BiLink: Bidirectional Meta-paths for Link Discovery in Billion-Scale Heterogeneous Graphs

Jinquan Hang
Rutgers University
Piscataway, NJ, USA
jinquan.hang@rutgers.edu

Zhiqing Hong*
Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, China
zhiqinghong@hkust-gz.edu.cn

Xinyue Feng
Rutgers University
Piscataway, NJ, USA
xinyue.feng@rutgers.edu

Desheng Zhang
Rutgers University
Piscataway, NJ, USA
desheng@cs.rutgers.edu

Haotian Wang
JD Logistics
Beijing, China
wanghaotian18@jd.com

Guang Wang
Florida State University
Tallahassee, FL, USA
guang@cs.fsu.edu

ABSTRACT

Link discovery in heterogeneous graphs, i.e., finding target edges between existing nodes, is a key task in many real-world applications. Since some path types between node pairs are related to target edges, the *meta-path* is widely used to represent these path types. However, the complexity of meta-path sampling grows exponentially with hop count, making long meta-paths impractical to sample on billion-scale graphs. Bidirectional search offers a natural way to split a long meta-path into two shorter meta-paths, each starting from one of its endpoints. However, different meta-paths have different structures, making it difficult to parallelize bidirectional meta-path sampling with uniform operations. Furthermore, existing models are not designed to jointly leverage the meta-path instances from bidirectional sampling. To address these challenges, we first introduce *bidirectional meta-path*, a unified path schema for bidirectional paths connecting node pairs. Based on this, we present BiLink, a framework for link discovery in billion-scale heterogeneous graphs. Specifically, BiLink represents all meta-paths in a unified table format and implements bidirectional meta-path sampling in distributed systems with uniform table operations. It also includes BiPathNN, a model that represents path instances as embedding sequences and jointly encodes them via the Transformer encoder for link prediction. Experiments show that bidirectional meta-path sampling is over 13 times faster than traditional meta-path sampling, while BiPathNN improves average precision by over 6% compared to state-of-the-art baselines. Moreover, BiLink has been deployed to discover companies' key personnel from 1.8 billion candidate pairs, increasing contact success rate by 86%.

PVLDB Reference Format:
Jinquan Hang, Zhiqing Hong, Xinyue Feng, Desheng Zhang, Haotian Wang, and Guang Wang. BiLink: Bidirectional Meta-paths for Link Discovery in Billion-Scale Heterogeneous Graphs. PVLDB, 19(12): 4316 - 4329, 2026.
doi:10.14778/3827998.3828035

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828035

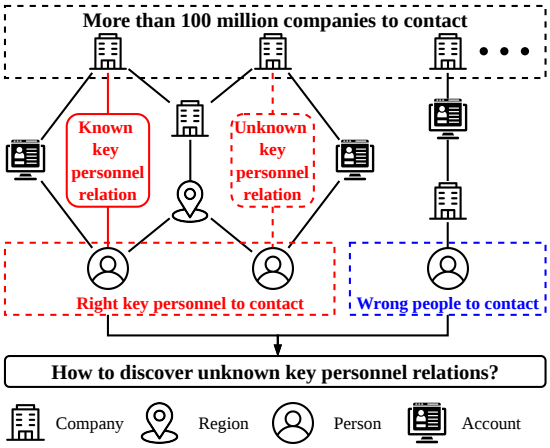

Figure 1: Problem of finding unknown key personnel.

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/JQHang/BiLink.

1 INTRODUCTION

Many real-world scenarios involve multiple types of entities connected by diverse relations, which are naturally represented as heterogeneous graphs [55, 77]. Link discovery [49, 63, 79] in heterogeneous graphs, i.e., finding target edges between existing nodes, has become increasingly important [16, 57, 67] and has been applied in areas such as social networks [24, 72, 85] and knowledge graph completion [7, 32, 65]. In this paper, we primarily focus on **link discovery in billion-scale heterogeneous graphs** and its application to discovering companies' key personnel. As illustrated in Figure 1, identifying key personnel enables service providers to contact the right people for product promotion. Moreover, our solution can be applied to other link discovery tasks, such as discovering authors' new references and coauthors.

Since unknown target edges cannot be directly queried from the graph, Heterogeneous Graph Neural Networks (HGNNs) [26, 63] are widely adopted to predict them by aggregating neighborhood information of node pairs. When applied to large-scale graphs [35,

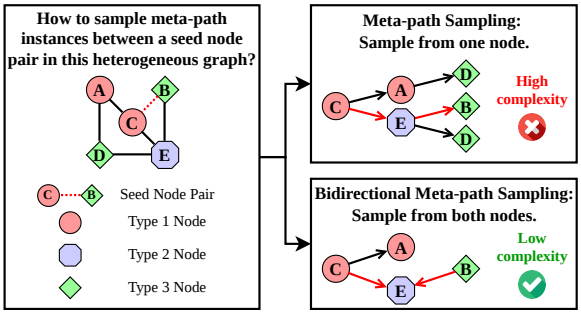

Figure 2: Bidirectional search reduces the complexity of sampling meta-path instances between a node pair.

37, 48], most HGNNs require subgraph sampling to control computational costs. The sampling approaches can be broadly categorized into k-hop sampling and meta-path sampling. K-hop sampling [23, 34] collects all neighbors within k hops of each node. HGNNs based on this approach, such as R-GCNs [53] and HGT [29], typically aggregate neighborhood information hop by hop. However, due to the neighbor explosion problem [66], k is typically limited to 2 for large-scale graphs [5], so these methods struggle to capture long-range structural information. To address this limitation, meta-path sampling was introduced. A meta-path [56] is a path schema defined by a sequence of node and edge types. For example, "Company → Account → Person" represents a meta-path for the graph in Figure 1. Given a meta-path, meta-path sampling starts from a seed node and samples paths following the meta-path. HGNNs based on this approach, such as MAGNN [19] and SeHGNN [71], typically aggregate neighborhood information along each meta-path. However, since the number of sampled paths still grows exponentially with path length, sampling long meta-paths remains time-consuming for large-scale graphs.

When sampling meta-paths between node pairs for link discovery, bidirectional search [54] provides a natural way to reduce complexity. In a graph with average degree d, the number of sampled paths in meta-path sampling grows by a factor of d at each hop, resulting in a complexity of $O(d^k)$ for a k-hop meta-path. As illustrated in Figure 2, instead of sampling from a single node, bidirectional meta-path sampling starts from both nodes of the pair. As each node only needs to sample a $\frac{k}{2}$-hop meta-path, this reduces complexity to $O(d^{\frac{k}{2}})$. Therefore, bidirectional meta-path sampling makes it feasible to sample long meta-paths between node pairs for large-scale link discovery.

However, applying this approach faces two key challenges:

- **(i) Bidirectional Sampling in Distributed Systems.** Running bidirectional meta-path sampling on billion-scale graphs requires implementation in distributed systems [2, 21, 74], which should leverage uniform operations [73] to process data in parallel. However, different meta-paths have different structures, making it challenging to implement bidirectional meta-path sampling with uniform operations.
- **(ii) Link Prediction with Bidirectional Meta-paths.** Bidirectional meta-path sampling first samples meta-paths from each node and then intersects them. This produces both meta-paths around each node and meta-paths connecting the node pair. However, most existing meta-path-based models [64, 71] are not designed to jointly leverage these meta-paths, requiring a specialized model for effective link prediction.

To address these challenges, we first propose *bidirectional meta-path*, a unified path schema for bidirectional paths connecting node pairs. Based on this, we present BiLink, a novel framework for link discovery in billion-scale heterogeneous graphs. Specifically, BiLink represents all meta-paths as rows in a unified table format and implements bidirectional meta-path sampling in distributed systems through uniform table operations. Moreover, it includes a model named Bidirectional Meta-path Neural Network (BiPathNN), which represents all path instances of each node pair as embedding sequences and jointly encodes them via the Transformer encoder for link prediction. Based on these methods, BiLink first uses bidirectional meta-path sampling to select candidate node pairs and sample corresponding meta-path instances. Then, BiLink trains BiPathNN on known target edges and applies it to discover unknown target edges among all candidate node pairs.

In summary, this paper makes the following contributions.

- To the best of our knowledge, we are the first to propose *bidirectional meta-path*, a unified path schema that enables leveraging longer meta-paths for link discovery on billion-scale heterogeneous graphs.
- We introduce a novel framework, BiLink, which designs a bidirectional meta-path sampling algorithm specifically for distributed systems and includes BiPathNN, an effective model that jointly leverages the sampling results for link prediction.
- Extensive experiments on three real-world tasks demonstrate that bidirectional meta-path sampling is over 13 times faster than traditional meta-path sampling, while BiPathNN improves average precision by over 6% compared to state-of-the-art baselines.
- We deploy BiLink to discover companies' key personnel from 1.8 billion candidate pairs, increasing contact success rate by 86%. We also apply it to discover authors' new references and coauthors, recalling over 31% of them within the top 1% of predicted pairs.

2 PRELIMINARY

2.1 Heterogeneous Graph

Although traditional Labeled Property Graphs (LPGs) [3, 18] in graph databases [11] can represent different node and edge types with diverse properties, heterogeneous graphs [39, 55] offer a more uniform structure compatible with Graph Neural Networks (GNNs):

Definition 2.1. **Heterogeneous Graph (HG).** A heterogeneous graph $HG = (V, E)$ is a graph consisting of different types of nodes (i.e., entities) and/or different types of edges (i.e., relations), each type having a fixed feature format (i.e., properties):

- Each node $v \in V$ is associated with a node type $o \in O$ and a feature vector $\mathbf{f}_v \in \mathbb{R}^{d_o}$, where d_o is the feature dimension.
- Each edge $e \in E$ is associated with an edge type $l \in L$ and a feature vector $\mathbf{f}_e \in \mathbb{R}^{d_l}$, where d_l is the feature dimension.
- Multiple node types or edge types exist: $|O| > 1$ or $|L| > 1$.

When applying GNNs to large-scale heterogeneous graphs, subgraph sampling is essential to control computational costs. A common approach is k-hop sampling, which collects neighbors within

k hops from a seed node. However, the number of sampled nodes grows exponentially with the hop count, making it impractical for large hop counts [66]. Another approach is meta-path sampling [56], which samples paths following a predefined meta-path:

Definition 2.2. **Meta-path.** A meta-path p is a path schema on a heterogeneous graph HG denoted in the sequential form $o_0 \xrightarrow{l_0} o_1 \xrightarrow{l_1} \cdots \xrightarrow{l_{k-1}} o_k$, where o_i is the type of the i-th node, l_i is the type of the i-th edge, and k is the hop count.

Given a meta-path, meta-path sampling collects corresponding path instances from a seed node:

Definition 2.3. **Meta-path Sampling.** In a heterogeneous graph $HG = (V, E)$, given a seed node v_0 and a meta-path p, meta-path sampling identifies sequences of nodes and edges $v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-1}} v_k$ starting from v_0 that follow p. These sequences are called meta-path instances.

2.2 Bidirectional Meta-path

As Figure 2 shows, bidirectional search can significantly reduce the cost of sampling meta-paths between node pairs. Although it is widely used in directed graphs [54], applying it to heterogeneous graphs requires a path schema for bidirectional paths connecting node pairs. Therefore, we propose the following definition:

Definition 2.4. **Bidirectional Meta-path.** A bidirectional meta-path p^{bi} is a path schema on a heterogeneous graph HG denoted in the bidirectional sequential form $o_0 \xrightarrow{l_0} \cdots \xrightarrow{l_{m-1}} o_m \xleftarrow{l_m} \cdots \xleftarrow{l_{k-1}} o_k$. Specifically, for each bidirectional meta-path:

- The forward path $o_0 \xrightarrow{l_0} \cdots \xrightarrow{l_{m-1}} o_m$ is referred to as the forward meta-path, denoted as p_{fwd}.
- The backward path $o_k \xrightarrow{l_{k-1}} \cdots \xrightarrow{l_m} o_m$ is referred to as the backward meta-path, denoted as p_{bwd}.
- o_m is the type of the join node, which connects the forward and backward meta-paths.

Given a bidirectional meta-path, bidirectional meta-path sampling collects corresponding path instances from a seed node pair:

Definition 2.5. **Bidirectional Meta-path Sampling.** In a heterogeneous graph $HG = (V, E)$, given a seed node pair (v_a, v_b) and a bidirectional meta-path $p^{\text{bi}} = (p_{\text{fwd}}, o_m, p_{\text{bwd}})$, bidirectional meta-path sampling first identifies:

- Forward meta-path instances $v_a \xrightarrow{e_0} \cdots \xrightarrow{e_{m-1}} v_m$ starting from v_a that follow p_{fwd}.
- Backward meta-path instances $v_b \xrightarrow{e_{k-1}} \cdots \xrightarrow{e_m} v_m$ starting from v_b that follow p_{bwd}.

Then, by joining forward and backward instances sharing the same join node v_m, it obtains bidirectional meta-path instances following the sequential form of p^{bi}: $v_a \xrightarrow{e_0} \cdots \xrightarrow{e_{m-1}} v_m \xleftarrow{e_m} \cdots \xleftarrow{e_{k-1}} v_b$.

Figure 3 illustrates an example of bidirectional meta-path sampling for a 2-hop bidirectional meta-path. Specifically, the bidirectional meta-path "*Type 1 node* → *Type 2 node* ← *Type 3 node*" consists of a forward meta-path "*Type 1 node* → *Type 2 node*" and a backward meta-path "*Type 3 node* → *Type 2 node*". Sampling

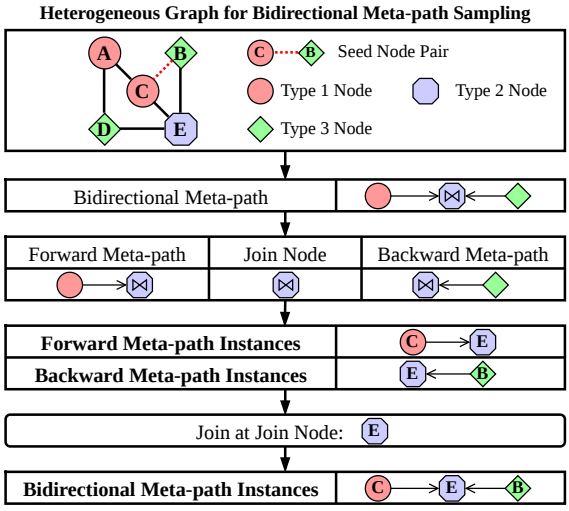

Figure 3: Example of bidirectional meta-path sampling.

from the seed node pair (C, B) yields 1 forward and 1 backward meta-path instance. By joining them at join node E, we obtain 1 bidirectional meta-path instance.

2.3 Problem Definition

In this paper, we focus on link discovery in billion-scale heterogeneous graphs, i.e., using known target edges to find unknown target edges between candidate nodes. Formally, given:

- A heterogeneous graph $HG = (V, E)$.
- A target edge type l^{tgt} connecting node types o_a^{tgt} and o_b^{tgt}, with known edge set $E_{\text{kn}}^{\text{tgt}}$.
- Candidate node sets V_a^{cand} and V_b^{cand} of types o_a^{tgt} and o_b^{tgt} respectively, containing nodes that can be connected by target edges of type l^{tgt}.

We aim to discover the unknown target edge set $E_{\text{unk}}^{\text{tgt}}$ between candidate node sets V_a^{cand} and V_b^{cand}.

3 DATA FORMULATION

3.1 Graph Construction

Traditional graph neural network platforms like PyG [17] and DGL [62] require loading the entire graph into memory, making them unsuitable for graphs that exceed memory limits [12, 61]. To handle such large-scale graphs, industrial platforms like GraphLearn [87] and AGL [78] adopt distributed systems to store and process graph data. Following these platforms, we store and process our billion-scale graph $HG = (V, E)$ in distributed systems as two types of tables:

- **Node Table:** Node table V stores all nodes in three columns: V.o for the node type $o \in O$, V.v for the node identifier, and V.f for the d_o-dimensional feature vector.
- **Edge Table:** Edge table E stores all edges in four columns: E.l for the edge type $l \in L$, connecting nodes of types o_a and o_b, E.v_a and E.v_b for the identifiers of the two connected nodes, and E.f for the d_l-dimensional feature vector.

Table 1: Example meta-path instance table.

P.k	P.O	P.L	P.E_a	P.E_b	P.V
2	[Company, Account, Person]	[Company-Account, Person-Account]	[0, 2]	[1, 1]	[Company A, Account A, Person A]

Table 2: Example subgraph table, with each row shown in two parts.

S.v_a	S.v_b	S.P.n_p	S.P.t	S.P.k	S.P.O	S.P.L	S.P.E_a	S.P.E_b	S.P.V
Company A	Person A	200	[forward, bidirectional, ...]	[1, 1, ...]	[[Account], [], ...]	[[Company-Account], [Person-Company], ...]	[[0], [1], ...]	[[1], [0], ...]	[[Account A], [], ...]

S.V.n_v	S.V.o	S.V.v	S.V.f	S.E.n_e	S.E.l	S.E.v_a	S.E.v_b	S.E.f
500	[Company, ...]	[Company A, ...]	[[0.3, ...], ...]	300	[Company-Account, ...]	[Company A, ...]	[Account A, ...]	[[0.4, ...], ...]

For link discovery tasks, we additionally define two types of tables for known target edges and candidate nodes:

- **Known Target Edge Table:** Known target edge table $\mathrm{E}_{\mathrm{kn}}^{\mathrm{tgt}}$ stores all known edges of target edge type l^{tgt} in two columns: $\mathrm{E}_{\mathrm{kn}}^{\mathrm{tgt}}.v_a$ and $\mathrm{E}_{\mathrm{kn}}^{\mathrm{tgt}}.v_b$ for the identifiers of the two connected nodes.
- **Candidate Node Tables:** Candidate node tables $\mathrm{V}_a^{\mathrm{cand}}$ and $\mathrm{V}_b^{\mathrm{cand}}$ store candidate nodes of types o_a^{tgt} and o_b^{tgt} respectively. Each table has one column: $\mathrm{V}_a^{\mathrm{cand}}.v$ or $\mathrm{V}_b^{\mathrm{cand}}.v$ for the node identifier.

3.2 Meta-path Representation

To implement bidirectional meta-path sampling in distributed systems, we store meta-paths and bidirectional meta-paths in a unified tabular format. Their instances are obtained by traversing edges in the edge table E, where each edge of type l_i connects nodes of types o_a and o_b. Since such an edge can be traversed in either direction, each hop $o_i \xrightarrow{l_i} o_{i+1}$ may correspond to either $o_a \xrightarrow{l_i} o_b$ or $o_b \xrightarrow{l_i} o_a$. Accordingly, we define the following table:

- **Meta-path Table:** Meta-path table M stores meta-paths and bidirectional meta-paths as their node type and edge type sequences. Each row stores one path schema with the following columns:
  - M.k stores the hop count k of the path schema.
  - M.O stores the node types along the path schema as a $(k+1)$-length array $[o_0, o_1, \ldots, o_k]$.
  - M.L stores the edge types along the path schema as a k-length array $[l_0, l_1, \ldots, l_{k-1}]$.
  - M.E_a and M.E_b are k-length arrays. For the i-th edge of type l_i connecting node types o_a and o_b, $\mathrm{M}.O[E_a[i]] = o_a$ and $\mathrm{M}.O[E_b[i]] = o_b$.

Based on the meta-path table, we define another table to store path instances:

- **Meta-path Instance Table:** Meta-path instance table P contains instances of both meta-paths and bidirectional meta-paths. Each row stores a path instance and its corresponding path schema with the following columns:
  - P.k, P.O, P.L, P.E_a, and P.E_b store the path schema for each instance, where each column corresponds to the same column in the Meta-path Table.
  - P.V stores the node identifiers along the path instance as a $(k+1)$-length array $[v_0, v_1, \ldots, v_k]$. Moreover, $\mathrm{P}.V[E_a[i]]$ and $\mathrm{P}.V[E_b[i]]$ store the node identifiers of v_a and v_b connected by the i-th edge e_i.

Table 1 shows a meta-path instance table with one meta-path instance "Company A → Account A → Person A" of the meta-path "Company → Account → Person".

Moreover, as defined in Definition 2.5, bidirectional meta-path sampling starts from seed node pairs. Seed node pairs can be stored as a seed node pair table $\mathrm{E}^{\mathrm{seed}}$, where each row contains a node pair (v_a, v_b) with their node types (o_a, o_b); or as two seed node tables $\mathrm{V}_a^{\mathrm{seed}}$ and $\mathrm{V}_b^{\mathrm{seed}}$, representing all pairs in $\mathrm{V}_a^{\mathrm{seed}} \times \mathrm{V}_b^{\mathrm{seed}}$.

3.3 Subgraph Representation

For link prediction using bidirectional meta-path sampling, each node pair requires a subgraph of its sampled paths and the feature vectors of the nodes and edges along them. However, these paths and feature vectors are stored in separate rows across the Node, Edge, and Meta-path Instance tables. Reading these rows individually requires many separate reads, which is slow in distributed systems. We therefore store each node pair's subgraph in a single row, defined as follows:

- **Subgraph Table:** Subgraph table S stores the subgraph formed by paths sampled for each node pair. Each row stores a node pair with its corresponding paths and feature vectors with the following columns:
  - S.v_a and S.v_b store the identifiers of the two nodes.
  - S.P stores all corresponding path instances:
    - S.P.n_p stores the number of path instances.
    - S.P.t is an n_p-length array marking each path instance as "forward", "backward", or "bidirectional".
    - S.P.k, S.P.O, S.P.L, S.P.E_a, S.P.E_b, and S.P.V are n_p-length arrays collected from the same columns in the Meta-path Instance Table. Moreover, S.P.O and S.P.V omit the nodes already known from S.v_a, S.v_b, and S.P.t.
  - S.V stores all unique nodes appearing in the subgraph:
    - S.V.n_v stores the number of nodes.
    - S.V.o, S.V.v, and S.V.f are n_v-length arrays storing the type, identifier, and feature vector of each node.
  - S.E stores all unique edges appearing in the subgraph:
    - S.E.n_e stores the number of edges.
    - S.E.l, S.E.v_a, S.E.v_b, and S.E.f are n_e-length arrays storing the type, the identifiers of the two connected nodes, and the feature vector of each edge.

Table 2 shows a subgraph table with one subgraph for node pair (Company A, Person A).

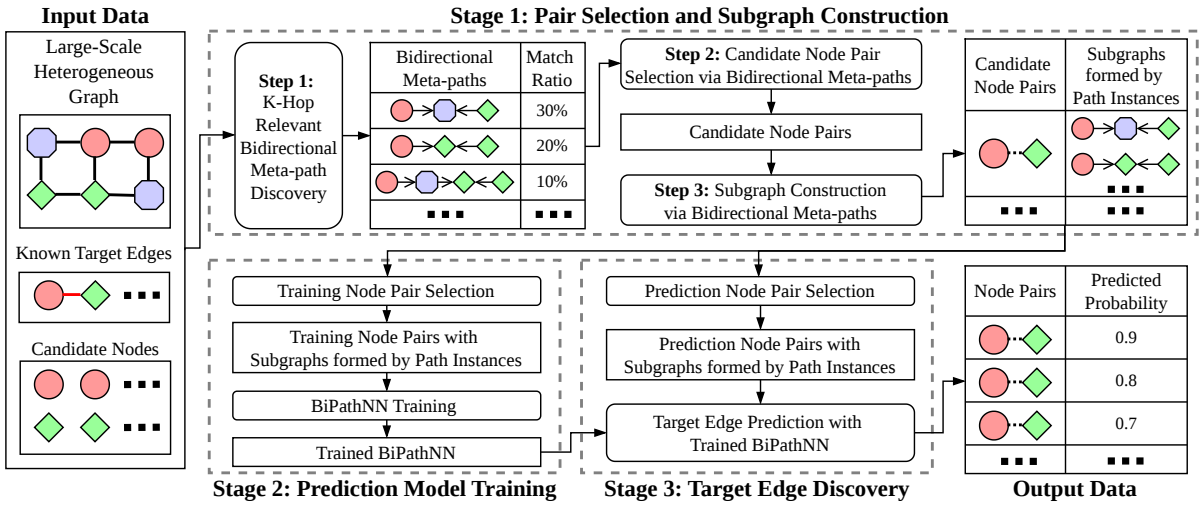

Figure 4: The overall structure of our BiLink framework.

Table 3: Notation used in this section.

Symbol	Meaning
k_{bi}	Max hop count for considered bidirectional meta-paths
n_{sample}	Max sampled path instances per meta-path for each node pair
n_{cand}, n_{sub}	# top-ranked bidirectional meta-paths used for candidate node pair selection / subgraph construction
n_{layer}, n_{head}	# Transformer encoder layers / self-attention heads
n_{neg}	# negative samples per positive sample
m_{group}	Max number of elements per path group
m_{pair}	Max number of path groups per node pair
d_{model}	Hidden dimension of our link prediction model
V, E	Node and edge tables defined in Section 3.1
V_a^{cand}, V_b^{cand}, $\text{E}_{\text{kn}}^{\text{tgt}}$	Candidate node and known target edge tables defined in Section 3.1
M, P	Meta-path and meta-path instance tables defined in Section 3.2
P_{fwd}, P_{bwd}, P_{bi}	Tables of forward / backward / bidirectional meta-path instances
M_{match}	Bidirectional meta-paths ranked by match ratio
E_{cand}	Candidate node pair table for link discovery
S	Subgraph table (defined in Section 3.3) for candidate node pairs
S_{label}, S_{pred}	Subgraph tables for model training / link discovery
Y_{label}	Binary labels of candidate node pairs selected for model training
Y_{train}, Y_{pred}	Predicted probabilities for model training / link discovery
$\text{I}_{\text{P}\to\text{X}}$, $\text{I}_{\text{P}\to\text{pos}}$	Feature / position indices for each path group element
$\text{I}_{\text{pair}\to\text{P}}$	Path group indices for each node pair
$\mathcal{X}$	Dictionary mapping each node / edge type to its feature matrix

4 METHODOLOGY

In this section, we introduce BiLink, our framework for link discovery in billion-scale heterogeneous graphs. Following existing link discovery methods [26, 32], BiLink first selects candidate node pairs to reduce the search space, then performs link prediction on them. As shown in Figure 4, BiLink consists of three stages: (1) select candidate node pairs and construct a subgraph for each pair; (2) train our link prediction model BiPathNN using candidate pairs labeled by known target edges; (3) apply the trained BiPathNN to discover unknown target edges. Table 3 summarizes the notation.

4.1 Stage 1: Pair Selection and Subgraph Construction

Motivation. Given candidate node sets V_a^{cand} and V_b^{cand}, the search space for unknown target edges is $V_a^{\text{cand}} \times V_b^{\text{cand}}$. This results in $O(N^2)$ possible pairs, where N is the candidate set size. In our tasks, N exceeds 10^8, making exhaustive link prediction infeasible. To reduce the search space, most existing methods [6] select candidate pairs more likely to be connected by target edges for link prediction.

Following this strategy, we use bidirectional meta-path sampling to identify candidate node pairs and construct a subgraph for each pair. As shown in Figure 4, this stage has three steps: (1) identify bidirectional meta-paths connecting the two target node types and rank them by relevance; (2) use the top-ranked meta-paths to select candidate node pairs; (3) sample path instances around candidate pairs to construct subgraphs. Since all three steps rely on bidirectional meta-path sampling on distributed systems, we first introduce its implementation, then describe each step in detail.

Bidirectional Meta-path Sampling. Our implementation of bidirectional meta-path sampling (BiPathSampling) is based on the tables defined in Section 3 and uses only table operations (e.g., join, filter, union), enabling efficient execution on distributed systems. Algorithm 1 shows the pseudo-code of BiPathSampling, with the input and output defined as:

$$\text{P}_{\text{fwd}}, \text{P}_{\text{bwd}}, \text{P}_{\text{bi}} = \text{BiPathSampling}\left(\begin{cases}\text{E}^{\text{seed}} \\ (\text{V}_a^{\text{seed}}, \text{V}_b^{\text{seed}})\end{cases}, \text{E}, k_{\max}, \text{M}_{\text{bi}}\right) \quad (1)$$

where the first input defines the seed node pairs, specified as a seed node pair table E^{seed} or seed node tables $(\text{V}_a^{\text{seed}}, \text{V}_b^{\text{seed}})$ (see Section 3.2). M_{bi} specifies which bidirectional meta-paths to sample; if omitted, BiPathSampling enumerates bidirectional meta-paths of up to $k_{\max}$ hops. Moreover, we limit each node pair to at most n_{sample} sampled path instances per meta-path and bidirectional meta-path.

Algorithm 1 has two blocks: Block 1 (Lines 1–13) iteratively samples forward and backward meta-path instances of up to $\lceil \frac{k_{\max}}{2} \rceil$ hops, and Block 2 (Lines 14–22) joins them at shared join nodes to form bidirectional meta-path instances.

Step 1: Relevant Bidirectional Meta-path Discovery. For identifying bidirectional meta-paths relevant to target edges, we first enumerate bidirectional meta-paths of up to k_{bi} hops. To rank them efficiently in parallel, we use each meta-path's match ratio against known target edges $\text{E}_{\text{kn}}^{\text{tgt}}$. First, we sample two seed node sets to

Algorithm 1 Bidirectional Meta-path Sampling (BiPathSampling).

Input: Seed node pairs, specified as seed node pair table $\mathrm{E}^{\mathrm{seed}}$ or seed node tables $(\mathrm{V}_a^{\mathrm{seed}}, \mathrm{V}_b^{\mathrm{seed}})$; edge table E; max hop $k_{\max}$; meta-path table M_{bi} of specified bidirectional meta-paths (optional).
Output: Path tables $\mathrm{P}_{\mathrm{fwd}}$, $\mathrm{P}_{\mathrm{bwd}}$, and P_{bi} of sampled forward, backward, and bidirectional meta-path instances, respectively.

//===== **Block 1: Unidirectional Path Sampling** =====//
1: $\mathrm{E}_{\mathrm{trav}} \leftarrow \mathrm{GetTraversalEdges}(\mathrm{E})$
2: **if** M_{bi} is provided **then**
3: $\quad \mathrm{M}_{\mathrm{fwd}}, \mathrm{M}_{\mathrm{bwd}} \leftarrow \mathrm{SplitPathSchema}(\mathrm{M}_{\mathrm{bi}})$
4: **if** $\mathrm{E}^{\mathrm{seed}}$ is provided **then**
5: $\quad \mathrm{V}_a^{\mathrm{seed}}, \mathrm{V}_b^{\mathrm{seed}} \leftarrow \mathrm{ExtractSeedNodes}(\mathrm{E}^{\mathrm{seed}})$
6: $\mathcal{P} \leftarrow \{0 : \mathrm{SeedNodeToPath}(\mathrm{V}_a^{\mathrm{seed}}, \mathrm{V}_b^{\mathrm{seed}})\}$
7: **for** $k \leftarrow 1$ **to** $\lceil \frac{k_{\max}}{2} \rceil$ **do**
8: $\quad \mathrm{P}_{\mathrm{prev}} \leftarrow \mathcal{P}[k-1]$
9: $\quad$ **if** M_{bi} is provided **then**
10: $\quad\quad \mathrm{M}_{\mathrm{fwd}}^{k} \leftarrow \mathrm{GetFirstKHops}(\mathrm{M}_{\mathrm{fwd}}, k)$
11: $\quad\quad \mathrm{M}_{\mathrm{bwd}}^{k} \leftarrow \mathrm{GetFirstKHops}(\mathrm{M}_{\mathrm{bwd}}, k)$
12: $\quad\quad \mathrm{P}_{\mathrm{prev}} \leftarrow \mathrm{AddNextEdgeType}(\mathrm{P}_{\mathrm{prev}}, \mathrm{M}_{\mathrm{fwd}}^{k}, \mathrm{M}_{\mathrm{bwd}}^{k})$
13: $\quad \mathcal{P}[k] \leftarrow \mathrm{JoinNextEdge}(\mathrm{P}_{\mathrm{prev}}, \mathrm{E}_{\mathrm{trav}})$
//===== **Block 2: Bidirectional Path Construction** =====//
14: $\mathrm{P}_{\mathrm{all}} \leftarrow \mathrm{Union}(\{\, \mathcal{P}[k] \mid k \in [0, \lceil \frac{k_{\max}}{2} \rceil] \,\})$
15: $\mathrm{P}_{\mathrm{fwd}}, \mathrm{P}_{\mathrm{bwd}} \leftarrow \mathrm{SeparatePaths}(\mathrm{P}_{\mathrm{all}})$
16: **if** M_{bi} is provided **then**
17: $\quad \mathrm{P}_{\mathrm{fwd}} \leftarrow \mathrm{FilterByMetaPath}(\mathrm{P}_{\mathrm{fwd}}, \mathrm{M}_{\mathrm{fwd}})$
18: $\quad \mathrm{P}_{\mathrm{bwd}} \leftarrow \mathrm{FilterByMetaPath}(\mathrm{P}_{\mathrm{bwd}}, \mathrm{M}_{\mathrm{bwd}})$
19: $\quad \mathrm{P}_{\mathrm{fwd}} \leftarrow \mathrm{AddBiPathType}(\mathrm{P}_{\mathrm{fwd}}, \mathrm{M}_{\mathrm{bi}})$
20: **if** $\mathrm{E}^{\mathrm{seed}}$ is provided **then**
21: $\quad \mathrm{P}_{\mathrm{fwd}} \leftarrow \mathrm{AddPairedSeedNode}(\mathrm{P}_{\mathrm{fwd}}, \mathrm{E}^{\mathrm{seed}})$
22: $\mathrm{P}_{\mathrm{bi}} \leftarrow \mathrm{JoinPaths}(\mathrm{P}_{\mathrm{fwd}}, \mathrm{P}_{\mathrm{bwd}})$
23: **return** $\mathrm{P}_{\mathrm{fwd}}, \mathrm{P}_{\mathrm{bwd}}, \mathrm{P}_{\mathrm{bi}}$

enumerate bidirectional meta-paths between them:

$$\mathrm{V}_a^{\mathrm{disc}}, \mathrm{V}_b^{\mathrm{disc}} = \mathrm{GetDiscoverySeeds}(\mathrm{E}_{\mathrm{kn}}^{\mathrm{tgt}}, \mathrm{V}_a^{\mathrm{cand}}, \mathrm{V}_b^{\mathrm{cand}}) \tag{2}$$

where GetDiscoverySeeds randomly samples node pairs from $\mathrm{E}_{\mathrm{kn}}^{\mathrm{tgt}}$ and individual nodes from $\mathrm{V}_a^{\mathrm{cand}}$ and $\mathrm{V}_b^{\mathrm{cand}}$ to form the two seed node sets.

Next, we use BiPathSampling to enumerate bidirectional meta-paths between the sampled seed nodes:

$$\mathrm{P}_{\mathrm{fwd}}^{\mathrm{disc}}, \mathrm{P}_{\mathrm{bwd}}^{\mathrm{disc}}, \mathrm{P}_{\mathrm{bi}}^{\mathrm{disc}} = \mathrm{BiPathSampling}((\mathrm{V}_a^{\mathrm{disc}}, \mathrm{V}_b^{\mathrm{disc}}), \mathrm{E}, k_{\mathrm{bi}}) \tag{3}$$

where no M_{bi} is specified, BiPathSampling enumerates bidirectional meta-paths of up to k_{bi} hops.

Finally, we rank each enumerated bidirectional meta-path by its match ratio:

$$\mathrm{M}_{\mathrm{match}} = \mathrm{RankByMatchRatio}(\mathrm{P}_{\mathrm{bi}}^{\mathrm{disc}}, \mathrm{E}_{\mathrm{kn}}^{\mathrm{tgt}}) \tag{4}$$

where RankByMatchRatio computes the match ratio of each bidirectional meta-path as the proportion of its connected node pairs that correspond to known target edges in $\mathrm{E}_{\mathrm{kn}}^{\mathrm{tgt}}$, and returns $\mathrm{M}_{\mathrm{match}}$ sorted by match ratio in descending order.

Step 2: Candidate Node Pair Selection. To select candidate node pairs for link prediction, we use the most relevant bidirectional meta-paths identified in the previous step to find node pairs that are likely connected by target edges. First, we select the top-ranked bidirectional meta-paths based on match ratio:

$$\mathrm{M}_{\mathrm{cand}} = \mathrm{Head}(\mathrm{M}_{\mathrm{match}}, n_{\mathrm{cand}}) \tag{5}$$

where Head returns the top n_{cand} bidirectional meta-paths in $\mathrm{M}_{\mathrm{match}}$.

Next, we sample path instances of these bidirectional meta-paths between all candidate nodes:

$$\mathrm{P}_{\mathrm{fwd}}^{\mathrm{cand}}, \mathrm{P}_{\mathrm{bwd}}^{\mathrm{cand}}, \mathrm{P}_{\mathrm{bi}}^{\mathrm{cand}} = \mathrm{BiPathSampling}((\mathrm{V}_a^{\mathrm{cand}}, \mathrm{V}_b^{\mathrm{cand}}), \mathrm{E}, k_{\mathrm{bi}}, \mathrm{M}_{\mathrm{cand}}) \tag{6}$$

where $\mathrm{M}_{\mathrm{cand}}$ restricts BiPathSampling to sample only the specified bidirectional meta-paths between candidate nodes $\mathrm{V}_a^{\mathrm{cand}}$ and $\mathrm{V}_b^{\mathrm{cand}}$.

Finally, we extract candidate node pairs from the sampled paths:

$$\mathrm{E}_{\mathrm{cand}} = \mathrm{SelectCandPairs}(\mathrm{P}_{\mathrm{bi}}^{\mathrm{cand}}) \tag{7}$$

where SelectCandPairs extracts node pairs connected by bidirectional meta-path instances in $\mathrm{P}_{\mathrm{bi}}^{\mathrm{cand}}$.

Step 3: Subgraph Construction. To discover target edges among candidate node pairs, we first select the top n_{sub} bidirectional meta-paths based on match ratio:

$$\mathrm{M}_{\mathrm{sub}} = \mathrm{Head}(\mathrm{M}_{\mathrm{match}}, n_{\mathrm{sub}}) \tag{8}$$

where n_{sub} should be large enough for the model to capture interactions among diverse meta-paths.

Next, we sample path instances following $\mathrm{M}_{\mathrm{sub}}$:

$$\mathrm{P}_{\mathrm{fwd}}^{\mathrm{sub}}, \mathrm{P}_{\mathrm{bwd}}^{\mathrm{sub}}, \mathrm{P}_{\mathrm{bi}}^{\mathrm{sub}} = \mathrm{BiPathSampling}(\mathrm{E}_{\mathrm{cand}}, \mathrm{E}, k_{\mathrm{bi}}, \mathrm{M}_{\mathrm{sub}}) \tag{9}$$

Finally, we construct the subgraph formed by the sampled path instances for each pair:

$$\mathrm{S} = \mathrm{ConstructSubgraphForPair}(\mathrm{E}_{\mathrm{cand}}, \mathrm{P}_{\mathrm{fwd}}^{\mathrm{sub}}, \mathrm{P}_{\mathrm{bwd}}^{\mathrm{sub}}, \mathrm{P}_{\mathrm{bi}}^{\mathrm{sub}}, \mathrm{V}, \mathrm{E}) \tag{10}$$

where S is the subgraph table as defined in Section 3.3.

4.2 Stage 2: Prediction Model Training

Motivation. We aim to train a link prediction model using known target edges. However, most existing meta-path-based models [64] are not designed to jointly leverage the forward, backward, and bidirectional meta-path instances in our subgraphs: they aggregate path instances into node-level [19, 71] or pair-level [26] embeddings, missing the interactions among them. Therefore, we design the Bidirectional Meta-path Neural Network (BiPathNN), which represents each path instance as a sequence of embeddings and jointly encodes them via the Transformer encoder [60] for effective link prediction. Since the Transformer encoder's complexity grows quadratically with sequence length, we divide path instances into groups, encoding each group first and then encoding all groups together. Below, we first introduce BiPathNN's design, then explain how we train it using known target edges.

BiPathNN. As shown in Figure 5, BiPathNN consists of four modules. First, a dataloader divides each node pair's path instances into path groups and converts them into matrix format. Second, we form a sequence of element embeddings for each path group by combining each element's feature and position embeddings. Third, we encode each path group's element embeddings into a path group embedding. Finally, we encode each node pair's path group embeddings into a pair embedding for link prediction. The inputs and outputs of BiPathNN can be expressed as follows:

$$\mathrm{Y} = \mathrm{BiPathNN}(\mathrm{S}, m_{\mathrm{group}}, O, L). \tag{11}$$

where $\mathrm{Y} \in \mathbb{R}^{n_{\mathrm{pair}}}$ represents the predicted probabilities for all node pairs in the subgraph table S, n_{pair} is the number of node pairs, and m_{group} is the max path group length. We detail each module below:

(1) BiPathNN Dataloader. Given the subgraph table S, the BiPathNN Dataloader divides each node pair's path instances into path groups and converts them into matrix format. Algorithm 2 shows the pseudo-code, which consists of three blocks. First (Block 1, Lines 1–7), it extracts unique node and edge features from the subgraph table to construct a feature dictionary. Then (Block 2, Lines 8–13), it divides path instances into path groups and constructs the feature and position indices for each path group element. Specifically,

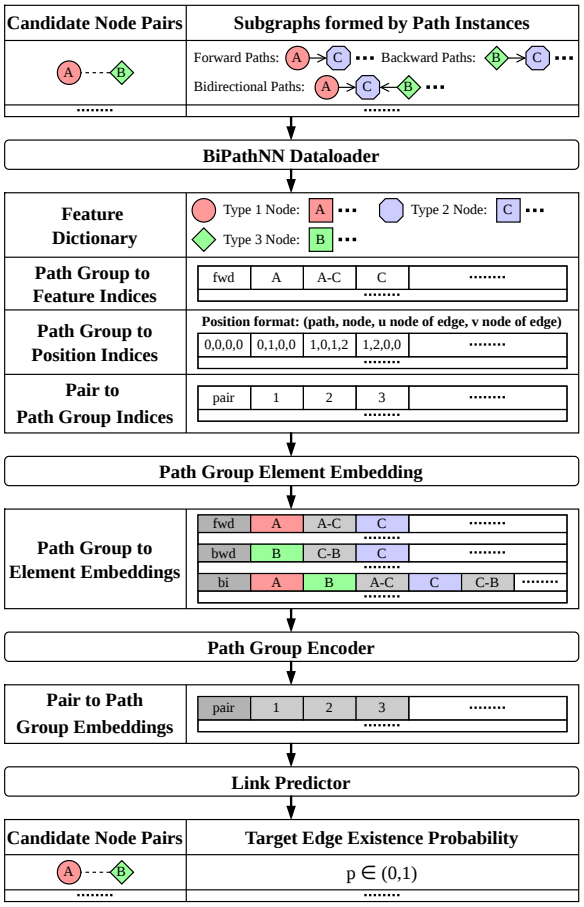

Figure 5: An illustration of BiPathNN.

each path group contains only forward, backward, or bidirectional meta-path instances, with at most m_{group} elements. Finally (Block 3, Lines 14–16), it constructs the mapping from node pairs to their path groups.

(2) Path Group Element Embedding. To represent each path group as a sequence of element embeddings, we construct each element embedding from a feature embedding and a position embedding. First, we construct feature embeddings for nodes and edges. For each type x in the feature dictionary $\mathcal{X}$, we use a dedicated MLP to project its features into a d_{model}-dimensional space and stack all projected feature embeddings together:

$$\mathrm{H}_{\text{proj}} = \text{Stack}(\text{MLP}_x(\mathcal{X}[x]) | x \in \mathcal{X}). \tag{12}$$

where $\mathrm{H}_{\text{proj}} \in \mathbb{R}^{n_{\text{feat}} \times d_{\text{model}}}$ contains the projected feature embeddings for all n_{feat} unique nodes and edges in $\mathcal{X}$.

Besides nodes and edges, path groups contain an element for padding and three elements denoting path groups containing forward, backward, and bidirectional meta-path instances, respectively. Since these elements have no raw features, we create learnable feature embeddings for them and stack the embeddings with H_{proj} to form the complete feature embedding matrix:

$$\mathrm{H}_{\mathrm{X}} = \text{Stack}(h_{\mathrm{X}}^{\text{pad}}, h_{\mathrm{X}}^{\text{fwd}}, h_{\mathrm{X}}^{\text{bwd}}, h_{\mathrm{X}}^{\text{bi}}, \mathrm{H}_{\text{proj}}) \tag{13}$$

where $h_{\mathrm{X}}^{\text{pad}}, h_{\mathrm{X}}^{\text{fwd}}, h_{\mathrm{X}}^{\text{bwd}}, h_{\mathrm{X}}^{\text{bi}} \in \mathbb{R}^{1 \times d_{\text{model}}}$ are learnable embeddings.

Algorithm 2 BiPathNN Dataloader.

Input: Subgraph table S for candidate node pairs, max path group length m_{group}, node types O and edge types L of the graph.
Output: Feature dictionary $\mathcal{X}$ mapping each node/edge type to feature vectors; path group to feature indices $\mathrm{I}_{\mathrm{P}\to\mathrm{X}}$ mapping path group elements to feature indices; path group to position indices $\mathrm{I}_{\mathrm{P}\to\text{pos}}$ mapping path group elements to position indices; pair to path group indices $\mathrm{I}_{\text{pair}\to\mathrm{P}}$ mapping node pairs to path group indices.

```
//===== Block 1: Feature Dictionary Construction =====//
X ← {}, I_X ← {}
for each o ∈ O do
    V_o ← UniqueNodesByType(S.V, o)
    X[o] ← V_o.f, I_X[o] ← V_o.v
for each l ∈ L do
    E_l ← UniqueEdgesByType(S.E, l)
    X[l] ← E_l.f, I_X[l] ← Stack(E_l.v_a, E_l.v_b)
//===== Block 2: Path Group Construction =====//
P_S ← ExpandPaths(S.P)
P_S, n_group ← AssignPathsToGroups(P_S, m_group)
I_P→X ← Zeros(n_group, m_group)
I_P→X ← FillFeatureIndices(I_P→X, P_S, S.v_a, S.v_b, I_X)
I_P→pos ← Zeros(n_group, m_group, 4)
I_P→pos ← FillPositionIndices(I_P→pos, P_S)
//===== Block 3: Pair to Path Group Mapping =====//
n_pair ← Count(S), m_pair ← MaxGroupsPerPair(P_S) + 1
I_pair→P ← Zeros(n_pair, m_pair)
I_pair→P ← FillPathGroupIndices(I_pair→P, P_S)
return X, I_P→X, I_P→pos, I_pair→P
```

Next, we use the path group to feature indices $\mathrm{I}_{\mathrm{P}\to\mathrm{X}}$ to retrieve the feature embeddings for each path group from H_{X}, and add position embeddings to construct the element embeddings for each path group:

$$\mathrm{H}_{\mathrm{P}}^{\mathrm{X}} = \mathrm{H}_{\mathrm{X}}[\mathrm{I}_{\mathrm{P}\to\mathrm{X}}] + \text{PositionEmbed}(\mathrm{I}_{\mathrm{P}\to\text{pos}}). \tag{14}$$

where $\mathrm{H}_{\mathrm{P}}^{\mathrm{X}} \in \mathbb{R}^{n_{\text{group}} \times m_{\text{group}} \times d_{\text{model}}}$ represents the element embedding sequences for all path groups, and PositionEmbed transforms the 4-dimensional position indices into position embeddings [60]. For elements corresponding to nodes or edges in path instances, the four dimensions represent the path index within the group, the node index within the path, and the edge's two connected node indices within the path.

(3) Path Group Encoder. Next, we encode each path group's element embedding sequence into a path group embedding via the Transformer encoder. First, we create a mask to exclude padding positions from encoding:

$$\mathrm{M}_{\mathrm{P}}^{\text{mask}} = \text{IsZero}(\mathrm{I}_{\mathrm{P}\to\mathrm{X}}) \tag{15}$$

where $\mathrm{M}_{\mathrm{P}}^{\text{mask}} \in \{0,1\}^{n_{\text{group}} \times m_{\text{group}}}$, and $\mathrm{M}_{\mathrm{P}}^{\text{mask}}[i,j] = 1$ if the j-th position in the i-th path group is a padding position, otherwise $\mathrm{M}_{\mathrm{P}}^{\text{mask}}[i,j] = 0$.

Then, we apply n_{layer} Transformer encoder layers with n_{head}-head self-attention to encode all element embeddings of each path group. For each layer $\ell \in \{1, \ldots, n_{\text{layer}}\}$, we compute:

$$\begin{aligned} \mathrm{H}_{\text{attn}}^{(\ell)} &= \text{SelfAttn}(\mathrm{H}_{\mathrm{P}}^{(\ell-1)}, \mathrm{M}_{\mathrm{P}}^{\text{mask}}) + \mathrm{H}_{\mathrm{P}}^{(\ell-1)} \\ \mathrm{H}_{\mathrm{P}}^{(\ell)} &= \text{FFN}(\mathrm{H}_{\text{attn}}^{(\ell)}) + \mathrm{H}_{\text{attn}}^{(\ell)} \end{aligned} \tag{16}$$

where $\mathrm{H}_{\mathrm{P}}^{(0)} = \mathrm{H}_{\mathrm{P}}^{\mathrm{X}}$, SelfAttn denotes a self-attention layer, and FFN denotes a feed-forward network.

Following common practice, we extract the embedding at position 0 as the path group embedding:

$$\mathrm{H}_{\mathrm{P}}^{\text{enc}} = \mathrm{H}_{\mathrm{P}}^{(n_{\text{layer}})}[:, 0, :]. \tag{17}$$

(4) Link Predictor. Next, we construct a sequence for each pair to encode its path group embeddings into a pair embedding via the Transformer encoder. Besides path group embeddings, each pair's sequence also includes an embedding for pair encoding and a padding embedding. We stack them with $\mathrm{H}_{\mathrm{P}}^{\mathrm{enc}}$:

$$\mathrm{H}_{\mathrm{P}} = \mathrm{Stack}(h_{\mathrm{P}}^{\mathrm{pad}}, h_{\mathrm{P}}^{\mathrm{pair}}, \mathrm{H}_{\mathrm{P}}^{\mathrm{enc}}) \tag{18}$$

where $h_{\mathrm{P}}^{\mathrm{pad}}$ and $h_{\mathrm{P}}^{\mathrm{pair}} \in \mathbb{R}^{1\times d_{\mathrm{model}}}$ are learnable embeddings.

Then, we use the pair to path group indices $\mathrm{I}_{\mathrm{pair}\to\mathrm{P}}$ to retrieve the path group embeddings for each pair from H_{P}:

$$\mathrm{H}_{\mathrm{pair}}^{\mathrm{P}} = \mathrm{H}_{\mathrm{P}}[\mathrm{I}_{\mathrm{pair}\to\mathrm{P}}]. \tag{19}$$

where $\mathrm{H}_{\mathrm{pair}}^{\mathrm{P}} \in \mathbb{R}^{n_{\mathrm{pair}}\times m_{\mathrm{pair}}\times d_{\mathrm{model}}}$ represents the path group embedding sequences for all node pairs.

Next, we similarly create a mask to exclude padding positions from encoding:

$$\mathrm{M}_{\mathrm{pair}}^{\mathrm{mask}} = \mathrm{IsZero}(\mathrm{I}_{\mathrm{pair}\to\mathrm{P}}) \tag{20}$$

Then, we apply another n_{layer} Transformer encoder layers with n_{head}-head self-attention to encode all path group embeddings of each node pair, following the same approach as Equation 16:

$$\begin{aligned} \mathrm{H}_{\mathrm{attn}}^{(\ell)} &= \mathrm{SelfAttn}(\mathrm{H}_{\mathrm{pair}}^{(\ell-1)}, \mathrm{M}_{\mathrm{pair}}^{\mathrm{mask}}) + \mathrm{H}_{\mathrm{pair}}^{(\ell-1)} \\ \mathrm{H}_{\mathrm{pair}}^{(\ell)} &= \mathrm{FFN}(\mathrm{H}_{\mathrm{attn}}^{(\ell)}) + \mathrm{H}_{\mathrm{attn}}^{(\ell)} \end{aligned} \tag{21}$$

where $\mathrm{H}_{\mathrm{pair}}^{(0)} = \mathrm{H}_{\mathrm{pair}}^{\mathrm{P}}$.

Finally, we extract the embedding at position 0 as the pair embedding and use an MLP to obtain the link prediction result:

$$\mathrm{Y}_{\mathrm{pair}} = \mathrm{MLP}_{\mathrm{pair}}(\mathrm{H}_{\mathrm{pair}}^{(n_{\mathrm{layer}})}[:, 0, :]). \tag{22}$$

Training Node Pair Selection. To train our BiPathNN, we label a subset of candidate node pairs using known target edges:

$$\mathrm{S}_{\mathrm{label}}, \mathrm{Y}_{\mathrm{label}} = \mathrm{LabeledPairs}(\mathrm{S}, \mathrm{E}_{\mathrm{kn}}^{\mathrm{tgt}}, n_{\mathrm{neg}}). \tag{23}$$

where $\mathrm{S}_{\mathrm{label}} \subseteq \mathrm{S}$, and $\mathrm{Y}_{\mathrm{label}} \in \{0,1\}^{n_{\mathrm{label}}}$ are the corresponding labels. $\mathrm{Y}_{\mathrm{label}}[i] = 1$ if the i-th node pair exists in $\mathrm{E}_{\mathrm{kn}}^{\mathrm{tgt}}$, otherwise 0. For each positive sample, we include n_{neg} negative samples.

BiPathNN Training. Then, we apply BiPathNN to obtain the predicted probabilities for labeled node pairs:

$$\mathrm{Y}_{\mathrm{train}} = \mathrm{BiPathNN}(\mathrm{S}_{\mathrm{label}}, m_{\mathrm{group}}, O, L). \tag{24}$$

Finally, we train BiPathNN by minimizing the binary cross-entropy (BCE) loss between predictions and ground-truth labels:

$$\mathcal{L} = \mathrm{BCE}(\mathrm{Y}_{\mathrm{train}}, \mathrm{Y}_{\mathrm{label}}). \tag{25}$$

4.3 Stage 3: Target Edge Discovery

Prediction Node Pair Selection. We first filter out node pairs that are already connected by known target edges:

$$\mathrm{S}_{\mathrm{pred}} = \mathrm{UnknownPairs}(\mathrm{S}, \mathrm{E}_{\mathrm{kn}}^{\mathrm{tgt}}), \tag{26}$$

where $\mathrm{S}_{\mathrm{pred}} \subseteq \mathrm{S}$ contains only node pairs not present in $\mathrm{E}_{\mathrm{kn}}^{\mathrm{tgt}}$.

Unknown Target Edge Prediction. We then apply the trained BiPathNN to predict unknown target edges:

$$\mathrm{Y}_{\mathrm{pred}} = \mathrm{BiPathNN}(\mathrm{S}_{\mathrm{pred}}, m_{\mathrm{group}}, O, L), \tag{27}$$

where $\mathrm{Y}_{\mathrm{pred}} \in \mathbb{R}^{n_{\mathrm{pred}}}$ represents the predicted probabilities for n_{pred} node pairs in $\mathrm{S}_{\mathrm{pred}}$.

Finally, we rank node pairs by their predicted probabilities and select top-k pairs as the discovered unknown target edges $\mathrm{E}_{\mathrm{unk}}^{\mathrm{tgt}}$.

Table 4: Heterogeneous Graphs for Link Discovery Tasks.

Task	Node/Edge Type	# Node/Edge	Feature Dim.
Companies' Key Personnel	Person Node	$> 10^8$	288
	Company Node	$> 10^8$	120
	Account Node	$> 10^8$	476
	Sub-Region Node	$> 10^6$	2,160
	Region Node	$> 10^4$	576
	Industry Node	$> 10^3$	0
	Person - Company	$> 10^8$	88
	Person - Account	$> 10^8$	240
	Person - Sub-Region	$> 10^9$	204
	Company - Account	$> 10^6$	150
	Company - Company	$> 10^7$	8
	Company - Sub-Region	$> 10^8$	0
	Company - Industry	$> 10^8$	0
	Sub-Region - Region	$> 10^6$	0
Authors' New References and Coauthors	Author Node	8.2×10^7	330
	Paper Node	2.1×10^8	53
	Institution Node	9.7×10^4	330
	Venue Node	2.1×10^5	360
	Topic Node	4.5×10^3	270
	Author - Paper	5.2×10^8	4
	Author - Institution	8.5×10^7	60
	Paper - Paper	1.8×10^9	0
	Paper - Institution	1.4×10^8	4
	Paper - Venue	1.6×10^8	12
	Paper - Topic	4.1×10^8	3

Table 5: Time periods of known target edges for each task.

	Companies' Key Personnel	Authors' New References	Authors' New Coauthors
Training Set	01/01/2024 - 08/31/2024	01/01/2021 - 12/31/2022	01/01/2021 - 12/31/2022
Validation Set	09/01/2024 - 10/31/2024	01/01/2023 - 06/30/2023	01/01/2023 - 06/30/2023
Test Set	11/01/2024 - 12/31/2024	07/01/2023 - 12/31/2023	07/01/2023 - 12/31/2023

5 EXPERIMENT

Our experiments aim to answer the following research questions:

- **RQ1:** Is our model more effective than other baselines?
- **RQ2:** How much more efficient is bidirectional meta-path sampling compared to traditional meta-path sampling?
- **RQ3:** What is the quality of discovered target edges?
- **RQ4:** How does each component of our model contribute to link prediction performance?
- **RQ5:** How do bidirectional meta-paths relate to target edges?
- **RQ6:** How do hyperparameters influence model performance?
- **RQ7:** Does our framework generalize to other applications?

5.1 Experimental Setup

Tasks and Datasets. We evaluate our approach on three link discovery tasks. As described in Section 2.3, each task requires a heterogeneous graph, known target edges, and candidate node sets.

- **Companies' Key Personnel**: This task aims to discover key personnel of potential customer companies. For known target edges, we use key personnel of existing customer companies. Candidate companies are those with potential demands, and candidate key personnel are those with available contact information.
- **Authors' New References**: This task aims to discover papers that authors will cite for the first time in future work. The graph

Table 6: Performance on link prediction.

Model	Companies' Key Personnel				Authors' New References				Authors' New Coauthors			
	AP	P@10k	P@20k	P@50k	AP	P@10k	P@20k	P@50k	AP	P@10k	P@20k	P@50k
R-GCNs	0.712	93.85%	89.13%	78.96%	0.545	81.78%	79.23%	65.27%	0.537	77.14%	73.94%	64.79%
HAN	0.639	83.09%	80.48%	72.24%	0.519	79.43%	74.53%	63.73%	0.528	76.05%	73.23%	63.98%
MAGNN	0.650	85.71%	80.94%	72.58%	0.541	81.48%	77.57%	65.53%	0.546	77.24%	74.19%	65.65%
HGT	0.718	91.07%	89.32%	79.16%	0.553	82.69%	78.94%	67.15%	0.565	78.72%	75.68%	67.86%
Simple-HGN	0.681	90.78%	85.25%	75.46%	0.578	83.08%	80.10%	69.12%	0.588	82.53%	79.31%	69.18%
SeHGNN	0.686	90.53%	84.76%	76.05%	0.547	82.14%	77.73%	66.74%	0.556	78.87%	75.21%	65.78%
Paths2Pair	0.734	94.73%	89.96%	79.89%	0.582	85.42%	81.45%	69.48%	0.595	83.80%	79.88%	70.08%
BiPathNN	**0.779**	**97.06%**	**91.95%**	**82.19%**	**0.621**	**87.48%**	**83.56%**	**72.82%**	**0.637**	**86.15%**	**82.36%**	**73.31%**

comes from OpenAlex [51], an open academic dataset with over 200 million publications. For known target edges, we use each author's first citation to each paper. Candidate authors are those with publications in the past five years, and candidate papers include all historical papers.

- **Authors' New Coauthors**: This task aims to discover new coauthors for authors in future work. It uses the same graph as the authors' new references task. For known target edges, we use each author's first collaboration with each coauthor. Candidate authors and coauthors are those with publications in the past five years.

Detailed statistics of the graphs used in these tasks are presented in Table 4. All edges can be traversed in both directions.

To prevent data leakage, each known target edge corresponds to the graph constructed with data available before its discovery time. Specifically, we use data before the discovery month for companies' key personnel and before the discovery year for authors' new references and coauthors. Based on chronological splitting of known target edges, we divide each task's dataset into training, validation, and test sets. Table 5 shows the specific time periods for each set. We randomly sample 300k, 100k, and 100k known target edges for the training, validation, and test sets respectively.

Execution Pipeline. First, we execute Stage 1 on a Spark cluster to select candidate node pairs and construct the subgraph table from our billion-scale heterogeneous graphs. Since these graphs have high-dimensional features, processing them requires TB-scale memory. To ensure each executor has sufficient memory for its allocated data while avoiding frequent inter-executor communication, we use 300 executors, each with 4 cores and 32 GB of memory. Beyond the graph data itself, bidirectional meta-path sampling produces large intermediate path tables, so each run samples paths for at most 200 million pairs. Then, we execute Stage 2 on a server with 32 CPUs, 180 GB of memory, and 4 A100 GPUs to train our model. Finally, we execute Stage 3 on the same server to predict target edge existence probabilities for all candidate node pairs.

Baselines. We compare our approach with seven representative HGNNs: R-GCNs [53], HAN [64], MAGNN [19], HGT [29], Simple-HGN [42], SeHGNN [71], and Paths2Pair [26].

These HGNNs fall into two categories: models based on k-hop sampling (R-GCNs, HGT, Simple-HGN) and models based on meta-path sampling (HAN, MAGNN, SeHGNN, Paths2Pair). For models based on k-hop sampling, we set $k = 2$ and limit each node to at most 20 sampled neighbors per edge type [23] due to memory constraints. For models based on meta-path sampling, we use the same path instances as our BiPathNN.

Evaluation Metrics. Following existing studies [19, 26], we evaluate the performance of all models using Average Precision (AP), Precision at Top 10k (P@10k), Precision at Top 20k (P@20k), and Precision at Top 50k (P@50k).

Parameter Configuration. For candidate pair selection and subgraph construction, we set different parameters for each task. For the companies' key personnel task: $k_{\text{bi}} = 6$, $n_{\text{cand}} = 10$, $n_{\text{sub}} = 100$. For the authors' new references task: $k_{\text{bi}} = 6$, $n_{\text{cand}} = 5$, $n_{\text{sub}} = 80$. For the authors' new coauthors task: $k_{\text{bi}} = 6$, $n_{\text{cand}} = 5$, $n_{\text{sub}} = 60$. For model evaluation, we set $n_{\text{neg}} = 10$ for the validation and test sets. Based on our computational resources, we set $m_{\text{group}} = 64$. The other hyperparameters of BiPathNN for each task are determined by the analysis in Section 5.7.

Skew Mitigation. Our bidirectional meta-path sampling (Algorithm 1) contains two joins (Lines 13 and 22) sensitive to data skew. This is because rows with the same join key value are shuffled to the same partition, so frequent values cause imbalanced data distribution across executors. Since we limit each pair to at most n_{sample} path instances per meta-path, we sample each join's right table to at most n_{sample} rows per join key value. This enables applying salted join to each frequent key value in three steps:

1. Rows of the key value in the left table are assigned a bucket id, with at most 10K rows per bucket id.
2. Rows of the key value in the right table are then replicated once per bucket id, with each copy containing its bucket id.
3. The two tables are then joined on the join key augmented with the bucket id, so rows of each frequent key value are now shuffled evenly across different partitions.

5.2 Overall Performance (RQ1)

To answer **RQ1**, we evaluate BiPathNN against seven baseline HGNNs across all three datasets. As shown in Table 6, BiPathNN consistently achieves the best performance across all datasets and metrics. Specifically, BiPathNN outperforms the strongest baseline based on k-hop sampling by 7.4% to 8.5% in AP across all datasets, which demonstrates that long-range bidirectional meta-paths between candidate node pairs are beneficial for link prediction. Moreover, BiPathNN outperforms the strongest baseline based on meta-path sampling by 6.1% to 7.1% in AP. Since BiPathNN uses the same path instances as these baselines but jointly encodes them to model the interactions among forward, backward, and bidirectional path instances, this improvement demonstrates the effectiveness of our model design. Furthermore, the decreasing precision from P@10k to P@50k shows that higher predicted probabilities correspond to

Table 7: Efficiency analysis.

	Meta-path Sampling	Bidirectional Meta-path Sampling	BiPathNN Prediction
Theoretical Complexity	$O(n_{\text{meta}} \cdot d^k)$	$O(n_{\text{meta}} \cdot d^{\frac{k}{2}} + n_{\text{join}})$	–
Companies' Key Personnel	23h 48m ± 2h	1h 46m ± 13m	8h 23m ± 18m
Authors' New References	46h 27m ± 4h	2h 51m ± 32m	16h 7m ± 41m
Authors' New Coauthors	32h 12m ± 3h	2h 25m ± 24m	11h 40m ± 27m

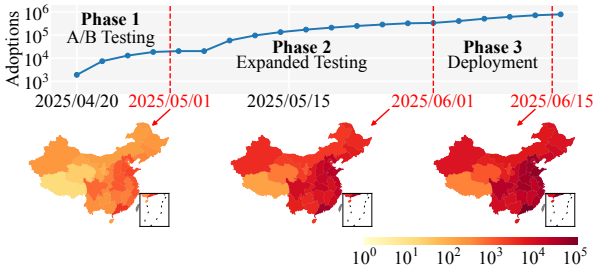

Figure 6: Three-phase real-world deployment results.

higher likelihood of true target edges. This validates our selection of top-k pairs as discovered target edges.

5.3 Efficiency Analysis (RQ2)

To answer **RQ2**, we conduct both theoretical and experimental analysis of bidirectional meta-path sampling versus traditional meta-path sampling. The results are presented in Table 7.

Theoretical analysis. Consider sampling path instances along n_{meta} k-hop meta-paths for a node pair in a graph with average degree d per edge type. Traditional meta-path sampling starts from one node, and the number of sampled paths grows by a factor of d at each hop, resulting in a complexity of $O(n_{\text{meta}} \cdot d^k)$. The exponential growth with k makes sampling long meta-paths impractical for large-scale graphs.

Instead of sampling k-hop meta-path instances from one node, bidirectional meta-path sampling only requires sampling $\frac{k}{2}$-hop meta-path instances from each node of the pair. The complexity of sampling these $\frac{k}{2}$-hop meta-paths is only $O(n_{\text{meta}} \cdot d^{\frac{k}{2}})$. The sampled $\frac{k}{2}$-hop meta-path instances from both nodes are then joined at the join node to obtain k-hop bidirectional meta-path instances. The complexity of the join operation is $O(n_{\text{join}})$, where n_{join} is the number of bidirectional meta-path instances between the node pair. Therefore, the total complexity of bidirectional meta-path sampling is $O(n_{\text{meta}} \cdot d^{\frac{k}{2}} + n_{\text{join}})$. In the worst case, n_{join} can reach $n_{\text{meta}} \cdot d^k$ when every pair of sampled path instances can be joined at the join node. However, most pairs of sampled path instances do not share the same join node in practice, so $n_{\text{join}} \ll n_{\text{meta}} \cdot d^{\frac{k}{2}}$. Thus, bidirectional meta-path sampling effectively reduces the complexity from $O(d^k)$ to $O(d^{\frac{k}{2}})$ in practice.

Table 8: Real-world A/B test results.

Method	Total Contacts	Successful Contacts	Success Rate
Previous	20,116	460	2.29%
BiLink	20,312	868	**4.27%**

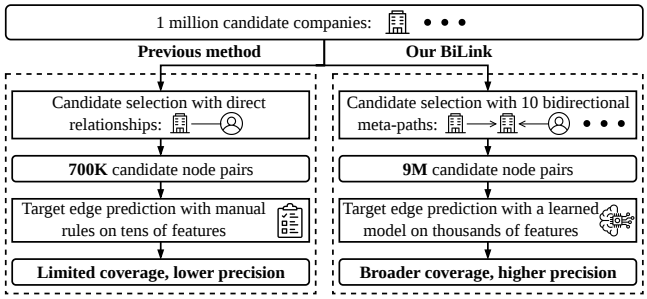

Figure 7: How BiLink improves deployment performance.

Empirical analysis. We measure the runtime of different sampling methods when sampling path instances for our model. Both methods are implemented on the same Spark cluster. For each task, we test on 5 groups of 200 million randomly selected node pairs. As shown in Table 7, bidirectional meta-path sampling achieves a 13 to 16 times speedup over traditional meta-path sampling. These results validate that bidirectional meta-path sampling can effectively reduce the complexity from $O(d^k)$ to $O(d^{\frac{k}{2}})$, making sampling long meta-paths practical for large-scale graphs.

Moreover, we measured the BiPathNN prediction time on the same test groups. As shown in Table 7, traditional meta-path sampling is significantly slower than BiPathNN prediction. Therefore, bidirectional meta-path sampling is essential to address the bottleneck of billion-scale link prediction.

5.4 Quality of Discovered Target Edges (RQ3)

To answer **RQ3**, we evaluate the quality of target edges discovered by BiLink on three tasks.

Companies' Key Personnel. We used the discovered key personnel for real-world customer expansion [14]. Specifically, the discovered key personnel and their predicted probabilities are precomputed and stored in a table. We then contact them in descending order of probability for product promotion. As shown in Figure 6, from April 2025 to June 2025, we continuously made calls and scaled up each week.

Our deployment consisted of three phases. In Phase 1, we compared BiLink with the previously deployed method designed by domain experts. For each method, we randomly selected 1 million candidate companies and applied the method to identify their most likely key personnel to contact. As shown in Table 8, BiLink achieved an 86% improvement in success rate. Figure 7 illustrates how BiLink achieves this improvement. For instance, key personnel who frequently use candidate companies' accounts are missed by the previous method. However, through the bidirectional meta-path "Company → Account ← Person", BiLink selects them as candidate node pairs and discovers the actual key personnel using thousands of features from the relevant paths.

Table 9: Ablation study.

Variant	Companies' Key Personnel				Authors' New References				Authors' New Coauthors			
	AP	P@10k	P@20k	P@50k	AP	P@10k	P@20k	P@50k	AP	P@10k	P@20k	P@50k
BiPathNN	0.779	97.06%	91.95%	82.19%	0.621	87.48%	83.56%	72.82%	0.637	86.15%	82.36%	73.31%
w/o forward meta-paths	0.742	95.06%	89.59%	79.94%	0.605	86.71%	82.46%	71.55%	0.590	83.98%	79.56%	69.41%
w/o backward meta-paths	0.749	95.89%	90.07%	80.15%	0.602	85.96%	81.86%	70.83%	0.596	83.04%	80.01%	69.89%
w/o bidirectional meta-paths	0.730	94.48%	89.19%	79.20%	0.576	85.09%	80.40%	68.38%	0.592	81.93%	78.28%	68.27%
Bi-LSTM encoder	0.758	96.09%	90.82%	80.78%	0.608	86.07%	82.13%	71.39%	0.622	85.37%	81.56%	71.81%

We then scaled up the deployment. In Phase 2, we expanded to 10 million candidate companies. The success rate remained consistently high. In Phase 3, we proceeded with full-scale deployment across all 200 million candidate companies. BiLink identified 1.8 billion candidate key personnel relationships and predicted their corresponding probabilities in approximately 4 days. By June 15, 2025, we had contacted 795,021 key personnel discovered by BiLink.

Authors' New References. We predicted papers that authors would cite for the first time in 2024–2025, using data before 2024. BiLink selected 1.6 billion candidate pairs and ranked them by predicted probability. Among the top 0.1%, 1%, and 10% ranked pairs, BiLink recalled 7.87%, 31.6%, and 75.8% of the 2.2M actual new references.

Authors' New Coauthors. We predicted new collaborations between authors in 2024–2025, using data before 2024. BiLink selected 1.4 billion candidate pairs and ranked them by predicted probability. Among the top 0.1%, 1%, and 10% ranked pairs, BiLink recalled 9.26%, 34.9%, and 79.9% of the 1.5M actual new collaborations.

Overall, these results across three tasks demonstrate BiLink's effectiveness in discovering target edges at billion scale.

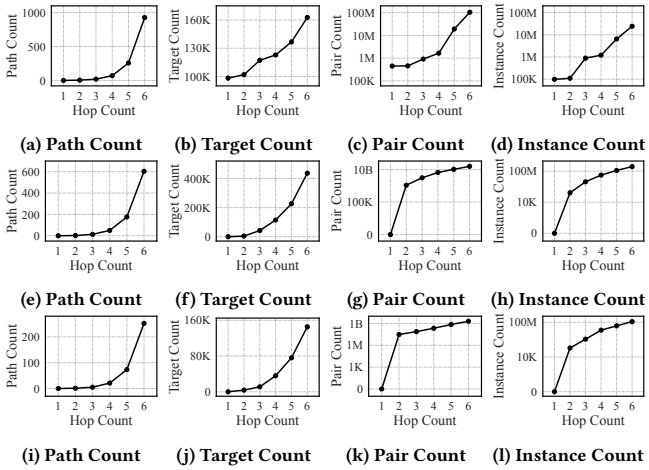

Figure 8: Analysis of relevant bidirectional meta-paths for three tasks: (a)–(d) companies' key personnel, (e)–(h) authors' new references, and (i)–(l) authors' new coauthors.

5.5 Ablation Study (RQ4)

To answer **RQ4**, we conduct two ablation studies on BiPathNN's inputs and encoder.

Input Meta-paths. We separately remove forward, backward, and bidirectional meta-paths from BiPathNN's input. As shown in Table 9, removing any type of meta-path consistently decreases performance across all tasks, validating the necessity of all three types for effective link prediction. Moreover, the relative importance of different meta-paths varies across tasks. For the companies' key personnel and authors' new references tasks, removing bidirectional meta-paths causes the largest performance drop, indicating that meta-paths directly connecting the node pair provide the most valuable information. For the authors' new coauthors task, removing each type leads to a similar performance drop, indicating that each type provides similarly valuable information.

Transformer Encoder. We replace BiPathNN's Transformer encoder (Equations 16 and 21) with a Bi-LSTM encoder using the same hyperparameters. As shown in Table 9, this replacement consistently decreases performance across all tasks, demonstrating that the Transformer encoder better captures the interactions among path elements and path groups.

5.6 Relevant Path Analysis (RQ5)

To answer **RQ5**, we focus on relevant bidirectional meta-paths, which connect the two node types of the target edge. Figure 8 shows how they relate to target edges as hop count increases. Specifically, it plots four metrics across the three tasks: the numbers of relevant bidirectional meta-paths, matched target edges, connected node pairs, and meta-path instances connecting target edges. The path count and target edge count in Figure 8 both grow rapidly with hop count, indicating that longer bidirectional meta-paths capture more types of relevant paths and cover more target edges. Meanwhile, the pair count grows much faster than the target edge count, indicating that the ratio of target edges among connected pairs decreases rapidly with hop count. Therefore, we only select candidate node pairs with the top-n_{cand} relevant bidirectional meta-paths to control the search space. The meta-path instance count also grows much faster, indicating that the number of meta-path instances connecting each target edge increases rapidly. Specifically, the average number of meta-path instances connecting each target edge exceeds 100 by hop count 6 across all three tasks, approaching the limit of our computational resources. Given that the computational cost of sampling longer meta-paths grows exponentially while these paths have lower match ratios to target edges, we set the maximum hop count k_{bi} of bidirectional meta-paths to 6 for all three tasks.

5.7 Hyperparameter Analysis (RQ6)

To answer **RQ6**, we analyze the impact of four key hyperparameters on BiPathNN performance across all tasks.

Hidden Dimension. Figures 9a, 9e and 9i show that performance improves consistently as d_{model} increases from 32 to 256 for all tasks, achieving the best results at $d_{\text{model}} = 256$.

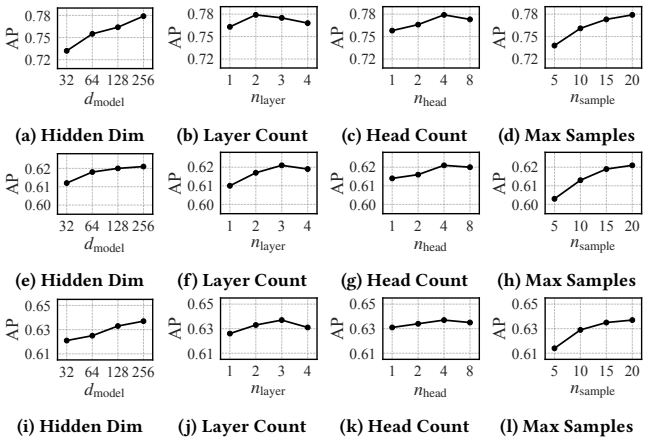

Figure 9: Hyperparameter analysis for three tasks: (a)–(d) companies' key personnel, (e)–(h) authors' new references, and (i)–(l) authors' new coauthors.

Layer Count. Figures 9b, 9f and 9j show that BiPathNN achieves optimal performance with $n_{\text{layer}} = 2$ for the companies' key personnel task and $n_{\text{layer}} = 3$ for the authors' new references and coauthors tasks.

Attention Heads. Figures 9c, 9g and 9k show that all tasks achieve optimal performance with $n_{\text{head}} = 4$.

Max Samples. Figures 9d, 9h and 9l show diminishing performance gains as n_{sample} increases. As our link prediction model's complexity grows quadratically with n_{sample}, we set $n_{\text{sample}} = 20$.

5.8 Generalization Analysis (RQ7)

To answer **RQ7**, we evaluate BiLink on two additional link discovery tasks to validate its generalization to other applications.

Users' New Items. The task aims to predict which items users will newly visit on an e-commerce platform. The corresponding graph is constructed from the Avito dataset [4] with 150M nodes across 5 types and 500M edges across 10 types. Using this graph, BiLink ranks 13.13% / 36.30% / 52.29% of 22K target edges in the top 1% / 5% / 10% of 57M candidate node pairs.

Conditions' New Interventions. The task aims to predict which interventions will be newly used to treat conditions in clinical trials. The corresponding graph is constructed from the AACT database [58] with 1M nodes across 5 types and 5M edges across 7 types. Using this graph, BiLink ranks 24.66% / 57.53% / 74.07% of 2K target edges in the top 1% / 5% / 10% of 2M candidate node pairs.

6 RELATED WORK

6.1 Link Discovery in Heterogeneous Graphs

Graphs are widely used to model the relational data generated in real-world applications, such as urban transportation [68, 69] and last-mile delivery [43–45]. Link discovery in heterogeneous graphs [80, 82, 84] aims to find hidden target edges in graphs with multiple node and edge types. This task has extensive applications in recommendation systems [1, 8, 13], social network analysis [10, 85], and knowledge graph completion [20, 81]. For large-scale graphs [30, 47, 52], existing heterogeneous graph neural networks (HGNNs) [15, 25, 31, 46] primarily rely on subgraph sampling to control computational costs.

Current sampling strategies fall into two main categories: k-hop sampling and meta-path sampling. K-hop sampling [29, 53] collects all neighbors within k hops of each node. HGNNs based on this strategy aggregate neighborhood information hop by hop. For instance, R-GCNs [53] employs relation-specific transformations to aggregate neighbors from different edge types. However, the number of sampled neighbors grows exponentially with the hop count [66], making k-hop sampling impractical for large hop counts.

To address this limitation, meta-path sampling was introduced, which restricts sampling to paths following predefined meta-paths. HGNNs based on meta-path sampling [64, 71] aggregate neighborhood information along each meta-path. For example, HAN [64] employs hierarchical attention mechanisms to learn semantic information from different meta-paths. However, the number of sampled paths still grows exponentially with path length, making long meta-path sampling time-consuming for large-scale graphs.

6.2 Bidirectional Search

Bidirectional search [28, 54] efficiently finds paths between two nodes by searching from both sides simultaneously. When applied to path sampling between node pairs, it reduces the complexity from $O(d^k)$ to $O(d^{\frac{k}{2}})$, where d is the average node degree and k is the path length. It has been applied to directed graph search [40, 50, 83], keyword search [33], path enumeration [27, 75, 76], and bipartite similarity computation [9, 38, 41, 70]. However, applying bidirectional search to meta-path sampling in billion-scale heterogeneous graphs has not been fully explored. Billion-scale graphs are typically processed in distributed systems [22, 36, 59, 86], which require operations to be parallelizable and stateless. Zhu et al. [88] proposed a bidirectional search method for discovering top-k important meta-paths, but their design does not consider these constraints. Therefore, to meet these requirements, we represent all meta-paths as rows in a unified table format and implement bidirectional meta-path sampling through uniform table operations.

7 CONCLUSION

In this paper, we proposed BiLink, a novel framework for link discovery in billion-scale heterogeneous graphs. Specifically, we first introduced *bidirectional meta-path*, a unified path schema for bidirectional paths connecting node pairs. Based on this, we implemented bidirectional meta-path sampling in distributed systems to efficiently select candidate node pairs and sample path instances between them. Then we proposed BiPathNN, a specialized model that jointly leverages these sampled path instances for effective link prediction. Experiments on three real-world tasks demonstrate that bidirectional meta-path sampling is over 13 times faster than traditional meta-path sampling, while BiPathNN improves average precision by over 6% compared to state-of-the-art baselines. In real-world deployment, BiLink discovered companies' key personnel from 1.8 billion candidate pairs, increasing contact success rate by 86%. Moreover, BiLink has been applied to discover authors' new references and coauthors, achieving over 31% recall using only the top 1% predicted pairs from billion-scale candidates.

REFERENCES

[1] Muhammad Adnan, Yassaman Ebrahimzadeh Maboud, Divya Mahajan, and Prashant J. Nair. 2021. Accelerating Recommendation System Training by Leveraging Popular Choices. *Proc. VLDB Endow.* 15, 1 (2021), 127–140. https://doi.org/10.14778/3485450.3485462

[2] Khaled Ammar and M. Tamer Ozsu. 2018. Experimental Analysis of Distributed Graph Systems. *Proc. VLDB Endow.* 11, 10 (2018), 1151–1164. https://doi.org/10.14778/3231751.3231764

[3] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Keith W. Hare, Jan Hidders, Victor E. Lee, Bei Li, Leonid Libkin, Wim Martens, Filip Murlak, Josh Perryman, Ognjen Savkovic, Michael Schmidt, Juan F. Sequeda, Slawek Staworko, and Dominik Tomaszuk. 2021. PG-Keys: Keys for Property Graphs. In *SIGMOD '21*. ACM, 2423–2436.

[4] Avito. 2015. Avito Context Ad Clicks. Kaggle competition. https://www.kaggle.com/c/avito-context-ad-clicks Accessed: 2026-05-01.

[5] Saurabh Bajaj, Hojae Son, Juelin Liu, Hui Guan, and Marco Serafini. 2024. Graph Neural Network Training Systems: A Performance Comparison of Full-Graph and Mini-Batch. *Proc. VLDB Endow.* 18, 4 (2024), 1196–1209.

[6] Caleb Belth, Alican Büyükçakir, and Danai Koutra. 2022. A hidden challenge of link prediction: which pairs to check? *Knowl. Inf. Syst.* 64, 3 (2022), 743–771.

[7] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *NIPS '13*. 2787–2795.

[8] Mengru Chen, Chao Huang, Lianghao Xia, Wei Wei, Yong Xu, and Ronghua Luo. 2023. Heterogeneous Graph Contrastive Learning for Recommendation. In *WSDM '23*. 544–552.

[9] Xiaoshuang Chen, Kai Wang, Xuemin Lin, Wenjie Zhang, Lu Qin, and Ying Zhang. 2021. Efficiently Answering Reachability and Path Queries on Temporal Bipartite Graphs. *Proc. VLDB Endow.* 14, 10 (2021), 1845–1858. https://doi.org/10.14778/3467861.3467873

[10] Audrey Cheng, Xiao Shi, Aaron Kabcenell, Shilpa Lawande, Hamza Qadeer, Jason Chan, Harrison Tin, Ryan Zhao, Peter Bailis, Mahesh Balakrishnan, Nathan Bronson, Natacha Crooks, and Ion Stoica. 2022. TAOBench: An End-to-End Benchmark for Social Network Workloads. *Proc. VLDB Endow.* 15, 9 (2022), 1965–1977.

[11] James Clarkson, Georgios Theodorakis, and Jim Webber. 2024. BIFROST: A Future Graph Database Runtime. In *ICDE '24*. 5605–5613. https://doi.org/10.1109/ICDE60146.2024.00448

[12] Pengjie Cui, Haotian Liu, Dong Jiang, Bo Tang, and Ye Yuan. 2025. Nezha: An Efficient Distributed Graph Processing System on Heterogeneous Hardware. *Proc. ACM Manag. Data* 3, 1, Article 57 (Feb. 2025), 27 pages. https://doi.org/10.1145/3709707

[13] Yixiang Fang, Kai Wang, Xuemin Lin, and Wenjie Zhang. 2021. Cohesive Subgraph Search over Big Heterogeneous Information Networks: Applications, Challenges, and Solutions. In *SIGMOD '21*. 2829–2838. https://doi.org/10.1145/3448016.3457538

[14] Xinyue Feng, Shuxin Zhong, Jinquan Hang, Wenjun Lyu, Yuequn Zhang, Guang Yang, Haotian Wang, Desheng Zhang, and Guang Wang. 2025. Hierarchical Structure Sharing Empowers Multi-task Heterogeneous GNNs for Customer Expansion. In *KDD '25*. 4424–4434. https://doi.org/10.1145/3711896.3737234

[15] Xinyue Feng, Shuxin Zhong, Jinquan Hang, Yuequn Zhang, Guang Yang, Haotian Wang, Desheng Zhang, and Guang Wang. 2025. NeighSqueeze: Compact Neighborhood Grouping for Efficient Billion-Scale Heterogeneous Graph Learning. In *CIKM '25*. 5658–5665. https://doi.org/10.1145/3746252.3761569

[16] Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. 2024. Position: Relational Deep Learning - Graph Representation Learning on Relational Databases. In *ICML '24*.

[17] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR (RLGM Workshop)*.

[18] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An Evolving Query Language for Property Graphs. In *SIGMOD '18*. 1433–1445. https://doi.org/10.1145/3183713.3190657

[19] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. MAGNN: Metapath Aggregated Graph Neural Network for Heterogeneous Graph Embedding. In *WWW '20*. 2331–2341.

[20] Junyang Gao, Xian Li, Yifan Ethan Xu, Bunyamin Sisman, Xin Luna Dong, and Jun Yang. 2019. Efficient Knowledge Graph Accuracy Evaluation. *Proc. VLDB Endow.* 12, 11 (2019), 1679–1691.

[21] Sen Gao, Jianwen Zhao, Hao Zhang, Shixuan Sun, Chen Liang, Gongye Chen, Wenliang Zhang, Bo Ren, Chao Liu, Chenyi Zhang, Quan Chen, Chao Li, Jingwen Leng, and Minyi Guo. 2025. GES: High-Performance Graph Processing Engine and Service in Huawei. In *SIGMOD Companion '25*. 391–403. https://doi.org/10.1145/3722212.3724439

[22] Milad Rezaei Hajidehi, Sraavan Sridhar, and Margo Seltzer. 2024. CUTTANA: Scalable Graph Partitioning for Faster Distributed Graph Databases and Analytics. *Proc. VLDB Endow.* 18, 1 (2024), 14–27.

[23] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS '17*.

[24] Jinquan Hang, Zheng Dong, Hongke Zhao, Xin Song, Peng Wang, and Hengshu Zhu. 2022. Outside In: Market-aware Heterogeneous Graph Neural Network for Employee Turnover Prediction. In *WSDM '22*. 353–362.

[25] Jinquan Hang, Zhiqing Hong, Xinyue Feng, Guang Wang, Dongjiang Cao, Jiayang Qiao, Haotian Wang, and Desheng Zhang. 2024. Complex-Path: Effective and Efficient Node Ranking with Paths in Billion-Scale Heterogeneous Graphs. *Proc. VLDB Endow.* 17, 12 (2024).

[26] Jinquan Hang, Zhiqing Hong, Xinyue Feng, Guang Wang, Guang Yang, Feng Li, Xining Song, and Desheng Zhang. 2024. Paths2Pair: Meta-path Based Link Prediction in Billion-Scale Commercial Heterogeneous Graphs. In *KDD '24*.

[27] Kongzhang Hao, Long Yuan, and Wenjie Zhang. 2021. Distributed Hop-Constrained s-t Simple Path Enumeration at Billion Scale. *Proc. VLDB Endow.* 15, 2 (2021), 169–182. https://doi.org/10.14778/3489496.3489499

[28] Robert Holte, Ariel Felner, Guni Sharon, and Nathan Sturtevant. 2016. Bidirectional search that is guaranteed to meet in the middle. In *AAAI '16*.

[29] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous Graph Transformer. In *WWW '20*. 2704–2710.

[30] Chengying Huan, Shuaiwen Leon Song, Santosh Pandey, Hang Liu, Yongchao Liu, Baptiste Lepers, Changhua He, Kang Chen, Jinlei Jiang, and Yongwei Wu. 2023. TEA: A General-Purpose Temporal Graph Random Walk Engine. In *EuroSys '23*. 182–198. https://doi.org/10.1145/3552326.3567491

[31] Houye Ji, Xiao Wang, Chuan Shi, Bai Wang, and Philip S. Yu. 2023. Heterogeneous Graph Propagation Network. *IEEE Trans. Knowl. Data Eng.* 35, 1 (2023), 521–532.

[32] Unmesh Joshi and Jacopo Urbani. 2020. Searching for Embeddings in a Haystack: Link Prediction on Knowledge Graphs with Subgraph Pruning. In *WWW '20*. 2817–2823.

[33] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. 2005. Bidirectional Expansion For Keyword Search on Graph Databases. In *VLDB '05*. 505–516.

[34] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR '17*.

[35] Anran Li, Yuanyuan Chen, Jian Zhang, Mingfei Cheng, Yihao Huang, Yueming Wu, Anh Tuan Luu, and Han Yu. 2024. Historical Embedding-Guided Efficient Large-Scale Federated Graph Learning. *Proc. ACM Manag. Data* 2, 3, Article 144 (May 2024), 24 pages. https://doi.org/10.1145/3654947

[36] Changji Li, Hongzhi Chen, Shuai Zhang, Yingqian Hu, Chao Chen, Zhenjie Zhang, Meng Li, Xiangchen Li, Dongqing Han, Xiaohui Chen, Xudong Wang, Huiming Zhu, Xuwei Fu, Tingwei Wu, Hongfei Tan, Hengtian Ding, Mengjin Liu, Kangcheng Wang, Ting Ye, Lei Li, Xin Li, Yu Wang, Chenguang Zheng, Hao Yang, and James Cheng. 2022. ByteGraph: A High-Performance Distributed Graph Database in ByteDance. *Proc. VLDB Endow.* 15, 12 (2022), 3306–3318. https://doi.org/10.14778/3554821.3554824

[37] Meihao Liao, Junjie Zhou, Rong-Hua Li, Qiangqiang Dai, Hongyang Chen, and Guoren Wang. 2024. Efficient and Provable Effective Resistance Computation on Large Graphs: An Index-based Approach. *Proc. ACM Manag. Data* 2, 3, Article 133 (May 2024), 27 pages. https://doi.org/10.1145/3654936

[38] Haoyu Liu and Siqiang Luo. 2024. BIRD: Efficient Approximation of Bidirectional Hidden Personalized PageRank. *Proc. VLDB Endow.* 17, 9 (2024), 2255–2268.

[39] Hao Liu, Qianwen Yang, Taoyong Cui, and Wei Wang. 2024. MSGNN: Masked Schema based Graph Neural Networks. *Proc. VLDB Endow.* 18, 3 (2024), 571–584.

[40] Shangqi Lu, Ru Wang, and Yufei Tao. 2025. Interactive Graph Search Made Simple. *Proc. ACM Manag. Data* 3, 3, Article 177 (June 2025), 25 pages. https://doi.org/10.1145/3725409

[41] Wensheng Luo, Qiaoyuan Yang, Yixiang Fang, and Xu Zhou. 2023. Efficient Core Maintenance in Large Bipartite Graphs. *Proc. ACM Manag. Data* 1, 3, Article 208 (Nov. 2023), 26 pages. https://doi.org/10.1145/3617329

[42] Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. 2021. Are We Really Making Much Progress? Revisiting, Benchmarking and Refining Heterogeneous Graph Neural Networks. In *KDD '21*. 1150–1160.

[43] Wenjun Lyu, Xiaolong Jin, Haotian Wang, Yiwei Song, Shuai Wang, Yunhuai Liu, Tian He, and Desheng Zhang. 2025. Towards Workload-Constrained Efficient Order Assignment in Last-Mile Delivery. *IEEE Trans. Mob. Comput.* 24, 2 (2025), 557–570. https://doi.org/10.1109/TMC.2024.3469236

[44] Wenjun Lyu, Haotian Wang, Zhiqing Hong, Guang Wang, Yu Yang, Yunhuai Liu, and Desheng Zhang. 2023. REDE: Exploring Relay Transportation for Efficient Last-mile Delivery. In *ICDE '23*. IEEE, 3003–3016.

[45] Wenjun Lyu, Shuxin Zhong, Guang Yang, Haotian Wang, Yi Ding, Shuai Wang, Yunhuai Liu, Tian He, and Desheng Zhang. 2025. InCo: Exploring Inter-Trip Cooperation for Efficient Last-mile Delivery. In *WWW '25*. 5183–5191.

[46] Li Ma, Yongchao Liu, Xiaofeng Gao, Peng Zhang, and Chuntao Hong. 2025. Building Robust and Trustworthy HGNN Models: A Learnable Threshold Approach for Node Classification. *ACM Trans. Knowl. Discov. Data* 19, 2, Article 50 (2025), 17 pages. https://doi.org/10.1145/3707645

[47] Lingkai Meng, Yu Shao, Long Yuan, Longbin Lai, Peng Cheng, Xue Li, Wenyuan Yu, Wenjie Zhang, Xuemin Lin, and Jingren Zhou. 2025. Revisiting Graph

Analytics Benchmark. *Proc. ACM Manag. Data* 3, 3, Article 208 (June 2025), 28 pages. https://doi.org/10.1145/3725345

[48] Jason Mohoney, Roger Waleffe, Henry Xu, Theodoros Rekatsinas, and Shivaram Venkataraman. 2021. Marius: Learning Massive Graph Embeddings on a Single Machine. In *OSDI '21*. 533–549.

[49] Trung-Kien Nguyen, Zemin Liu, and Yuan Fang. 2023. Link Prediction on Latent Heterogeneous Graphs. In *WWW '23*. 263–273.

[50] Ira Pohl. 1971. Bi-directional Search. In *Machine Intelligence 6*, Bernard Meltzer and Donald Michie (Eds.). Edinburgh University Press, Edinburgh, UK, 127–140.

[51] Jason Priem, Heather Piwowar, and Richard Orr. 2022. OpenAlex: A fully-open index of scholarly works, authors, venues, institutions, and concepts. *arXiv* (2022). https://doi.org/10.48550/arXiv.2205.01833 Accessed: 2025-08-05.

[52] Sherif Sakr, Angela Bonifati, Hannes Voigt, Alexandru Iosup, Khaled Ammar, Renzo Angles, Walid Aref, Marcelo Arenas, Maciej Besta, Peter A. Boncz, Khuzaima Daudjee, Emanuele Della Valle, Stefania Dumbrava, Olaf Hartig, Bernhard Haslhofer, Tim Hegeman, Jan Hidders, Katja Hose, Adriana Iamnitchi, Vasiliki Kalavri, Hugo Kapp, Wim Martens, M. Tamer Özsu, Eric Peukert, Stefan Plantikow, Mohamed Ragab, Matei R. Ripeanu, Semih Salihoglu, Christian Schulz, Petra Selmer, Juan F. Sequeda, Joshua Shinavier, Gábor Szárnyas, Riccardo Tommasini, Antonino Tumeo, Alexandru Uta, Ana Lucia Varbanescu, Hsiang-Yun Wu, Nikolay Yakovets, Da Yan, and Eiko Yoneki. 2021. The Future is Big Graphs: A Community View on Graph Processing Systems. *Commun. ACM* 64, 9 (2021), 62–71. https://doi.org/10.1145/3434642

[53] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *ESWC '18*. Springer, 593–607.

[54] Nathan R. Sturtevant and Ariel Felner. 2018. A Brief History and Recent Achievements in Bidirectional Search. In *AAAI '18*.

[55] Yizhou Sun and Jiawei Han. 2013. Mining Heterogeneous Information Networks: A Structural Analysis Approach. *SIGKDD Explor. Newsl.* 14, 2 (2013), 20–28.

[56] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. *Proc. VLDB Endow.* 4 (2011), 992–1003.

[57] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2022. Heterogeneous Information Networks: the Past, the Present, and the Future. *Proc. VLDB Endow.* 15, 12 (2022), 3807–3811.

[58] Asba Tasneem, Laura Aberle, Hari Ananth, Swati Chakraborty, Karen Chiswell, Brian J. McCourt, and Ricardo Pietrobon. 2012. The Database for Aggregate Analysis of ClinicalTrials.gov (AACT) and Subsequent Regrouping by Clinical Specialty. *PLOS ONE* 7, 3 (2012), e33677. https://doi.org/10.1371/journal.pone.0033677

[59] Bing Tong, Yan Zhou, Chen Zhang, Jianheng Tang, Jing Tang, Leihong Yang, Qiye Li, Manwu Lin, Zhongxin Bao, Jia Li, and Lei Chen. 2024. GalaxyBase: A High Performance Native Distributed Graph Database for HTAP. *Proc. VLDB Endow.* 17, 12 (2024), 3893–3905.

[60] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *NIPS '17*. 6000–6010.

[61] Roger Waleffe, Jason Mohoney, Theodoros Rekatsinas, and Shivaram Venkataraman. 2023. MariusGNN: Resource-Efficient Out-of-Core Training of Graph Neural Networks. In *EuroSys '23*. 144–161. https://doi.org/10.1145/3552326.3567501

[62] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv* (2019).

[63] Ping Wang, Khushbu Agarwal, Colby Ham, Sutanay Choudhury, and Chandan K. Reddy. 2021. Self-Supervised Learning of Contextual Embeddings for Link Prediction in Heterogeneous Networks. In *WWW '21*. 2946–2957.

[64] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous Graph Attention Network. In *WWW '19*. 2022–2032.

[65] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI '14*. 1112–1119.

[66] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Netw. Learn. Syst.* 32, 1 (2021), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386

[67] C. Yang, Y. Xiao, Y. Zhang, Y. Sun, and J. Han. 2022. Heterogeneous Network Representation Learning: A Unified Framework With Survey and Benchmark. *IEEE Trans. Knowl. Data Eng.* 34 (2022), 4854–4873.

[68] Guang Yang, Meiqi Tu, Zelong Li, Jinquan Hang, Taichi Liu, Ruofeng Liu, Yi Ding, Yu Yang, and Desheng Zhang. 2024. Behavior-Aware Hypergraph Convolutional Network for Illegal Parking Prediction with Multi-Source Contextual Information. In *CIKM '24*. 2827–2836. https://doi.org/10.1145/3627673.3679563

[69] Guang Yang, Yuequn Zhang, Jinquan Hang, Xinyue Feng, Zejun Xie, Desheng Zhang, and Yu Yang. 2023. CARPG: Cross-City Knowledge Transfer for Traffic Accident Prediction via Attentive Region-Level Parameter Generation. In *CIKM '23*. 2939–2948. https://doi.org/10.1145/3583780.3614802

[70] Renchi Yang and Jieming Shi. 2024. Efficient High-Quality Clustering for Large Bipartite Graphs. *Proc. ACM Manag. Data* 2, 1, Article 23 (March 2024), 27 pages. https://doi.org/10.1145/3639278

[71] Xiaocheng Yang, Mingyu Yan, Shirui Pan, Xiaochun Ye, and Dongrui Fan. 2023. Simple and Efficient Heterogeneous Graph Neural Network. In *AAAI '23*.

[72] Yuyang Ye, Zheng Dong, Hengshu Zhu, Tong Xu, Xin Song, Runlong Yu, and Hui Xiong. 2023. MANE: Organizational Network Embedding With Multiplex Attentive Neural Networks. *IEEE Trans. Knowl. Data Eng.* 35, 4 (2023), 4047–4061.

[73] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *NSDI '12*. 15–28.

[74] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *HotCloud '10*.

[75] Yuanyuan Zeng, Yixiang Fang, Wensheng Luo, and Chenhao Ma. 2025. Accelerating Skyline Path Enumeration with a Core Attribute Index on Multi-attribute Graphs. *Proc. ACM Manag. Data* 3, 3, Article 124 (June 2025), 26 pages. https://doi.org/10.1145/3725261

[76] Yuanyuan Zeng, Yixiang Fang, Chenhao Ma, Xu Zhou, and Kenli Li. 2024. Efficient Distributed Hop-Constrained Path Enumeration on Large-Scale Graphs. *Proc. ACM Manag. Data* 2, 1, Article 22 (March 2024), 25 pages. https://doi.org/10.1145/3639277

[77] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V. Chawla. 2019. Heterogeneous Graph Neural Network. In *KDD '19*. 793–803.

[78] Dalong Zhang, Xin Huang, Ziqi Liu, Jun Zhou, Zhiyang Hu, Xianzheng Song, Zhibang Ge, Lin Wang, Zhiqiang Zhang, and Yuan Qi. 2020. AGL: A Scalable System for Industrial-Purpose Graph Machine Learning. *Proc. VLDB Endow.* 13, 12 (2020), 3125–3137.

[79] Han Zhang, Yu Hao, Xin Cao, Yixiang Fang, Won-Yong Shin, and Wei Wang. 2021. Relation Prediction via Graph Neural Network in Heterogeneous Information Networks with Missing Type Information. In *CIKM '21*. 2517–2526.

[80] Jiawei Zhang, Jianhui Chen, Shi Zhi, Yi Chang, Philip S. Yu, and Jiawei Han. 2017. Link Prediction across Aligned Networks with Sparse and Low Rank Matrix Estimation. In *ICDE '17*. 971–982.

[81] Jiasheng Zhang, Rex Ying, and Jie Shao. 2024. Online Detection of Anomalies in Temporal Knowledge Graphs with Interpretability. *Proc. ACM Manag. Data* 2, 6, Article 247 (Dec. 2024), 26 pages. https://doi.org/10.1145/3698823

[82] Muhan Zhang and Yixin Chen. 2018. Link Prediction Based on Graph Neural Networks. In *NeurIPS '18*. 5171–5181.

[83] Yalong Zhang, Rong-Hua Li, Longlong Lin, Qi Zhang, Lu Qin, and Guoren Wang. 2025. Integral Densest Subgraph Search on Directed Graphs. *Proc. ACM Manag. Data* 3, 3, Article 176 (June 2025), 26 pages. https://doi.org/10.1145/3725313

[84] Yufeng Zhang, Weiqing Wang, Hongzhi Yin, Pengpeng Zhao, Wei Chen, and Lei Zhao. 2023. Disconnected Emerging Knowledge Graph Oriented Inductive Link Prediction. In *ICDE '23*. 381–393.

[85] Yiming Zhang, Lingfei Wu, Qi Shen, Yitong Pang, Zhihua Wei, Fangli Xu, Ethan Chang, and Bo Long. 2023. Graph Learning Augmented Heterogeneous Graph Neural Network for Social Recommendation. *ACM Trans. Recomm. Syst.* 1, 4, Article 16 (oct 2023), 22 pages.

[86] Wenyue Zhao, Yang Cao, Peter Buneman, Jia Li, and Nikos Ntarmos. 2024. Automating Vectorized Distributed Graph Computation. *Proc. ACM Manag. Data* 2, 6, Article 254 (2024), 27 pages. https://doi.org/10.1145/3698833

[87] Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. 2019. AliGraph: a comprehensive graph neural network platform. *Proc. VLDB Endow.* 12, 12 (2019), 2094–2105.

[88] Zichen Zhu, Tsz Nam Chan, Reynold Cheng, Loc Do, Zhipeng Huang, and Haoci Zhang. 2022. Effective and efficient discovery of top-k meta paths in heterogeneous information networks. *IEEE Trans. Knowl. Data Eng.* 34, 9 (2022), 4172–4185. https://doi.org/10.1109/TKDE.2020.3037218