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


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

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

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

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

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

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 **Hyper**graph framework for **Join**able 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 by integrating global message passing across hyperedges with local message passing between columns and hyperedges. To strengthen coherence and internal consistency in the result columns, we cast online ranking as a coherence-aware top-$K$ column selection problem. We then introduce a reranking module that leverages a maximum spanning tree algorithm to prune noisy connections and maximize coherence. Experiments demonstrate the superiority of HyperJoin, achieving average improvements of 21.4% (Precision@15) and 17.2% (Recall@15) over the best baseline.



**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/T-Lab/HyperJoin.




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


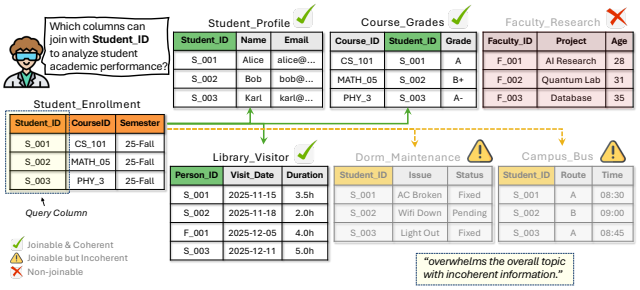


**Figure 1: An example of joinable table discovery in data lake.**

## 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 [7, 10, 12, 26, 31] 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 [13] for an in-depth discussion). Early non-learning methods generally cast joinable table discovery as a set-similarity search problem [23]. Representative approaches, such

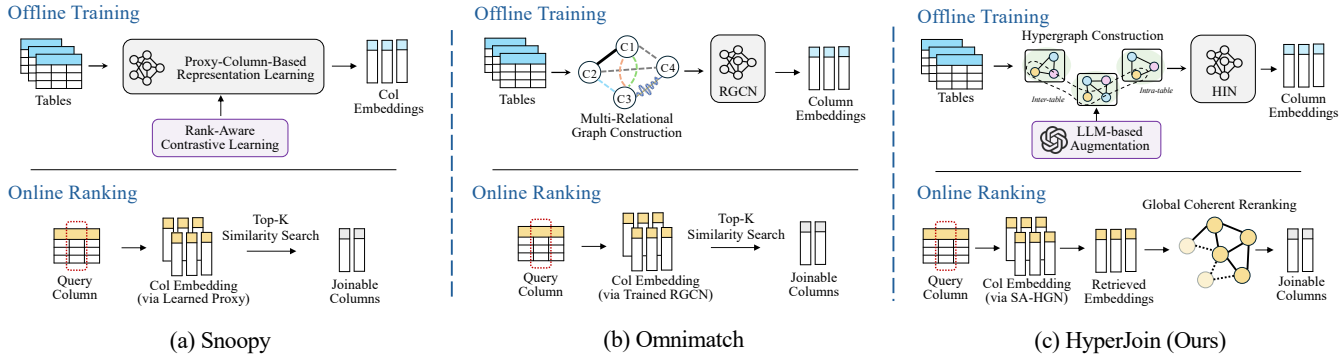


**Figure 2: Framework comparisons of joinable table discovery methods.**

as JOSIE [49] and LSH Ensemble [50], 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 [14] and DistilBERT [33]) for more effective joinable table discovery. Representative methods include DeepJoin [16], Snoopy [19] and Omnimatch [25]. As illustrated in Figure 2(a) and Figure 2(b), these methods typically integrate an offline training phase with an online ranking 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 within tables, 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 [16] and Snoopy [19] process columns individually to learn their representations. Although Omnimatch [25] attempts to utilize a graph to learn to model the column relationships. However, it still primarily focuses on pairwise relationships.

Secondly, in the online ranking 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 with a unified framework for effective offline representation learning?* 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 discovery.

*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 4, 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 **Hyper**graph-based framework that recasts the problem of **Join**able 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 ranking 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 formulate the task of global coherent joinable table selection as a coherence-aware top-$K$ column selection problem that balances individual query relevance with inter-candidate coherence. We prove that this problem is NP-hard. Thus, the proposed global coherent reranking module first constructs a weighted complete graph spanning the query and target columns to capture the structural context of columns after the first-stage top-B retrieval ($K < B$). An efficient greedy approximation algorithm that leverages a Maximum Spanning Tree (MST) algorithm are employed to prune noisy connections and maximize the global matching score.

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

- We propose HyperJoin, a new 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 ranking 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 benchmarks show the superiority of HyperJoin, achieving average improvements of 21.4% in Precision@15 and 17.2% 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 [13, 15–17, 19]. A data lake contains a table repository $\mathcal{T} = \{T_1, T_2, \ldots, T_M\}$ 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 $\mathcal{C} = \{C_1, C_2, \ldots, C_N\}$ containing $N$ columns across all tables. Each column $C$ is represented as a set of cell values $\{c_1, c_2, \ldots, c_{|C|}\}$ and may also include associated metadata such as column names, where $|C|$ denotes the cardinality (number of 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.

**Table 1: Symbols and Descriptions**

| Notation | Description |
|---|---|
| $\mathcal{T}$ | table repository |
| $T$ | a table, consisting of multiple columns |
| $\mathcal{C}$ | 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 |

Definition 1 (Pairwise Column Joinability). *Given a query column $C_q$ and a candidate column $C_t$ from the repository $\mathcal{C}$, their pairwise joinability score $J(C_q, C_t)$ is defined as the average best-match similarity for each value in the query column:*

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

*This directional score measures how well $C_q$ is covered by $C_t$. $sim(v_q, v_t) \in [0, 1]$ 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 $\mathcal{R}^* \subseteq \mathcal{C}$ of size $K$ that each $C_t \in \mathcal{R}$ has a high joinability with $C_q$.*

### 2.2 Hypergraph

A hypergraph is a generalization of a graph in which an edge (called a *hyperedge*) can connect *any number* of vertices rather than only two [3, 27]. Formally, a hypergraph $\mathcal{H} = (V, E)$ consists of a vertex set $V$ and a hyperedge set $E$, where each hyperedge $e \in E$ denotes a *group-wise* relation among multiple vertices (i.e., $\emptyset \neq e \subseteq V$). This capability makes hypergraphs a natural modeling tool for higher-order interactions in real-world data, such as co-authorship [44] and biological complexes [29].

### 2.3 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 [24]. A prevailing architecture in this domain is the auto-regressive (AR) model, which generates sequences by iteratively predicting the next token conditioned on

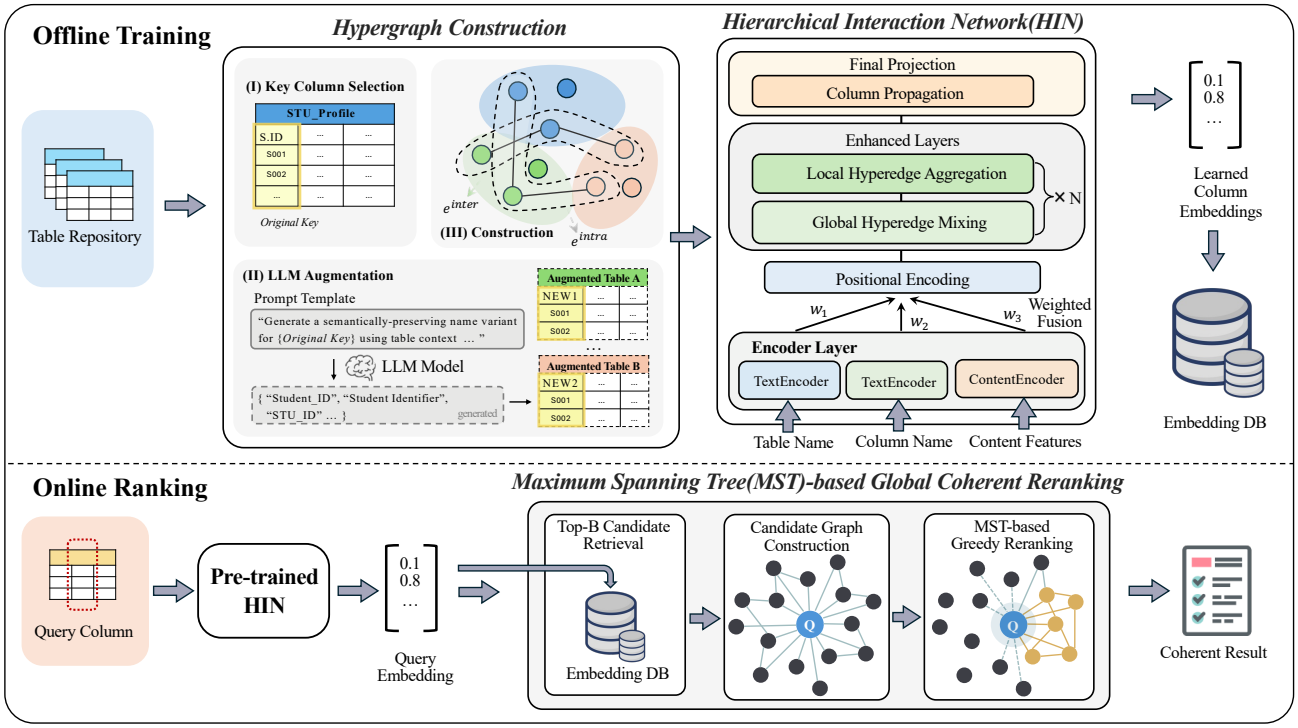


**Figure 3: Framework overview of HyperJoin.**

the preceding context [40, 41]. Building upon these foundations, the field has evolved from PLMs to LLMs [38, 39, 47].

PLMs, such as the auto-encoding BERT [14] and its variants DistilBERT [33] and MPNet [35], 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 [32]) represent a significant leap in scale, often containing tens or hundreds of billions of parameters. While LLMs share the pre-training paradigm 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., IVFPQ [22] or HNSW [30] in the FAISS library) is often built for efficient retrieval [42]. 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 as the joinable tables.

Despite this shared framework, these methods vary in how they obtain column embeddings. DeepJoin [16] 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 [19] introduces proxy columns to overcome limitations of direct PLM encoding. It learns a set of proxy column embeddings through rank-aware contrastive learning. Omnimatch [25] 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 as a coherence-aware top-$K$ column 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 $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathbf{X}^v, \mathbf{X}^e, \mathbf{\Pi})$, where $\mathcal{V} = \{v_1, \ldots, v_N\}$ denotes the set of $N$ nodes, each corresponding to a column in the data lake, and $\mathcal{E} = \{e_1, \ldots, e_M\}$ denotes the set of $M$ hyperedges. Each node $v_i \in \mathcal{V}$ is associated with an initial feature vector $\mathbf{x}^v_{v_i} \in \mathbb{R}^d$, 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 $\mathbf{x}^e_{e_j} \in \mathbb{R}^{d_e}$, 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 $\mathbf{\Pi} \in \{0,1\}^{N\times M}$, where $\mathbf{\Pi}_{ij} = 1$ 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 $\mathbf{x}^v_{v_i}$ 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 $\mathcal{S} = \{\texttt{table}, \texttt{col}, \texttt{content}\}$ denote the components and $\mathbf{w} \in \mathbb{R}^3$ be learnable fusion weights; we compute

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

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

*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 [28].

*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:

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

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

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

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 $\mathcal{F}_{multi}$ and $\mathcal{F}_{single}$ defined above, the Bayes-optimal risk satisfies*

$$\Psi^*(\mathcal{F}_{multi}) \;\le\; \Psi^*(\mathcal{F}_{single}).$$

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

**Theorem 2.** *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 $\mathcal{F}_{single}$ (i.e., column-wise encoding without using $\mathcal{N}^{intra}$ or $\mathcal{N}^{inter}$).*

## 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 [20], 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* [43]. 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

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

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

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

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 $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ and project them through a two-layer MLP:

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

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 is:

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

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 [4, 9], 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:

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

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:

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

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:

$$\mathbf{X}^e = \mathbf{D}_e^{-1}\mathbf{\Pi}^\top\mathbf{H}, \tag{10}$$

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

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):

$$\tilde{\mathbf{Z}}_{\text{inter}} = \mathbf{Z}_{\text{inter}}\mathbf{W}_{\text{inter}}, \tag{11}$$

$$\tilde{\mathbf{Z}}_{\text{intra}} = \mathbf{Z}_{\text{intra}}\mathbf{W}_{\text{intra}}, \tag{12}$$

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 [36], 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 $j$:

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

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

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

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: $\mathbf{Z}_{\text{HI}} = \tilde{\mathbf{Z}} + \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}_O$.

For HR, we apply a two-layer MLP:

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

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 all the layers.

**Propagating Information Back to Columns.** We propagate enriched hyperedge embeddings back to columns:

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

where $\mathcal{N}(v) = \{j \mid \mathbf{\Pi}_{vj} = 1\}$ 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:

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

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

The overall forward process of HIN is summarized in Algorithm 2 in the appendix of the online full version [28]. The following theorem shows that our model is more expressive than 1-WL test. The proof is in the online full version [28].

**Theorem 3** (HIN is strictly more expressive than 1-WL message passing)**.** *Let $\mathcal{H}_{base}$ 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 $\mathcal{H}_{pe}$ be the class obtained by augmenting inputs with*

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

*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 $\mathcal{H}_{base} \subsetneq \mathcal{H}_{pe}$.*

**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 $\mathbf{X}^e = \mathbf{D}_e^{-1}\mathbf{\Pi}^\top\mathbf{H}$ costs $O(\|\mathbf{\Pi}\|_0 d)$. In Stage 3, each mixer layer costs $O(M^2d + Md^2)$, yielding $O(L_{\text{mixer}}(M^2d + Md^2))$ in total. Thus, the overall online complexity is $O(LNd^2 + L_{\text{mixer}}(M^2d + Md^2))$, with $\|\mathbf{\Pi}\|_0 \leq 2N$ and typically $M \ll N$ (e.g., $M \approx 0.1N$–$0.3N$).

**Training Objectives.** We adopt a self-supervised training strategy without labels 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 2, we define the cosine similarity between columns $C_a$ and $C_b$ as

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

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

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

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

## 5 ONLINE RANKING

We present the online ranking phase of HyperJoin, which efficiently retrieves coherent joinable columns from large-scale data lakes. Section 5.1 discusses the objective and its computational complexity. Then, Section 5.2 presents an efficient global coherent reranking module: we first retrieve a manageable candidate pool via similarity search and construct a candidate graph, and then perform reranking using a greedy algorithm with MST.

### 5.1 Hardness Analysis

Given a query column $C_q$, an initial candidate pool $C_B$ returned by a fast similarity search 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 use a pairwise affinity score $w(\cdot, \cdot)$ over columns, where $w(C_i, C_j) \in \mathbb{R}^+$ reflects their joinability and semantic similarity. A coherent result set should be both relevant to the query and well-connected internally. A natural way to capture this trade-off is to score a size-$K$ set by the following objective:

$$\mathcal{R}^* = \underset{\mathcal{R}\subseteq C_B, |\mathcal{R}|=K}{\arg\max}\; G(\mathcal{R}), \tag{21}$$

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

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

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.

Selecting the optimal $K$-column subset under Eq. 21 is a combinatorial optimization problem. The following theorem presents the hardness analysis for solving such problem. The proof is in the online full version [28].

**Theorem 4** (NP-hardness of Coherence-Aware Selection)**.** *Maximizing the objective $G(\mathcal{R})$ in Eq. 21 under the cardinality constraint $|\mathcal{R}| = K$ is NP-hard when Coherence($\mathcal{R}$) is instantiated as the weight of a maximum spanning tree rooted at $C_q$ on the induced subgraph over $\{C_q\} \cup \mathcal{R}$.*

## 5.2 Global Coherent Search

Given a query column $C_q$, the offline phase provides a structure-aware embedding $h_C$ for every column $C$ in the data lake. At query time, we aim to solve the coherence-aware top-$K$ selection objective in Eq. 21. Theorem 4 shows that selecting the best size-$K$ subset is NP-hard. We therefore adopt a two-stage framework that balances scalability and effectiveness:(1) retrieve a manageable candidate pool and build a weighted candidate graph; and (2) perform MST-based greedy reranking to return a globally coherent top-$K$ set.

**Candidate Graph Construction.** Given a query column $C_q$, the data lake contains $N$ total columns. To avoid the intractable search space $\binom{N}{K}$, we first retrieve a manageable candidate pool of size $B$ (where $K < B \ll N$) via efficient similarity search, and then perform coherence-aware optimization only within this pool, yielding a candidate set $C_B$.

Definition 3 (Candidate Graph). *Given a query column $C_q$ and a candidate set $C_B$ obtained via similarity search (typically $|C_B| = B > K$), 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 $w : \mathcal{E}_C \to \mathbb{R}^+$ assigns each edge a positive weight:*

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

*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, $C_B = \{C_1, C_2, \ldots, C_B\}$, 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 $w(C_q, C_i) = \text{sim}(h_{C_q}, h_{C_i})$ 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 $w(C_i, C_j) = \text{sim}(h_{C_i}, h_{C_j})$ reflecting the joinability and semantic similarity between the two candidates. This stage reduces the search space from $\binom{N}{K}$ to $\binom{B}{K}$ by restricting selection to the top-$B$ candidate while preserving high recall.

**MST-based Greedy Reranking.** With the candidate graph $\mathcal{G}_C$ built, the core challenge is to efficiently select a set $\mathcal{R} \subseteq C_B$ that maximizes the objective $G(\mathcal{R})$ in Eq. 21. This requires instantiating the coherence term, Coherence($\mathcal{R}$), which quantifies how well the selected columns can be integrated for joint analysis. A seemingly straightforward way to enforce coherence is to require the returned columns to form a dense clique, i.e., every pair is strongly joinable. This assumption is overly restrictive for real data lakes where join graphs are typically sparse and multi-hop joins through bridge tables are common.

Instead, coherence should ensure that every column in $\mathcal{R}$ is reachable from the query $C_q$ through a sequence of strong join relationships. This notion is naturally quantified by the weight of a *maximum spanning tree* (MST) constructed on the induced subgraph over $C_q \cup \mathcal{R}$. The MST extracts the set of strongest possible edges that connect all columns without cycles, ensuring reachability from the query anchor to every result while accommodates multi-hop joins common in real data lakes, and avoids the over-strictness of dense subgraph measures. Formally, for a vertex set $\mathcal{V}' = C_q \cup \mathcal{R}$, let $\mathcal{G}_C[\mathcal{V}']$ denote the subgraph induced by $\mathcal{V}'$ in $\mathcal{G}_C$. The coherence of $\mathcal{R}$ is defined as the weight of a MST of $\mathcal{G}_C[\mathcal{V}']$:

$$\text{Coherence}(\mathcal{R}) = \max_{T \in \mathcal{S}(\mathcal{V}')} \sum_{(u,v) \in E(T)} w(u, v), \tag{24}$$

where $\mathcal{S}(\mathcal{V}')$ is the set of all spanning trees of $\mathcal{G}_C[\mathcal{V}']$, and $E(T)$ denotes the edge set of tree $T$.

Evaluating $G(\mathcal{R})$ by explicitly constructing an MST for each candidate subset $\mathcal{R}$ is computationally prohibitive for online search. We therefore propose a greedy algorithm that cleverly combines the logic of Prim's MST construction with subset selection, avoiding the need to recompute an MST from scratch at each iteration. The algorithm incrementally builds the result set $\mathcal{R}$ by simulating the growth of an MST *rooted at the query $C_q$* as follows:

- **Initialization:** Start with $\mathcal{R} = \emptyset$ and define the current connected component is $S = \{C_q\}$. Since $S$ initially contains only the query node, the coherence gain term for any candidate $C$ is simply $w(C, C_q)$; hence the first selected node is the one with the highest query relevance.
- **Selection rule:** At each step, for every candidate $C \in C_B \setminus \mathcal{R}$, compute the marginal gain

$$\Delta(C \mid \mathcal{R}) = w(C_q, C) + \lambda \cdot \max_{C' \in S} w(C, C'), \tag{25}$$

  where $S = \{C_q\} \cup \mathcal{R}$. The first term captures the relevance to the query, while the second term measures the strongest possible connection between $C$ and the current component $S$—this mirrors the cut step in Prim's algorithm, which attaches a new node via the heaviest edge crossing to the growing tree. The algorithm then selects the candidate

$$C^* = \arg \max_{C \in C_B \setminus \mathcal{R}} \Delta(C \mid \mathcal{R}).$$

- **Maintenance:** Update the result set $\mathcal{R} \leftarrow \mathcal{R} \cup \{C^*\}$ and the connected component $S \leftarrow S \cup \{C^*\}$. This incremental expansion ensures that each newly added node is linked to the existing structure by the strongest available edge, thereby progressively building a high-weight connectivity backbone. Repeat the selection and maintenance steps until $|\mathcal{R}| = K$.

**Algorithm 1:** Global Coherent Search

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

// **Phase 1: Candidate Graph Construction**
1 $C_B \leftarrow$ Top-$B$ similarity search using $h_{C_q}$
2 Construct candidate graph $\mathcal{G}_C = (\{C_q\} \cup C_B, E_C, w)$
3 Compute edge weights $w(C_i, C_j) \leftarrow \text{sim}(h_{C_i}, h_{C_j})$ for edges
// **Phase 2: MST-based Greedy Reranking**
4 $\mathcal{R} \leftarrow \emptyset$
5 $U \leftarrow C_B$ // unselected candidates
6 **foreach** $C \in U$ **do**
7 $\text{rel}[C] \leftarrow w(C_q, C)$
8 $\text{key}[C] \leftarrow w(C_q, C)$ // best attachment to $S = \{C_q\}$
9 **for** $i = 1, \ldots, K$ **do**
// Select candidate maximizing marginal gain
10 $C^* \leftarrow \arg\max_{C \in U} \left(\text{rel}[C] + \lambda \cdot \text{key}[C]\right)$
11 $\mathcal{R} \leftarrow \mathcal{R} \cup \{C^*\}$
12 $U \leftarrow U \setminus \{C^*\}$
// Update keys for remaining candidates
13 **foreach** $C \in U$ **do**
14 **if** $w(C, C^*) > \text{key}[C]$ **then**
15 $\text{key}[C] \leftarrow w(C, C^*)$
16 **return** $\mathcal{R}$

This iterative process ensures that the final set $\mathcal{R}$ is connected to $C_q$ via a tree composed of the strongest edges chosen at each step. The sequence of max operations implicitly constructs a connectivity backbone rooted at $C_q$ that approximates the MST gain; the exact MST-based coherence of the final set can be obtained by running a MST algorithm on the induced subgraph $\mathcal{G}_C[\{C_q\} \cup \mathcal{R}]$ if needed, without requiring recomputation for every intermediate subset during the online search.

**Algorithm Description.** The complete global coherent search procedure is summarized in Algorithm 1, which integrates both candidate graph construction and MST-based reranking. Given a query column $C_q$, the algorithm first retrieves the Top-$B$ most relevant candidates via similarity search and constructs a weighted candidate graph $\mathcal{G}_C$ (lines 1–3). In the reranking phase (lines 4–16), the algorithm grows a result set $\mathcal{R}$ by iteratively selecting the candidate with the highest marginal gain, as defined in Eq. 25. This gain combines the candidate's relevance to $C_q$ with its strongest connection to the current tree (including $C_q$ itself), thereby simulating a Prim-style expansion of a maximum spanning tree rooted at the query. The process continues until $K$ columns are selected, yielding a coherent result set optimized for both relevance and internal joinability.

**Approximation Guarantee.** We provide a theoretical justification for the greedy reranking algorithm. We denote by $\Delta(C \mid \mathcal{R})$ the surrogate marginal gain defined in Eq. 25. We prove that the greedy rule in Eq. 25 optimizes a tractable surrogate whose marginal gain provably lower-bounds the true marginal gain contributed by the MST-based coherence term. The proof is in the online full version [28].

**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 |

**Lemma 1** (MST Gain Lower Bound). *For any $S \ni C_q$ and $v \notin S$, let $MST(S)$ be the MaxST weight on $\mathcal{G}_C[S]$. Then*

$$MST(S \cup \{v\}) - MST(S) \;\geq\; \max_{u \in S} w(v, u).$$

**Theorem 5** (Surrogate Marginal Gain Property). *Let $S = \{C_q\} \cup \mathcal{R}$ and let $C^*$ be the candidate selected by maximizing the surrogate marginal gain $\Delta(C \mid \mathcal{R})$ in Eq. 25. Then*

$$G(\mathcal{R} \cup \{C^*\}) - G(\mathcal{R}) \;\geq\; \Delta(C^* \mid \mathcal{R}).$$

*Moreover, if $u_t \in \arg\max_{u \in S_{t-1}} w(C^{(t)}, u)$, then $\{(C^{(t)}, u_t)\}_{t=1}^{K}$ forms a spanning tree over $\{C_q\} \cup \mathcal{R}$ and*

$$Coherence(\mathcal{R}) = MST(\{C_q\} \cup \mathcal{R}) \;\geq\; \sum_{t=1}^{K} w(C^{(t)}, u_t).$$

This analysis shows Eq. 25 is a conservative estimate of the true marginal gain in $G(\cdot)$: at each iteration, the objective increases by at least the computed surrogate gain. Moreover, the maintained attachment form a spanning tree over the final vertex set, whose total weight is lower-bounded by the MST-based coherence.

**Complexity Analysis.** The per-query time complexity is dominated by two components: candidate retrieval and graph construction cost $O(Nd + N \log B + B^2 d)$, and greedy reranking cost $O(BK)$. Specifically, Stage 1 retrieves $B$ candidates via similarity search over $N$ columns ($O(Nd + N \log B)$) and constructs a candidate graph with $O(B^2)$ edges whose weights are computed in $O(B^2 d)$ time. Stage 2 maintains for each candidate its best connection (key) to the current tree; each of the $K$ iterations involves selecting the candidate with maximum marginal gain in $O(B)$ time and updating the keys of the remaining candidates in $O(B)$ time, yielding an $O(BK)$ total cost. The overall complexity is $O(Nd + N \log B + B^2 d + BK)$, and the space complexity is $O(B^2)$ to store the candidate graph's pairwise weights, which is efficient for typical parameters.

# 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 proir works [5, 15, 16, 37, 46, 49, 50]. Table 2 reports their statistics.

**Baselines.** We compare the following methods. (1) JOSIE [49]: Token-based set similarity search using inverted index and posting lists with cost-model-based greedy merge algorithm. (2) LSH Ensemble [50]: MinHash-based containment search using Locality Sensitive Hashing with dynamic partitioning. (3) DeepJoin [16]: Pretrained transformer model for column embedding. (4) Snoopy [19]: State-of-the-art column embedding model that uses proxy columns to improve semantic join discovery. (5) Omnimatch [25]: Multi-relational graph neural network that combines multiple similarity signals for table matching and joinability discovery.

**Metrics.** Following prior work [13], 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:

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

$$\text{Recall@K} = \frac{1}{|Q|} \sum_{q \in Q} \frac{|\mathcal{R}_K(q) \cap \mathcal{R}^*(q)|}{|\mathcal{R}^*(q)|}. \tag{27}$$

We report results for $K \in \{5, 15, 25\}$ to assess performance at 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 ~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. 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 $4 \times 10^{-4}$, batch size 64, and train for 30 epochs. We use triplet loss with margin 1.0 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 global coherent reranking, we set candidate size $B = 50$ and coherence weight $\lambda = 1.0$. 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. Overall, HyperJoin achieves the best Precision@K and Recall@K across *WebTable*, *USA*, *CAN* and *UK_SG*. As shown in the table, LSH performs best among traditional baselines and PLM-based baseline generally demonstrate superior performance compared to traditional methods. Among the PLM-based models, Snoopy yields the best results. Notably, HyperJoin further outperforms Snoopy, achieving the highest overall accuracy. Specifically, at $K = 15$, HyperJoin improves P@15 by 21.4 points and R@15 by 17.2 points on average compared to Snoopy. The improvements are most pronounced on government-derived benchmarks, where HyperJoin exceeds Snoopy by margins of up to 28.7 points in P@15 (on UK_SG). This stems from the inherent properties of government data, which are characterized by strong structural and semantic regularities such as consistent column naming, rich metadata, and clear domain hierarchies. HyperJoin's LLM-augmented hypergraph and hierarchical message passing are explicitly designed to capture these precise forms of contextual signals. In contrast, WebTable consists predominantly of numerical content where such semantic signals are inherently less frequent, yet HyperJoin still achieves substantial gains by effectively utilizing the available structural and metadata context. These consistent margins demonstrate that HyperJoin provides not only higher accuracy but also stronger generalization across heterogeneous datasets. Compared to traditional set-overlap methods (JOSIE, LSH Ensemble), HyperJoin demonstrates even larger improvements, with average R@25 gains exceeding 40% across 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 P@5 of 87.7% and improves over Snoopy by 10.5 points in P@5 and 3.0 points in R@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 15.1 points in P@25 and 19.5 points in R@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 +6.4 points at $K = 5$ to +34.8 points at $K = 25$. Overall, the results suggest that HyperJoin is competitive even at small $K$, while its gains are larger at higher $K$. This shows jointly modeling relevance and coherence becomes increasingly beneficial as broader discovery makes relevance alone insufficient.

**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 coherent reranking (CR) by comparing the full model with and without CR. CR improves performance consistently across datasets, with average gains of +4.1 P@15. The largest benefit is on CAN (P@15: 75.56→81.78, +6.22; R@15: 51.73→55.89, +4.16), while the improvement remains clear on USA (+4.66 P@15, +3.22 R@15), suggesting that set-level coherence suppresses 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. The most severe drop occurs 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; R@15: 75.23→53.19) and is also notable on USA (P@15: -18.34; R@15: -12.86), indicating that structural information is crucial when global regularities are strong. 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.

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

| Method | Webtable | | | | | | USA | | | | | | CAN | | | | | | 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 | 92.0 | 95.3 | 83.6 | 21.3 | 66.4 | 97.1 | 83.3 | 81.8 | 65.9 | 18.9 | 55.9 | 74.2 | 94.0 | 89.0 | 70.2 | 27.2 | 77.1 | 98.8 |

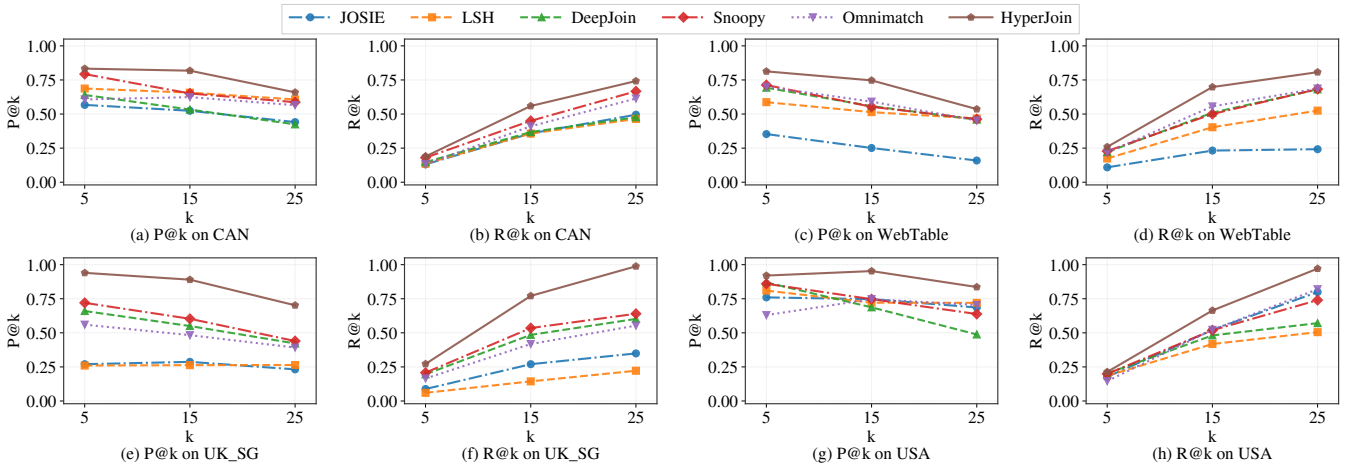


**Figure 4: Performance comparison across different datasets and k values.**

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

| Variant | Webtable | | USA | | CAN | | UK_SG | |
|---|---|---|---|---|---|---|---|---|
| | P@15 | R@15 | P@15 | R@15 | P@15 | R@15 | P@15 | R@15 |
| Full | **74.67** | **69.75** | **95.33** | **66.42** | **81.78** | **55.89** | **89.00** | **77.08** |
| w/o CR | 71.33 | 67.09 | 90.67 | 63.20 | 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 |

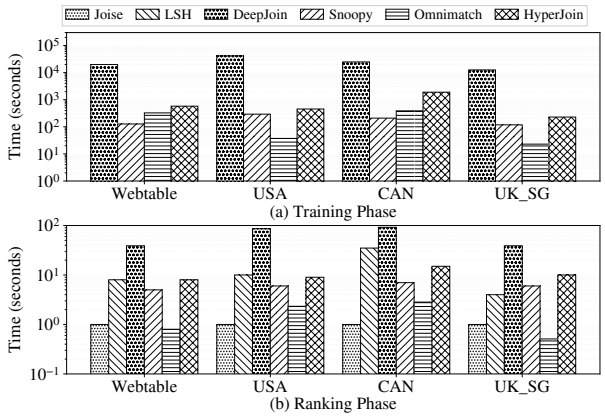


**Figure 5: Training and evaluation efficiency across datasets.**

**Exp-3: Efficiency Evaluation.** Figure 5 compares 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). This is expected since DeepJoin performs model fine-tuning (based on DistilBERT), which introduces a significantly higher training cost than lightweight training objectives. In addition, OmniMatch incurs a substantial offline overhead for constructing column similarity graphs due to extensive pairwise column operations; in our setting, this preprocessing alone takes on the order of hours (e.g., 30.94 hours on USA and 43.37 hours on CAN). For the ranking 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. Note that the reported DeepJoin time is measured end-to-end and includes rebuilding the target-side index for each run, rather than only the online top-$k$ retrieval step.

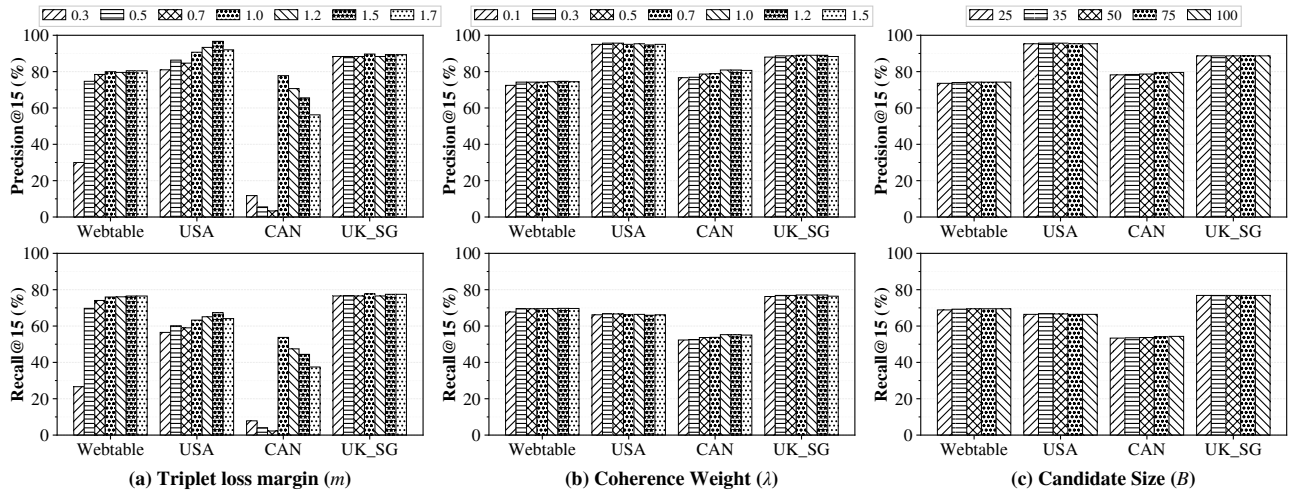


**Figure 6: Hyperparameter sensitivity analysis on datasets. (a) Triplet loss margin $m$, (b) Global Cohenrent Seaech coherence weight $\lambda$, and (c) Global Cohenrent Seaech candidate size $B$.**

## 6.3 Robustness Evaluation

**Exp-4: Triplet Loss Margin $m$.** We evaluate triplet loss margins $m \in \{0.3, 0.5, 0.7, 1.0, 1.2, 1.5, 1.7\}$ across four datasets. As shown in Figure 6(a), the triplet loss margin is critical. Too small margins can trigger severe degradation or even collapse, particularly on CAN and Webtable (e.g., CAN drops to 3.33% P@15 / 2.25% R@15 at $m = 0.7$ and Webtable to 30.00% P@15 / 26.63% R@15 at $m = 0.3$). Performance becomes stable at $m = 1.0$, yielding consistently reliable results across datasets and avoiding catastrophic failures. Larger margins can further benefit some corpora (e.g., $m = 1.5$ performs best on USA at 96.67% / 67.36% and also improves Webtable to 80.44% / 76.52%), but may slightly hurt CAN and brings only minor changes on UK_SG (peak at $m = 1.0$ with 89.67% / 77.71%). Overall, we use $m = 1.0$ as a robust default in all experiments.

**Exp-5: Coherence Weight $\lambda$.** We evaluate $\lambda \in \{0.1, 0.3, 0.5, 0.7, 1.0, 1.2, 1.5\}$ across four datasets. Figure 6(b) indicates that dataset-dependent optima, but a stable plateau around $\lambda \in [0.7, 1.2]$. Within this range, UK_SG and CAN generally perform best, while Webtable improves steadily up to $\lambda = 1.2$ (74.67% P@15 / 69.74% R@15) and then changes little. In contrast, USA prefers smaller $\lambda$ (best around $\lambda = 0.3$–$0.5$ with 95.67% / 66.66%) and degrades mildly for larger values, suggesting over-regularization by coherence. CAN gains substantially from coherence and peaks at $\lambda = 1.0$–$1.2$ (80.89% P@15 and 55.26% R@15 at $\lambda = 1.0$), followed by a slight decline. Overall, we set $\lambda = 1.0$ as a robust default, delivering near-optimal results on three datasets while remaining competitive on USA.

**Exp-6: Candidate Size $B$.** We vary the candidate pool sizes $B \in \{25, 35, 50, 75, 100\}$ for Global Cohenrent Seaech. As shown in Figure 6(c), performance is largely stable w.r.t. $B$, with only minor fluctuations across datasets (max variation $\leq$1.34 in P@15 and $\leq$0.94 in R@15). In particular, results quickly saturate at moderate pool sizes: UK_SG is unchanged across all $B$, and Webtable/USA plateau around $B \approx 35$–$50$. CAN shows a mild monotonic improvement with larger pools, but the gains beyond $B = 50$ are marginal. Overall, we set $B = 50$ as the default, balancing near-optimal quality and computational cost.

# 7 RELATED WORK

Existing methods for joinable table discovery [6, 8, 13, 15, 17, 21] can be broadly classified into traditional set-overlap methods and deep learning-based methods. Traditional approaches, such as Jaccard similarity [18], containment coefficients [11, 45, 48], and LSH [50], 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, such as DeepJoin [16] and Snoopy [19], employ transformers (e.g., BERT) to encode column values into semantic embeddings. Graph neural network approaches, such as OmniMatch [25], model table repositories as multi-relational graphs and use RGCN [34] to learn structure-aware column embeddings. To the best of our knowledge, HyperJoin is the first hypergraph-based framework for joinable table discovery. It models both intra-table and inter-table structural interactions in a unified representation learning phase, and further introduces a MST-based global coherent reranking strategy to return globally consistent join candidates.

# 8 CONCLUSION

In this paper, we address the problem of joinable table discovery in data lakes and propose HyperJoin, a hypergraph-based framework. HyperJoin follows a two-phase design with offline representation learning and an online ranking. In the offline phase, we propose an LLM-augmented hypergraph that models both intra-table structure and inter-table joinability via dual-type hyperedges, and learns expressive column representations with HIN. In the online phase, we introduce efficient global coherent reranking module for a globally coherent result set, balancing scalability and effectiveness. Extensive experiments on four public benchmarks show that HyperJoin consistently outperforms state-of-the-art methods, achieving superior Precision and Recall overall, while returning more coherent and semantically relevant join candidates.

# REFERENCES


[1] [n.d.]. Government Open Data Portals (USA, CAN, UK, SG). https://www.data.gov/; https://open.canada.ca/; https://data.gov.uk/; https://data.gov.sg/.

[2] [n.d.]. WebTable. https://webdatacommons.org/webtables/.

[3] Naheed Anjum Arafat, Arijit Khan, Arpit Kumar Rai, and Bishwamittra Ghosh. 2023. Neighborhood-based Hypergraph Core Decomposition. *Proceedings of the VLDB Endowment* 16, 9 (2023), 2061–2074.

[4] Adrián Arnaiz-Rodríguez, Ahmed Begga, Francisco Escolano, and Nuria Oliver. 2022. Diffwire: Inductive graph rewiring via the lov\'asz bound. *arXiv preprint arXiv:2206.07369* (2022).

[5] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. Tabel: Entity linking in web tables. In *International Semantic Web Conference*. Springer, 425–441.

[6] Alex Bogatu, Alvaro AA Fernandes, Norman W Paton, and Nikolaos Konstantinou. 2020. Dataset discovery in data lakes. In *2020 ieee 36th international conference on data engineering (icde)*. IEEE, 709–720.

[7] Chengliang Chai, Lei Cao, Guoliang Li, Jian Li, Yuyu Luo, and Samuel Madden. 2020. Human-in-the-loop outlier detection. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 19–33.

[8] Tianji Cong, Fatemeh Nargesian, and HV Jagadish. 2023. Pylon: Semantic table union search in data lakes. *arXiv preprint arXiv:2301.04901* (2023).

[9] Andreea Deac, Marc Lackenby, and Petar Veličković. 2022. Expander graph propagation. In *Learning on Graphs Conference*. PMLR, 38–1.

[10] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, Sibo Wang, Michael Stonebraker, Ahmed K Elmagarmid, Ihab F Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. 2017. The Data Civilizer System.. In *Cidr*.

[11] Dong Deng, Albert Kim, Samuel Madden, and Michael Stonebraker. 2017. SILKMOTH: An Efficient Method for Finding Related Sets with Maximum Matching Constraints. *Proceedings of the VLDB Endowment* 10, 10 (2017).

[12] Yuhao Deng, Chengliang Chai, Lei Cao, Nan Tang, Jiayi Wang, Ju Fan, Ye Yuan, and Guoren Wang. 2024. Misdetect: Iterative mislabel detection using early loss. Association for Computing Machinery (ACM).

[13] Yuhao Deng, Chengliang Chai, Lei Cao, Qin Yuan, Siyuan Chen, Yanrui 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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).

[20] Xiaoxin He, Bryan Hooi, Thomas Laurent, Adam Perold, Yann LeCun, and Xavier Bresson. 2023. A generalization of vit/mlp-mixer to graphs. In *International conference on machine learning*. PMLR, 12724–12745.

[21] Xuming Hu, Shen Wang, Xiao Qin, Chuan Lei, Zhengyuan Shen, Christos Faloutsos, Asterios Katsifodimos, George Karypis, Lijie Wen, and Philip S Yu. 2023. Automatic table union search with tabular representation learning. In *Findings of the Association for Computational Linguistics: ACL 2023*. 3786–3800.

[22] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[23] Aamod Khatiwada, Roee Shraga, Wolfgang Gatterbauer, and Renée J Miller. 2022. Integrating data lake tables. *Proceedings of the VLDB Endowment* 16, 4 (2022), 932–945.

[24] Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 39–48.

[25] Christos Koutras, Jiani 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.

[26] 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).

[27] Geon Lee, Jihoon Ko, and Kijung Shin. 2020. Hypergraph motifs: concepts, algorithms, and discoveries. *VLDB* 13, 12 (2020), 2256–2269.

[28] Shiyuan Liu, Jianwei Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2026. HyperJoin: Full Version. https://github.com/T-Lab/HyperJoin/tree/main/full_version.

[29] Jose Lugo-Martinez, Daniel Zeiberg, Thomas Gaudelet, Noël Malod-Dognin, Natasa Przulj, and Predrag Radivojac. 2021. Classification in biological networks with hypergraphlet kernels. *Bioinformatics* 37, 7 (2021), 1000–1007.

[30] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[31] Fatemeh Nargesian, Erkang Zhu, Renée J Miller, Ken Q Pu, and Patricia C Arocena. 2019. Data lake management: challenges and opportunities. *Proceedings of the VLDB Endowment* 12, 12 (2019), 1986–1989.

[32] Alec Radford. 2018. Improving language understanding by generative pre-training. *Preprint* (2018).

[33] 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).

[34] 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.

[35] 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.

[36] 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.

[37] Fei Wang, Kexuan Sun, Muhao 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.

[38] Jianwei Wang, Kai Wang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2024. Efficient unsupervised community search with pre-trained graph transformer. *arXiv preprint arXiv:2403.18869* (2024).

[39] Jianwei Wang, Kai Wang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2024. Neural attributed community search at billion scale. *Proceedings of the ACM on Management of Data* 1, 4 (2024), 1–25.

[40] Jianwei Wang, Kai Wang, Ying Zhang, Wenjie Zhang, Xiwei Xu, and Xuemin Lin. 2025. On llm-enhanced mixed-type data imputation with high-order message passing. *arXiv preprint arXiv:2501.02191* (2025).

[41] Jianwei Wang, Ying Zhang, Kai Wang, Xuemin Lin, and Wenjie Zhang. 2024. Missing data imputation with uncertainty-driven network. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–25.

[42] Yiqi Wang, Long Yuan, Wenjie Zhang, Zi Chen, Xuemin Lin, and Qing Liu. 2024. Simpler is More: Efficient Top-K Nearest Neighbors Search on Large Road Networks. *Proceedings of the VLDB Endowment* 17, 13 (2024), 4683–4695.

[43] 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.

[44] Hanrui Wu, Yuguang Yan, and Michael Kwok-Po Ng. 2022. Hypergraph collaborative network on vertices and hyperedges. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 3 (2022), 3245–3258.

[45] 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.

[46] 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.

[47] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2023. Large language models as data preprocessors. *arXiv:2308.16361* (2023).

[48] 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.

[49] 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.

[50] 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

## 9.1 Overall forward process of HIN.

Algorithm 2 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.

## 9.2 Proofs of Theoretical Results

This section collects proofs deferred from the main paper. For readability, we group proofs by the section where the corresponding theorem appears.

### 9.2.1 Proof of Theorem 1.

*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:

$$\begin{aligned} \mathcal{F}_{\text{single}}(i, j) &= \{\mathbf{x}_i, \mathbf{x}_j\}, \\ \mathcal{F}_{\text{multi}}(i, j) &= \Big\{\mathbf{x}_i, \mathbf{x}_j,\ \mathcal{N}^{\text{intra}}(v_i), \mathcal{N}^{\text{intra}}(v_j), \\ &\qquad \mathcal{N}^{\text{inter}}(v_i), \mathcal{N}^{\text{inter}}(v_j)\Big\}. \end{aligned} \tag{28}$$

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

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

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 $\mathcal{F}_{multi}$ and $\mathcal{F}_{single}$ defined above, the Bayes-optimal risk satisfies*

$$\Psi^*(\mathcal{F}_{multi}) \;\le\; \Psi^*(\mathcal{F}_{single}).$$

*Moreover, if the added contexts are informative in the sense that $\mathbb{P}(Y_{ij} = 1 \mid \mathcal{F}_{multi}) \neq \mathbb{P}(Y_{ij} = 1 \mid \mathcal{F}_{single})$ 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,

$$\begin{aligned} \Psi^*(\mathcal{F}_{\text{multi}}) &= \inf_{g_{\text{multi}}} \mathbb{E}\big[\ell\big(g_{\text{multi}}(\mathcal{F}_{\text{multi}}), Y_{ij}\big)\big] \\ &\le \inf_{g_{\text{single}}} \mathbb{E}\big[\ell\big(g_{\text{single}}(\mathcal{F}_{\text{single}}), Y_{ij}\big)\big] = \Psi^*(\mathcal{F}_{\text{single}}). \end{aligned} \tag{29}$$

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. □

**Algorithm 2:** Hierarchical Interaction Network (HIN)

**Input:** Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathbf{X}^v, \mathbf{X}^e, \mathbf{\Pi})$, Table IDs, Joinable pairs $\mathcal{P}_{\text{join}}$
**Output:** Final column embeddings $\{\mathbf{h}_v^{\text{final}}\}_{v \in \mathcal{V}}$

**// Stage 1: Positional Encoding**
1 Precompute Laplacian eigenvectors (offline):
2 $\quad \mathbf{A} \leftarrow \text{BuildAdjacencyMatrix}(\mathcal{P}_{\text{join}})$
3 $\quad \mathbf{L}_{\text{norm}} = \mathbf{I}_N - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$
4 $\quad \mathbf{V}_{\text{pe}} \leftarrow \text{EigenVectors}(\mathbf{L}_{\text{norm}}, k = 16)$
5 Inject table-level and column-level positional encodings:
6 $\quad$ **for** *each column/node $v$ with table ID $t$ and index $i$* **do**
7 $\quad\quad \mathbf{h}_v^{(0)} \leftarrow \mathbf{x}_v^v + \alpha \cdot \mathbf{E}_{\text{tbl}}[t] + \beta \cdot \text{MLP}_{\text{pe}}(\mathbf{V}_{\text{pe}}[i, :])$

**// Stage 2: Local Hyperedge Aggregation**
8 $\mathbf{H} \leftarrow [\mathbf{h}_{v_1}^{(0)}, \ldots, \mathbf{h}_{v_N}^{(0)}]^T$
9 **for** $\ell = 1$ *to* $L = 2$ **do**
10 $\quad \mathbf{H} \leftarrow \text{LayerNorm}(\text{ReLU}(\mathbf{H}\mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)}))$
11 Aggregate to hyperedge level with domain-specific transforms:
12 $\quad \mathbf{d}_e \leftarrow \mathbf{\Pi}^T \mathbf{1}_N$ `// Hyperedge degrees`
13 $\quad \mathbf{D}_e \leftarrow \text{Diag}(\mathbf{d}_e)$
14 $\quad \mathbf{X}^e \leftarrow \mathbf{D}_e^{-1}\mathbf{\Pi}^T\mathbf{H}$
15 $\quad \mathbf{Z}^{(0)} \leftarrow \mathbf{X}^e$ `// mixer input tokens`
16 $\quad$ Partition $\mathbf{Z}^{(0)}$ into $\mathbf{Z}_{\text{inter}}^{(0)}$ and $\mathbf{Z}_{\text{intra}}^{(0)}$
17 $\quad \tilde{\mathbf{Z}}_{\text{inter}} \leftarrow \mathbf{Z}_{\text{inter}}^{(0)}\mathbf{W}_{\text{inter}}$
18 $\quad \tilde{\mathbf{Z}}_{\text{intra}} \leftarrow \mathbf{Z}_{\text{intra}}^{(0)}\mathbf{W}_{\text{intra}}$
19 $\quad \tilde{\mathbf{Z}} \leftarrow [\tilde{\mathbf{Z}}_{\text{inter}}; \tilde{\mathbf{Z}}_{\text{intra}}]$

**// Stage 3: Global Hyperedge Mixing**
20 $\mathbf{A}_{\text{hyperedge}} \leftarrow \text{RowNormalize}(\mathbf{\Pi}^T\mathbf{\Pi} - \text{diag}(\mathbf{\Pi}^T\mathbf{\Pi}))$
21 **for** $\ell = 1$ *to* $L_{mixer} = 2$ **do**
$\quad$ `// Hyperedge Interaction with structure bias`
22 $\quad \mathbf{Q}, \mathbf{K}, \mathbf{V} \leftarrow \text{LayerNorm}(\tilde{\mathbf{Z}})[\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V]$
23 $\quad \mathbf{Z}_{\text{HI}} \leftarrow$
$\quad\quad \tilde{\mathbf{Z}} + \text{MultiHead}\left(\text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \lambda\mathbf{A}_{\text{hyperedge}}\right)\mathbf{V}\right)\mathbf{W}_O$
$\quad$ `// Hyperedge Refinement`
24 $\quad \tilde{\mathbf{Z}} \leftarrow \mathbf{Z}_{\text{HI}} + \mathbf{W}_2\text{GELU}(\mathbf{W}_1\text{LayerNorm}(\mathbf{Z}_{\text{HI}})^T)^T$
25 $\mathbf{Z}_{\text{out}} \leftarrow \tilde{\mathbf{Z}}$ `// final hyperedge embeddings`
26 Let $\mathbf{z}_{e_j}$ denote the $j$-th row of $\mathbf{Z}_{\text{out}}$.
**// Propagate hyperedge back to columns**
27 **for** $i = 1$ *to* $N$ **do**
28 $\quad \mathcal{N}(v_i) \leftarrow \{j \mid \mathbf{\Pi}_{ij} = 1\}$
29 $\quad \mathbf{h}_{v_i}^{\text{hyperedge}} \leftarrow \frac{1}{|\mathcal{N}(v_i)|}\sum_{j \in \mathcal{N}(v_i)} \mathbf{W}_{\text{h2c}}\mathbf{z}_{e_j}$
30 $\quad \mathbf{h}_{v_i}^{\text{final}} \leftarrow \text{L2Norm}(\text{LayerNorm}(\mathbf{h}_{v_i} + \mathbf{h}_{v_i}^{\text{hyperedge}}))$
31 **return** $\{\mathbf{h}_v^{\text{final}}\}_{v \in \mathcal{V}}$

### 9.2.2 Proof of Theorem 2.

**Theorem 2.** *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 $\mathcal{F}_{single}$ (i.e., column-wise encoding without using $\mathcal{N}^{intra}$ or $\mathcal{N}^{inter}$).*

Proof. This can be achieved by choosing $\mathcal{E}$ as $N$ singleton hyperedges $\mathcal{E} = \{\{v_1\}, \ldots, \{v_N\}\}$ (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 $\mathcal{N}^{\text{intra}}/\mathcal{N}^{\text{inter}}$. □

#### 9.2.3 Proof of Theorem 3.

**Theorem 3** (HIN is strictly more expressive than 1-WL message passing)**.** *Let $\mathcal{H}_{base}$ 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 $\mathcal{H}_{pe}$ be the class obtained by augmenting inputs with*

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

*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 $\mathcal{H}_{base} \subsetneq \mathcal{H}_{pe}$.*

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

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

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, $\mathcal{H}_{\text{base}} \subsetneq \mathcal{H}_{\text{pe}}$. □

#### 9.2.4 Proof of Theorem 4.

**Theorem 4** (NP-hardness of Coherence-Aware Selection)**.** *Maximizing the objective $G(\mathcal{R})$ in Eq. 21 under the cardinality constraint $|\mathcal{R}| = K$ is NP-hard when Coherence($\mathcal{R}$) is instantiated as the weight of a maximum spanning tree rooted at $C_q$ on the induced subgraph over $\{C_q\} \cup \mathcal{R}$.*

Proof. We prove NP-hardness by reduction from the 0–1 Knapsack problem. Given a knapsack instance with items $\{1, \ldots, n\}$, 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 $\{u_{i,1}, \ldots, u_{i,w_i}\}$. We add an edge between $C_q$ and $u_{i,1}$ with weight 0. For each $j = 1, \ldots, w_i - 2$, 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 $\{d_1, \ldots, d_W\}$ 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 $\{u_{i,1}, \ldots, u_{i,w_i-1}\}$. 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. □

#### 9.2.5 Proof of Theorem 5.

**Lemma 1** (MST Gain Lower Bound)**.** *For any $S \ni C_q$ and $v \notin S$, let $MST(S)$ be the MaxST weight on $\mathcal{G}_C[S]$. Then*

$$MST(S \cup \{v\}) - MST(S) \;\geq\; \max_{u \in S} w(v, u).$$

Proof. Let $T$ be a MaxST of $\mathcal{G}_C[S]$ and $u^* \in \arg\max_{u \in S} w(v, u)$. Then $T \cup \{(v, u^*)\}$ is a spanning tree of $\mathcal{G}_C[S \cup \{v\}]$, so $\text{MST}(S \cup \{v\}) \geq \text{MST}(S) + \max_{u \in S} w(v, u)$. □

**Theorem 5** (Surrogate Marginal Gain Property)**.** *Let $S = \{C_q\} \cup \mathcal{R}$ and let $C^*$ be the candidate selected by maximizing the surrogate marginal gain $\Delta(C \mid \mathcal{R})$ in Eq. 25. Then*

$$G(\mathcal{R} \cup \{C^*\}) - G(\mathcal{R}) \;\geq\; \Delta(C^* \mid \mathcal{R}).$$

*Moreover, if* $u_t \in \arg\max_{u \in S_{t-1}} w(C^{(t)}, u)$, *then* $\{(C^{(t)}, u_t)\}_{t=1}^{K}$ *forms a spanning tree over* $\{C_q\} \cup \mathcal{R}$ *and*

$$\mathit{Coherence}(\mathcal{R}) = \mathit{MST}(\{C_q\} \cup \mathcal{R}) \;\geq\; \sum_{t=1}^{K} w(C^{(t)}, u_t).$$

Proof.

$$G(\mathcal{R} \cup \{C^*\}) - G(\mathcal{R}) = w(C_q, C^*) + \lambda\big(\mathrm{MST}(S \cup \{C^*\}) - \mathrm{MST}(S)\big)$$

and Lemma 1 gives $\mathrm{MST}(S \cup \{C^*\}) - \mathrm{MST}(S) \geq \max_{u \in S} w(C^*, u)$, hence the first claim. The second claim follows since each iteration adds one new vertex and one attachment edge to the connected component rooted at $C_q$, and the MST maximizes weight over all spanning trees. □