 5.4, we observed that the main source of error in head prediction, i.e., $(?, r, t)$ queries, is confusion between the correct head and candidates whose training neighborhoods overlap with it. Consider a test query $(?, r, t)$ whose correct answer is h_1, and a candidate h_2 for which (h_2, r, t) is not a known triple, so that h_2 remains a negative under the filtered protocol. If h_1 and h_2 share training pairs, e.g., both (h_1, r', t') and (h_2, r', t') are in $\mathcal{T}$, their neighborhoods look alike, and a model that ranks h_2 above h_1 lowers the MRR. We call such candidates *doppelgangers* of the correct head. In the empirical analysis below, a candidate counts as a doppelganger of the correct head h^* if it shares at least one training (r, t) pair with h^*.

Detailed error analysis on WikiKG2 reveals that doppelganger confusion is the dominant failure mode in head prediction. Of gHAWK's wrong predictions at rank 1, 93% share at least one training (r, t) pair with the correct head. The problem is not concentrated in a few pathological queries: averaging across all test queries with masked heads, 240 of the 500 candidate negatives (48%) are doppelgangers of the correct head. Because the head distribution is the long tail of WikiKG2 (Sec. 5.3.1), the asymmetry is severe: head MRR for RotatE is 0.087 while tail MRR is 0.78. Doppelganger confusion is a head-side phenomenon; the $+0.47$ headroom above gHAWK's head MRR (0.463) that we characterize below is exclusive to head prediction.

Mechanism of doppelganger confusion. The phenomenon is a structural consequence of how query-independent representations are built. Two entities e_1 and e_2 that share a training pair (r', t') receive the same contribution from it: the same bits in their Bloom filters, the same count in RelDist, and, during gradient training, the same pressure to score $(\cdot, r', t')$ well. The fewer training pairs they do *not* share, the closer their representations become, and at query time the scoring function sees two similar representations that it cannot reliably tell apart. Proposition A (App. F) makes this precise for every class of query-independent feature we consider, and Corollary A carries it over to the decoder's scores.

Scoring with a doppelganger oracle. Assume that we have an oracle that tells us the correct head h^* for each test query $(?, r, t)$. We define an *oracle score* for each candidate $c \neq h^*$ as

$$\hat{s}_{\text{oracle}}(c) = \hat{s}(c) - \alpha \cdot S(c, h^*), \qquad S(c, h^*) = |\mathcal{N}(c) \cap \mathcal{N}(h^*)|,$$

and leave the correct head's score unchanged, $\hat{s}_{\text{oracle}}(h^*) = \hat{s}(h^*)$. Here $\hat{s}(\cdot)$ is the RotatE decoder score of the trained RotatE + gHAWK model, and $\mathcal{N}$ is the training neighborhood defined in App. F. The oracle penalizes each candidate by the size of its neighborhood overlap with the correct answer, scaled by α. We sweep $\alpha \in \{1, 2, 5, 10, 20\}$; at $\alpha = 5$ the oracle score achieves 0.9340 head MRR on WikiKG2, the reference level reported in Sec. 5.4.

The oracle is not a deployable method, since it requires knowing h^* to compute $S(c, h^*)$. Its purpose is to estimate the head MRR reachable once doppelganger confusion is resolved, which we find to be 0.9340. The natural way to close the gap to this level is query conditioning: two similar entities always look similar if one looks only at the entities, but the query itself can be used to tell them apart.

Stratification by overlap magnitude. Head MRR is not uniform across queries. Stratifying head queries by the overlap in (r, t) pairs between the highest-ranked wrong head and the correct head reveals a non-monotonic pattern, shown in Tab. 24.

Table 24: Test head queries on WikiKG2 stratified by overlap in (r, t) pairs between the highest-ranked wrong head and the correct head.

Overlap	N queries	% of total	Head MRR
0	56,696	9.5%	0.564
1–2	323,760	54.1%	0.423
3–5	215,499	36.0%	0.494
6–10	2,567	0.4%	0.649

The 1–2 overlap bucket contains the majority of queries (54.1%) and has the lowest head MRR (0.423). These are entities that share just enough structure to confuse the model but not enough to provide a clear disambiguating signal. This bucket is where modest query-conditioned signal would

have maximum impact. The higher-overlap buckets have higher head MRR. This is consistent with Proposition A, which bounds the distance between two representations by the number of training pairs the entities do *not* share rather than by their overlap: two entities with a large overlap can still differ in many pairs. This stratification identifies where any query-conditioned method would need to intervene most aggressively.

F Theoretical Analysis of the Doppelganger Effect

F.1 Setup and Proposition Statement

For an entity $e \in \mathcal{E}$, define its outgoing training neighborhood $\mathcal{N}(e) = \{(r,t) : (e,r,t) \in \mathcal{T}\}$ and its bidirectional neighborhood $\widetilde{\mathcal{N}}(e) = \mathcal{N}(e) \cup \{(r,h) : (h,r,e) \in \mathcal{T}\}$. gHAWK uses $\widetilde{\mathcal{N}}$ in practice. The proofs use $\mathcal{N}$ for readability, and every statement holds verbatim with $\widetilde{\mathcal{N}}$ in its place. For Lemma 3 and App. F.3, the pairs in $\widetilde{\mathcal{N}}$ are taken with their direction (equivalently, an incoming pair (r,h) is written with the inverse relation r^{-1}), since an entity's vector enters a loss term differently as a head and as a tail. For a candidate c and the correct answer h^*, write $S(c,h^*) = |\mathcal{N}(c) \cap \mathcal{N}(h^*)|$ and $\Delta(c,h^*) = \mathcal{N}(c) \triangle \mathcal{N}(h^*)$, so that $|\Delta(c,h^*)| = |\mathcal{N}(c)| + |\mathcal{N}(h^*)| - 2S(c,h^*)$. A feature $f : \mathcal{E} \to \mathbb{R}^d$ is *query-independent* if $f(e)$ is computed without reference to any specific query.

> **Proposition A (Query-independent ceiling).** Let f be a query-independent structural feature from one of three classes: *(i)* hash-based (Bloom filters); *(ii)* histogram-based (RelDist); or *(iii)* gradient-trained (TransE, DistMult, ComplEx, jointly learned entity tables), under the assumptions of Lemma 3. Then $\|f(c) - f(h^*)\| \le D_f(|\Delta(c,h^*)|)$ for a non-decreasing function D_f with $D_f(0) = 0$ that does not depend on the pair (c,h^*). The same holds for any concatenation of features from these classes, with or without a subsequent Lipschitz map (e.g., a linear projection or a ReLU MLP).

F.2 Proof of Proposition A

F.2.1 Lemma 1: Hash-Based Features

$\|f_{\mathrm{Bloom}}(c) - f_{\mathrm{Bloom}}(h^*)\|_1 \le k \cdot |\Delta(c,h^*)|$.

Proof. The ℓ_1 distance between two binary filters is the number of bits set in exactly one of them. Without loss of generality, let bit b be set in $f_{\mathrm{Bloom}}(c)$ but not in $f_{\mathrm{Bloom}}(h^*)$: some pair $\sigma \in \mathcal{N}(c)$ hashes to b under one of the k functions, and no pair in $\mathcal{N}(h^*)$ does. Hence $\sigma \notin \mathcal{N}(h^*)$, so $\sigma \in \Delta(c,h^*)$. Each pair in $\Delta(c,h^*)$ sets at most k bits, so at most $k|\Delta(c,h^*)|$ bits differ. □

Remark (MinHash). A MinHash signature (Sec. 5.4) satisfies the same conclusion in expectation. Coordinate i differs between c and h^* only if the pair minimizing h_i over $\mathcal{N}(c) \cup \mathcal{N}(h^*)$ lies in $\Delta(c,h^*)$; for random hash functions this happens with probability $|\Delta(c,h^*)|/|\mathcal{N}(c) \cup \mathcal{N}(h^*)|$, so the expected number of differing coordinates is $k|\Delta(c,h^*)|/|\mathcal{N}(c) \cup \mathcal{N}(h^*)| \le k|\Delta(c,h^*)|$.

F.2.2 Lemma 2: Histogram Features

Let the histogram have bins $\{P_j\}$ that partition the possible (r,x) pairs (for RelDist, one bin per relation, $P_r = \{(r,x) : x \in \mathcal{E}\}$), with $f_j(e) = |\mathcal{N}(e) \cap P_j|$. Then $\|f_{\mathrm{hist}}(c) - f_{\mathrm{hist}}(h^*)\|_1 \le |\Delta(c,h^*)|$.

Proof. For each bin, $|f_j(c) - f_j(h^*)| \le |(\mathcal{N}(c) \triangle \mathcal{N}(h^*)) \cap P_j|$, because pairs in both neighborhoods contribute equally to both counts. Summing over the disjoint bins gives at most $|\Delta(c,h^*)|$. □

F.2.3 Lemma 3: Gradient-Trained Features

Let $f_{\mathrm{grad}}(e) = \theta_e^*$ be the entity vectors at a stationary point θ^* of the training loss L. Fix all parameters other than θ_e at θ^*, and assume the following hold for $e \in \{c, h^*\}$, with constants G and μ common to all entity pairs considered.

(A1) *Per-entity decomposition.* The terms of L that depend on θ_e are $F_e(\theta) = \sum_{\sigma \in \mathcal{N}(e)} \phi_\sigma(\theta) + \Psi(\theta)$, where ϕ_σ is the loss term of training pair $\sigma = (r, t)$ viewed as a function of the entity's own vector, which is the same function for every entity whose neighborhood contains σ, and Ψ collects terms that are the same for every entity (e.g., a per-entity regularizer, or the expected negative-sampling term under uniform corruption).

(A2) *Bounded gradients.* $\|\nabla \phi_\sigma\|_2 \le G$ on a convex set U containing θ_c^* and $\theta_{h^*}^*$.

(A3) *Local strong convexity.* F_e is μ-strongly convex on U.

Then

$$\|f_{\mathrm{grad}}(c) - f_{\mathrm{grad}}(h^*)\|_2 \le \frac{G}{\mu} |\Delta(c, h^*)|.$$

Proof. Stationarity gives $\nabla F_c(\theta_c^*) = 0$ and $\nabla F_{h^*}(\theta_{h^*}^*) = 0$. By (A1), the shared pairs and Ψ cancel in $F_c - F_{h^*}$, which therefore contains only the pairs in $\Delta(c, h^*)$:

$$\nabla F_c(\theta_{h^*}^*) = \nabla F_c(\theta_{h^*}^*) - \nabla F_{h^*}(\theta_{h^*}^*) = \sum_{\sigma \in \mathcal{N}(c) \setminus \mathcal{N}(h^*)} \nabla \phi_\sigma(\theta_{h^*}^*) - \sum_{\sigma \in \mathcal{N}(h^*) \setminus \mathcal{N}(c)} \nabla \phi_\sigma(\theta_{h^*}^*),$$

so $\|\nabla F_c(\theta_{h^*}^*)\|_2 \le G|\Delta(c, h^*)|$ by (A2). By (A3),

$$\mu \|\theta_{h^*}^* - \theta_c^*\|_2^2 \le \left\langle \nabla F_c(\theta_{h^*}^*) - \nabla F_c(\theta_c^*),\, \theta_{h^*}^* - \theta_c^* \right\rangle \le \|\nabla F_c(\theta_{h^*}^*)\|_2 \, \|\theta_{h^*}^* - \theta_c^*\|_2,$$

hence $\|\theta_c^* - \theta_{h^*}^*\|_2 \le \|\nabla F_c(\theta_{h^*}^*)\|_2 / \mu \le G|\Delta(c, h^*)|/\mu$. □

Remarks on Lemma 3. *Scaling with overlap.* If each ϕ_σ is μ_0-strongly convex on U and Ψ is μ_Ψ-strongly convex, then $\mu \ge \mu_\Psi + \mu_0 |\mathcal{N}(c)|$; since $|\mathcal{N}(c)| \ge S(c, h^*)$, the bound becomes $G|\Delta(c, h^*)|/(\mu_\Psi + \mu_0 S(c, h^*))$, which tightens as the shared overlap grows for a fixed number of unshared pairs. *On the assumptions.* KGE objectives are not globally convex, so (A3) is an assumption about the region around the solution reached by training, and the trained embeddings are idealized as a stationary point. Strong convexity can come from the pair terms themselves (e.g., a squared-distance objective) or from per-entity ℓ_2 regularization. Where (A3) fails, Lemma 3 gives no guarantee; Lemmas 1 and 2 hold unconditionally.

F.2.4 Combination Under Concatenation and Lipschitz Maps

Let $f_1, \ldots, f_n$ satisfy $\|f_i(c) - f_i(h^*)\|_2 \le D_i(|\Delta(c, h^*)|)$ (the ℓ_1 bounds of Lemmas 1 and 2 also bound the ℓ_2 distance). The concatenation $f = [f_1; \ldots; f_n]$ satisfies $\|f(c) - f(h^*)\|_2 \le \left(\sum_i D_i^2\right)^{1/2}$, which is non-decreasing in $|\Delta(c, h^*)|$ and zero when $\Delta(c, h^*) = \emptyset$. An L-Lipschitz map M applied afterwards preserves this up to the factor L: $\|M(f(c)) - M(f(h^*))\|_2 \le L\left(\sum_i D_i^2\right)^{1/2}$. This covers gHAWK's projection and fusion MLPs (a ReLU MLP is Lipschitz with constant at most the product of its layers' operator norms; dropout is inactive at evaluation) and the gate of Eq. (2), which is linear in its inputs.

Corollary A (score collapse). Let $z(e)$ be a representation covered by Proposition A, with $\|z(c) - z(h^*)\| \le D(|\Delta(c, h^*)|)$, and let the decoder score $\hat{s}_q(e)$ of a query q be L_s-Lipschitz in $z(e)$. Then, for every query q,

$$|\hat{s}_q(c) - \hat{s}_q(h^*)| \le L_s \, D(|\Delta(c, h^*)|).$$

For a RotatE head query $(?, r, t)$, the score $\hat{s}_q(e) = \gamma - \|z(e) \circ \mathbf{r} - \mathbf{t}\|$ is Lipschitz in $z(e)$, with constant 1 in the norm used by the distance, because rotation by $\mathbf{r}$ ($|r_i| = 1$) preserves the modulus of each coordinate. Candidates whose neighborhoods nearly coincide with that of h^* therefore receive nearly the same score as h^* on every query, so no representation covered by Proposition A can reliably rank h^* above them. This is the sense in which Proposition A implies a ceiling.

Implication. In the decoder-only configuration on WikiKG2, which has no domain-specific features, every input to the decoder is covered by Proposition A: the structural priors, the jointly learned entity table, and their gated fusion. App. F.3 shows that message passing over the training graph does not separate entities with identical neighborhoods either. Separating such entities requires conditioning on the query.

F.3 Connection to Message Passing and 1-Weisfeiler–Leman

Consider a message-passing GNN in which each layer computes an entity's new representation from its current representation and the multiset of messages from its neighbors, $\{(r, h_x^{(l)}) : (r, x) \in \widetilde{\mathcal{N}}(e)\}$, applied over full neighborhoods (as at inference). This covers relational convolutions, mean aggregation, and attention, whose weights depend only on these same inputs. If $\widetilde{\mathcal{N}}(c) = \widetilde{\mathcal{N}}(h^*)$ and $h_c^{(0)} = h_{h^*}^{(0)}$, then by induction every layer applies the same function to the same arguments for both entities, so $h_c^{(l)} = h_{h^*}^{(l)}$ at every depth. Since $\widetilde{\mathcal{N}}(c) = \widetilde{\mathcal{N}}(h^*)$ implies $\Delta(c, h^*) = \emptyset$, Proposition A gives equal query-independent inputs, and message passing therefore cannot separate the two entities. This is the simplest instance of the 1-WL limitation of message-passing GNNs [42]. Structural priors and message-passing GNNs thus share the same limitation for entities with identical neighborhoods. Query conditioning, as in labeling tricks [46] and NBFNet [49], changes the inputs per query and falls outside this argument.

G Extended Related Work

This appendix expands on the classification of related work presented in Sec. 6.

Scalable GNN training. Sampling-based methods reduce per-step computation by working on subsets of the graph. GraphSAGE [18] samples a fixed-size neighborhood per node; FastGCN [5] uses importance sampling at the layer level; ClusterGCN [7] constructs mini-batches from graph partitions, and GraphSAINT [45] from sampled subgraphs. TIGER [38] extends inductive GNN training to KG reasoning at scale through a structured training pipeline. gHAWK is complementary to all of these: rather than reducing message-passing cost, it precomputes structural features that remain informative regardless of which subset is sampled at each step, directly compensating for the structural information that sampling discards. The Bloom filter and global TransE are computed once over the full training graph and accessed in constant time per entity.

Precomputed structural features. DeepWalk [31] and node2vec [16] learn node embeddings from random walks on homogeneous graphs. Positional encodings for GNNs [11] augment node features with structural information about position in the graph. SIGN [13] precomputes propagated features so that training needs no message passing; it is a backbone rather than a structural prior, and it composes with gHAWK, which lifts SIGN from 37.54% to 40.46% on OGB-MAG (App. D.2). Bloom signatures [47] augment message-passing GNNs with compact hashed encodings of node neighborhoods for scalable link prediction on homogeneous graphs. For heterogeneous KGs, NodePiece [14] represents each entity by a fixed set of anchor tokens and its relational context, whose embeddings are learned jointly with the link-prediction objective. gHAWK differs from these in three ways. First, its structural priors stay frozen across tasks, while NodePiece learns its anchor embeddings for each downstream objective. Second, it combines *both* local (Bloom) and global (TransE) signals, while Bloom signatures [47] and similar methods rely on a single feature type. Third, it operates on top of the learned embedding through a gated fusion rather than replacing it, allowing the model to balance precomputed structural signal against task-specific learning during training.

Local–global representation learning. Combining local and global information is a recurring idea in other domains: time-series forecasting [9], multimodal sentiment analysis [10], node classification [12], protein–ligand binding affinity prediction [6], and self-supervised learning on molecular graphs [43]. In each case, both representations are learned jointly with the training objective on inputs that fit in memory. gHAWK differs in that its priors are frozen and computed once before any task training, target heterogeneous KGs with hundreds of relation types, and are reused unchanged across tasks and backbones.

Heterogeneous graph methods. Heterogeneous graph transformers such as HGT [22] apply type-aware attention over meta-relations, and PHGT [26] extends this with poly-tokenization for heterogeneous benchmark performance, achieving state-of-the-art results on DBLP, ACM, and IMDb through a heavyweight transformer architecture trained per-dataset. HOPE [48] and HGAMLP [23] are more recent approaches in the same space, optimizing different aspects of heterogeneous graph

learning. gHAWK reaches comparable or stronger accuracy on the same benchmarks with a GraphSAGE backbone and a single preprocessing recipe (Bloom + TransE) applied without per-dataset tuning, supporting the paper's claim that precomputed structural priors can substitute substantially for purpose-built architectures.

Query-conditioned KG completion. NBFNet [49] and A*Net [50] compute query-specific entity representations through path-based propagation. More generally, labeling tricks [46] condition node representations on the target link, which provably separates links that independently computed node representations cannot. These methods escape the limit characterized by Proposition A, but at a per-query cost: NBFNet exhausts GPU memory on WikiKG2 (as documented in [50]), and A*Net's accuracy falls below the gHAWK decoder despite its higher representational capacity per query. gHAWK occupies a different point in the design space, trading per-query expressiveness for the scalability of precomputed query-independent features, a trade-off that proves favorable on benchmarks where rare entities dominate the query distribution.

Long-tail entity prediction. Popularity-agnostic evaluation [29] documents that standard benchmarks under-weight rare-entity performance, with embedding model accuracy degrading sharply as entity popularity decreases. Most existing approaches address this indirectly through training-time interventions such as adjusted negative sampling or loss reweighting. gHAWK addresses long-tail entities directly: structural features remain informative regardless of training frequency, and Sec. 5.3.1's stratified analysis documents the benefit of capturing the long tail empirically across the bottom three frequency buckets that account for 99% of test queries.