

# HyperJoin: LLM-augmented Hypergraph Link Prediction for Joinable Table Discovery

Shiyuan Liu

University of Technology Sydney  
shiyuan.liu-1@student.uts.edu.au

Lu Qin

University of Technology Sydney  
lu.qin@uts.edu.au

Jianwei Wang

University of New South Wales  
jianwei.wang1@unsw.edu.au

Wenjie Zhang

University of New South Wales  
wenjie.zhang@unsw.edu.au

Xuemin Lin

ACEM, Shanghai Jiao Tong University  
xuemin.lin@sjtu.edu.cn

Ying Zhang

University of Technology Sydney  
ying.zhang@uts.edu.au

## ABSTRACT

As a pivotal task in data lake management, joinable table discovery has attracted widespread interest. While existing language model-based methods achieve remarkable performance by combining offline column representation learning with online ranking, their design insufficiently accounts for the underlying structural interactions: (1) offline, they directly model tables into isolated or pairwise columns, thereby struggling to capture the rich inter-table and intra-table structural information; and (2) online, they rank candidate columns based solely on query-candidate similarity, ignoring the mutual interactions among the candidates, leading to incoherent result sets. To address these limitations, we propose HyperJoin, a large language model (LLM)-augmented **Hypergraph** framework for **Joinable** table discovery. Specifically, we first construct a hypergraph to model tables using both the intra-table hyperedges and the LLM-augmented inter-table hyperedges. Consequently, the task of joinable table discovery is formulated as link prediction on this constructed hypergraph. We then design HIN, a Hierarchical Interaction Network that learns expressive column representations through bidirectional message passing over columns and hyperedges. To strengthen coherence and internal consistency in the result columns, we cast online ranking as a coherence-aware subgraph selection problem defined on a weighted graph spanning the query and target columns. We then introduce a reranking module that leverages a maximum spanning tree algorithm to prune noisy connections and maximize coherence. Experiments over 4 public benchmarks demonstrate the superiority of HyperJoin, achieving an average xxx% improvement over state-of-the-art methods.

## PVLDB Reference Format:

Shiyuan Liu, Jianwei Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. HyperJoin: LLM-augmented Hypergraph Link Prediction for Joinable Table Discovery. PVLDB, 17(9): 2227 - 2240, 2024.  
doi:10.14778/3665844.3665853

## PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/XXX>.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 17, No. 9 ISSN 2150-8097.  
doi:10.14778/3665844.3665853

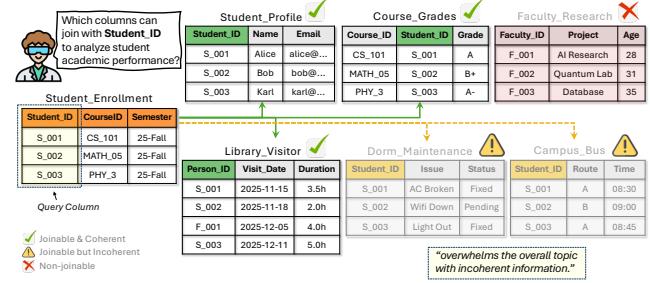


Figure 1: An example of joinable table discovery in data lake.

## 1 INTRODUCTION

The proliferation of open data portals and enterprise data lakes has led to massive collections of tabular data, creating tremendous opportunities for data analysis and machine learning. However, these tables are often fragmented, making it difficult to leverage complementary information from other tables to enhance analysis. Prior studies [13] highlight the prevalence of this concern, reporting that approximately 89% of real-world business intelligence projects involve multiple tables that require joins. In this work, we investigate joinable table discovery. Given a table repository  $\mathcal{T}$  and a query column  $C_q$ , joinable table discovery aims to find columns  $C$  from  $\mathcal{T}$  that exhibit a large number of semantically matched cells with  $C_q$ . Thus, the table containing column  $C$  can be joined with the query table to enrich features and enhance downstream data analysis or machine learning tasks.

**EXAMPLE 1.** As illustrated in Figure 1, a data scientist analyzes student academic performance using the *Student\_Enrollment* table and searches for columns joinable with the *Student\_ID* ( $C_q$ ). The data lake contains several semantically joinable columns (green checkmarks). However, columns from tables like *Dorm\_Maintenance* and *Campus\_Bus*, while joinable, originate from unrelated service domains and may overwhelm the analysis with incoherent information (yellow warnings). An effective join discovery method should therefore identify joinable columns while ensuring that the returned results remain coherent to better support the analysis.

Recognizing the importance and practical value of joinable table search, a set of methods has been developed (refer to the survey and benchmark paper [5] for an in-depth discussion). Early non-learning methods generally cast joinable table discovery as a set-similarity search problem. Representative approaches, such

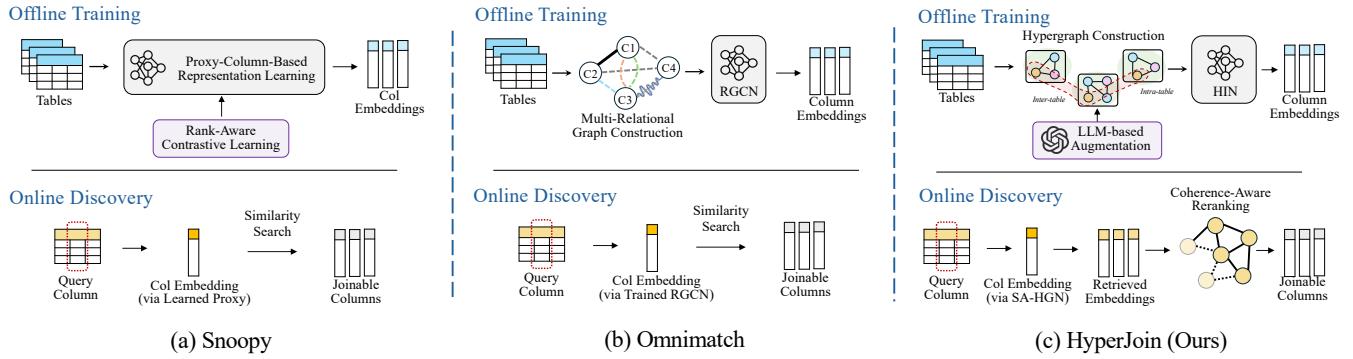


Figure 2: Framework comparisons of joinable table discovery methods.

as JOSIE [24] and LSH Ensemble [25], model each column as a set of distinct values and identify joinable columns based on their overlap, supported by inverted indexes or MinHash-based LSH for efficiency. However, these methods limit the join type to equi-join based on exact string matching, overlooking numerous semantically matched cells (e.g., “NY” semantically matches “New York”), thus underestimating column joinabilities.

Recently, pre-trained language models (PLMs)-based methods are introduced to exploit the semantic capabilities of PLMs (e.g., BERT [6] and DistilBERT [15]) for more effective joinable table discovery. Representative methods include DeepJoin [8], Snoopy [11] and Omnimatch [12]. As illustrated in Figure 2(a) and Figure 2(b), these methods typically integrate an offline training phase with an online discovery phase to optimize both efficiency and effectiveness. During the offline phase, a model is trained to learn column representations, encoding the content of each column into a dense vector. In the online phase, given a query column, the system identifies joinable columns based on the similarity between the query and candidate columns.

**Motivations.** Although these PLMs-based methods achieve promising performance, their designs insufficiently account for the underlying structural interactions, which are critical for accurate joinable table discovery. Specifically:

Firstly, during the offline training phase, these methods typically partition tables into isolated or pairwise columns. This approach struggles to capture rich intra-table structural information (such as high-order dependencies among columns and the hierarchical relationships between tables and columns) as well as inter-table structural information (such as global join topologies). As a result, the learned representation lacks sufficient expressiveness to capture these intricate dependencies, leading to suboptimal accuracy. Specifically, DeepJoin [8] and Snoopy [11] process columns individually to learn their representations. Although Omnimatch [12] attempts to utilize a graph to learn to model the column relationships. However, it still primarily focuses on pairwise relationships.

Secondly, in the online discovery phase, existing methods typically perform simple Top-K ranking based solely on the query-candidate similarity scores. This approach neglects the mutual correlations and latent dependencies among the candidate columns. Consequently, this yields incoherent result sets. Integrating such columns may introduce notable noise, which overwhelms the overall insights with irrelevant information. For example, as shown in

Figure 1, when the query is `Student_ID`, a Top-K retrieval may include columns from `Campus_Bus` and `Dorm_Maintenance`. Although these columns are joinable, they should be ranked lower since they originate from unrelated domains and fail to support the intended analysis of student academic performance.

**Challenges.** To design an accurate joinable table discovery method, two challenges exist below:

*Challenge I: How to capture the complex structural relationships for effective offline representation learning within a unified framework?* Table repositories exhibit complex structural relationships comprising both inter-table and intra-table relationships. Moreover, these relationships are not isolated; rather, they are strongly interdependent. For example, inter-table column joinability may only be meaningful when contextualized with fine-grained intra-table semantics, making isolated modeling approaches inadequate. Therefore, a significant challenge lies in devising a unified framework that jointly employs these complex structural relationships for joinable table search.

*Challenge II: How to effectively and efficiently search globally relevant and coherent joinable tables from large-scale data lakes?* Data lakes typically contain hundreds to thousands of heterogeneous tables. When accounting for mutual context, the number of potential join paths grows exponentially. Furthermore, as we proved in Theorem 9, identifying the optimal  $K$ -column subset that maximizes joint coherence is NP-hard, causing exhaustive search to be intractable for large-scale data lakes.

**Our approaches.** Driven by the aforementioned challenges, we propose HyperJoin, a new **Hypergraph**-based framework that recasts the problem of **Joinable** table search as link prediction on the hypergraph to better leverage the structural context of tables. As illustrated in Figure 2(c), similar to previous PLM-based methods, HyperJoin also employs a two-phase architecture: an offline representation learning phase and an online discovery phase.

To address Challenge I, in the offline phase, we propose to use a hypergraph to model the structure prior of tables and learn column representations. Formally, the hypergraph is formulated with columns serving as nodes and comprising two types of hyperedges: (1) *Large language model (LLM)-augmented inter-table hyperedges* and (2) *Intra-table hyperedges*. Specifically, the LLM-augmented hyperedges connect all columns that belong to the same connected component of joinable columns, explicitly modeling inter-table information of joinability. We first employ LLMs to augment data

by generating semantically equivalent column name variants, and then treat each original column together with its LLM-generated variants as a unified join-key entity, using this expanded set to construct inter-table hyperedges that propagate joinability signals across tables and their LLM-generated alternatives. In addition, the intra-table hyperedges connect all columns within the same table, capturing high-order relationships among columns and column-table relationships. Furthermore, we design HIN, a Hierarchical Interaction Network that performs hierarchical message passing aligned with our two types of hyperedges: it first aggregates column information within intra-table and LLM-augmented inter-table hyperedges, and then enables all-to-all interaction across hyperedges through a global mixing step, allowing each column to better leverage both the intra-table schema relationships and inter-table joinability signals.

To solve Challenge II, in the online phase, we introduce a global coherent reranking module to improve both efficiency and accuracy. Specifically, we first construct a weighted complete graph spanning the query and target columns to capture the structural context of columns after the first-stage top-B retrieval ( $K < B$ ). We then formulate the task of global coherent joinable table selection as a coherence-aware subgraph selection that balances individual query relevance with inter-candidate coherence. We prove that this problem is NP-hard (via reduction from the Densest k-Subgraph problem). We then propose an efficient greedy approximation algorithm that leverages a maximum spanning tree algorithm to prune noisy connections and maximize the global matching score.

**Contributions.** The main contributions are as follows:

- We propose HyperJoin, a novel hypergraph-based framework for joinable table discovery that exploits structure prior via dual-type hypergraphs. It contains the offline representation learning phase and the online discovery phase.
- In the offline phase, we build a hypergraph with LLM-augmented inter-table hyperedges and intra-table hyperedges, and design the HIN for effective message passing and learn expressive column representation.
- In the online phase, we propose a global coherent reranking module that leverages an efficient greedy approximation algorithm with the maximum spanning tree to tractably discover the coherent result columns.
- Experiments across 4 public benchmarks demonstrate the superiority of HyperJoin, achieving average improvements of 15.3% in Precision@15 and 18.7% in Recall@15 over previous state-of-the-art methods.

## 2 PRELIMINARIES

### 2.1 Problem Definition

We follow the typical setting of joinable table discovery [5, 7–9, 11]. A data lake contains a table repository  $\mathcal{T} = \{T_1, T_2, \dots, T_M\}$  with  $M$  tables. Each table  $T_i$  consists of  $n_i$  rows (tuples) and  $m_i$  columns (attributes), where each cell value is denoted as  $c_{ij}$ . We focus on textual columns and extract all of them from  $\mathcal{T}$  into a column repository  $C = \{C_1, C_2, \dots, C_N\}$  containing  $N$  columns across all tables. Each column  $C$  is represented as a set of cell values  $\{c_1, c_2, \dots, c_{|C|}\}$  and may also include associated metadata such as column names, where  $|C|$  denotes the cardinality (number of

**Table 1: Symbols and Descriptions**

Notation	Description
$\mathcal{T}$	table repository
$T$	a table, consisting of multiple columns
$C$	column repository (all columns in $\mathcal{T}$ )
$C$	a column, represented as a set of cell values
$C_q$	query column
$ C $	cardinality (number of cells) of column $C$
$\mathbf{h}_C \in \mathbb{R}^d$	embedding of column $C$
$J(C_q, C_t)$	joinability score between $C_q$ and $C_t$
$K$	number of results to return
$\mathcal{R}, \mathcal{R}^*$	result set / optimal result set of size $K$
$\mathcal{H}(V, E)$	hypergraph with nodes $V$ and hyperedges $E$
$E_{\text{inter}}$	inter-table hyperedges
$E_{\text{intra}}$	intra-table hyperedges
$f^\theta(\cdot)$	neural network model with parameters $\theta$
$\lambda$	coherence weight in reranking objective

cells) of column  $C$ . We use  $C_q$  to denote the query column.  $\mathcal{R}^*$  and  $\mathcal{R}$  are utilized to denote the ground-truth optimal result set and a predicted result set, respectively. Next, we give the formal definition of joinable table discovery.

**DEFINITION 1 (PAIRWISE COLUMN JOINABILITY).** *Given a query column  $C_q$  and a candidate column  $C_t$  from the repository  $C$ , their pairwise joinability score  $J(C_q, C_t)$  is defined as the average best-match similarity for each value in the query column:*

$$J(C_q, C_t) = \frac{1}{|C_q|} \sum_{v_q \in C_q} \max_{v_t \in C_t} \text{sim}(v_q, v_t) \quad (1)$$

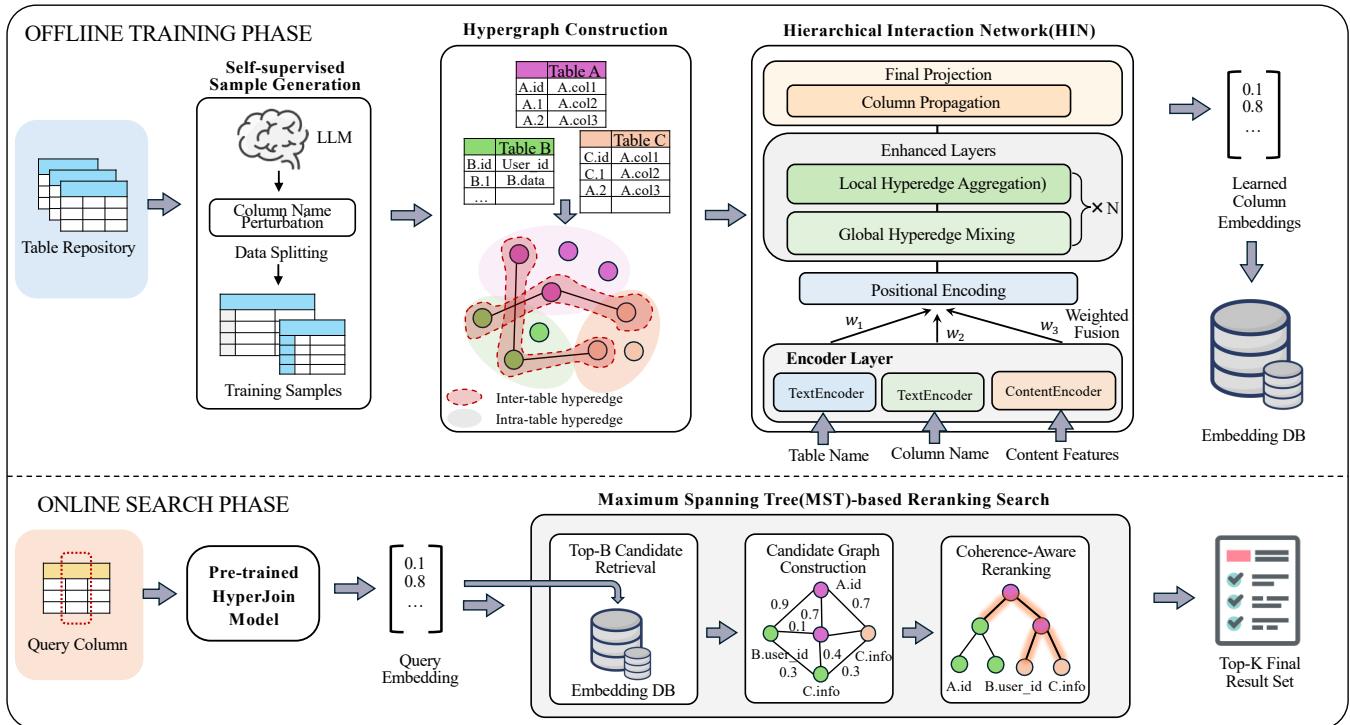
This directional score measures how well  $C_q$  is covered by  $C_t$ .  $\text{sim}(v_q, v_t) \in [0, 1]$  is the semantic value correspondence that quantifies the semantic relatedness between two cell values  $v_i$  and  $v_j$ .

**DEFINITION 2 (JOINABLE TABLE DISCOVERY).** *Given a query column  $C_q$ , a table repository  $\mathcal{T}$ , and an integer  $K$ , the task of joinable table discovery is to select a result set  $\mathcal{R}^* \subseteq C$  of size  $K$  that each  $C_t \in \mathcal{R}$  has a high joinability with  $C_q$ .*

### 2.2 Language model and state of the art

Language models serve as the cornerstone of modern Natural Language Processing (NLP), fundamentally aiming to model the probability distribution of text sequences. A prevailing architecture in this domain is the auto-regressive (AR) model, which generates sequences by iteratively predicting the next token conditioned on the preceding context. Building upon these foundations, the field has evolved from PLMs to LLMs.

PLMs, such as the auto-encoding BERT [6] and its variants DistilBERT [15] and MPNet [17], revolutionized NLP by learning contextualized representations from massive text corpora. These models map textual sequences into dense vectors that capture semantic similarity, typically requiring task-specific fine-tuning for downstream applications. In contrast, LLMs (e.g., the GPT series [14]) represent a significant leap in scale, often containing tens or hundreds of billions of parameters. While LLMs share the pre-training paradigm



**Figure 3: Framework overview of HyperJoin.**

with PLMs, their massive scale unlocks “emergent abilities”, such as zero-shot reasoning and in-context learning, enabling them to solve complex tasks without explicit parameter updates.

**State of the art.** Current state-of-the-art solutions for joinable table discovery are mostly PLM-based, which follow a common offline/online paradigm. In the offline stage, a model is trained (or fine-tuned), and embeddings are precomputed for all columns in the repository. An approximate nearest neighbor index (e.g., FAISS) is often built for efficient retrieval. In the online stage, the query column is encoded using the same model, and its top- $k$  most similar columns are retrieved from the index.

Despite this shared framework, these methods vary in how they obtain column embeddings. DeepJoin [8] fine-tunes the DistilBERT to embed columns into fixed-dimensional vectors. It transforms column contents and metadata into a textual sequence using a set of contextualization options. Snoopy [11] introduces proxy columns to overcome limitations of direct PLM encoding. It learns a set of proxy column embeddings through rank-aware contrastive learning. Offline, Snoopy trains the proxy model and precomputes column embeddings. Omnimatch [12] constructs a similarity graph where nodes represent columns and edges encode multiple signals. It uses a Relational Graph Convolutional Network (RGCN) to learn column embeddings that aggregate neighborhood information.

### 3 OVERVIEW

The overall architecture of HyperJoin is illustrated in Figure 3. Instead of treating tables as isolated or pairwise columns, we model the entire data lake as a hypergraph, where nodes represent columns,

and hyperedges explicitly encode both intra-table schema constraints and inter-table correlations. Consequently, we recast the joinable table discovery task as a link prediction problem on this hypergraph. To effectively solve this link prediction task, we design the HIN. This network learns expressive column embeddings by performing hierarchical message passing. It consists of three key components: (1) a positional encoding module that captures table identity and global structural roles; (2) a Local Hyperedge Aggregation module that propagates information within the specific intra- and inter-table contexts; and (3) a Global Hyperedge Mixing module that enables all-to-all interaction across hyperedges. In the online search phase, we introduce a global coherent reranking module to select the joinable column set. We formulate the task of top- $k$  coherent column selection as a subgraph selection problem on the candidate graph and solve it using an MST-based algorithm.

## 4 OFFLINE REPRESENTATION LEARNING

In this section, we present the offline representation learning phase of HyperJoin. The section is organized into two main components. First, Section 4.1 describes how we construct a unified hypergraph from the data lake, including initial column feature encoding, LLM-augmented inter-table hyperedge and intra-table hyperedge construction. Second, Section 4.2 presents our Hierarchical Interaction Network that operates on the constructed hypergraph, detailing the network architecture and training objectives.

### 4.1 Hypergraph Construction

To more comprehensively capture the structure context of the table lake, we model the data lake as a hypergraph where a single

hyperedge can connect multiple columns simultaneously, enabling unified modeling of these complex structural relationships.

**Hypergraph Definition.** Formally, we have a hypergraph  $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathbf{X}^v, \mathbf{X}^e, \Pi)$ , where  $\mathcal{V} = \{v_1, \dots, v_N\}$  denotes the set of  $N$  nodes, each corresponding to a column in the data lake, and  $\mathcal{E} = \{e_1, \dots, e_M\}$  denotes the set of  $M$  hyperedges. Each node  $v_i \in \mathcal{V}$  is associated with an initial feature vector  $\mathbf{x}_{v_i}^v \in \mathbb{R}^d$ , and  $\mathbf{X}^v \in \mathbb{R}^{N \times d}$  stacks all node features row-wise. Each hyperedge  $e_j \subseteq \mathcal{V}$  represents a column context (e.g., intra-table or inter-table) and is optionally associated with an initial hyperedge feature vector  $\mathbf{x}_{e_j}^e \in \mathbb{R}^{d_e}$ , where  $\mathbf{X}^e \in \mathbb{R}^{M \times d_e}$  stacks all hyperedge features row-wise. Hyperedges are allowed to overlap, meaning a column may participate in multiple contextual groups. The hypergraph structure is represented by a node–hyperedge incidence matrix  $\Pi \in \{0, 1\}^{N \times M}$ , where  $\Pi_{ij} = 1$  if  $v_i \in e_j$  and 0 otherwise.

**Hypergraph Construction.** The construction of  $\mathcal{H}$  proceeds in three steps: (1) we encode initial node features  $\mathbf{X}^v$  by fusing semantic signals from schema names and data content; (2) we construct intra-table hyperedges  $\mathcal{E}_{\text{intra}}$  based on physical table schemas; and (3) we construct LLM-augmented inter-table hyperedges  $\mathcal{E}_{\text{inter}}$  by identifying join-connected column groups, augmented with LLM-generated semantic variants.

*Step 1: Initial Column Feature Encoding.* Columns in a data lake contain rich semantic information from multiple sources: the table name provides high-level context, the column name describes specific semantics, and the cell values offer direct evidence. A robust initial representation should fuse all these sources to provide a strong foundation for the subsequent structure-aware learning.

Each column node  $v_i$  is initialized with a feature vector  $\mathbf{x}_{v_i}^v$  by fusing three components: table name, column name, and cell values. We encode the table name and column name using two independent text encoders, producing  $\mathbf{h}_i^{\text{table}} \in \mathbb{R}^{d_0}$  and  $\mathbf{h}_i^{\text{col}} \in \mathbb{R}^{d_0}$ . For cell values, we obtain a content representation by aggregating pre-trained value embeddings with simple statistical pooling and passing the result through a two-layer MLP, yielding  $\mathbf{h}_i^{\text{content}} \in \mathbb{R}^{d_0}$ . Let  $\mathcal{S} = \{\text{table}, \text{col}, \text{content}\}$  denote the components and  $\mathbf{w} \in \mathbb{R}^3$  be learnable fusion weights; we compute

$$\mathbf{h}_i^{(0)} = \sum_{s \in \mathcal{S}} \text{softmax}(\mathbf{w})_s \cdot \mathbf{h}_i^s, \quad (2)$$

followed by a linear projection to obtain  $\mathbf{x}_{v_i}^v = \mathbf{W}_0 \mathbf{h}_i^{(0)} \in \mathbb{R}^d$ . Stacking  $\mathbf{x}_{v_i}^v$  yields the node feature matrix  $\mathbf{X}^v \in \mathbb{R}^{N \times d}$  used in the hypergraph  $\mathcal{H}$ .

*Step 2: Intra-Table Hyperedges.* For each table  $T$ , we construct an intra-table hyperedge  $e_T^{\text{intra}} = \{v_i \mid v_i \text{ corresponds to a column in } T\}$ . This is motivated by two points: (i) tables naturally group semantically related columns; (ii) columns co-occurring in the same table often share context beyond join relationships. Thus, these hyperedges capture schema-level co-occurrence signals.

*Step 3: Inter-Table Hyperedges.* To construct  $\mathcal{E}_{\text{inter}}$ , we build an auxiliary join graph  $\mathcal{G}_{\text{join}} = (\mathcal{V}, \mathcal{P}_{\text{join}})$ , where  $(v_i, v_j) \in \mathcal{P}_{\text{join}}$  indicates that the corresponding column pair is joinable. We apply Union-Find to obtain connected components  $\{C_k\}$  of  $\mathcal{G}_{\text{join}}$ ; each component with  $|C_k| \geq 2$  induces an inter-table hyperedge  $e_k^{\text{inter}} = C_k$ , and we set  $\mathcal{E}_{\text{inter}} = \{e_k^{\text{inter}}\}$ . To enrich the joinability signals without ground-truth join information, we generate the join graph with

LLM-generated semantic variants as follows: (i) *key column selection*. We identify candidate join-key columns using standard keyness heuristics (e.g., completeness/uniqueness signals); (ii) *LLM augmentation*. For each selected join-key column  $C_{\text{key}}$ , we use an LLM to generate a semantically-preserving name variant  $C'_{\text{key}}$  conditioned on table context (e.g., table/column names and sample rows). The prompt template is shown in Box 4.1; (iii) *inter-table hyperedge generation*. We add edges  $(C_{\text{key}}, C'_{\text{key}})$  to  $\mathcal{G}_{\text{join}}$ , then recompute connected components to form the final  $\mathcal{E}_{\text{inter}}$ .

### Prompt Design

**Input context.** Column name, table name, all column names in the table, and 3–5 sampled rows.

**Transformation guidance.** Generate a semantically equivalent variant that follows common database naming conventions:

- Separator changes (e.g., `CustomerID` → `Customer_ID`, `customer-id`)
- Abbreviations (e.g., `CustomerID` → `CustID`)
- Case conventions (e.g., `CustomerID` → `customerId`, `CUSTOMERID`)
- Synonyms (e.g., `CustomerID` → `ClientID`)

**Output.** Return a single most plausible variant in a structured format (e.g., JSON: `{"perturbed": "Cust_ID"}`).

**Theoretical Analysis.** We provide a formal justification for why incorporating both intra-table and inter-table contexts can improve joinable table discovery, and show that prior SOTA methods are special cases of our formulation. All the proofs are in our online full version [].

*Setup.* Let  $Y_{ij} \in \{0, 1\}$  denote whether two columns  $(v_i, v_j)$  are joinable. Let  $\mathbf{x}_i$  be the initial feature of column  $v_i$ . For each column  $v_i$ , define two context variables induced by the constructed hypergraph: (i) its intra-table context  $\mathcal{N}^{\text{intra}}(v_i)$  (the set of columns co-located with  $v_i$  in the same table), and (ii) its inter-table context  $\mathcal{N}^{\text{inter}}(v_i)$  (the set of columns connected with  $v_i$  through joinable connectivity, e.g., within the same join-connected component). We write the available information for deciding joinability as a feature set:

$$\begin{aligned} \mathcal{F}_{\text{single}}(i, j) &= \{\mathbf{x}_i, \mathbf{x}_j\}, \\ \mathcal{F}_{\text{multi}}(i, j) &= \left\{ \mathbf{x}_i, \mathbf{x}_j, \mathcal{N}^{\text{intra}}(v_i), \mathcal{N}^{\text{intra}}(v_j), \right. \\ &\quad \left. \mathcal{N}^{\text{inter}}(v_i), \mathcal{N}^{\text{inter}}(v_j) \right\}. \end{aligned} \quad (3)$$

Let  $\Psi^*(\mathcal{F})$  denote the minimum achievable expected discovery risk under feature set  $\mathcal{F}$ :

$$\Psi^*(\mathcal{F}) = \inf_g \mathbb{E}[\ell(g(\mathcal{F}), Y_{ij})],$$

where  $g$  is any measurable scoring function and  $\ell(\cdot)$  is a bounded loss (e.g., logistic or 0-1 loss).

**THEOREM 1.** *For the two feature sets  $\mathcal{F}_{\text{multi}}$  and  $\mathcal{F}_{\text{single}}$  defined above, the Bayes-optimal risk satisfies*

$$\Psi^*(\mathcal{F}_{\text{multi}}) \leq \Psi^*(\mathcal{F}_{\text{single}}).$$

Moreover, if the added contexts are informative in the sense that  $\mathbb{P}(Y_{ij} = 1 \mid \mathcal{F}_{\text{multi}}) \neq \mathbb{P}(Y_{ij} = 1 \mid \mathcal{F}_{\text{single}})$  with non-zero probability, then the inequality is strict.

**THEOREM 2 (PRIOR METHODS ARE SPECIAL CASES).** Consider our framework that learns column embeddings via a model  $f_\theta$  operating on the constructed hypergraph. DeepJoin and Snoopy can be viewed as special cases obtained by restricting the utilized context to  $\mathcal{F}_{\text{single}}$  (i.e., column-wise encoding without using  $\mathcal{N}^{\text{intra}}$  or  $\mathcal{N}^{\text{inter}}$ ).

## 4.2 HIN Model Design

We present our HIN, which operates on the constructed hypergraph to learn expressive column representations. Our architecture consists of three components: (1) *Positional Encoding*, (2) *Local Hyperedge Aggregation*, and (3) *Global Hyperedge Mixing*.

**Positional Encoding.** Standard message-passing graph neural networks (GNNs) suffer from position agnosticism [], where nodes with identical local neighborhoods may receive identical representations even if they play different global roles in the joinability graph. This limitation is formally characterized by the *1-dimensional Weisfeiler–Lehman (1-WL) test* [20]. We inject two types of positional encodings to mitigate this issue: (i) a *table-level positional encoding* and (ii) a *column-level Laplacian positional encoding*.

*Table-level positional encoding.* We maintain a learnable lookup matrix  $\mathbf{E}_{\text{tbl}} \in \mathbb{R}^{|\mathcal{T}| \times d}$ , where each row stores a trainable  $d$ -dimensional vector for one table ID. For a column  $v$  belonging to table  $t(v)$ , we inject table identity by

$$\mathbf{h}_v^{\text{tbl-pe}} = \mathbf{x}_v^v + \alpha \cdot \mathbf{E}_{\text{tbl}}[t(v)], \quad (4)$$

where  $\alpha$  is a learnable scalar weight (initialized as 0.1) and optimized jointly with the model.

*Column-level positional encoding.* We adopt Laplacian positional encoding to capture global structural roles of columns in the joinability topology. We construct a pairwise column graph  $\mathcal{G}_{\text{col}} = (\mathcal{V}, \mathcal{P}_{\text{join}})$ , where each edge  $(v_i, v_j) \in \mathcal{P}_{\text{join}}$  indicates that the corresponding column pair is joinable. We build the symmetric adjacency matrix  $\mathbf{A} \in \{0, 1\}^{N \times N}$  by setting  $A_{ij} = A_{ji} = 1$  iff  $(v_i, v_j) \in \mathcal{P}_{\text{join}}$ , and 0 otherwise. We compute the normalized Laplacian

$$\mathbf{L}_{\text{norm}} = \mathbf{I}_N - \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}, \quad (5)$$

where  $\mathbf{I}_N$  is the  $N \times N$  identity matrix and  $\mathbf{D}$  is the diagonal degree matrix. We extract the smallest  $k$  eigenvectors  $\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$  and project them through a two-layer MLP:

$$\mathbf{PE}_{\text{col}}(v_i) = \mathbf{W}_2 \cdot \sigma(\mathbf{W}_1 \cdot [\mathbf{v}_1[i], \dots, \mathbf{v}_k[i]]^\top). \quad (6)$$

where  $\mathbf{W}_1 \in \mathbb{R}^{d/2 \times k}$  and  $\mathbf{W}_2 \in \mathbb{R}^{d \times d/2}$  are learnable projection matrices and  $\sigma(\cdot)$  is a non-linear activation (e.g., ReLU). This encoding captures global spectral structure of the joinability graph. For efficiency, we precompute the eigenvectors offline and cache  $\mathbf{PE}_{\text{col}}$  for lookup during training.

*Unified positional encoding.* The final positional encoding combines both levels:

$$\mathbf{h}_v = \mathbf{x}_v^v + \alpha \cdot \mathbf{E}_{\text{tbl}}[t(v)] + \beta \cdot \mathbf{PE}_{\text{col}}(v), \quad (7)$$

where  $\mathbf{x}_v^v$  is the raw initial column embedding,  $\mathbf{E}_{\text{tbl}}[t(v)]$  is the table-level embedding for the table containing column/node  $v$ ,  $\mathbf{PE}_{\text{col}}(v)$  is the Laplacian-based column positional encoding and  $\beta$  is a learnable

scalar weight initialized identically to  $\alpha$ . Together, they allow the model to automatically adjust the strength of each positional signal.

**Local Hyperedge Aggregation.** Traditional GNNs suffer from over-squashing [], where information from distant nodes gets exponentially compressed through multiple layers. We adopt a hierarchical hyperedge-based approach to aggregate both the local and global information: by first aggregating columns into  $M$  hyperedges ( $M \ll N$ ) and performing message passing at the hyperedge level, we shorten path lengths and alleviate information bottlenecks.

*Stage 1: Node-level feature transformation.* Given PE-enhanced embeddings  $\{\mathbf{h}_v^{(0)}\}_{v \in \mathcal{V}}$ , we apply  $L = 2$  layers of node transformation:

$$\mathbf{h}_v^{(\ell)} = \text{LayerNorm}(\sigma(\mathbf{W}^{(\ell)} \mathbf{h}_v^{(\ell-1)} + \mathbf{b}^{(\ell)})), \quad (8)$$

where  $\sigma(\cdot)$  is ReLU activation and  $\mathbf{h}_v^{(0)} = \mathbf{h}_v$ . Unlike traditional GNN layers that aggregate neighbor information at this stage, we defer aggregation to Stage 2 at the hyperedge level.

*Stage 2: Aggregation to hyperedge level.* We aggregate node embeddings to the hyperedge level via mean pooling:

$$\mathbf{x}_{e_j}^e = \frac{1}{|e_j|} \sum_{v \in e_j} \mathbf{h}_v^{(L)}, \quad (9)$$

where  $e_j \subseteq \mathcal{V}$  denotes the node set of the  $j$ -th hyperedge. Let  $\mathbf{H} \in \mathbb{R}^{N \times d}$  stack node embeddings row-wise, i.e.,  $\mathbf{H}[v, :] = \mathbf{h}_v^{(L)}$ , and let  $\mathbf{X}^e \in \mathbb{R}^{M \times d}$  stack hyperedge embeddings row-wise, i.e.,  $\mathbf{X}^e[j, :] = \mathbf{x}_{e_j}^e$ . This is efficiently computed via sparse matrix multiplication:

$$\mathbf{X}^e = \mathbf{D}_e^{-1} \boldsymbol{\Pi}^\top \mathbf{H}, \quad (10)$$

where  $\boldsymbol{\Pi} \in \{0, 1\}^{N \times M}$  is the node–hyperedge incidence matrix and  $\mathbf{D}_e \in \mathbb{R}^{M \times M}$  is diagonal with  $(\mathbf{D}_e)_{jj} = \sum_{i=1}^N \boldsymbol{\Pi}_{ij} = |e_j|$ . Each node belongs to at most two hyperedges (one intra-table and at most one inter-table), making  $\boldsymbol{\Pi}$  very sparse with time complexity  $O(\|\boldsymbol{\Pi}\|_0 \cdot d)$  where  $\|\boldsymbol{\Pi}\|_0 \leq 2N$ .

We denote the aggregated hyperedge embeddings as  $\mathbf{Z}^{(0)} = \mathbf{X}^e$  and partition them by hyperedge type. Inter-table and intra-table hyperedges capture fundamentally different relationships: cross-table join semantics versus intra-table schema semantics. We apply separate linear transformations to learn domain-specific patterns (lowercase = single embedding, uppercase = stacked matrix):

$$\tilde{\mathbf{Z}}_{\text{inter}} = \mathbf{Z}_{\text{inter}} \mathbf{W}_{\text{inter}}, \quad (11)$$

$$\tilde{\mathbf{Z}}_{\text{intra}} = \mathbf{Z}_{\text{intra}} \mathbf{W}_{\text{intra}}, \quad (12)$$

and concatenate them:  $\tilde{\mathbf{Z}} = [\tilde{\mathbf{Z}}_{\text{inter}}; \tilde{\mathbf{Z}}_{\text{intra}}] \in \mathbb{R}^{M \times d}$ .

**Global Hyperedge Mixing.** Although local message passing on the hyperedge graph captures short-range dependencies, we still need global information exchange across hyperedges to model long-range join relationships. However, standard hypergraph message passing is inherently local (typically 1-hop per layer) and thus requires multiple layers to achieve global coverage, which increases computation and can exacerbate over-squashing. We design a global mixing mechanism that achieves all-to-all communication in a single layer, inspired by MLP-Mixer [18], while preserving hypergraph structural priors through structure-aware attention.

**Mixer Architecture.** Our mixer consists of two sequential operations per layer: Hyperedge Interaction (HI) across hyperedges and Hyperedge Refinement (HR) within each hyperedge embedding.

For HI, we use structure-aware multi-head self-attention that combines content similarity with structural proximity. To capture structural relationships, we first construct a hyperedge adjacency matrix  $\mathbf{A}_{\text{hyperedge}} \in \mathbb{R}^{M \times M}$  where entry  $[i, j]$  reflects the number of shared columns between hyperedges  $i$  and  $j$ :

$$\mathbf{A}_{\text{hyperedge}} = \text{RowNormalize}\left(\Pi^T \Pi - \text{diag}(\Pi^T \Pi)\right). \quad (13)$$

We inject this structural prior into multi-head self-attention via an additive bias term:

$$\text{head}_i = \text{softmax}\left(\frac{\mathbf{Q}_i \mathbf{K}_i^T}{\sqrt{d_k}} + \lambda \cdot \mathbf{A}_{\text{hyperedge}}\right) \mathbf{V}_i, \quad (14)$$

where  $\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i$  are query, key, value matrices for head  $i$ , and  $\lambda$  is a learnable scalar initialized to 0.5. This allows hyperedges with high structural connection but low content similarity to still exchange information. We use  $h = 8$  heads and combine outputs with residual connection:  $\mathbf{Z}_{\text{HI}} = \tilde{\mathbf{Z}} + \text{Concat}(\text{head}_1, \dots, \text{head}_h) \mathbf{W}_O$ .

For HR, we apply a two-layer MLP:

$$\mathbf{Z}_{\text{out}} = \mathbf{Z}_{\text{HI}} + \mathbf{W}_2 \cdot \text{GELU}(\mathbf{W}_1 \cdot \text{LayerNorm}(\mathbf{Z}_{\text{HI}})^T)^T, \quad (15)$$

where  $\mathbf{W}_1 \in \mathbb{R}^{4d \times d}$  and  $\mathbf{W}_2 \in \mathbb{R}^{d \times 4d}$  expand and project back with factor 4. We stack  $L_{\text{mixer}} = 2$  layers, with  $\mathbf{A}_{\text{hyperedge}}$  shared across layers.

**Propagating information back to columns.** We propagate enriched hyperedge embeddings back to columns:

$$\mathbf{h}_v^{\text{hyperedge}} = \frac{1}{|\mathcal{N}(v)|} \sum_{j \in \mathcal{N}(v)} \mathbf{W}_{\text{h2c}} \mathbf{z}_{ej}^{(L_{\text{mixer}})}, \quad (16)$$

where  $\mathcal{N}(v) = \{j \mid \Pi_{vj} = 1\}$  is the set of hyperedges containing column  $v$ , and  $\mathbf{W}_{\text{h2c}} \in \mathbb{R}^{d \times d}$  adapts hyperedge embeddings to the column-level space. The final column embedding combines original and hyperedge-enhanced information:

$$\mathbf{h}_v^{\text{final}} = \text{L2Norm}(\text{LayerNorm}(\mathbf{h}_v + \mathbf{h}_v^{\text{hyperedge}})). \quad (17)$$

The residual connection preserves original semantics while L2 normalization enables efficient cosine similarity search.

The overall forward process of HIN is summarized in Algorithm ?? in the appendix of the online full version []. The following theorem shows that our model is more expressive than 1-WL test. The proof is in the online full version []

**THEOREM 3 (HIN IS STRICTLY MORE EXPRESSIVE THAN 1-WL MESSAGE PASSING).** Let  $\mathcal{H}_{\text{base}}$  be the hypothesis class of permutation-equivariant message-passing encoders operating on the joinability graph with raw node features  $\mathbf{h}^{(0)}$  (i.e., 1-WL-initialized inputs). Let  $\mathcal{H}_{\text{pe}}$  be the class obtained by augmenting inputs with

$$PE(v) = \alpha \mathbf{E}_{\text{tbl}}[t(v)] + \beta \mathbf{PE}_{\text{col}}(v), \quad (18)$$

and using an injective node-wise update (as in HIN). Assume there exist two nodes  $u$  and  $v$  such that  $\mathbf{h}_u^{(0)} = \mathbf{h}_v^{(0)}$  and  $u, v$  are 1-WL-indistinguishable under the raw initialization, but  $PE(u) \neq PE(v)$ . Then  $\mathcal{H}_{\text{base}} \subsetneq \mathcal{H}_{\text{pe}}$ .

**Complexity Analysis.** In Stage 1, computing the top- $k$  Laplacian eigenvectors is done offline (dense  $O(N^3)$  for small graphs, or sparse iterative solvers for large/sparse graphs); online positional injection costs  $O(Nd)$ . In Stage 2, the  $L$ -layer node-wise transformations cost  $O(LNd^2)$ , and pooling nodes to hyperedges via  $\mathbf{X}^e = \mathbf{D}_e^{-1} \Pi^T \mathbf{H}$

costs  $O(\|\Pi\|_0 d)$ . In Stage 3, each mixer layer costs  $O(M^2 d + Md^2)$ , yielding  $O(L_{\text{mixer}}(M^2 d + Md^2))$  in total. Thus, the overall online complexity is  $O(LNd^2 + L_{\text{mixer}}(M^2 d + Md^2))$ , with  $\|\Pi\|_0 \leq 2N$  and typically  $M \ll N$  (e.g.,  $M \approx 0.1N - 0.3N$ ).

**Training Objectives.** We adopt a self-supervised training strategy without manual annotations by splitting each table into two overlapping subsets, treating join-key columns as positive pairs (including LLM-generated name variants), and sampling negatives from non-key columns within the same split and across different tables. Our training objective encourages the model to assign higher similarity to joinable column pairs than to non-joinable pairs by a margin. Given final column embeddings  $\mathbf{h}_C^{\text{final}}$  produced by Algorithm ??, we define the cosine similarity between columns  $C_a$  and  $C_b$  as

$$s(C_a, C_b) = \frac{\mathbf{h}_{C_a}^{\text{final}} \mathbf{h}_{C_b}^{\text{final}}}{\|\mathbf{h}_{C_a}^{\text{final}}\|_2 \|\mathbf{h}_{C_b}^{\text{final}}\|_2}. \quad (19)$$

Let  $\mathcal{D}_{\text{train}}^+$  denote the set of joinable (positive) pairs and  $\mathcal{D}_{\text{train}}^-$  denote the set of non-joinable (negative) pairs. For each anchor column  $C_i$ , we sample a positive  $C_i^+$  with  $(C_i, C_i^+) \in \mathcal{D}_{\text{train}}^+$  and a negative  $C_i^-$  with  $(C_i, C_i^-) \in \mathcal{D}_{\text{train}}^-$ . We optimize the margin-based triplet ranking loss

$$\mathcal{L}_{\text{triplet}} = \frac{1}{|\mathcal{D}_{\text{train}}^+|} \sum_{(C_i, C_i^+) \in \mathcal{D}_{\text{train}}^+} \left[ m + s(C_i, C_i^-) - s(C_i, C_i^+) \right]_+, \quad (20)$$

where  $[\cdot]_+ = \max(\cdot, 0)$  and  $m$  is the margin hyperparameter.

## 5 COHERENT ONLINE SEARCH

In this chapter, we present the online search phase of HyperJoin, which efficiently retrieves coherent joinable columns from large-scale data lakes. Section 5.1 formalizes global coherent top-K selection and establishes its NP-hardness. Then, Section 5.2.1 and Section 5.2.2 present an efficient global coherent reranking module: we first retrieve a manageable candidate pool via similarity search and construct a candidate graph, and then perform coherence-aware reranking using a greedy algorithm with MST.

### 5.1 Global Coherent Search

Given a query column  $C_q$ , a candidate pool  $C_B$  returned by an initial retrieval stage, and a target size  $K$ , our goal is to select a subset  $\mathcal{R}^* \subseteq C_B$  with  $|\mathcal{R}^*| = K$  that is both highly relevant to  $C_q$  and mutually joinable as a coherent result set. **To quantify relevance and coherence, we assume a pairwise affinity function  $w(\cdot, \cdot)$  over columns, where  $w(C_i, C_j) \in \mathbb{R}^+$  reflects their joinability and semantic similarity.** We then formulate coherence-aware top-K selection as a weighted objective that balances query relevance and result set coherence:

$$\mathcal{R}^* = \arg \max_{\mathcal{R} \subseteq C_B, |\mathcal{R}|=K} G(\mathcal{R}), \quad (21)$$

where the objective function  $G(\mathcal{R})$  is:

$$G(\mathcal{R}) = \sum_{C \in \mathcal{R}} w(C_q, C) + \lambda \cdot \text{Coherence}(\mathcal{R}) \quad (22)$$

The first term aggregates individual query relevance, while the second term measures how tightly the selected columns are interconnected. The hyperparameter  $\lambda \geq 0$  controls the trade-off between relevance and coherence.

**Computational complexity.** Selecting the optimal  $K$ -column subset is a combinatorial optimization problem because the coherence term is defined over the induced subgraph on  $\{C_q\} \cup \mathcal{R}$  and depends on a maximum spanning tree rooted at  $C_q$  under a constraint on the number of selected nodes. We show that optimizing this coherence-aware objective is computationally intractable.

**THEOREM 4 (NP-HARDNESS OF COHERENCE-AWARE Top- $K$  SELECTION).** *The problem of selecting  $K$  columns from a candidate set to maximize query relevance together with the MST-based coherence defined on the induced subgraph and rooted at  $C_q$  is NP-hard.*

The proof is in the online full version [1].

## 5.2 Two-Stage Coherence-Aware Reranking

Given the NP-hardness of the coherence-aware top- $K$  selection problem (Theorem 9), directly optimizing over all  $N$  columns in the data lake is computationally intractable. We propose a two-stage framework that balances efficiency and effectiveness:

**Stage 1: Candidate Graph Construction.** Given a query column  $C_q$  and the entire data lake containing  $N$  columns, we first perform efficient similarity search to retrieve the Top- $B$  most relevant candidates (where  $K \ll B \ll N$ ), yielding a candidate set  $C_B$ . We then construct a candidate graph  $\mathcal{G}_C = (\mathcal{V}_C, \mathcal{E}_C, w)$  that models both query-to-candidate relationships and inter-candidate relationships. This stage reduces the search space from  $\binom{N}{K}$  to  $\binom{B}{K}$  while ensuring high recall.

**Stage 2: MST-based Greedy Reranking.** Within the candidate graph  $\mathcal{G}_C$ , we perform coherence-aware optimization to select the final top- $K$  result set  $\mathcal{R}$ . We employ a greedy approximation algorithm that iteratively selects candidates with maximum marginal gain, balancing individual query relevance with inter-candidate coherence.

This two-stage approach enables scalable coherence-aware search in large data lakes by combining efficient retrieval with principled coherence optimization.

**5.2.1 Stage 1: Candidate Graph Construction.** Given a query column  $C_q$ , the data lake contains  $N$  total columns. Directly optimizing for a coherent top- $K$  result set over all  $N$  columns would require evaluating  $\binom{N}{K}$  possible subsets, which is computationally intractable. To address this scalability challenge, we adopt a two-stage approach: first retrieve a manageable candidate pool of size  $B$  (where  $K \ll B \ll N$ ) via efficient similarity search, then perform coherence-aware optimization within this narrowed search space. This candidate retrieval stage filters the vast data lake to the most promising candidates while ensuring high recall, providing a tractable foundation for subsequent coherence optimization.

**DEFINITION 3 (CANDIDATE GRAPH).** *Given a query column  $C_q$  and a candidate set  $C_B$  obtained via similarity search (typically  $|C_B| = B \gg K$ ), the candidate graph is defined as a weighted undirected graph  $\mathcal{G}_C = (\mathcal{V}_C, \mathcal{E}_C, w)$  where  $\mathcal{V}_C = \{C_q\} \cup C_B$  contains the query column and all candidate columns, and the edge set  $\mathcal{E}_C = \mathcal{E}_{query} \cup \mathcal{E}_{cand}$  consists of query-to-candidate edges and*

*candidate-to-candidate edges. The weight function  $w : \mathcal{E}_C \rightarrow \mathbb{R}^+$  assigns a positive weight to each edge:*

$$w(C_i, C_j) = sim(h_{C_i}, h_{C_j}), \quad (23)$$

*where  $h_{C_i}, h_{C_j}$  are the structure-aware column embeddings learned by HIN (Section 4.2).*

The candidate graph is constructed through the following steps. First, for a given query column  $C_q$ , we perform efficient similarity search over all columns in the data lake based on cosine similarity between their learned embeddings. This yields a ranked list of the Top- $B$  most relevant candidates,  $C_B = \{C_1, C_2, \dots, C_B\}$ , ensuring high recall. Second, we establish query-to-candidate edges  $\mathcal{E}_{query} = \{(C_q, C) \mid C \in C_B\}$  connecting the query column to all candidates, with edge weights  $w(C_q, C_i) = sim(h_{C_q}, h_{C_i})$  reflecting the relevance of each candidate to the query. Third, to capture the inter-candidate coherence, we establish candidate-to-candidate edges  $\mathcal{E}_{cand} = \{(C_i, C_j) \mid C_i, C_j \in C_B, i \neq j\}$  between all pairs of candidates, with edge weights  $w(C_i, C_j) = sim(h_{C_i}, h_{C_j})$  reflecting the joinability and semantic similarity between the two candidates.

### 5.2.2 Stage 2: MST-based Greedy Reranking.

**THEOREM 5 (PRIM-STYLE MST EXPANSION).** *Let  $\mathcal{G}_C$  be the weighted candidate graph with non-negative edge weights  $w$ . Consider the Maximum Spanning Tree (MST) rooted at  $C_q$  on the induced subgraph over  $\{C_q\} \cup \mathcal{R}$ . In Algorithm 1, the coherence term used in Eq. 24 selects, at each iteration, the candidate whose best connection to the current selected set is maximal, i.e.,  $\max_{C_j \in \mathcal{R}} w(C, C_j)$ . This follows the cut step of Prim's algorithm for constructing a maximum spanning tree, while additionally incorporating query relevance.*

Although the coherence-aware top- $K$  selection problem is NP-hard, this greedy construction enables efficient MaxST-based reranking.

We propose a greedy approximation algorithm via marginal gain. Although computing the exact optimal solution remains NP-hard, we can efficiently compute the marginal gain of adding each candidate column. For a candidate column  $C \in C_B \setminus \mathcal{R}$  not yet in the current result set  $\mathcal{R}$ , we define its marginal gain as:

$$\Delta(C \mid \mathcal{R}) = w(C_q, C) + \lambda \cdot \max_{C_j \in \mathcal{R}} w(C, C_j) \quad (24)$$

When  $\mathcal{R} = \emptyset$ , we define  $\max_{C_j \in \mathcal{R}} w(C, C_j) = 0$ . The first term measures how relevant  $C$  is to the query. The second term measures how well  $C$  connects to the existing columns in  $\mathcal{R}$ , thereby increasing the overall coherence of the result set. The hyperparameter  $\lambda \geq 0$  balances these two objectives.

**Algorithm description.** The complete coherence-aware online search procedure is summarized in Algorithm 1, integrating both candidate graph construction and MST-based reranking. The algorithm inputs the query column, embedding database, target size  $K$ , candidate size  $B$ , and coherence weight  $\lambda$ , and outputs the final result set  $\mathcal{R}$ .

The algorithm operates in two phases. In Phase 1 (lines 1 to 7), we construct the candidate graph. We first retrieve the Top- $B$  most similar candidates using cosine similarity search (line 1). We then construct the vertex set (line 2), query-to-candidate edges (line 3), candidate-to-candidate edges (line 4), and combine them into the edge set (line 5). Edge weights are computed as the cosine similarity

**Algorithm 1:** Coherence-Aware Online Search

---

**Input:** Query column  $C_q$ , Embedding DB  $\{h_C \mid C \in C\}$ , Target size  $K$ , Candidate size  $B$ , Coherence weight  $\lambda$ .

**Output:** Final result set  $\mathcal{R}$ .

// Phase 1: Candidate Graph Construction

- 1  $C_B \leftarrow \text{TopK}(\{h_C \mid C \in C\}, h_{C_q}, B)$
- 2  $\mathcal{V}_C \leftarrow \{C_q\} \cup C_B$
- 3  $\mathcal{E}_{\text{query}} \leftarrow \{(C_q, C) \mid C \in C_B\}$
- 4  $\mathcal{E}_{\text{cand}} \leftarrow \{(C_i, C_j) \mid C_i, C_j \in C_B, i \neq j\}$
- 5  $\mathcal{E}_C \leftarrow \mathcal{E}_{\text{query}} \cup \mathcal{E}_{\text{cand}}$
- 6 **for** each edge  $(C_i, C_j) \in \mathcal{E}_C$  **do**
- 7    $w(C_i, C_j) \leftarrow \text{sim}(h_{C_i}, h_{C_j})$
- // Phase 2: MST-based Greedy Reranking
- 8  $\mathcal{R} \leftarrow \emptyset$
- 9 **for**  $i = 1, \dots, K$  **do**
- 10    $C^* \leftarrow \text{null}$
- 11    $\text{max\_gain} \leftarrow -\infty$
- 12   **for** each candidate  $C \in C_B \setminus \mathcal{R}$  **do**
- 13      $\text{relevance} \leftarrow w(C_q, C)$
- 14     **if**  $\mathcal{R} = \emptyset$  **then**
- 15        $\text{coherence} \leftarrow 0$
- 16     **else**
- 17        $\text{coherence} \leftarrow \max_{C_j \in \mathcal{R}} w(C, C_j)$
- 18      $\text{gain} \leftarrow \text{relevance} + \lambda \cdot \text{coherence}$
- 19     **if**  $\text{gain} > \text{max\_gain}$  **then**
- 20        $\text{max\_gain} \leftarrow \text{gain}$
- 21        $C^* \leftarrow C$
- 22    $\mathcal{R} \leftarrow \mathcal{R} \cup \{C^*\}$
- 23 **return**  $\mathcal{R}$

---

between column embeddings (lines 6 to 7). In Phase 2 (lines 8 to 20), we perform MST-based greedy reranking. Starting from an empty result set (line 8), we iteratively select  $K$  candidates (lines 9 to 20). At each iteration, we initialize the best candidate and maximum gain (lines 10 to 11). For each remaining candidate (line 12), we compute its query relevance (line 13) and coherence increment (line 14), and calculate the total marginal gain (line 15). If the current candidate achieves higher gain than the previous best (line 16), we update the maximum gain and best candidate (lines 17 to 18). Finally, we add the selected candidate to the result set (line 19). The algorithm returns the final result set of size  $K$  (line 20).

**Complexity analysis.** For each query column  $C_q$ , Stage 1 performs a linear scan over all  $N$  column embeddings to compute cosine similarities, costing  $O(Nd)$ , followed by Top- $B$  selection with  $O(N \log B)$  time. In Stage 2, constructing the candidate graph requires computing query–candidate weights in  $O(Bd)$  and candidate–candidate weights in  $O(B^2d)$ . The greedy reranking runs for  $K$  iterations. In each iteration, it scans all remaining candidates (at most  $B$ ) and computes the coherence gain  $\max_{C_j \in \mathcal{R}} w(C, C_j)$ , which takes  $O(|\mathcal{R}|) \leq O(K)$  time. Therefore, the total time complexity of Stage 2 is  $\sum_{i=1}^K O((B-i) \cdot i) = O(BK^2)$ . Overall, the per-query time complexity is  $O(Nd + N \log B + B^2d + BK^2)$  up to logarithmic

**Table 2: Statistics of benchmark datasets**

Dataset	#Tables	#Columns	#Queries	#Join Pairs
USA	2,270	21,702	20	15,584
CAN	9,590	70,982	30	57,124
UK_SG	1,166	11,293	20	8,164
Webtable	2,922	21,787	30	21,029

factors, and the space complexity is  $O(B^2)$  to store the pairwise candidate weights.

## 6 EXPERIMENTAL EVALUATION

### 6.1 Experimental Setup

**Datasets.** We evaluate joinable table discovery on four real-world data lake benchmarks. Three benchmarks are derived from government open data portals in the United States, Canada, and the United Kingdom/Singapore [1]; for convenience, we refer to them as USA, CAN, and UK\_SG, respectively. We also use Webtable [2], a benchmark derived from the WDC Web Table Corpus, and keep only the English relational web tables. For each table, we extract the key column as specified by the benchmark metadata. All benchmarks provide metadata such as table titles, column names, and cell context, and have been used in prior works [3, 7, 8, 19, 22, 24, 25]. Table 2 reports their statistics.

**Baselines.** We compare the following methods. (1) JOSIE [24]: Token-based set similarity search using inverted index and posting lists with cost-model-based greedy merge algorithm. (2) LSH Ensemble [25]: MinHash-based containment search using Locality Sensitive Hashing with dynamic partitioning. (3) DeepJoin [8]: Pre-trained transformer model for column embedding. (4) Snoopy [11]: State-of-the-art embedding model with contrastive learning. (5) Ommimatch [12]: Multi-relational graph neural network for table matching.

**Metrics.** Following prior work [5], we evaluate retrieval quality using Precision@K and Recall@K. Let  $Q$  denote the set of query columns,  $\mathcal{R}_K(q)$  denote the set of Top- $K$  retrieved columns for query  $q$ , and  $\mathcal{R}^*(q)$  denote the set of ground-truth joinable columns for  $q$ . We compute:

$$\text{Precision}@K = \frac{1}{|Q|} \sum_{q \in Q} \frac{|\mathcal{R}_K(q) \cap \mathcal{R}^*(q)|}{K}, \quad (25)$$

$$\text{Recall}@K = \frac{1}{|Q|} \sum_{q \in Q} \frac{|\mathcal{R}_K(q) \cap \mathcal{R}^*(q)|}{|\mathcal{R}^*(q)|}. \quad (26)$$

We report results for  $K \in \{5, 15, 25\}$  to assess performance at different cutoffs.

**Implementation Details.** We implement HyperJoin in PyTorch 2.0.1 with Python 3.9. For initial column encoding, we use a text encoder with learnable token embeddings (vocabulary size  $\sim 1500$ ) to encode table names and column names, followed by mean pooling. For column content features, we use pre-trained fastText embeddings to encode cell values, which are then aggregated and projected through a two-layer MLP with ReLU activation and dropout. The model architecture consists of 2 hypergraph convolutional layers

and 2 MLP-Mixer layers, with embedding dimension 512 and hidden dimension 512. We use three-level positional encoding with dimension 16 for table-level and column-level structural information, computed via Laplacian eigenvector decomposition on the joinable graph. For training, we employ the Adam optimizer with learning rate  $5 \times 10^{-4}$ , batch size 64, and train for 30 epochs. We use triplet loss with margin 1.0 and temperature 0.1 for contrastive learning, with hard negative mining enabled. Dropout rate is set to 0.05 to prevent overfitting. For self-supervised learning, we generate augmented columns using DeepSeek-V3 via API with temperature 0.7 and top-p sampling 0.9. For MST-based coherent reranking, we set candidate size  $B = 50$  and coherence weight  $\lambda = 0.5$ . All experiments are conducted on a workstation with an Intel Xeon E E-2488 CPU (8 cores, 16 threads), an NVIDIA A2 GPU with 15GB memory, and 128 GB RAM.

## 6.2 Overall Evaluation

**Exp-1: Overall Performance Comparison.** Table 3 and Figure 4 present the comprehensive comparison across all methods on four benchmark datasets. HyperJoin consistently achieves the best Precision@K and Recall@K across *WebTable*, *USA*, *Canada* and *UK\_SG*. To quantify the improvement margin over the strongest baseline, we take Snoopy as a reference competitor and compute average absolute gains across datasets. At  $K = 15$ , HyperJoin improves Precision@15 by 20.2 percentage points and Recall@15 by 16.4 percentage points on average. The improvements are most pronounced on *UK\_SG*, where HyperJoin exceeds Snoopy by 27.7 points in Precision@15 and 22.8 points in Recall@15. On *WebTable*, the gain is similarly substantial, with +19.4 points in Precision@15 and +19.8 points in Recall@15. These consistent margins demonstrate that HyperJoin provides not only higher accuracy but also stronger generalization across heterogeneous datasets. For DeepJoin, we initially explored dataset-specific fine-tuning on all benchmarks; however, the outcomes were mixed and revealed a critical failure mode on *Webtable*. While fine-tuning yields a modest improvement on *USA* (Precision@5 increases by approximately 4% compared to zero-shot), it catastrophically degrades performance on *Webtable*, where Precision@5 drops from 69.33% in the zero-shot setting to 0.67% after fine-tuning. Due to this severe instability, we report the *Webtable* results using the zero-shot DistilBERT configuration to ensure a reliable and fair comparison, while for the remaining datasets we report the fine-tuned DeepJoin results. Compared to traditional set-overlap methods (JOSIE, LSH Ensemble), HyperJoin demonstrates even larger improvements, with average Recall@25 gains exceeding 40% across all datasets.

We further analyze whether HyperJoin’s advantage is more evident when the number of discovered tables is limited (small  $K$ ) or comparatively higher (large  $K$ ). At  $K = 5$ , HyperJoin achieves an average Precision@5 of 89.2% and improves over Snoopy by 12.0 points in Precision@5 and 3.35 points in Recall@5, suggesting better early-stage ranking quality under limited discovery. As  $K$  increases, the advantage becomes more pronounced, especially in recall. At  $K = 25$ , HyperJoin exceeds Snoopy by 12.9 points in Precision@25 and 16.9 points in Recall@25 on average, indicating that it can discover substantially more joinable tables while maintaining strong precision. This pattern is particularly clear on *UK\_SG*, where

the recall margin grows from +6.5 points at  $K = 5$  to +34.1 points at  $K = 25$ . Overall, the results suggest that HyperJoin is competitive even at small  $K$ , while its gains are larger at higher  $K$ , where broader discovery makes it harder to preserve both relevance and coverage.

**Exp-2: Ablation Study** We conduct an ablation study to analyze the contribution of each component in HyperJoin. Table 4 reports P@15 and R@15 on four datasets. We first examine the effect coherent reranking (CR) by comparing the full model with and without CR. CR improves performance consistently on all datasets, yielding average absolute gains of +3.9 points in Precision@15 and +2.8 points in Recall@15. The largest benefit is observed on *Canada* (P@15: 75.56 → 81.78, +6.22; R@15: 51.73 → 55.89, +4.16), while the improvements on *USA* (P@15: +3.67; R@15: +2.52) and *UK\_SG* (P@15: +2.33; R@15: +1.85) remain clear, indicating that set-level coherence helps suppress candidates that are locally plausible but globally inconsistent. We then remove key modules to understand where the performance comes from (under the same setting as w/o CR). Removing HIN causes substantial and consistent degradations across all datasets, with the most severe drop on *WebTable* (P@15: 71.33 → 51.11, -20.22; R@15: 67.09 → 46.88, -20.21), highlighting the importance of fine-grained interaction modeling for filtering numerous superficially similar distractors in heterogeneous web tables. Removing the hypergraph (HG) module shows a more dataset-dependent impact: the drop is particularly large on *UK\_SG* (P@15: 86.67 → 59.67, -27.00; R@15: 75.23 → 53.19, -22.04) and is also notable on *USA* (P@15: -15.67; R@15: -11.08), suggesting that higher-order relational structure is crucial for datasets with stronger global regularities. Overall, the ablations demonstrate that HIN is a core contributor across datasets, HG is especially important on more structured corpora, and CR further complements both by improving set-level coherence. We also visualize these trends in a bar chart (Figure 5) to highlight the distinct roles of CR, HIN, and HG across datasets.

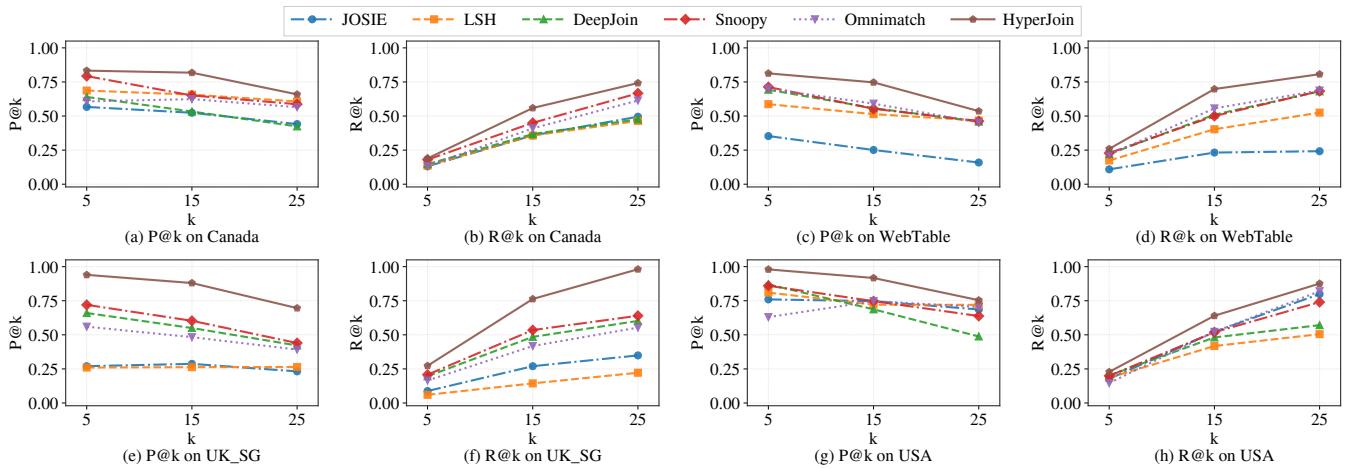
**Exp-3: Efficiency Evaluation.** Figure 6 compare the training and evaluation efficiency of HyperJoin against baselines. For the training phase, HyperJoin completes within 3.8–31.6 minutes across the four datasets (e.g., 31.6 minutes on CAN), which is substantially faster than DeepJoin that requires hours (e.g., 6.9 hours on CAN; 11.5 hours on USA). We also note that OmniMatch incurs a substantial offline overhead for constructing column similarity graphs due to extensive pairwise column operations; in our setting, this preprocessing takes on the order of hours (e.g., 30.94 hours on *USA* and 43.37 hours on CAN). For the discovery/query phase, HyperJoin answers queries in 8–15 seconds, and is consistently faster than DeepJoin on larger datasets, while remaining in the same order of magnitude as LSH and Snoopy.

## 6.3 Robustness Evaluation

**Exp-4: Triplet Loss Margin.** We evaluate triplet loss margins  $m \in \{0.3, 0.5, 0.7, 1.0, 1.5\}$  across four datasets. Table 5 shows that the margin is a critical hyperparameter: small margins can cause severe degradation or collapse, particularly on CAN and webtable (e.g., CAN drops to 4.22% P@15 at  $m = 0.7$  and webtable to 32.44% at  $m = 0.3$ ). Performance stabilizes at  $m = 1.0$ , which achieves strong results across all datasets and avoids catastrophic failures.

**Table 3: Performance comparison across four benchmark datasets.**

Method	webtable						USA						Canada						UK_SG					
	Precision			Recall			Precision			Recall			Precision			Recall			Precision			Recall		
	@5	@15	@25	@5	@15	@25	@5	@15	@25	@5	@15	@25	@5	@15	@25	@5	@15	@25	@5	@15	@25	@5	@15	@25
JOSIE	35.3	25.1	15.9	10.9	23.2	24.2	76.0	74.7	68.6	17.7	52.2	79.9	56.7	52.4	44.1	13.1	35.8	49.5	27.0	28.7	23.2	8.8	27.0	34.9
LSH	58.7	51.4	46.7	17.4	40.3	52.5	81.0	72.1	72.0	18.5	41.8	50.5	68.7	65.7	60.7	13.6	35.7	46.4	26.0	26.3	26.4	6.0	14.4	22.2
DeepJoin	69.3	55.8	45.9	22.1	51.0	68.1	87.0	68.7	48.8	20.2	48.1	57.1	64.0	53.3	42.4	14.7	36.8	47.6	66.0	55.0	42.2	19.6	48.5	60.2
Snoopy	71.3	55.3	46.3	22.8	50.0	68.3	86.0	74.7	63.8	19.9	51.9	74.0	79.3	65.1	58.7	18.0	45.0	66.7	72.0	60.3	44.0	20.8	53.5	64.0
Omnimatch	70.0	59.1	45.3	21.8	55.6	68.7	63.0	74.7	70.4	14.6	52.2	82.0	60.7	62.4	56.5	13.3	41.0	61.4	56.0	48.3	39.2	16.5	41.8	55.3
HyperJoin	81.3	74.7	53.6	25.9	69.8	80.7	98.0	91.7	75.4	22.8	64.0	87.6	83.3	81.8	65.9	18.9	55.9	74.2	94.0	88.0	69.6	27.3	76.3	98.1



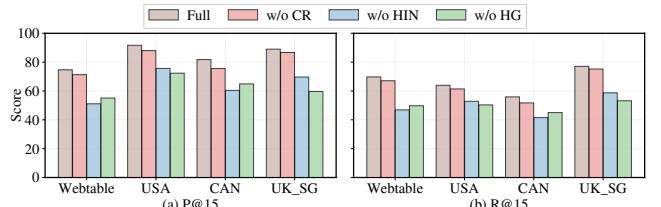
**Figure 4: Performance comparison across different datasets and k values.**

**Table 4: Ablation study on four datasets using P@15 and R@15 (percentage). We quantify the effect of CR by comparing the full model with and without CR, and evaluate core module ablations by removing HIN or HG.**

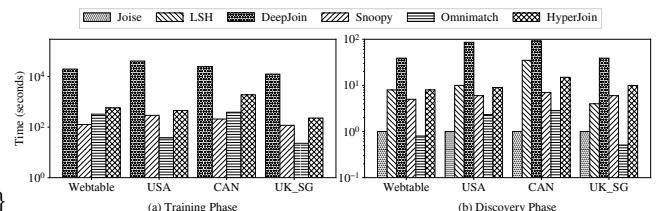
Variant	webtable		USA		Canada		UK_SG	
	P@15	R@15	P@15	R@15	P@15	R@15	P@15	R@15
Full	<b>74.67</b>	<b>69.75</b>	<b>91.67</b>	<b>63.94</b>	<b>81.78</b>	<b>55.89</b>	<b>89.00</b>	<b>77.08</b>
w/o CR	71.33	67.09	88.00	61.42	75.56	51.73	86.67	75.23
w/o HIN	51.11	46.88	75.67	52.83	60.44	41.54	69.67	58.73
w/o HG	55.11	49.75	72.33	50.34	64.89	45.01	59.67	53.19

Larger margins further improve UK\_SG, webtable, and USA (best at  $m = 1.5$ ), but hurt CAN, indicating over-constraint on large and diverse data. Overall,  $m = 1.0$  is a robust default, with  $m = 1.5$  preferred when optimizing for UK\_SG/USA/webtable specifically.

**Exp-5: MST Coherence Weight  $\lambda$ .** We evaluate  $\lambda \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$  across four datasets. Table 6 highlights clear dataset-dependent preferences: UK\_SG favors larger  $\lambda$  (best at  $\lambda = 0.9$ ), indicating stronger benefits from coherence-aware reranking, whereas USA prefers smaller  $\lambda$  (best at  $\lambda = 0.3$ ), suggesting that relevance signals are already strong and excessive coherence can be restrictive. webtable shows a relatively flat trend with the best performance at



**Figure 5: Ablation study on four datasets measured by P@15 and R@15.**



**Figure 6: Training and evaluation efficiency across datasets.**

a moderate-to-high setting ( $\lambda = 0.7$ ). CAN is comparatively insensitive, with similar performance at both low and high  $\lambda$ . Overall,

**Table 5: Impact of triplet loss margin on model performance. Results report P@15 and R@15 with MST reranking across four datasets. Bold indicates the best performance per dataset/metric.**

Margin	UK_SG		webtable		USA		CAN	
	P@15	R@15	P@15	R@15	P@15	R@15	P@15	R@15
0.3	86.33	74.91	32.44	29.14	76.67	53.53	14.67	9.42
0.5	87.00	75.32	75.33	70.59	90.67	63.19	13.11	8.33
0.7	86.67	75.10	78.22	74.13	90.67	63.17	4.22	2.39
1.0	88.00	76.25	79.56	75.61	91.67	63.95	<b>76.89</b>	<b>52.94</b>
1.5	<b>88.67</b>	<b>76.86</b>	<b>80.22</b>	<b>76.20</b>	<b>94.67</b>	<b>65.99</b>	64.22	43.55

**Table 6: Impact of MST coherence weight  $\lambda$  on retrieval performance. Results report P@15 and R@15 with MST reranking across four datasets. Bold indicates the best performance per dataset/metric.**

$\lambda$	UK_SG		webtable		USA		CAN	
	P@15	R@15	P@15	R@15	P@15	R@15	P@15	R@15
0.1	86.67	75.16	78.89	75.02	91.67	63.95	77.11	<b>53.19</b>
0.3	88.00	76.25	79.11	75.22	<b>92.00</b>	<b>64.19</b>	76.89	52.99
0.5	88.00	76.25	79.56	75.61	91.67	63.95	76.89	52.94
0.7	88.33	76.50	<b>79.78</b>	<b>75.78</b>	90.33	63.00	76.89	52.93
0.9	<b>88.67</b>	<b>76.88</b>	79.33	75.39	90.33	63.00	<b>77.11</b>	53.08

**Table 7: Impact of MST candidate size  $B$  on retrieval performance. Results report P@25 and R@25 across four datasets. Bold indicates the best performance per dataset/metric.**

$B$	UK_SG		webtable		USA		CAN	
	P@25	R@25	P@25	R@25	P@25	R@25	P@25	R@25
25	69.40	97.82	57.87	88.12	75.00	87.19	61.73	70.28
35	<b>69.60</b>	<b>98.06</b>	58.40	88.73	75.20	87.37	63.60	72.54
50	<b>69.60</b>	<b>98.06</b>	<b>58.53</b>	<b>88.87</b>	<b>75.40</b>	<b>87.58</b>	63.60	72.53
70	<b>69.60</b>	<b>98.06</b>	<b>58.53</b>	<b>88.87</b>	<b>75.40</b>	<b>87.58</b>	<b>63.87</b>	<b>72.82</b>
100	<b>69.60</b>	<b>98.06</b>	<b>58.53</b>	<b>88.87</b>	<b>75.40</b>	<b>87.58</b>	<b>63.87</b>	<b>72.82</b>

$\lambda = 0.5$  serves as a robust default, while dataset-specific tuning can yield modest gains.

**Exp-6: MST Candidate Size  $B$ .** We evaluate candidate pool sizes  $B \in \{25, 35, 50, 70, 100\}$  for MST reranking using P@25 and R@25. Table 7 shows that larger candidate pools yield better performance but quickly saturate: UK\_SG reaches its optimum at  $B = 35$ , while webtable and USA saturate at  $B = 50$  (identical results for  $B \geq 50$ ). CAN continues to improve up to  $B = 70$ , but the gain beyond  $B = 50$  is marginal. Overall,  $B = 50$  provides a robust default, achieving near-optimal quality across all datasets with moderate computational cost.

## 7 RELATED WORK

Existing methods for joinable table discovery can be broadly classified into traditional set-overlap methods and deep learning-based methods. Traditional approaches, such as Jaccard similarity [10], containment coefficients [4, 21, 23], and Locality-Sensitive Hashing

(LSH) [25], measure column overlap using set-based metrics and exact or fuzzy string matching. While scalable, these methods rely on syntactic matching and fail to capture semantic relationships. Deep learning-based methods address this limitation through two main paradigms. Pre-trained language model approaches, exemplified by DeepJoin [8] and Snoopy [11], employ transformers (e.g., BERT) to encode column values into semantic embeddings. Graph neural network approaches, such as OmniMatch [12], model table repositories as multi-relational graphs and use RGCN [16] to learn structure-aware column embeddings. However, these graph-based methods are limited to pairwise relationships and cannot explicitly capture higher-order correlations, such as multi-hop join paths involving three or more columns or multi-column structures within tables. HyperJoin advances beyond these limitations by introducing a multi-context hypergraph representation that explicitly models both high-order join relationships across tables (Type-1 hyperedges) and intra-table structural dependencies (Type-2 hyperedges), combined with a coherence-aware reranking mechanism to ensure globally consistent result sets.

## 8 CONCLUSION

In this paper, we address the problem of joinable table discovery in data lakes and propose HyperJoin, a hypergraph-based framework that explicitly models high-order structural relationships. HyperJoin employs a two-phase architecture consisting of an offline representation learning phase and an online search phase. In the offline phase, we design a multi-context hypergraph with inter-table and intra-table hyperedges, along with a HIN for representation learning, and introduce a self-supervised learning strategy via table splitting and LLM-based data augmentation to enable training without human annotations. In the online search phase, we formulate the coherence-aware top-K selection problem, prove its NP-hardness, and propose an efficient greedy approximation algorithm. Extensive experiments across 4 public benchmarks demonstrate that HyperJoin achieves superior performance over state-of-the-art methods in both supervised and label-free settings.

## REFERENCES

- [1] [n.d.]. OpenData. <https://open.canada.ca/>. Accessed: n.d.
- [2] [n.d.]. WebTable. <https://webdatcommons.org/webtables/>. Accessed: n.d.
- [3] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. TabEntity linking in web tables. In *International Semantic Web Conference*. Springer, 425–441.
- [4] Dong Deng, Albert Kim, Samuel Madden, and Michael Stonebraker. 2017. SILK-MOTH: An Efficient Method for Finding Related Sets with Maximum Matching Constraints. *Proceedings of the VLDB Endowment* 10, 10 (2017).
- [5] Yuhao Deng, Chengliang Chai, Lei Cao, Qin Yuan, Siyuan Chen, Yanru Yu, Zhaoze Sun, Junyi Wang, Jiajun Li, Ziqi Cao, et al. 2024. Lakebench: A benchmark for discovering joinable and unionable tables in data lakes. *Proceedings of the VLDB Endowment* 17, 8 (2024), 1925–1938.
- [6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.
- [7] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 456–467.
- [8] Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. 2023. DeepJoin: Joinable Table Discovery with Pre-Trained Language Models. *Proceedings of the VLDB Endowment* 16, 10 (2023), 2458–2470.

- [9] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J Miller. 2023. Semantics-Aware Dataset Discovery from Data Lakes with Contextualized Column-Based Representation Learning. *Proceedings of the VLDB Endowment* 16, 7 (2023), 1726–1739.
- [10] Raul Castro Fernandez, Jisoo Min, Demitri Nava, and Samuel Madden. 2019. Lazo: A cardinality-based method for coupled estimation of jaccard similarity and containment. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1190–1201.
- [11] Yuxiang Guo, Yuren Mao, Zhonghao Hu, Lu Chen, and Yunjun Gao. 2025. Snoopy: Effective and Efficient Semantic Join Discovery via Proxy Columns. *IEEE Transactions on Knowledge and Data Engineering* (2025).
- [12] Christos Koutras, Jian Zhang, Xiao Qin, Chuan Lei, Vasileios Ioannidis, Christos Faloutsos, George Karypis, and Asterios Katsifodimos. 2025. OmniMatch: Joinability Discovery in Data Products. *Proceedings of the VLDB Endowment* 18, 11 (2025), 4588–4601.
- [13] Eugenie Y Lai, Yeye He, and Surajit Chaudhuri. 2025. Auto-Prep: Holistic Prediction of Data Preparation Steps for Self-Service Business Intelligence. *arXiv preprint arXiv:2504.11627* (2025).
- [14] Alec Radford. 2018. Improving language understanding by generative pre-training. *Preprint* (2018).
- [15] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).
- [16] 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 *European semantic web conference*. Springer, 593–607.
- [17] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding. *Advances in neural information processing systems* 33 (2020), 16857–16867.
- [18] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. 2021. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems* 34 (2021), 24261–24272.
- [19] Fei Wang, Kexuan Sun, Muhan Chen, Jay Pujara, and Pedro Szekely. 2021. Retrieving complex tables with multi-granular graph representation learning. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1472–1482.
- [20] Boris Weisfeiler and Andrei Leman. 1968. The reduction of a graph to canonical form and the algebra which appears therein. *nti, Series 2*, 9 (1968), 12–16.
- [21] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 97–108.
- [22] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *ACL: Association for Computational Linguistics*, 8413–8426.
- [23] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M Procopiuc, and Divesh Srivastava. 2010. On multi-column foreign key discovery. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 805–814.
- [24] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J Miller. 2019. Josie: Overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data*, 847–864.
- [25] Erkang Zhu, Fatemeh Nargesian, Ken Q Pu, and Renée J Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. *Proceedings of the VLDB Endowment* 9, 12 (2016).

## 9 APPENDIX

*Setup.* Let  $Y_{ij} \in \{0, 1\}$  denote whether two columns  $(v_i, v_j)$  are joinable. Let  $\mathbf{x}_i$  be the initial feature of column  $v_i$ . For each column  $v_i$ , define two context variables induced by the constructed hypergraph: (i) its intra-table context  $\mathcal{N}^{\text{intra}}(v_i)$  (the set of columns co-located with  $v_i$  in the same table), and (ii) its inter-table context  $\mathcal{N}^{\text{inter}}(v_i)$  (the set of columns connected with  $v_i$  through joinable connectivity, e.g., within the same join-connected component). We write the available information for deciding joinability as a feature set:

$$\begin{aligned}\mathcal{F}_{\text{single}}(i, j) &= \{\mathbf{x}_i, \mathbf{x}_j\}, \\ \mathcal{F}_{\text{multi}}(i, j) &= \left\{ \mathbf{x}_i, \mathbf{x}_j, \mathcal{N}^{\text{intra}}(v_i), \mathcal{N}^{\text{intra}}(v_j), \right. \\ &\quad \left. \mathcal{N}^{\text{inter}}(v_i), \mathcal{N}^{\text{inter}}(v_j) \right\}.\end{aligned}\quad (27)$$

Let  $\Psi^*(\mathcal{F})$  denote the minimum achievable expected discovery risk under feature set  $\mathcal{F}$ :

$$\Psi^*(\mathcal{F}) = \inf_g \mathbb{E}[\ell(g(\mathcal{F}), Y_{ij})],$$

where  $g$  is any measurable scoring function and  $\ell(\cdot)$  is a bounded loss (e.g., logistic or 0-1 loss).

**THEOREM 6.** *For the two feature sets  $\mathcal{F}_{\text{multi}}$  and  $\mathcal{F}_{\text{single}}$  defined above, the Bayes-optimal risk satisfies*

$$\Psi^*(\mathcal{F}_{\text{multi}}) \leq \Psi^*(\mathcal{F}_{\text{single}}).$$

*Moreover, if the added contexts are informative in the sense that  $\mathbb{P}(Y_{ij} = 1 | \mathcal{F}_{\text{multi}}) \neq \mathbb{P}(Y_{ij} = 1 | \mathcal{F}_{\text{single}})$  with non-zero probability, then the inequality is strict.*

**PROOF.** Any predictor  $g_{\text{single}}(\mathcal{F}_{\text{single}})$  is also feasible under  $\mathcal{F}_{\text{multi}}$  by defining  $g_{\text{multi}}(\mathcal{F}_{\text{multi}}) = g_{\text{single}}(\mathcal{F}_{\text{single}})$ , i.e., simply ignoring the additional context variables. Therefore,

$$\begin{aligned}\Psi^*(\mathcal{F}_{\text{multi}}) &= \inf_{g_{\text{multi}}} \mathbb{E}[\ell(g_{\text{multi}}(\mathcal{F}_{\text{multi}}), Y_{ij})] \\ &\leq \inf_{g_{\text{single}}} \mathbb{E}[\ell(g_{\text{single}}(\mathcal{F}_{\text{single}}), Y_{ij})] = \Psi^*(\mathcal{F}_{\text{single}}).\end{aligned}\quad (28)$$

If the conditional distribution of  $Y_{ij}$  changes when conditioning on the added contexts, then the Bayes decision rule under  $\mathcal{F}_{\text{multi}}$  differs from that under  $\mathcal{F}_{\text{single}}$  on a set of non-zero measure, yielding strictly smaller Bayes risk.  $\square$

**THEOREM 7 (PRIOR METHODS ARE SPECIAL CASES).** *Consider our framework that learns column embeddings via a model  $f_\theta$  operating on the constructed hypergraph. DeepJoin and Snoopy can be viewed as special cases obtained by restricting the utilized context to  $\mathcal{F}_{\text{single}}$  (i.e., column-wise encoding without using  $\mathcal{N}^{\text{intra}}$  or  $\mathcal{N}^{\text{inter}}$ ).*

**PROOF.** This can be achieved by choosing  $\mathcal{E}$  as  $N$  singleton hyperedges  $\mathcal{E} = \{\{v_1\}, \dots, \{v_N\}\}$  (or equivalently disabling hyperedge message passing). If every hyperedge is singleton, then each column belongs to exactly one hyperedge of size 1. Any within-hyperedge aggregation (mean/sum/attention) over a singleton returns the column itself, thus Local Hyperedge Aggregation leaves representations unchanged. Moreover, since hyperedges share no nodes, hyperedge adjacency becomes diagonal and Global Hyperedge Mixing degenerates to per-hyperedge (thus per-column) transformations without cross-column exchange. Therefore, the resulting embedding has the form  $\mathbf{h}_i = \phi_\theta(\mathbf{x}_i)$ . DeepJoin instantiates

$\phi_\theta$  as a PLM-based column encoder trained with self-supervised contrastive learning, and Snoopy further applies a deterministic proxy-based transformation on top of such column-wise embeddings. Hence both methods are recovered as restricted instances that do not leverage  $N^{\text{intra}}/N^{\text{inter}}$ .  $\square$

**THEOREM 8 (POSITIONALIZED HIN IS STRICTLY MORE EXPRESSIVE THAN 1-WL MESSAGE PASSING).** *Let  $\mathcal{H}_{\text{base}}$  be the hypothesis class of permutation-equivariant message-passing encoders operating on the joinability graph with raw node features  $\mathbf{h}^{(0)}$  (i.e., 1-WL-initialized inputs). Let  $\mathcal{H}_{\text{pe}}$  be the class obtained by augmenting inputs with*

$$PE(v) = \alpha E_{\text{tbl}}[t(v)] + \beta PE_{\text{col}}(v), \quad (29)$$

*and using an injective node-wise update (as in HIN). Assume there exist two nodes  $u$  and  $v$  such that  $\mathbf{h}_u^{(0)} = \mathbf{h}_v^{(0)}$  and  $u, v$  are 1-WL-indistinguishable under the raw initialization, but  $PE(u) \neq PE(v)$ . Then  $\mathcal{H}_{\text{base}} \subsetneq \mathcal{H}_{\text{pe}}$ .*

**PROOF.** We first note that  $\mathcal{H}_{\text{base}} \subseteq \mathcal{H}_{\text{pe}}$  since any encoder in  $\mathcal{H}_{\text{base}}$  can be realized in  $\mathcal{H}_{\text{pe}}$  by ignoring the positional terms (e.g., setting  $\alpha = \beta = 0$ ).

Since the encoder in  $\mathcal{H}_{\text{base}}$  is permutation-equivariant and  $u, v$  are 1-WL-indistinguishable with identical raw features, it must output identical representations for  $u$  and  $v$  at every layer; hence no hypothesis in  $\mathcal{H}_{\text{base}}$  can separate  $u$  and  $v$ .

In contrast, under  $\mathcal{H}_{\text{pe}}$ , the augmented inputs satisfy

$$\mathbf{h}_u = \mathbf{h}_u^{(0)} + PE(u) \neq \mathbf{h}_v^{(0)} + PE(v) = \mathbf{h}_v.$$

With an injective node-wise update, this distinction is preserved, so there exists a hypothesis in  $\mathcal{H}_{\text{pe}}$  whose outputs separate  $u$  and  $v$  (e.g., via a linear readout). Therefore,  $\mathcal{H}_{\text{base}} \subsetneq \mathcal{H}_{\text{pe}}$ .  $\square$

**THEOREM 9 (NP-HARDNESS OF COHERENCE-AWARE Top-K SELECTION).** *The problem of selecting  $K$  columns from a candidate set to maximize query relevance together with the MST-based coherence defined on the induced subgraph and rooted at  $C_q$  is NP-hard.*

**PROOF.** We prove NP-hardness by reduction from the 0–1 Knapsack problem. Given a knapsack instance with items  $\{1, \dots, n\}$ , each item  $i$  having integer weight  $w_i$  and value  $v_i$ , and a capacity  $W$ , the goal is to choose a subset of items with total weight at most  $W$  that maximizes the total value.

We construct an instance of our selection problem as follows. We create a root node corresponding to the query column  $C_q$ . For each item  $i$ , we create a path of  $w_i$  candidate nodes  $\{u_{i,1}, \dots, u_{i,w_i}\}$ . We add an edge between  $C_q$  and  $u_{i,1}$  with weight 0. For each  $j = 1, \dots, w_i - 2$ , we add an edge  $(u_{i,j}, u_{i,j+1})$  with weight 0. We assign the item value to the last edge on the path by setting the edge weight  $(u_{i,w_i-1}, u_{i,w_i})$  to be  $v_i$ . We further add  $W$  dummy candidate nodes  $\{d_1, \dots, d_W\}$  and connect each  $d_t$  to  $C_q$  with an edge weight 0. We set the result size to  $K = W$  and set all query relevance weights to zero, so the objective is determined solely by the coherence term.

We claim that in the induced subgraph on  $\{C_q\} \cup \mathcal{R}$ , the coherence value equals the total value of the knapsack items encoded by  $\mathcal{R}$ . Because coherence is computed on the induced subgraph, any selected node must be connected to  $C_q$  using only nodes in  $\mathcal{R} \cup \{C_q\}$ . If  $\mathcal{R}$  contains  $u_{i,w_i}$ , then to make  $u_{i,w_i}$  connected to  $C_q$  in the induced subgraph,  $\mathcal{R}$  must contain the entire prefix  $\{u_{i,1}, \dots, u_{i,w_i-1}\}$ .

Therefore, obtaining the positive weight  $v_i$  on the last edge necessarily consumes exactly  $w_i$  selected nodes. Selecting any strict prefix of the path yields no positive contribution because all edges on the prefix have weight 0.

Given any feasible knapsack solution  $S$  with  $\sum_{i \in S} w_i \leq W$ , we form a selection set  $\mathcal{R}$  by including all nodes on the paths of items in  $S$  and filling the remaining budget with dummy nodes so that  $|\mathcal{R}| = W$ . In the induced subgraph, a maximum spanning tree rooted at  $C_q$  includes the last edge of each chosen item path, contributing exactly  $\sum_{i \in S} v_i$ , and all other required edges have weight 0, so the coherence equals the knapsack value. Conversely, from any selection set  $\mathcal{R}$  of size  $W$ , we extract the corresponding item subset  $S$  consisting of those indices  $i$  for which  $u_{i,w_i} \in \mathcal{R}$ . By the connectivity requirement on the induced subgraph, this implies that all  $w_i$  nodes on the path are selected, so  $\sum_{i \in S} w_i \leq W$ . The coherence of  $\mathcal{R}$  is then exactly  $\sum_{i \in S} v_i$  because positive weights only appear on the last edges of selected item paths.

Thus, solving the selection problem optimally would solve the 0–1 Knapsack problem optimally, which is NP-hard. Therefore, the coherence-aware top- $K$  selection problem is NP-hard.  $\square$

## 9.1 Overall forward process of HIN.

Algorithm ?? summarizes the complete forward pass of HIN. Lines 2–7 inject positional encodings by precomputing Laplacian eigenvectors offline and adding table-level and column-level encodings. Lines 9–14 apply node-level GNN transformation and aggregate to hyperedge level with domain-specific transforms. Lines 16–24 perform global hyperedge mixing via structure-aware attention and channel mixing. Lines 26–29 propagate hyperedge embeddings back to columns with residual connections. Finally, line 31 returns the final embeddings.



