Graph Neural Networks for Databases: A Survey

Ziming Li[1] , **Youhuan Li**[1*] , **Yuyu Luo**[2] , **Guoliang Li**[3] , **Chuxu Zhang**[4]
[1]College of Computer Science and Electronic Engineering, Hunan Univerisity
[2]The Hong Kong University of Science and Technology (Guangzhou)
[3]Tsinghua University
[4]University of Connecticut
{zimingli, liyouhuan}@hnu.edu.cn, yuyuluo@hkust-gz.edu.cn,
liguoliang@tsinghua.edu.cn, chuxu.zhang@uconn.edu

Abstract

Graph neural networks (GNNs) are powerful deep learning models for graph-structured data, demonstrating remarkable success across diverse domains. Recently, the database (DB) community has increasingly recognized the potentiality of GNNs, prompting a surge of researches focusing on improving database systems through GNN-based approaches. However, despite notable advances, There is a lack of a comprehensive review and understanding of how GNNs could improve DB systems. Therefore, this survey aims to bridge this gap by providing a structured and in-depth overview of GNNs for DB systems. Specifically, we propose a new taxonomy that classifies existing methods into two key categories: (1) Relational Databases, which includes tasks like performance prediction, query optimization, and Text-to-SQL, and (2) Graph Databases, addressing challenges like efficient graph query processing and graph similarity computation. We systematically review key methods in each category, highlighting their contributions and practical implications. Finally, we suggest promising avenues for integrating GNNs into Database systems.

1 Introduction

Graph Neural Networks (GNNs) [Kipf and Welling, 2016; Veličković *et al.*, 2018; Schlichtkrull *et al.*, 2018; Zhang *et al.*, 2019] have achieved remarkable success across various domains by leveraging recursive message-passing and aggregation mechanisms, which enables effective solutions for downstream tasks such as node classification and graph clustering. In the field of databases (DB) - a well-established field with decades of research, the potential of GNNs to optimize complex DB problems has recently gained significant attention. GNNs have been applied to various challenges in DB, including query optimization, performance prediction, and graph query processing. Despite these advances, there is still a lack of comprehensive surveys that systematically categorize and summarize these contributions.

[*]Corresponding author

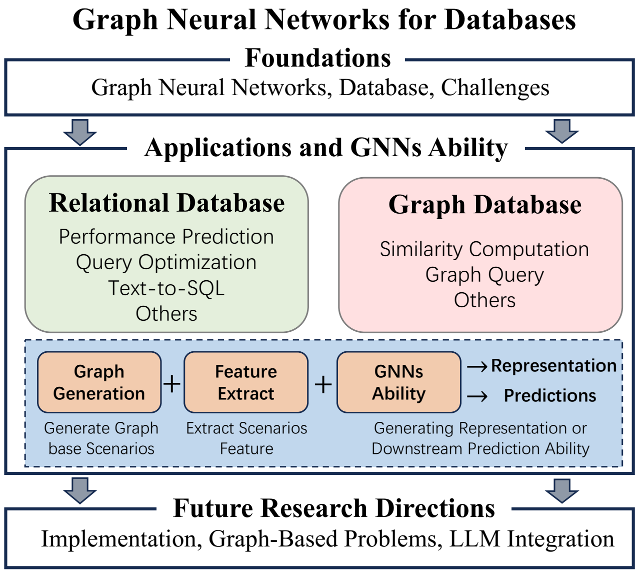
Graph Neural Networks for Databases
Foundations
Graph Neural Networks, Database, Challenges
Applications and GNNs Ability
Relational Database
Performance Prediction
Query Optimization
Text-to-SQL
Others
Graph Database
Similarity Computation
Graph Query
Others
Graph Generation
Feature Extract
GNNs Ability
Representation
Predictions
Generate Graph base Scenarios
Extract Scenarios Feature
Generating Representation or Downstream Prediction Ability
Future Research Directions
Implementation, Graph-Based Problems, LLM Integration

Figure 1: Overview of survey.

To fill this gap, this paper provides a comprehensive review of GNNs for DB. As illustrated in Fig. 1, we begin by introducing foundations of GNNs for DB and propose a taxonomy to categorize existing techniques. Following this, we discuss the key challenges associated with harnessing GNNs' capabilities in DB. We then delve into a detailed review of these techniques, focusing on three key questions: (1) What types of graph are utilized in specific scenarios? (2) How are features extracted from these graphs? (3) What roles do GNNs play in these problems - do they primarily generate data representations for downstream tasks, support downstream prediction, or perform both functions? For each category, we include a summary section that highlights how GNNs enhance the efficiency and capability of DB. Finally, we outline research directions to inspire future exploration of GNNs for DB.

- To the best of our knowledge, this is the first comprehensive survey that focuses exclusively on the applications of GNNs in DB systems.
- We systematically review and categorize existing methods, providing insights into how graphs are constructed, how features are initialized, and the specific roles that GNNs play in each study.

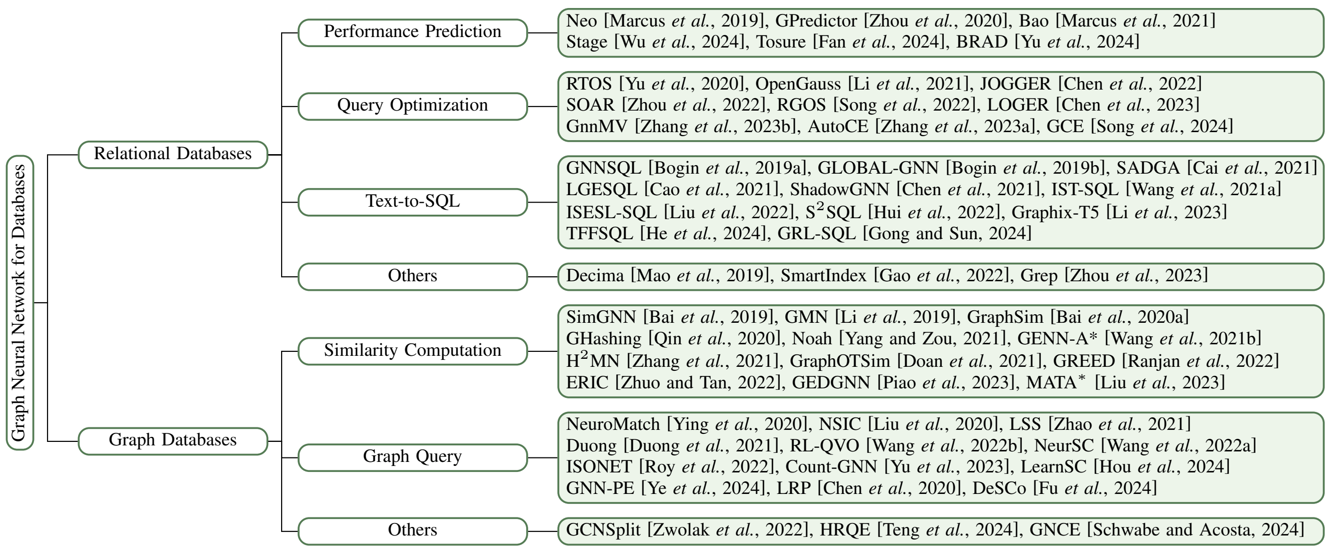
Graph Neural Network for Databases
Relational Databases
Performance Prediction
Neo [Marcus et al., 2019], GPredictor [Zhou et al., 2020], Bao [Marcus et al., 2021]
Stage [Wu et al., 2024], Tosure [Fan et al., 2024], BRAD [Yu et al., 2024]
Query Optimization
RTOS [Yu et al., 2020], OpenGauss [Li et al., 2021], JOGGER [Chen et al., 2022]
SOAR [Zhou et al., 2022], RGOS [Song et al., 2022], LOGER [Chen et al., 2023]
GnnMV [Zhang et al., 2023b], AutoCE [Zhang et al., 2023a], GCE [Song et al., 2024]
Text-to-SQL
GNNSQL [Bogin et al., 2019a], GLOBAL-GNN [Bogin et al., 2019b], SADGA [Cai et al., 2021]
LGESQL [Cao et al., 2021], ShadowGNN [Chen et al., 2021], IST-SQL [Wang et al., 2021a]
ISESL-SQL [Liu et al., 2022], S^{2}SQL [Hui et al., 2022], Graphix-T5 [Li et al., 2023]
TFFSQL [He et al., 2024], GRL-SQL [Gong and Sun, 2024]
Others
Decima [Mao et al., 2019], SmartIndex [Gao et al., 2022], Grep [Zhou et al., 2023]
Graph Databases
Similarity Computation
SimGNN [Bai et al., 2019], GMN [Li et al., 2019], GraphSim [Bai et al., 2020a]
GHashing [Qin et al., 2020], Noah [Yang and Zou, 2021], GENN-A* [Wang et al., 2021b]
H^{2}MN [Zhang et al., 2021], GraphOTSim [Doan et al., 2021], GREED [Ranjan et al., 2022]
ERIC [Zhuo and Tan, 2022], GEDGNN [Piao et al., 2023], MATA* [Liu et al., 2023]
Graph Query
NeuroMatch [Ying et al., 2020], NSIC [Liu et al., 2020], LSS [Zhao et al., 2021]
Duong [Duong et al., 2021], RL-QVO [Wang et al., 2022b], NeurSC [Wang et al., 2022a]
ISONET [Roy et al., 2022], Count-GNN [Yu et al., 2023], LearnSC [Hou et al., 2024]
GNN-PE [Ye et al., 2024], LRP [Chen et al., 2020], DeSCo [Fu et al., 2024]
Others
GCNSplit [Zwolak et al., 2022], HRQE [Teng et al., 2024], GNCE [Schwabe and Acosta, 2024]

Figure 2: A Taxonomy of GNNs for Databases. The technologies of GNNs for DBs can be categorized into two types based on application scenarios: relational databases and graph databases.

- We discuss promising avenues for future research at the intersection of GNNs and DB.

2 Foundations of GNNs for Databases

In this section, we introduce the foundations of GNNs for DB, including their definitions. Then, we discuss the challenges of integrating GNNs with DB, emphasizing aspects that necessitate careful consideration.

2.1 Graph Neural Networks

Graph Neural Networks (GNNs) use a message-passing paradigm over graphs to aggregate information across nodes and edges. A graph is defined as $G = (\mathcal{V}, \mathcal{E}, X)$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of edges, and X indicate node features. GNNs iteratively update the embeddings of each node $v \in \mathcal{V}$ and merge these data with the current embedding of the node. The node embedding update process on the $(l+1)$-th layer is expressed as:

$$h_v^{l+1} = \text{COM}\left(h_v^l, \text{AGG}\left(\{h_u^l \mid \forall u \in \mathcal{N}_v\}\right)\right),$$

where h_v^l and h_u^l the embeddings of nodes v and u at the l-th layer, respectively. The aggregation function (AGG$(\cdot)$) collects information from neighboring nodes, and the combination function (COM$(\cdot)$) integrates this information with the current node's embedding. The initial node embedding h_v^0 is initialized with the corresponding node attributes X_v. For a complete representation of the entire graph G, a READOUT function is used, conveyed as:

$$h_G^l = \text{READOUT}\left(\{h_v^l \mid \forall v \in \mathcal{V}\}\right),$$

where the READOUT operation can be any permutation-invariant function, such as summation, mean, or maximum pooling.

2.2 Databases

A database is a structured repository designed for efficient information storage, manipulation, and retrieval. It solves the challenge of managing large amounts of structured and unstructured data by providing systems that ensure consistency, security, and scalability while supporting data storage, querying, and updating. Modern databases enable complex queries, transactional processing, and data analysis through well-defined schemas, indexing mechanisms, and optimization techniques. Over the past few decades, database research has significantly matured, leading to well-established theoretical foundations. This progress has fueled the development of robust database management systems [Stonebraker and Rowe, 1986; Li *et al.*, 2021; Guia *et al.*, 2017].

2.3 Challenges

The application of GNNs to DB systems presents unique challenges due to graph-based message-passing mechanisms:

- **Graph Representation of Database Problems.** To harness the full potential of GNNs for addressing database-related challenges, it is crucial to efficiently encode the structured elements, constraints, and problems of databases into graph representations.
- **Feature Engineering for Database Contexts.** Designing node and edge features tailored to database contexts is essential but demanding. Nodes should represent database entities, whereas edges should encapsulate relational semantics. Selecting relevant features for the constructed graph is crucial to address different problems.
- **GNNs' Capabilities in Databases.** In database contexts, GNNs' capabilities must align with specific applications, such as query optimization or schema matching. The challenge lies in selecting or designing GNN architectures that balance computational efficiency with effectiveness.

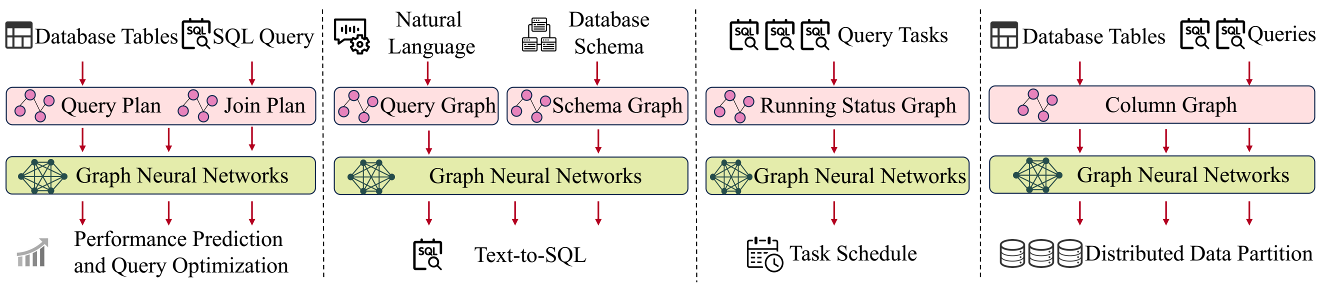

Figure 3: GNNs for Relational Databases. GNNs for relational databases typically represent element relationships or physical operations in relational databases as graphs, leveraging GNNs to generate corresponding representations for downstream tasks. GNNs may also serve as the models for these downstream tasks.

In the following sections, we explore how GNNs contribute to each area and highlighting the differences among the various methods by discussing above challenges. Fig. 2 presents a taxonomy that classifies existing GNNs for DB works into relational databases and graph databases.

3 GNNs for Relational Databases

At first, we explore how GNN-based approaches have been used to address different problems in relational databases. As shown in Fig. 3, GNNs for Relational DB encode element relationships or operations as graphs, generating representations or serving directly as models for downstream tasks.

3.1 Performance Prediction

Performance prediction is vital for resource management and reliable database performance. Since GNNs excel at capturing relationships between related features (e.g., buffer size, computation costs), many works have leveraged GNNs to improve performance prediction.

Neo [Marcus *et al.*, 2019] and Bao [Marcus *et al.*, 2021] convert logical and physical query plans into graph representations. They use GNNs to predict execution times. Particularly, Bao utilizes a vector tree to represent a query plan. Each tree node features a vector embodying the operator's one-hot encoding, cardinality, cost, and cache information. The execution time of these vector trees is then predicted using a tree convolutional neural network. GPredictor [Zhou *et al.*, 2020] extends performance prediction to handle concurrent queries by extracting workload features that impact performance. It identifies physical operators in query plans, analyzes relationships such as data sharing and lock conflicts, and gathers database configuration parameters and active tasks. The concurrent query behavior is modeled as a graph, with nodes representing operators and edges capturing their relationships. GPredictor employs a graph-based learning model to generate embeddings and a graph prediction network to predict performance. To enhance the generalization of performance prediction, Tosure [Fan *et al.*, 2024] uses Tree-LSTM to encode query plans, extracting transferable features independent of specific database instances.

Additionally, some works focus on performance prediction in cloud database engines. BRAD [Yu *et al.*, 2024] improves performance prediction by using SQL text and query features to estimate query runtimes. It creates a feature graph through SQL statement parsing, incorporating logical features. The model utilizes a GNN with a unique query representation framework centered on these logical query attributes and data statistics, including estimated join selectivity. Stage [Wu *et al.*, 2024] employs a three-tiered strategy to address performance prediction. In particular, Stage's global model is constructed using a GCN. The model accepts a query's physical execution plan as a graph and predicts execution times. Each node in the input graph represents a physical operator.

The common framework for query performance prediction using GNNs transforms query plans into graph representations, where nodes and edges encode the relationships in the execution plan. GNNs learn feature representations from these graph structures to estimate query execution times or costs. They capture dependencies and interactions between query plan components, such as tables, indexes, and resources, to predict the overall impact of the performance.

3.2 Query Optimization

Query processing plays a key role in ensuring the efficiency of database systems. This section explores how GNN-based approaches enhance query optimization tasks.

Join Order Selection(JOS)

A query can have multiple join plans that yield the same result, but each plan incurs a different cost. GNNs, with their ability to capture relationships between tables and columns, can optimize join orders to enhance query performance.

RTOS [Yu *et al.*, 2020] uses Tree-LSTM to construct a join tree for a query in a reinforcement learning framework. During construction, Tree-LSTM combines information from columns and tables to estimate query execution costs as the long-term reward for a partial join tree. The join tree is incrementally expanded by adding one node at a time until all required join operators are incorporated. This approach effectively captures the join tree structure and adapts dynamically to schema changes. JOGGER [Chen *et al.*, 2022] addresses the JOS problem by considering table connections. It first constructs a schema graph based on primary-foreign key relationships in the database and applies the DeepWalk algorithm to generate table representations. For each query, JOGGER creates an undirected query graph $\mathcal{T}$. $\mathcal{T}$ is then encoded using GCNs along with the table representations to capture the relationships among the participating tables and their connections.

Contrary to the assumption in previous studies that each segment of a join tree equally impacts cost prediction, SOAR [Zhou *et al.*, 2022] accounts for the varied influence of different components. It resolves the JOS issue using a GAT. Firstly, it utilizes GAT to encapsulate the join trees' structure. Secondly, SOAR leverages GAT to highlight the differing impacts of join tree components on long-term rewards.

LOGER [Chen *et al.*, 2023] leverages the knowledge of the DBMS optimizer to refine the JOS search space. A query is naturally represented as a join graph, where nodes correspond to tables and edges denote join predicates. To address the JOS problem, LOGER employs a Graph Transformer to facilitate information exchange between adjacent table nodes while embedding structural information into each node. This approach effectively captures relationships between tables and join predicates, improving optimization quality.

Specific GNN models optimized for certain databases can enhance JOS performance. RGOS [Song *et al.*, 2022] introduces a join order selection algorithm specifically designed for TiDB. RGOS integrates Tree-LSTM and GCN with TiDB's architecture to represent states in the reinforcement learning environment. The Tree-LSTM encodes the existing subplan tree, while the GCN encodes the new subplan when it joins with the remaining tables, capturing graph relationships. These graph convolutional features are then input into the reinforcement learning Q-network to guide decision-making.

Existing frameworks for addressing the JOS problem with GNNs involve representing join orders as graph structures, where nodes correspond to tables or columns, and edges represent join predicates or relationships. GNNs are used to learn representations of these join trees or graphs, capturing complex interactions and dependencies. This process is often integrated with reinforcement learning techniques.

Cardinality Estimation

Cardinality estimation (CE) is a critical component of database query optimization, tasked with predicting the number of rows (or tuples) a query or its specific components will return. To improve CE precision, AutoCE [Zhang *et al.*, 2023a] introduces a model advisor that quickly selects a suitable CE model based on the data set and specified metrics. In AutoCE, dataset characteristics are represented as a feature graph, such as column correlation, skewness, and domain size. Nodes in the graph encapsulate the features of individual tables, while the edges represent joins between tables. During model selection training, AutoCE employs a GIN to encode these feature graphs, facilitating accurate and efficient model recommendations. For distributed relational databases, GCE [Song *et al.*, 2024] introduces a novel architecture tailored to TiDB. GCE employs GNNs to create query plan representations specifically for CE tasks. In this approach, Tree-LSTM is utilized to capture the structural features of the query plan tree, while a GCN is used to extract the relationships and connection topology in the query plan.

Materialized View

Materialized Views (MVs) operate akin to standard tables. Precomputed and stored query results offered by MVs notably enhance query performance. GnnMV [Zhang *et al.*, 2023b] tackles dynamic materialized view management by representing dynamic query workloads as a query graph. A GNN model is trained on this graph, taking node features as input and outputting benefit estimations for using an MV to answer a query. Key query features (e.g., operator types, metadata, and predicates) are encoded into the GNN.

3.3 Text-to-SQL

SQL, as the query language in databases, continues to be a critical area. One notable focus is Text-to-SQL, which enables users to query databases using natural language, improving accessibility and usability [Liu *et al.*, 2024; Luo *et al.*, 2021]. GNNs model structural relationships in databases, aligning database elements with natural language.

In GNNSQL [Bogin *et al.*, 2019a], the database schema is represented as a graph using GNNs, including tables, columns, and key relationships. This novel method employs a RGCN to compute global node representations, which are then integrated into an encoder-decoder parser. This enables the conversion of natural language questions into SQL queries. Building upon GNNSQL, GLOBAL-GNN [Bogin *et al.*, 2019b] integrates global schema reasoning and database constants. It leverages GNNs in three distinct ways: (1) for relevance scoring, it uses a gated GCN to identify database constants likely to feature in the query; (2) for representation learning, a GCN calculates representations for database constants, used by the decoder to generate candidate queries; and (3) for candidate re-ranking, a re-ranking GCN selects the most appropriate query from the generated candidates. IST-SQL [Wang *et al.*, 2021a] addresses the Text-to-SQL problem in multi-turn scenarios. ISR-SQL constructs a schema-state graph that captures relationships between tables and previously generated SQL keywords associated with the schema. Then, it utilizes an RGNN to encode schema-state representations for further SQL generation.

Previous work on text-to-SQL problems often overlooked the role of edge information in representation generation. LGESQL [Cao *et al.*, 2021] transforms the original node-centric schema graph into an edge-centric graph to better model the edge topology. LGESQL employs an RGAT to enhance the representations of both node-centric and edge-centric graphs. SADGA [Cai *et al.*, 2021] points out the limitation of treating questions as sequences while representing database schemas as structured graphs. SADGA bridges this gap with a unified Gated Graph Neural Network (GGNN) encoder that cohesively models both the question and the schema. To further improve generalization capabilities, ShadowGNN [Chen *et al.*, 2021] tackles the challenge of limited cross-domain generalization caused by schema semantics. It employs a Graph Projection Neural Network built on RGCN to abstract natural language queries and semantic schemas. Similarly, TTFSQL [He *et al.*, 2024] integrates a cross-graph attention mechanism into RGAT, allowing the model to dynamically adjust the attention weights between different graphs.

With the rise of pre-trained language models (PLMs) in various fields [Qin *et al.*, 2018; Luo *et al.*, 2018], researchers have begun integrating PLMs with GNNs to address Text-to-SQL problems. ISESL-SQL [Liu *et al.*, 2022] and S^{2}SQL [Hui *et al.*, 2022] uses PLMs to construct schema-linking graphs. They employ a modified RGAT to learn joint embeddings for questions and schema nodes. Similarly, Graphix-T5 [Li *et al.*, 2023] combines the T5 pre-trained language model with

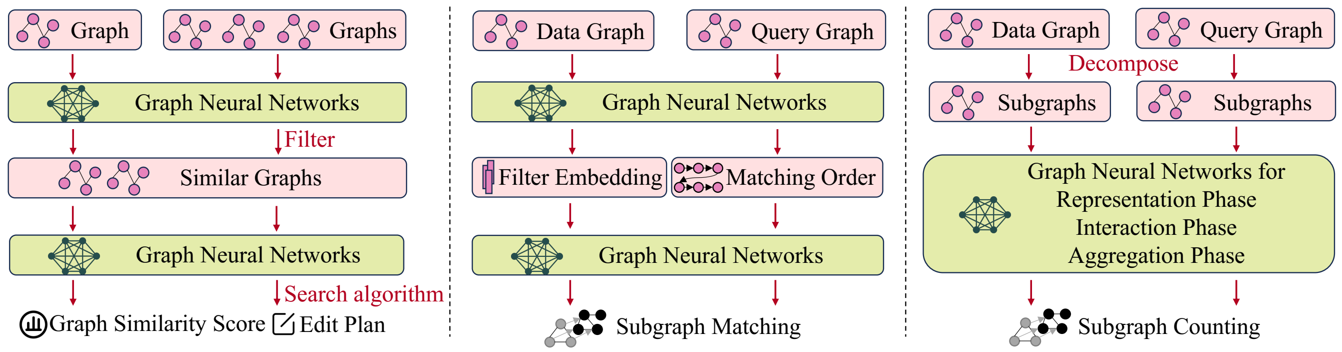

Figure 4: GNNs for Graph Databases. GNNs for graph databases typically transform graph structures into representation space for comparison or use these representations to accelerate traditional algorithms.

a semi-pretrained RGNN to integrate semantic information and establish schema linkage relationships. GRL-SQL [Gong and Sun, 2024] also merges PLMs with GNNs, focusing on disconnected nodes in the question-schema graph.

3.4 Others Problems in Relational Databases

Scheduling. Optimal scheduling enhances the computational effectiveness of a database. Decima [Mao *et al.*, 2019] begins by encoding ongoing job stages and their dependencies as directed acyclic graphs (DAGs). These DAGs are processed using a GCN to generate the corresponding representations. The representations include information such as job stage attributes and the dependency structure of the DAG.
Data Partition. A well-designed partition improves database performance by allowing queries to access the relevant data efficiently. Grep [Zhou *et al.*, 2023] improves the quality of data partition by introducing GNN-based technology. It begins by creating a column graph, where columns are vertices and joins are edges. Each vertex has a feature vector. Grep then uses attention mechanisms to select key columns for partitioning. A GNN is applied to capture global structures for each vertex. A classifier is used to choose the best partition keys, ensuring optimal partitioning decisions. To avoid the cost of actual partitioning, Grep uses a graph learning model to predict database performance.
Index Selection. Indexes are essential for accelerating database queries. SmartIndex [Gao *et al.*, 2022] addresses this issue by using a GCN-powered cost estimation model for selecting workload-specific indexes. The process begins with an LSTM model that encodes nodes in query plans, where each node represents a specific DBMS operation. At the same time, SmartIndex traverses the query plan to get its query plan's adjacency matrix. The GCN processes the LSTM outputs and adjacency matrix to generate feature representations. A ResNet model is then used to estimate the cost of each query plan and index pair.

3.5 Summary of Methods on Relational Databases

As shown in Fig. 3, the above methods significantly enhance the efficiency of the relational database by modeling database operations and relationships as graphs. One approach involves representing database actions, such as query plans and join operations, as graphs. In these graphs, nodes correspond to operators or intermediate results, and edges represent dependencies or execution sequences. GNNs are used to encode these actions into detailed representations that can predict key metrics, including performance and execution cost. Such predictions enable query optimizers to select the most efficient execution plans. Another approach focuses on modeling relationships in relational databases. Schema graphs capture the connections between columns and tables, reflecting data dependencies and structural relationships. Similarly, workload graphs describe the dependency structures of jobs, emphasizing shared resources and execution priorities. By processing these relationships, GNNs generate representations that support downstream tasks that contribute to more efficient database operations.

4 GNNs for Graph Databases

Graph databases store, query, and analyze data in graph form. This representation provides significant opportunities for leveraging GNNs power. As shown in Fig. 4, this survey emphasizes graph database tasks, including graph similarity computation and graph query.

4.1 Similarity Computation

Graph similarity computation is a fundamental operation in graph databases. Since GNNs can effectively transform graphs into embeddings and extract their features, many studies have integrated GNNs into graph similarity computing problems.

Graph Edit Distance (GED)

GED is widely used to identify a set of graphs from a graph database that are structurally similar to a given query graph and compute the edit distances between them. GNNs excel at transforming graphs into embeddings, enabling efficient comparisons in the embedding space. Furthermore, GNNs can help traditional algorithms in the GED problem, such as A*, by guiding them to better search directions. In this section, we discuss these methods in detail.

SimGNN [Bai *et al.*, 2019] reformulates graph similarity computation as a learning problem. It starts by using GCNs to generate node embeddings for each graph, encoding both node features and structural properties. Subsequently, SimGNN calculates graph similarity through interaction between graph embeddings, yielding an graph similarity score. At the same time, GMN [Li *et al.*, 2019] introduces graph matching networks to improve verification accuracy. It employs a cross-graph attention mechanism to associate nodes across graphs, identifying structural differences to compute similarity scores.

Previous approaches often use fixed-length vectors to represent graphs, but real-world graphs typically vary in size. To address this, GraphSim [Bai *et al.*, 2020b] directly performs neural operations directly on two sets of node embeddings. It employs a GCN to generate node representations and introduces a multiscale framework that utilizes outputs from multiple GCN layers to construct similarity matrices. However, ERIC [Zhuo and Tan, 2022] challenges the necessity of computationally expensive node-to-node matching. It proposes a streamlined regularization technique, Alignment Regularization (AReg). During training, AReg enforces a node-graph correspondence constraint in the GNN encoder. At inference, only the graph-level representations are used to compute similarity scores. MGMN [Ling *et al.*, 2023] combines a node–graph matching network for capturing interactions and a siamese GNN for learning global-level interactions.

Although many works improved efficiency and accuracy for GED, they required scanning the entire graph database. To address this, GHashing [Qin *et al.*, 2020] introduces GNN-based semantic hashing for efficient GED computation on large datasets. It trains a GNN to generate graph embeddings, which are then converted into hash codes by an MLP to represent graph types, enabling constant-time lookups. Similarly, GREED [Ranjan *et al.*, 2022] employs an indexing strategy to enhance the pruning process's efficiency. It uses a Siamese GIN to generate independent embeddings for graphs in a pair-independent manner.

To better capture the relationships in graphs, H^{2}MN [Zhang *et al.*, 2021] introduces a novel approach by transforming each graph into a hypergraph to capture non-pairwise relationships. Then, H^2 MN uses a hypergraph neural network to model complex structural relationships and achieve superior performance. For explainability, GOTSim [Doan *et al.*, 2021] generates node embeddings across multiple GCN layers and derives the optimal transformation cost. The final similarity score is aggregated from these costs across all layers.

Search-based works address limitations of prior learning-based methods cannot recover the actual edit path. Noah [Yang and Zou, 2021], GENN-A* [Wang *et al.*, 2021b] and MATA* [Liu *et al.*, 2023] focus on guiding the search directions of the A* algorithm. Noah uses GIN to learn the cost estimation function $h(\cdot)$. GENN-A* introduces the Graph Edit Neural Network (GENN). GENN consists of GCN and SplineCNN to generate node embeddings. These embeddings are cached for further dynamic graph embedding during the A* search process. MATA* reframes GED computation as a node-matching problem. It uses a structure-enhanced GNN (SEGcn) to identify candidate matching nodes, effectively pruning unpromising search directions in the A* algorithm. GEDGNN [Piao *et al.*, 2023] employs a two-step process to recover the edit path. First, it predicts the GED value and a matching matrix using GNNs. Second, the k-best matching algorithm generates multiple node matching from this matrix to derive the edit path.

4.2 Graph Query

Subgraph matching and subgraph counting involve identifying all subgraphs in a data graph that match a given query graph or determining the count of such subgraphs. Traditional algorithms face significant scalability challenges due to the tasks' NP-complete nature. GNN-based methods enable more scalable solutions for subgraph matching and counting.

Subgraph Matching

GNN-based methods for subgraph matching can be broadly divided into two categories: learning-based approaches that rely entirely on GNNs and hybrid approaches that combine GNN techniques with traditional heuristic search methods to assist in the matching process.

There are some learning-based subgraph matching works focus on approximate subgraph matching result. Neuromatch [Ying *et al.*, 2020] leverages a GNN to embed nodes in the data graph. For a given query graph, it selects a central node and maps the query graph into the embedding space using the same GNN. A data vertex is retained as a candidate only if its embedding is located in the top-right quadrant of the query graph's central node. This approach efficiently identifies candidate subgraphs likely to match the query graph. ISONET [Roy *et al.*, 2022] extends the scope of GNNs by proposing a framework incorporating context-sensitive representations for nodes and edges.

Hybrid methods can be divided into two types. The first type uses GNN embeddings as an indexing mechanism. The second type leverages embeddings to generate better traditional matching orders. For the first type, Duong [Duong *et al.*, 2021] and GNN-PE [Ye *et al.*, 2024] use GNN-based embeddings to filter unpromising search branches. Duong introduces an indexing mechanism, mapping graphs into a numerical space where structurally similar nodes and subgraphs are close to each other. When a query graph is received, the search is narrowed to candidate regions near the query's embedding, reducing the search space. GNN-PE employs GNN-based embeddings for graph paths. These embeddings define a dominant relationship, ensuring that paths with dominant embeddings maintain a subgraph relationship. This approach enables GNN-PE to prune candidates effectively and transform subgraph matching into a dominating region search in the embedding space. For the second type, RL-QVO [Wang *et al.*, 2022b] use GNNs to generate high-quality matching order. The process begins by generating feature vector for each node in the query graph, based on statistical heuristics such as node degree and label frequency. RL-QVO then employs GNNs to produce vector representations that capture comprehensive graph-level information. These representations are used in a reinforcement learning framework to generate optimal matching orders.

Overall, GNNs for subgraph matching contribute to two key areas. First, they utilize GNNs to map graph relationships into a unified embedding space. These embeddings can directly support learning-based subgraph matching or act as auxiliary filters in hybrid methods. Second, GNNs capture graph features to generate higher-quality matching order.

Subgraph Counting

GNNs are employed in two primary directions for subgraph counting problem. First, they are used to generate better embeddings for data graphs and query graphs. Second, they are leveraged to enhance scalability, enabling subgraph counting for larger and more complex graphs.

NSIC [Liu *et al.*, 2020] treats the data graph as an information source and the query graph as a retrieval query. It employs RGCN/RGIN to learn representations from adjacency matrices and vertex features. These representations are processed through an interaction module to capture correlations and are further combined with size information to predict subgraph counts. At the same time, [Chen *et al.*, 2020] demonstrates that 2-invariant Graph Networks can effectively count subgraphs for star-shaped patterns. To capture fine-grained structural information, Count-GNN [Yu *et al.*, 2023] introduces a subgraph counting framework based on an edge-centric GNN that propagates and aggregates messages specifically for edges.

Several studies aim to improve the efficiency of subgraph counting in large graphs by decomposing the data graph, the query graph, or both. LSS [Zhao *et al.*, 2021] decomposes query graphs and introduces a subgraph counting framework based on sketch learning. This method first extracts features for each substructure from the query graph as a fixed-length vectorized representation. Specifically, ALSS employs GNNs to represent substructures. It then uses a self-attention mechanism to learn an aggregation function, followed by an MLP to predict the final count. On the contrary, NeurSC [Wang *et al.*, 2022a] extracts substructures from the data graph. NeurSC uses an estimator for subgraph counting. The estimator comprises an intra-graph neural network, an inter-graph neural network, and a wasserstein discriminator. Similarly, DeSCo [Fu *et al.*, 2024] divides the data graph into small neighborhoods, encoding their local information using a GNN with subgraph-based heterogeneous message passing. LearnSC [Hou *et al.*, 2024] simultaneously decomposes both the query graph and data graph. Additionally, LearnSC presents a comprehensive subgraph counting framework with five phases: decomposition, representation, interaction, estimation, and aggregation. LearnSC uses GIN to acquire each node's representation in the representation phase and employs GIN as the base network in a cross-graph learning model during the interaction phase.

4.3 Other Problems in Graph Databases

Graph Partitioning. Graph partitioning plays a critical role in graph database. GCNSplit [Zwolak *et al.*, 2022] introducing a novel framework designed for graph partitioning in streaming environments. It utilizes an immutable, fixed-size model to capture the characteristics of the graph stream. GCNSplit leverages GCN to generate vertex embeddings and enhances the model with load constraints and assignment heuristics.
Knowledge Graph. Knowledge graphs are pivotal to graph databases. This paper concentrates on database-specific facets such as cardinality estimation. SPARQL enables querying via graph pattern matching. GNCE [Schwabe and Acosta, 2024] leverages GNNs to model the query structures of knowledge graphs. It generates RDF2Vec embeddings, trains a GNN to minimize cardinality discrepancies, and predicts cardinalities for new queries. HRQE [Teng *et al.*, 2024] extends this to hyper-relational knowledge graphs by incorporating qualifier information, using a pre-trained module, enhanced GNN layers, and an MLP decoder for cardinality prediction.

4.4 Summary of Methods on Graph Databases

GNN-based methods often transform traditional graphs into representations, effectively capturing essential features. These representations can be broadly categorized into three main types. The first category involves comparing graphs in the representation space. This capability is particularly useful for similarity-based queries, where graphs are matched based on proximity in the embedding space. Additionally, these representations play a crucial role in filtering operations. For instance, in subgraph matching, the representations are used to prune unpromising candidate graphs early in the process. The second category leverages these representations as an auxiliary tool to enhance traditional graph algorithms. For example, GNN-generated representations guide A^* algorithms by providing heuristic search directions. The third category focuses on utilizing these representations to enhance the performance of graph databases. For example, GNN representations improve data partitioning by capturing structural nuances.

5 Future Directions

The study of GNNs for databases is advancing quickly, offering numerous challenges and opportunities. Here, we explore some of them.
Database Implementation. While many GNN-based methods aim to improve database efficiency and capability, few have been directly implemented in business databases. Future work could explore optimizing GNN-based techniques tailored to specific database frameworks. Moreover, implementing GNN-based methods in cloud-native environments presents opportunities to real-time inference capabilities.
Relational Databases Problems based on Graph. As shown in Fig. 3, many GNN-based methods for relational databases rely on transforming database schemas and their underlying relationships into graph representations. By representing database elements, like tables and columns, and relationships, like graph nodes and edges, researchers can unlock new possibilities for addressing a wide range of database problems.
Integration with Large Language Models. The combination of large language models (LLMs) and graph learning methods has emerged as a significant area of research. [Tang *et al.*, 2024; Xie *et al.*, 2024]. LLMs excel at understanding and processing unstructured textual data, while GNNs provide powerful tools for modeling structured relationships and dependencies in graph-structured data. By combining the strengths of both paradigms, researchers can address a broader spectrum of database challenges, from natural language query understanding to optimizing graph-based queries and enhancing data retrieval accuracy.

6 Conclusion

GNNs have shown strong potential in databases by capturing complex structural patterns. Many studies have explored this interdisciplinary area to enhance relational and graph database performance. In this survey, we first introduce key concepts in this joint area. We then categorize existing efforts into relational and graph database contexts. For each category, we highlighted the specific problems addressed and the roles GNNs played in solving them. Finally, we outline several promising directions for future research in this emerging field.

References

[Bai *et al.*, 2019] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. In *WSDM*, 2019.

[Bai *et al.*, 2020a] Yunsheng Bai, Hao Ding, Ken Gu, Yizhou Sun, and Wei Wang. Learning-based efficient graph similarity computation via multi-scale convolutional set matching. In *AAAI*, 2020.

[Bai *et al.*, 2020b] Yunsheng Bai, Hao Ding, Ken Gu, Yizhou Sun, and Wei Wang. Learning-based efficient graph similarity computation via multi-scale convolutional set matching. In *AAAI*, 2020.

[Bogin *et al.*, 2019a] Ben Bogin, Jonathan Berant, and Matt Gardner. Representing schema structure with graph neural networks for text-to-sql parsing. In *ACL*, 2019.

[Bogin *et al.*, 2019b] Ben Bogin, Matt Gardner, and Jonathan Berant. Global reasoning over database structures for text-to-sql parsing. In *EMNLP*, 2019.

[Cai *et al.*, 2021] Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao. SADGA: structure-aware dual graph aggregation network for text-to-sql. In *NIPS*, 2021.

[Cao *et al.*, 2021] Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. LGESQL: line graph enhanced text-to-sql model with mixed local and non-local relations. In *ACL*, 2021.

[Chen *et al.*, 2020] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? In *NIPS*, 2020.

[Chen *et al.*, 2021] Zhi Chen, Lu Chen, Yanbin Zhao, Ruisheng Cao, Zihan Xu, Su Zhu, and Kai Yu. Shadowgnn: Graph projection neural network for text-to-sql parser. In *ACL*, 2021.

[Chen *et al.*, 2022] Jin Chen, Guanyu Ye, Yan Zhao, Shuncheng Liu, Liwei Deng, Xu Chen, Rui Zhou, and Kai Zheng. Efficient join order selection learning with graph-based representation. In *KDD*, 2022.

[Chen *et al.*, 2023] Tianyi Chen, Jun Gao, Hedui Chen, and Yaofeng Tu. LOGER: A learned optimizer towards generating efficient and robust query execution plans. *VLDB*, 2023.

[Doan *et al.*, 2021] Khoa D. Doan, Saurav Manchanda, Suchismit Mahapatra, and Chandan K. Reddy. Interpretable graph similarity computation via differentiable optimal alignment of node embeddings. In *SIGIR*, 2021.

[Duong *et al.*, 2021] Chi Thang Duong, Dung Hoang, Hongzhi Yin, Matthias Weidlich, Quoc Viet Hung Nguyen, and Karl Aberer. Efficient streaming subgraph isomorphism with graph neural networks. *VLDB*, 2021.

[Fan *et al.*, 2024] Shuhuan Fan, Mengshu Hou, Rui Xi, and Wenwen Ma. Precision meets resilience: Cross-database generalization with uncertainty quantification for robust cost estimation. In *CIKM*, 2024.

[Fu *et al.*, 2024] Tianyu Fu, Chiyue Wei, Yu Wang, and Rex Ying. Desco: Towards generalizable and scalable deep subgraph counting. In *WSDM*, 2024.

[Gao *et al.*, 2022] Jianling Gao, Nan Zhao, Ning Wang, and Shuang Hao. Smartindex: An index advisor with learned cost estimator. In *CIKM*, 2022.

[Gong and Sun, 2024] Zheng Gong and Ying Sun. Graph reasoning enhanced language models for text-to-sql. In *SIGIR*, 2024.

[Guia *et al.*, 2017] José Guia, Valéria Gonçalves Soares, and Jorge Bernardino. Graph databases: Neo4j analysis. In *ICEIS*, 2017.

[He *et al.*, 2024] Yaozhen He, Enpei Huang, and Rongzhi Qi. Tffsql: Enhancing graph neural network through text feature fusion for text-to-sql. In *AIAHPC*, 2024.

[Hou *et al.*, 2024] Wenzhe Hou, Xiang Zhao, and Bo Tang. Learnsc: An efficient and unified learning-based framework for subgraph counting problem. In *ICDE*, 2024.

[Hui *et al.*, 2022] Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Yanyang Li, Bowen Li, Jian Sun, and Yongbin Li. S^2SQL: Injecting syntax to question-schema interaction graph encoder for text-to-SQL parsers. In *ACL*, 2022.

[Kipf and Welling, 2016] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arxiv*, 2016.

[Li *et al.*, 2019] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *ICML*, 2019.

[Li *et al.*, 2021] Guoliang Li, Xuanhe Zhou, Ji Sun, Xiang Yu, Yue Han, Lianyuan Jin, Wenbo Li, Tianqing Wang, and Shifu Li. opengauss: An autonomous database system. *VLDB*, 2021.

[Li *et al.*, 2023] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *AAAI*, 2023.

[Ling *et al.*, 2023] Xiang Ling, Lingfei Wu, Saizhuo Wang, Tengfei Ma, Fangli Xu, Alex X. Liu, Chunming Wu, and Shouling Ji. Multilevel graph matching networks for deep graph similarity learning. *TNNLS*, 2023.

[Liu *et al.*, 2020] Xin Liu, Haojie Pan, Mutian He, Yangqiu Song, Xin Jiang, and Lifeng Shang. Neural subgraph isomorphism counting. In *KDD*, 2020.

[Liu *et al.*, 2022] Aiwei Liu, Xuming Hu, Li Lin, and Lijie Wen. Semantic enhanced text-to-sql parsing via iteratively learning schema linking graph. In *KDD*, 2022.

[Liu *et al.*, 2023] Junfeng Liu, Min Zhou, Shuai Ma, and Lujia Pan. Mata*: Combining learnable node matching with a* algorithm for approximate graph edit distance computation. In *CIKM*, 2023.

[Liu *et al.*, 2024] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. A survey of nl2sql with large language models: Where are we, and where are we going?, 2024.

[Luo *et al.*, 2018] Yuyu Luo, Xuedi Qin, Nan Tang, Guoliang Li, and Xinran Wang. Deepeye: Creating good data visualizations by keyword search. In *SIGMOD*, 2018.

[Luo *et al.*, 2021] Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. Synthesizing natural language to visualization (NL2VIS) benchmarks from NL2SQL benchmarks. In *SIGMOD*, 2021.

[Mao *et al.*, 2019] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *SIGCOMM*, 2019.

[Marcus *et al.*, 2019] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. Neo: A learned query optimizer. *VLDB*, 2019.

[Marcus *et al.*, 2021] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. Bao: Making learned query optimization practical. In *SIGMOD*, 2021.

[Piao *et al.*, 2023] Chengzhi Piao, Tingyang Xu, Xiangguo Sun, Yu Rong, Kangfei Zhao, and Hong Cheng. Computing graph edit distance via neural graph matching. *VLDB*, 2023.

[Qin *et al.*, 2018] Xuedi Qin, Yuyu Luo, Nan Tang, and Guoliang Li. Deepeye: Visualizing your data by keyword search. In *EDBT*, 2018.

[Qin *et al.*, 2020] Zongyue Qin, Yunsheng Bai, and Yizhou Sun. Ghashing: Semantic graph hashing for approximate similarity search in graph databases. In *KDD*, 2020.

[Ranjan *et al.*, 2022] Rishabh Ranjan, Siddharth Grover, Sourav Medya, Venkatesan T. Chakaravarthy, Yogish Sabharwal, and Sayan Ranu. GREED: A neural framework for learning graph distance functions. In *NIPS*, 2022.

[Roy *et al.*, 2022] Indradyumna Roy, Venkata Sai Baba Reddy Velugoti, Soumen Chakrabarti, and Abir De. Interpretable neural subgraph matching for graph retrieval. In *AAAI*, 2022.

[Schlichtkrull *et al.*, 2018] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *ESWC*, 2018.

[Schwabe and Acosta, 2024] Tim Schwabe and Maribel Acosta. Cardinality estimation over knowledge graphs with embeddings and graph neural networks. *SIGMOD*, 2024.

[Song *et al.*, 2022] Yuanfeng Song, Yuqiang Li, Shuhuan Fan, Dongsheng He, and Jianming Liao. A new graph neural network-based join optimization algorithm. In *ADMIT*, 2022.

[Song *et al.*, 2024] Yuan Feng Song, Xiao Dong Li, Dan Ni Zhang, Shu Huan Fan, and Dong Sheng He. A new cardinality estimation method based on graph neural networks. In *FAIML*, 2024.

[Stonebraker and Rowe, 1986] Michael Stonebraker and Lawrence A. Rowe. The design of postgres. In *SIGMOD*, 1986.

[Tang *et al.*, 2024] Nan Tang, Chenyu Yang, Ju Fan, Lei Cao, Yuyu Luo, and Alon Y. Halevy. Verifai: Verified generative AI. In *CIDR*, 2024.

[Teng *et al.*, 2024] Fei Teng, Haoyang Li, Shimin Di, and Lei Chen. Cardinality estimation on hyper-relational knowledge graphs. *arxiv*, 2024.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[Wang *et al.*, 2021a] Runze Wang, Zhen-Hua Ling, Jingbo Zhou, and Yu Hu. Tracking interaction states for multi-turn text-to-sql semantic parsing. In *AAAI*, 2021.

[Wang *et al.*, 2021b] Runzhong Wang, Tianqi Zhang, Tianshu Yu, Junchi Yan, and Xiaokang Yang. Combinatorial learning of graph edit distance via dynamic embedding. In *CVPR*, 2021.

[Wang *et al.*, 2022a] Hanchen Wang, Rong Hu, Ying Zhang, Lu Qin, Wei Wang, and Wenjie Zhang. Neural subgraph counting with wasserstein estimator. In *SIGMOD*, 2022.

[Wang *et al.*, 2022b] Hanchen Wang, Ying Zhang, Lu Qin, Wei Wang, Wenjie Zhang, and Xuemin Lin. Reinforcement learning based query vertex ordering model for subgraph matching. In *ICDE*, 2022.

[Wu *et al.*, 2024] Ziniu Wu, Ryan Marcus, Zhengchun Liu, Parimarjan Negi, Vikram Nathan, Pascal Pfeil, Gaurav Saxena, Mohammad Rahman, Balakrishnan Narayanaswamy, and Tim Kraska. Stage: Query execution time prediction in amazon redshift. In *SIGMOD*, 2024.

[Xie *et al.*, 2024] Yupeng Xie, Yuyu Luo, Guoliang Li, and Nan Tang. Haichart: Human and AI paired visualization system. *VLDB*, 2024.

[Yang and Zou, 2021] Lei Yang and Lei Zou. Noah: Neural-optimized a* search algorithm for graph edit distance computation. In *ICDE*, 2021.

[Ye *et al.*, 2024] Yutong Ye, Xiang Lian, and Mingsong Chen. Efficient exact subgraph matching via gnn-based path dominance embedding. *VLDB*, 2024.

[Ying *et al.*, 2020] Rex Ying, Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, and Jure Leskovec. Neural subgraph matching. *arxiv*, 2020.

[Yu *et al.*, 2020] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. Reinforcement learning with tree-lstm for join order selection. In *ICDE*, 2020.

[Yu *et al.*, 2023] Xingtong Yu, Zemin Liu, Yuan Fang, and Xinming Zhang. Learning to count isomorphisms with graph neural networks. In *AAAI*, 2023.

[Yu *et al.*, 2024] Geoffrey X. Yu, Ziniu Wu, Ferdi Kossmann, Tianyu Li, Markos Markakis, Amadou Ngom, Samuel Madden, and Tim Kraska. Blueprinting the cloud: Unifying and automatically optimizing cloud data infrastructures with brad. *VDLB*, 2024.

[Zhang *et al.*, 2019] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V. Chawla. Heterogeneous graph neural network. In *KDD*, 2019.

[Zhang *et al.*, 2021] Zhen Zhang, Jiajun Bu, Martin Ester, Zhao Li, Chengwei Yao, Zhi Yu, and Can Wang. H2MN: graph similarity learning with hierarchical hypergraph matching networks. In *KDD*, 2021.

[Zhang *et al.*, 2023a] Jintao Zhang, Chao Zhang, Guoliang Li, and Chengliang Chai. Autoce: An accurate and efficient model advisor for learned cardinality estimation. In *ICDE*, 2023.

[Zhang *et al.*, 2023b] Jintao Zhang, Chao Zhang, Guoliang Li, and Chengliang Chai. Dynamic materialized view management using graph neural network. In *ICDE*, 2023.

[Zhao *et al.*, 2021] Kangfei Zhao, Jeffrey Xu Yu, Hao Zhang, Qiyan Li, and Yu Rong. A learned sketch for subgraph counting. In *SIGMOD*, 2021.

[Zhou *et al.*, 2020] Xuanhe Zhou, Ji Sun, Guoliang Li, and Jianhua Feng. Query performance prediction for concurrent queries using graph embedding. *VLDB*, 2020.

[Zhou *et al.*, 2022] Weiqing Zhou, Siyu Zhan, Bo Dai, and Lei Guo. SOAR: A learned join order selector with graph attention mechanism. In *IJCNN*, 2022.

[Zhou *et al.*, 2023] Xuanhe Zhou, Guoliang Li, Jianhua Feng, Luyang Liu, and Wei Guo. Grep: A graph learning based database partitioning system. *SIGMOD*, 2023.

[Zhuo and Tan, 2022] Wei Zhuo and Guang Tan. Efficient graph similarity computation with alignment regularization. In *NIPS*, 2022.

[Zwolak *et al.*, 2022] Michal Zwolak, Zainab Abbas, Sonia Horchidan, Paris Carbone, and Vasiliki Kalavri. *GCNSplit*: bounding the state of streaming graph partitioning. In *aiDM*, 2022.