GT-Muzero: SQL Query Plan Optimization via Graph Transformers and Model-Based Reinforcement Learning

Fuming Ye, Wenting Li, Qiong Zhou, Jie Ye, Mengzhu Liu*
College of Computer and Information Engineering, Guizhou University of Commerce, Guiyang 550014, China
*E-mail: mengzhuliu12@163.com

Keywords: Structured query language, reinforcement learning, graph transformer, muzero, learning-based optimizer

Received: June 24, 2013

Learning-based Structured Query Language (SQL) optimizers often face low sample efficiency and high training costs. To address these challenges, this study proposes a query optimization framework, GT-MuZero, which integrates a Graph Transformer (GT) with the model-based reinforcement learning (RL) algorithm MuZero. The framework converts SQL queries into heterogeneous graphs containing tables, predicates, and join operators. Structural encoding is performed via Laplacian feature vectors. GT's global self-attention mechanism effectively overcomes the over-smoothing problem encountered by traditional Graph Neural Networks (GNNs) when processing deep execution trees. MuZero reduces reliance on costly real-database interactions by performing virtual forward planning in a learned latent space. Experiments on a high-performance server equipped with NVIDIA A100 GPUs, using the 100 GB TPC-DS benchmark datasets, demonstrated exceptional sample efficiency: GT-MuZero achieved 96.73% policy consistency with only 10,000 real training samples, whereas conventional methods such as GCN-PPO required more than 50,000 samples. Quantitative evaluation showed a geometric mean performance ratio (GMPR) of 2.81. Compared with the PostgreSQL baseline, latency for complex queries was reduced by more than 3.8 times. Although the average inference latency of 155 ms exhibits diminishing returns for minimal queries, the framework's high sample efficiency and closed-loop robustness under large-scale, complex analytical workloads demonstrate its practical effectiveness and scientific value for building high-performance, adaptive intelligent database systems.

Povzetek: Študija predstavlja učinkovit pristop za optimizacijo SQL poizvedb z uporabo grafovnih transformatorjev in spodbujevanega učenja, ki zmanjšuje potrebo po velikem številu učnih vzorcev ter izboljšuje zmogljivost pri zahtevnih poizvedbah.

1 Introduction

With the continuous advancement of global digital transformation, data has become the central driving force of modern society. From business intelligence and scientific computing to artificial intelligence (AI) applications, systems are increasingly generating and processing massive volumes of structured data [1]. Structured Query Language (SQL), as the standard interface for interacting with relational databases, directly impacts the efficiency and responsiveness of advanced applications [2, 3]. However, the rapid growth of data and increasingly complex analytical requirements pose unprecedented challenges to traditional database query optimizers. Traditional optimizers rely heavily on static, sample-based cost models to estimate execution costs and use heuristic rules or limited dynamic programming to prune the search space [4]. This approach often results in a performance gap between the selected "optimal" plan and the true optimal plan, forming a key bottleneck that limits overall system performance.

To address this challenge, both academia and industry have turned to machine learning, giving rise to the research frontier of learning-based query optimization. The core idea is to enable optimizers to "learn" from historical experience and make better decisions. Early AI-based methods primarily focused on improving cardinality estimation using deep neural networks, but they did not fundamentally address the combinatorial optimization problem. Later, end-to-end reinforcement learning (RL)–based optimizers emerged, modeling query optimization as a sequential decision-making problem in which an agent interacts with the database environment [5]. Through continuous trial-and-error and feedback, the agent outputs strategies for efficiently executing query plans. However, each trial requires executing actual queries, resulting in extremely low sample efficiency and high training costs, which severely restrict large-scale deployment in production environments.

To overcome these limitations, this study proposes an adaptive SQL query optimization paradigm that integrates the Graph Transformer (GT) with the MuZero algorithm. The core innovation lies in combining GT's capability to capture global structural dependencies with MuZero's model-based RL, enabling high-fidelity plan representation and efficient look-ahead planning in latent space. To systematically evaluate the proposed framework, this study established the following research objectives. These objectives aimed to experimentally

verify whether the framework addressed the core bottlenecks in the current field. The specific research questions are as follows:

- RQ1: can global semantic modeling with a GT effectively mitigate the over-smoothing problem of traditional graph neural networks (GNNs) and thereby improve plan selection accuracy under complex SQL topologies?
- RQ2: can the introduction of the muzero-based model-based RL paradigm achieve faster model convergence and higher sample efficiency while significantly reducing interaction with the real database environment, that is, lowering sample requirements?

The main contributions of this study are as follows:

- Overcoming representation bottlenecks. This study introduces a GT to address the over-smoothing problem that occurs when traditional GNNs process deep execution trees. By leveraging a global self-attention mechanism, the model captures the semantic structure of complex SQL topologies more effectively.
- A novel decision-making paradigm. The forward planning mechanism of muzero is applied to SQL query optimization for the first time. By performing simulated planning in an abstract latent space, the framework substantially improves sample efficiency and reduces reliance on costly real-environment feedback, which has been a fundamental limitation of learning-based optimizers.
- Validation of synergistic effects. Extensive ablation studies verify the synergistic benefit of the dual-driven architecture that combines deep representation learning with virtual planning.

2 Related work

2.1 Traditional optimization and machine learning approaches

Traditional query optimization techniques primarily rely on cost-based models. However, they often struggle with complex queries in real-time large-scale data analytics scenarios. Rahman et al. [6] noted that static statistics and heuristic rules were insufficient under dynamic workloads. In large-scale data environments, learning-based query optimizers have demonstrated clear advantages over traditional cost models. Marcus et al. [7] introduced a learning-guided mechanism that enabled the deployment of learning-based optimizers in production systems. Their results showed that experience-driven learning significantly reduced the execution cost of complex queries. Compared with earlier approaches that focused on specific query types, such as K-NN, this experience-based optimization paradigm exhibited stronger robustness under dynamic workloads. Although Milicevic and Babovic [8] provided a systematic review of deep learning applications in database execution and recognized their potential, early efforts remained limited. For example, the K-NN–based optimization approach studied by Choi et al. [9] focused on specific query types and lacked the ability to model complex SQL join relationships.

2.2 Limitations of RL and graph learning

To overcome the rigidity of rule-based engines, Sassi and Jaziri [10] explored combining RL with graph techniques to capture SQL structural relationships. Khan et al. [11] further demonstrated the advantages of representing query plans as heterogeneous graphs. However, conventional GNNs suffer from over-smoothing when stacked deeply, which reduces node feature discriminability. To address representation accuracy, Yan et al. [12] systematically analyzed the foundations, techniques, and challenges of deep reinforcement learning (DRL) for join order selection. In addition, Marcus and Papaemmanouil [13] demonstrated the robustness of model-based DRL in optimizing complex systems. This finding suggests that constructing an internal predictive model, such as MuZero, can significantly improve sample efficiency by reducing reliance on costly interactions with the real environment.

2.3 From attention mechanisms to deep SQL representation

The self-attention mechanism in the Transformer architecture provides an effective means of modeling long-range dependencies. Bhopale and Tiwari [14] demonstrated the superiority of Transformers in contextual representation for information retrieval. This global alignment capability was also validated in multimodal domains. Tan et al. [15] investigated the feasibility of using large language models (LLMs) as relational database query optimizers, demonstrating that Transformer-based architectures excel at capturing complex SQL semantics and performing execution logic reasoning. Meanwhile, Yao et al. [16] proposed a novel query optimization approach leveraging large models, showing that self-attention mechanisms could model global relationships between query operators more accurately, thereby significantly improving the quality of complex query plan selection. These advances indicate that GT can compensate for the limited receptive fields of traditional GNNs by modeling global structural dependencies. Moreover, as learning-based optimizers became more widely deployed, model security emerged as an important concern. Oliynyk et al. [17] conducted in-depth research on defense strategies for machine learning models, providing valuable insights for building industrial-grade database optimization systems.

Table 1 summarizes the differences between the proposed GT-MuZero framework and representative mainstream techniques.

Table 1: Comparison of existing query optimization approaches and the proposed framework

Method Category	Representative Literature	Core Techniques and Contributions	Typical Quantitative Metrics (SOTA)	Limitations and Missing SOTA Aspects
Traditional Optimization	Rahman et al. [6]	Based on static statistical information and hard-coded heuristic rules.	geometric mean performance ratio (GMPR): 1.0 (baseline); Optimization latency: < 1 ms	Static and rigid: unable to adapt to dynamic data distributions; low accuracy under complex queries.
Domain-Specific Machine Learning	Choi et al. [9]	Applied traditional ML models such as K-NN to specific query types.	Effective in specific scenarios; no universal GMPR metric reported.	Weak generalization: limited to specific query types; incapable of modeling complex joins relationships.
Graph-Based Reinforcement Learning (GNN-RL)	Sassi et al. [10]; Khan et al. [11]; Yan et al. (survey) [12]	Modeled query plans as graphs; used GNNs to extract local features and combined with DRL for policy learning.	Accuracy: 70%–75%; Sample requirement: 50,000+	Representation bottleneck: over-smoothing issue in GNNs leads to loss of deep structural information. Efficiency bottleneck: extremely low sample efficiency, requiring tens of thousands of real database interactions.
Model-Based RL and Robust Planning	Marcus & Papaemmanouil [13]	Emphasized robustness of model-based DRL in complex system optimization; provided theoretical foundation for virtual planning.	Conceptual contribution; improved system robustness.	Lacked deep integration with advanced graph representations (e.g., Transformer).
Transformer-Based Global Semantic Modeling	Bhopale et al. [14]; Tan et al. [15]; Yao et al. [16]	Leveraged self-attention to capture global SQL semantics, or directly applied LLMs for end-to-end optimization.	High-precision semantic understanding; logical reasoning capability under LLM-driven frameworks.	Decoupling of representation and planning: strong representation capacity not integrated with efficient virtual planning. High inference overhead: LLM latency unsuitable for real-time decision-making.
Practical Systems and Security Considerations	Marcus et al. (Bao) [7]; Oliynyk et al. [17]	Bao: proposed a learning-based guidance mechanism to enhance optimizer practicality; Oliynyk: focused on model security and defense strategies.	Bao GMPR: 2.1–2.4; No direct performance metrics for security studies.	Bao: still relied on traditional optimizers with limited representation capacity. Security: orthogonal to performance optimization, but critical for industrial deployment.
GT-MuZero (This Study)	This Study	Deep integration: combined the global representation capability of GT with the efficient virtual planning of MuZero.	GMPR: 2.81; Accuracy: 96.73%; Sample requirement: 10,000	Filled the SOTA gap: first unified framework integrating top-tier global representation models with efficient virtual planning paradigm, simultaneously addressing both representation and efficiency bottlenecks.

In summary, although learning-based query optimization has made notable progress in representation learning (e.g., GNNs and Transformers) and practical system deployment (e.g., Bao), existing state-of-the-art methods face a core technical gap. Specifically, strong global representation capability ("seeing clearly") has not been deeply integrated with efficient virtual planning ("thinking ahead"). The proposed GT-MuZero framework aims to bridge this gap. By combining GT-based global semantic modeling with MuZero's forward virtual planning, the framework performs efficient rollouts in a learned latent space. This design achieves a dramatic reduction—by an order of magnitude—in sample requirements while significantly surpassing prior optimization performance, laying a solid practical foundation for next-generation, highly autonomous intelligent database systems.

3 GT-driven SQL query optimization method with muzero

3.1 Problem formalization and model design

The core scientific question of this study was how to capture the complex global dependencies of SQL queries while significantly reducing the reliance of learning-based optimizers on costly interactions with real database environments. To address this issue, the GT-MuZero framework was designed with two primary motivations:

a) GT: The GT mitigated the semantic information loss caused by the over-smoothing problem of traditional GNNs when handling multi-join queries, thereby providing high-quality global state representations.

b) MuZero algorithm: MuZero constructed an internal world model to perform forward planning, which alleviated the long-standing problem of low sample efficiency in RL for query optimization.

Correspondingly, the problem was formally defined as follows: given an SQL query Q, the objective was to determine an optimal join-order sequence $a_{1:T}$ that minimized the execution cost (latency). This process was modeled as a Markov Decision Process (MDP) [18], characterized by the five-tuple (S,A,P,R,γ). Specifically, S denoted the global state space generated by the GT, and A represented the action space consisting of valid join operations. To enhance modeling rigor, Table 2 summarizes the principal mathematical symbols used in this study along with their definitions.

Table 2: Description of core mathematical symbols

Symbol	Definition	Dimension / Type
S	State space composed of global feature vectors encoded by GT	Set of vectors
$a_{1:T}$	Sequence of actions (join operations) from the initial state to the terminal state	Sequence
h_i	Initial raw feature vector of node (i)	d_h=128
p_i	Structured encoding based on Laplacian eigenvectors	k=8
d_{model}	Hidden dimension of the Transformer layers and latent space	256
$g(\cdot)$	MuZero Dynamics function	MLP
$f(\cdot)$	MuZero Prediction function	MLP

Based on this formalization, this study designed the end-to-end model architecture illustrated in Figure 1.

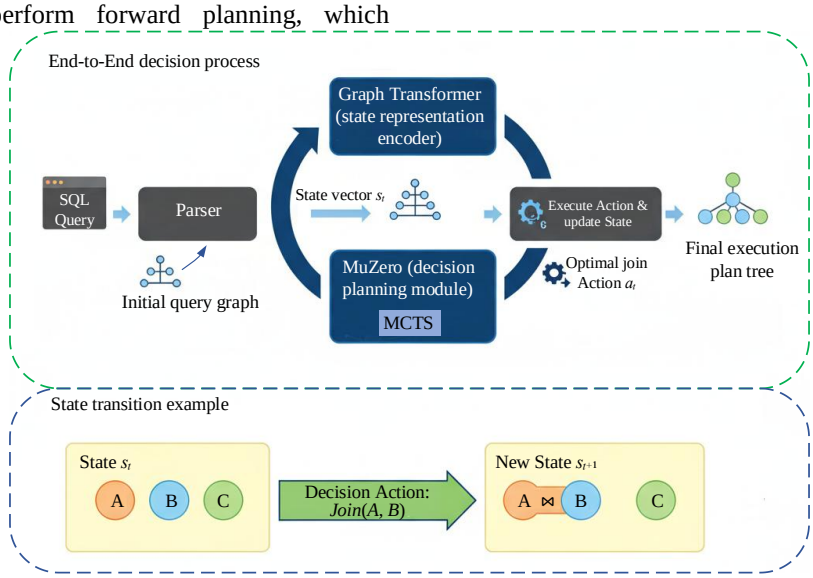


Figure 1: Overall model architecture and state transition illustration

3.2 State representation module using the GT

A high-quality state representation is critical for enabling the decision module to effectively "understand" SQL queries. The traditional GNN propagates information by layer-by-layer neighborhood aggregation, which will lead to problems such as excessive smoothness and limited receptive field when dealing with complex query graphs [19]. In order to overcome these limitations, this study adopts GT as the state representation module. Its main advantage lies in the self-attention mechanism, which allows any two nodes in the graph to directly calculate the influence weight between them. The architecture of state representation module based on GT is shown in Figure 2.

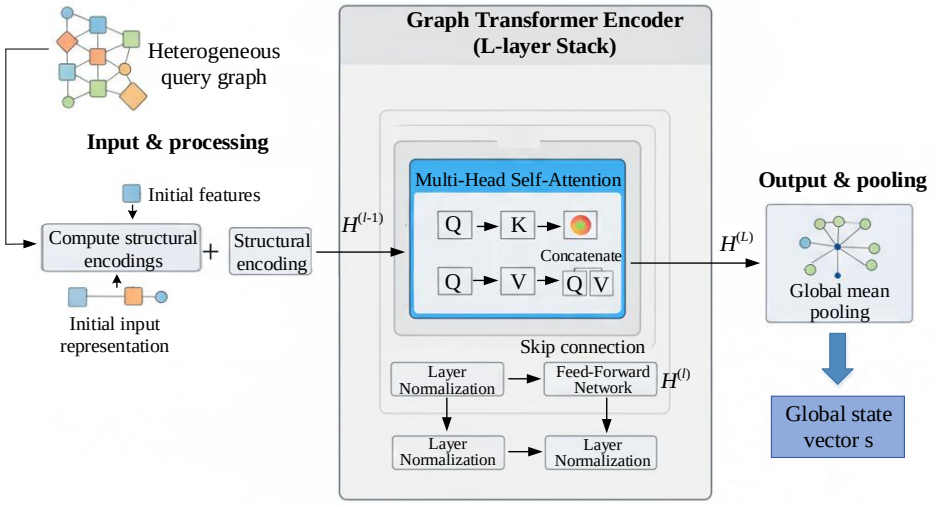


Figure 2: State representation module based on GT

Before applying the GT, the input SQL query is first parsed into a heterogeneous graph. Nodes in this graph are categorized into different types, such as table nodes, filter condition nodes, and join condition nodes, each containing its own initial feature vector $h_i \in R^{d_h}$, where i is the node index and d_h is the feature dimension.

To obtain high-fidelity state representations, this study designs a pipeline that converts SQL queries into heterogeneous graphs.

a) Node definition. Each SQL query is parsed into three node types:

(1) Table nodes, which store table size and cardinality statistics. (2) Filter nodes, which include selectivity estimates and operator types. (3) Join nodes, which represent join operations such as Hash Join and Nested Loop.

b) Edge definition. Edges are created based on primary–foreign key relationships and query predicates. This forms a heterogeneous graph that captures structural dependencies.

c) Complex syntax handling.

For GROUP BY and ORDER BY clauses, the framework generates dedicated attribute aggregation nodes. For nested queries, a hierarchical graph construction strategy is used. Subqueries are abstracted as composite feature nodes within the parent graph. This design preserves structural compactness and semantic integrity.

To clearly illustrate the transformation process of complex SQL syntax, a representative TPC-DS nested query was taken as an example: SELECT ... FROM TableA WHERE id IN (SELECT a_id FROM TableB GROUP BY a_id). Within the proposed framework, this query was transformed into a hierarchical heterogeneous graph. First, the subquery was abstracted as a *composite feature node* with aggregation attributes. Second, the GROUP BY clause was represented by introducing a dedicated *Aggregation Node* connected to the TableB node. This aggregation node encoded both the type of aggregation operator and the estimated cardinality reduction ratio. The nested logic (e.g., IN or EXISTS) was modeled as a *semi-join edge* connecting the main-query table node to the subquery composite node. This design preserved the compactness of the graph structure while fully retaining the logical semantics of the SQL statement.

Since the Transformer architecture is inherently permutation-invariant and lacks explicit positional bias, structural encoding based on Laplacian eigenvectors was introduced to incorporate graph topology information. Specifically, for the initial query graph, the normalized Laplacian matrix was computed as: $L = I - D^{-1/2}AD^{-1/2}$, where A denoted the adjacency matrix and D the degree matrix. The first k=8 smallest non-trivial eigenvectors of L were selected as structural features, denoted by $p_i \in R^{d_h}$. The initial node input representation defined in Equation (1) was expressed as:

$$x_i^{(0)} = h_i + p_i \quad (1)$$

This design projected the *k*-dimensional structural information into the d_{model}=256-dimensional latent space via a lightweight multilayer perceptron and combined it additively with the linearly transformed raw node features. Such integration ensured that the model captured both attribute-level information (e.g., table cardinality, selectivity) and the spatial topological position of each node within complex join trees.

The core of this module is the Multi-Head Self-Attention mechanism. For the node input representation matrix $X^{(l-1)} \in R^{N\times d_{model}}$ at layer l, this study first projects it into three distinct subspaces—Query (Q), Key (K), and Value (V)—via learnable linear transformation matrices $W_Q^{(l)}, W_K^{(l)}, W_V^{(l)} \in R^{d_{model}\times d_k}$, as shown in Equations (2)-(4):

$$Q^{(l)} = X^{(l-1)}W_Q^{(l)} \quad (2)$$

$$K^{(l)} = X^{(l-1)}W_K^{(l)} \quad (3)$$

$$V^{(l)} = X^{(l-1)}W_V^{(l)} \quad (4)$$

where *N* denotes the number of nodes in the graph, d_{model} is the model's hidden dimension, and Dd_k is the key vector dimension. The attention weights are then computed as Equation (5):

$$Attention(Q^{(l)}, K^{(l)}, V^{(l)}) = softmax\left(\frac{Q^{(l)}(K^{(l)})^T}{\sqrt{d_k}}\right)V^{(l)} \quad (5)$$

where $Q^{(l)}\left(K^{(l)}\right)^T$ calculates the raw attention scores between each pair of nodes, with element (*i*,*j*) representing the importance of node *i* to node *j*. The scaling factor $\sqrt{d_k}$ prevents excessively large dot products, stabilizing the softmax gradients during training.

To allow the model to jointly learn information from multiple representation subspaces, a multi-head attention mechanism is used. *Q*, *K*, and V are further split into *H* "heads," attention is computed in parallel, and the results are concatenated. The output of a single attention head *h* is given by Equation (6).

$$head_h = Attention(Q_h, K_h, V_h) \quad (6)$$

The outputs of all heads are concatenated and linearly transformed by $W_O \in R^{Hd_v\times d_{model}}$ to produce the final multi-head attention output, as Equation (7).

$$MultiHead(Q,K,V) = Concat(head_1,...,head_H)W_O \quad (7)$$

where d_v denotes the dimension of the value vector, typically set to d_k. The multi-head attention output is then passed through a residual connection and LN, followed by a Feed-Forward Network (FFN) for non-linear transformation. A complete GT layer can be summarized as Equations (8)-(9).

$$Z^{(l)} = LN(X^{(l-1)} + MultiHead(X^{(l-1)})) \quad (8)$$

$$X^{(l)} = LN(Z^{(l)} + FFN(Z^{(l)})) \quad (9)$$

After processing through *L* GT encoder layers, this study obtains the final updated embeddings $x_i^{(L)}$ for each node in the graph. To generate a fixed-dimensional vector representing the current state of the entire query plan, a graph pooling layer is introduced. The approach adopts global average pooling, computing the element-wise mean of all node embeddings, as shown in Equation (10).

$$s_0 = \frac{1}{N}\sum_{i=1}^{N} x_i^{(L)} \quad (10)$$

The resulting s_0 is a global graph embedding vector that condenses both the structural and semantic information of the query graph. This vector serves as the final representation of the current query plan state and is used as the output of the representation function *h* in the MuZero algorithm, providing a high-quality, interpretable input for subsequent decision-making and planning modules.

3.3 Decision planning module supported by the muzero algorithm

The scientific motivation of this study lies in integrating the GT with the MuZero algorithm (Schrittwieser et al. (Nature 2020)) [20] to fundamentally address the inherent tension in learning-based query optimizers between complex topological representations and costly real-environment interactions. To overcome the semantic loss bottleneck caused by traditional GNNs, which suffer from limited receptive fields and "over-smoothing," the GT provides high-fidelity, panoramic state embeddings for SQL plan trees through a global self-attention mechanism. This ensures that the decision-making module can perceive long-range cost dependencies among operators.

Building on this, the core novelty of the framework lies in using the MuZero–constructed internal "world model" to perform offline look-ahead planning in the latent space. This coupling of deep global representation and virtual forward planning allows the system to simulate operator execution and predict long-term returns in an abstract space based on the precise semantic information generated by the GT.

When selecting an action for the current state, MuZero does not directly rely on the raw policy p_0 output by the prediction function *f*. Instead, it performs a brief but effective MCTS planning process to enhance the policy. At each node *s* in the search tree, the next action *a* is chosen using the Polynomial Upper Confidence Trees (PUCT) algorithm, which balances exploitation of known good actions and exploration of potential unknown actions, as shown in Equation (11):

$$a^* = \underset{a}{argmax}\left[Q(s,a) + P(s,a)\cdot\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}\cdot\left(c_1 + log\left(\frac{\sum_b N(s,b)+c_2+1}{c_2}\right)\right)\right] \quad (11)$$

$Q(s,a)$ represents the average Q-value of action *a*, $P(s,a)$ is the prior probability from the prediction function *f*, $N(s,a)$ is the visit count of action *a*, and c_1 and c_2 are hyperparameters controlling the degree of exploration. Each MCTS iteration consists of three stages: selection, expansion, and backup. Starting from the root node, the PUCT formula is recursively applied to select a path to a leaf node; at the leaf node, the prediction function *f* is called to expand new child nodes; then, the obtained value *v* is backpropagated along the path, updating statistics for all parent nodes. The Q-value and visit count are updated according to Equations (12)-(13):

$$N(s,a) \leftarrow N(s,a) + 1 \quad (12)$$

$$Q(s,a) \leftarrow \frac{N(s,a)\cdot Q(s,a)+v}{N(s,a)+1} \quad (13)$$

After a fixed number of simulations (e.g., hundreds), MCTS constructs a search tree with deep reasoning. The final action is not taken from the raw policy p_0, but rather from the visit count distribution π at the root node after MCTS enhancement, typically adjusted with a temperature parameter τ, as shown in Equation (14):

$$\pi(a \mid s_0) = \frac{N(s_0,a)^{1/\tau}}{\sum_b N(s_0,b)^{1/\tau}} g(s_t, a_{t+1}) f(s_t) \quad (14)$$

During training, the goal is to make the predictions of the three networks as close as possible to the MCTS-enhanced search results and the true final reward z. This is achieved by minimizing a combined loss function consisting of three components: value loss, policy loss, and reward loss, as expressed in Equation (15):

$$L = \sum_{k=0}^{K} \left((z_k - v_k)^2 - \pi_k \cdot log(p_k) + (r_{k-1} - r_k)^2 \right) \quad (15)$$

The value loss encourages the predicted value v_k to match the MCTS-enhanced value or final return z_k. The policy loss (cross-entropy) encourages the predicted policy p_k to mimic the MCTS search policy π_k. The reward loss optimizes the dynamics function g for reward prediction.

A pseudocode flow of the decision planning module supported by the MuZero algorithm is shown below.

Algorithm: GT-MuZero based SQL Query Optimization Planning

```
# Input: SQL Heterogeneous Graph (o_t), Database Statistics, Config
# Output: Optimal Join Action (a_t), Updated Model Parameters (theta)

class GT-MuZeroOptimizer:
  def _init_(self, theta):
    self.h = GraphTransformerEncoder() # Representation function: SQL Graph -> State s
    self.g = DynamicsNetwork()      # Dynamics function: (s, a) -> (r, s')
    self.f = PredictionNetwork()    # Prediction function: s -> (p, v)

  def plan_join_order(self, sql_graph):
    # 1. State Representation via GT
    # Encodes SQL graph using Laplacian encodings and global self-attention
    s_0 = self.h(sql_graph)
    root_node = Node(state=s_0)

    # 2. Monte Carlo Tree Search (MCTS) in Abstract Hidden Space
    for _ in range(Config.num_simulations):
      node = root_node
      search_path = [node]

      # Selection Phase: Balance exploration & exploitation via PUCT formula
      while node.is_expanded():
        action, node = node.select_child_via_puct()
        search_path.append(node)

      # Expansion & Evaluation: Planning in the "World Model"
      # No real DB execution; using dynamics g and prediction f for virtual feedback
      parent = search_path[-2] if len(search_path) > 1 else None

    # 3. Action Selection: Choose action based on MCTS visit count distribution
    return select_action_by_visit_count(root_node, temp=Config.temperature)

  def update_model(self, buffer):
    # Joint Loss Optimization (Eq. 15): Optimizing Value, Policy, and Reward losses
    loss = compute_total_loss(buffer, self.h, self.g, self.f)
     adam_optimizer.step(loss)
```

In the integrated implementation of the MuZero architecture, the core innovation lay in the construction of an internal world model through the dynamics network g and the prediction network f. The dynamics network $g(s_t, a_{t+1})$ adopted a three-layer multilayer perceptron (MLP) architecture, with 256 hidden neurons in each layer. Each layer incorporated a ReLU activation function and Layer Normalization (LN). Its purpose was to infer the next latent state s_{t+1} and the immediate reward r from the current latent state and the selected action. In parallel, the prediction network $f(s_t)$ consisted of an independent Policy Head and a Value Head. The Policy Head generated the action distribution π using a Softmax function. The Value Head estimated the scalar value v of the current state. To ensure stable convergence when handling SQL

execution latencies that varied by several orders of magnitude, the reward signal r was normalized using a global Min–Max strategy based on historical observations. This procedure mapped all reward values to the interval [0,1]. This tightly coupled parameterization enabled high-frequency virtual look-ahead planning within an abstract latent space. The process operated without interaction with the physical database engine. As a result, the framework significantly reduced sample requirements and captured complex long-range cost dependencies among operators.

4 Results and discussion

4.1 Experimental setup

This study adopted the widely recognized TPC-DS benchmark and used the *dsdgen* tool to generate a 100 GB dataset, which was deployed on PostgreSQL 14. The ANALYZE command was executed to ensure that database statistics remained accurate and up to date. To provide supervisory signals, an automated data collection pipeline was constructed. A total of 12,500 query instances were generated from 99 query templates and were split into training, validation, and test sets at a ratio of 8:1:1. The pipeline leveraged the *pg_hint_plan* extension to enforce the execution of multiple logically equivalent query plans. The reciprocal of the true execution latency was computed and normalized using Min–Max scaling to serve as the reward signal. Logical equivalence among plans was strictly verified by comparing operator output checksums. During preprocessing, each SQL query was parsed into a heterogeneous graph. Node features, such as table cardinality and filter selectivity, were derived from database statistical catalogs, including pg_class.

All experiments were conducted on a high-performance server equipped with an NVIDIA A100 80GB GPU, 512 GB of memory, and an Intel Xeon Platinum 8358 CPU. The software environment was based on Ubuntu and included Python, PyTorch, and PyTorch Geometric as core components. To ensure fairness and reproducibility, all models, including baseline methods, were executed under the same hardware and software configuration, with the random seed fixed at 42. The key hyperparameters of the proposed GT-MuZero model, including the number of Transformer layers, the number of attention heads, and the number of MCTS simulations, are detailed in Table 3 to facilitate reproducibility in future research.

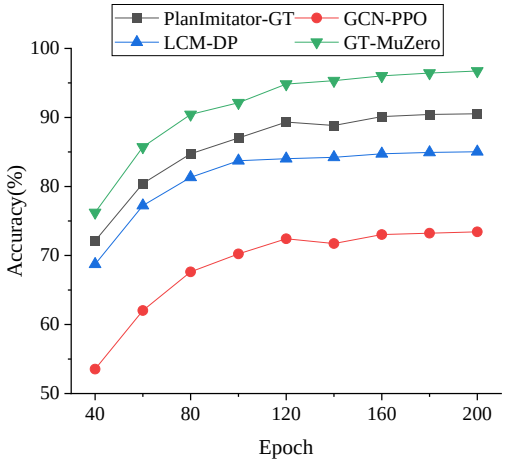
Table 3: Model hyperparameter settings

Module	Parameter	Value
GT	Model hidden dimension (d_model)	256
	Number of attention heads (H)	8
	Number of Transformer layers (L)	6
	Feed-forward network dimension	512
MuZero	Number of MCTS simulations	100
	PUCT constants (c_1, c_2)	1.25, 19652
Training	Optimizer	AdamW
	Learning rate	$1e^{-4}$
	Batch size	64
	Epochs	200
	Discount factor (γ)	0.99

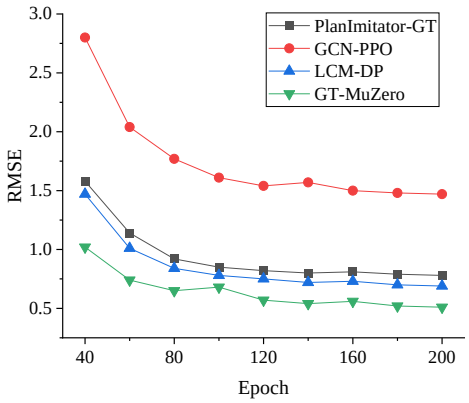
4.2 Performance evaluation

(1) Prediction Accuracy of Different Models

In order to comprehensively evaluate the proposed GT-MuZero algorithm, we compare it with the following baseline algorithms: graph convolution network and near-end strategy optimization (GCN-PPO) [21], learning cost model based on dynamic programming (LCM-DP) [22], and plan simulator-GT based on GT. The evaluation indexes include accuracy and root mean square error (RMSE) [23], and the results are shown in Figure. 3.



(a) **Policy consistency accuracy across models**



(b) Root Mean Square Error (RMSE) in value and cost prediction

Figure 3: Comparative performance evaluation of different models

As illustrated in Figure 3, GT-MuZero demonstrated the fastest convergence speed and the highest predictive accuracy among all models. Its final policy consistency accuracy reached 96.73%, and the value prediction RMSE was only 0.51. These results significantly outperformed PlanImitator-GT (90.53%) and GCN-PPO (73.43%, RMSE 1.47). The findings indicate that GT-MuZero maintained strong stability and reliability when handling complex joins and filtering predicates in the TPC-DS benchmark. The model accurately estimated candidate plan costs and consistently selected optimal decisions. These results empirically validated the effectiveness and generalization capability of integrating deep graph-level semantic understanding with MuZero-based planning in complex query optimization scenarios.

(2) Comprehensive Performance Evaluation and Ablation Study

To further validate the superiority of GT-MuZero, the core performance comparison and ablation study were jointly analyzed. The comparative models included: GNN-MuZero, which replaced the GT with a traditional Graph Convolutional Network (GCN); GT-PPO, which replaced MuZero with the model-free RL algorithm PPO. As shown in Table 4, GT-MuZero outperformed all baseline models across key evaluation metrics. The ablation analysis revealed that the GT component increased policy consistency accuracy from 82.15% to 96.73%, effectively mitigating the over-smoothing problem inherent in GNN architectures. The MuZero component reduced the required convergence samples from 40,000 to 10,000, thereby demonstrating the substantial improvement in sample efficiency achieved through latent-space virtual planning. All reported metrics were averaged over 10 independent runs. Statistical validity was confirmed using the Shapiro–Wilk normality test.

Table 4: Comprehensive performance comparison and ablation results of learning-based optimizers

Model	Core Architecture	Policy Consistency Accuracy (%)	Value Prediction RMSE	GMPR	Samples Required for Convergence
PostgreSQL	Rule- and cost-based	–	–	1.00	–
GCN-PPO	3-layer GCN + PPO	73.43 ± 1.25	1.47 ± 0.08	1.47	50,000 +
LCM-DP	MLP + Dynamic Programming	84.15 ± 1.10	0.95 ± 0.06	2.15	35,000
GNN-MuZero	3-layer GCN + MuZero	82.15 ± 0.92	0.78 ± 0.05	1.98	15,000
GT-PPO	GT + PPO	90.53 ± 0.85	0.82 ± 0.05	2.12	40,000
GT-MuZero (This Study)	GT + MuZero	96.73 ± 0.42	0.51 ± 0.03	2.81	10,000

(3) Comparison of Core Performance Metrics

To further compare algorithm performance using Query Execution Latency, four representative TPC-DS queries were selected for analysis: Q11 (simple), Q25 (medium), Q72 (complex), and Q5 (special), as shown in Figure 4.

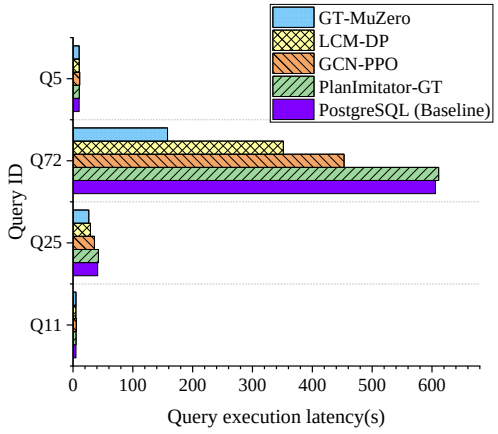


Figure 4: Execution latency comparison for representative queries under different algorithms

As shown in Figure 4, the detailed microscopic analysis of typical query execution delay clearly shows the performance of each optimizer under different query complexity. For simple query (Q11), the performance of all models is equivalent, and there is no significant difference. However, with the query complexity increasing to medium (Q25) and complex (Q72) levels, the advantages of learning-based optimizer are gradually emerging, among which the GT-MuZero model proposed in this study is particularly outstanding. For the representative complex query Q72, GT-MuZero reduces execution time dramatically from 605.8 seconds (baseline) to 157.9 seconds, achieving a performance improvement of over 3.8×. Moreover, it demonstrates consistent superiority across all categories of complex queries in the TPC-DS benchmark. It is worth noting that for a specific query, Q5, the traditional PostgreSQL heuristic rules outperform, highlighting that each optimizer has scenarios where it excels.

To further validate the reliability of GT-MuZero's performance gains over the PostgreSQL baseline, this study conducted 10 independent runs on four representative queries with varying levels of complexity (Q11, Q25, Q72, and Q5). Paired-sample t-tests were then

performed based on the collected results. The statistical outcomes are summarized in Table 5.

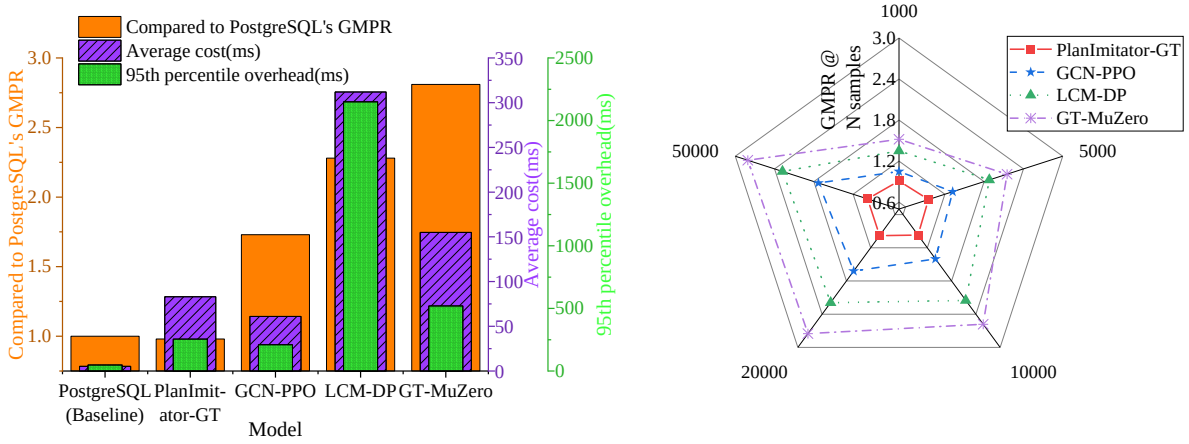
Table 5: Paired-sample t-test results comparing GT-muzero and postgresql baseline

Query ID	Complexity	PostgreSQL Mean (s)	GT-MuZero Mean (s)	Performance Gain (%)	t Statistic	p-value	Significance
Q11	Simple	5.00	4.90	2.0%	1.12	0.294	ns
Q25	Medium	41.30	26.40	36.1%	12.45	< 0.001	***
Q72	Complex	605.80	157.90	73.9%	45.82	< 0.001	***
Q5	Special	10.20	10.30	-1.0%	-0.85	0.418	ns

*Note: Significance levels are indicated as: *** for p < 0.001, ns for p > 0.05.*

As shown in Table 5, GT-MuZero achieves highly significant improvements ($p < 0.001$) for both complex queries (Q72) and medium-complexity queries (Q25), with t statistics of 45.82 and 12.45, respectively. These results provide strong evidence that the model's performance gains in multi-table joins and large-scale data scans are not incidental but stem from its superior global semantic representation and look-ahead planning capabilities.

To further assess each model's performance and learning efficiency on the full test set, Figure 5 provides a comparison of GT-MuZero with the baseline algorithms. The comparison focuses on three metrics: GMPR, optimization overhead, and sample convergence speed.



(a) Comparison of GMPR and optimization overhead

(b) GMPR convergence trajectories

Figure 5 Comparative Analysis of Overall Performance and Sample Efficiency on the Full Test Set.

As shown in Figure 5a, the macro evaluation of the model performance shows that the GMPR of GT-MuZero reaches 2.81. Although its optimization overhead (average 155 milliseconds) is higher than that of the traditional optimizer, compared with the significant execution time savings it brings, this cost can be ignored, and it is far lower than the LCM-DP model with the same excellent performance, showing excellent practicability. More importantly, Figure 5b highlights the excellent sample efficiency of GT-MuZero: with only 10,000 training samples, its performance has surpassed the final results of all other learning-based algorithms trained with 50,000 samples. This efficiency significantly reduces the high-cost actual query interaction required for model training, and verifies the potential of the model-based RL method proposed in this study in building an efficient and practical intelligent query optimizer.

(4) Computational Cost and Hyperparameter Sensitivity Analysis

The computational cost of the models is summarized in Table 6. Although GT-MuZero incurs additional inference overhead due to MCTS simulations (average single-query inference latency of 155 ms), this cost is negligible compared with the hundreds of seconds saved for terabyte-scale queries. Sensitivity analysis shows that performing 100 MCTS simulations achieves an optimal balance between performance and latency.

Table 6: Computational cost and inference latency comparison

Model	GPU Hours	Average Inference Latency (ms/query)	Peak Memory Usage (GB)	Failure Rate (Simple Queries)

PostgreSQL (Heuristic)	-	< 1	< 0.1	0%
GCN-PPO	18.5	45	1.2	8.40%
GT-MuZero	12.2	155	2.4	5.20%

4.3 Discussion

GT-MuZero achieved a 3.8× performance gain on complex queries, such as Q72. This improvement stems primarily from the panoramic semantic modeling provided by the GT. Li et al. [24] emphasized that representing structured data as a graph was crucial for capturing complex logical dependencies. Building on this principle, GT-MuZero outperformed prior approaches, including Akioyamen et al. [25], who highlighted the "unreasonable effectiveness" of language models in query optimization, and Sulimov et al. [26], who proposed the generative subplan-based optimizer GenJoin. By leveraging global self-attention, GT-MuZero enables direct, lossless semantic propagation. Low-level filter operator selectivity can influence top-level join strategies regardless of operator depth. This design yields exceptional decision accuracy in computation-intensive tasks, such as multi-table joins. The approach represents a shift in query optimization from traditional pattern matching to intelligent agents capable of logical reasoning [27].

Quantitative failure analysis revealed the boundaries of this paradigm. For roughly 5.2% of extremely simple queries (e.g., single-table point queries Q5 or Q11), the inference overhead of the deep learning model (~155 ms) outweighed the modest gains from plan improvements. As a result, GT-MuZero slightly underperformed compared with PostgreSQL's native heuristic optimizer. This finding suggests that for high-frequency, sub-millisecond interactive workloads, the marginal benefits of panoramic representation and deep planning decrease as query complexity drops. To address this limitation, the study proposes an adaptive switching strategy. The system monitors query topology in real time. For shallow tree structures or queries involving few tables, it automatically reverts to lightweight algorithms. GT-MuZero is activated only for analytical workloads. This hybrid approach maintains maximum optimization potential without imposing unnecessary computational overhead.

From a systems-science perspective, GT-MuZero operates as a closed-loop decision system with an internal world model. Inspired by nonlinear optimal control, the framework treats MuZero's look-ahead planning as a self-correcting loop within the control cycle. This mechanism filters statistical drift and noise in dynamic workloads, ensuring robustness and bounded regret when handling unseen query patterns. The combination of high sample efficiency and deep semantic understanding makes GT-MuZero suitable for practical, system-level applications. These include cloud-native autonomous database tuning, adaptive resource scheduling, and cross-engine query co-optimization. By integrating predictive modeling with adaptive control, this architecture offers a solid paradigm for building next-generation high-performance, self-healing intelligent data management systems.

5 Conclusion

This study validated an intelligent SQL query optimization framework that integrated a GT with the MuZero algorithm. The results demonstrate that by combining deep global representation with model-based RL and virtual planning, GT-MuZero shifts the paradigm from "passive environmental feedback" to "active logical planning." Experiments confirmed that the framework reduced the requirement for real database interaction samples by 80% while significantly improving the accuracy of plan selection. Its core contribution lies in leveraging GT's global modeling capability to overcome information loss in deep execution trees and using MuZero's internal predictive model to simulate optimal execution paths in latent space. This enables performance beyond the state-of-the-art on complex analytical queries.

Despite these advances, several limitations remain. First, in extremely simple scenarios, such as single-table point queries, the model's inference overhead (~155 ms) can offset the optimization gains, illustrating the diminishing marginal returns under low-complexity workloads. Second, the current evaluation focuses primarily on PostgreSQL and the TPC-DS benchmark. Its generalizability to distributed or heterogeneous database management systems and the potential risk of overfitting require further investigation. Future research will focus on developing adaptive computation switching mechanisms that allow the system to select the appropriate optimization depth based on query complexity. In parallel, efforts will explore model compression techniques to reduce inference overhead and aim for generalized deployment of the framework in cross-engine query optimization and large-scale cloud-native database tuning.

Founding

This work was supported by the Provincial_level First-Class Course "Database System" [2022]SXXH2022235, the Department of Science and Technology of Guizhou Province, China under Grant [2023]159, the Guizhou Province of the second batch of provincial "Golden Courses"(2023SJJK02), the Guizhou University Engineering Research Center ([2023] No. 041, the Guizhou Provincial Department of Education Youth Science and Technology Talents Growth Project, the Natural Science Research Project of Guizhou Provincial Department of Education - Youth Science and Technology Talent Growth Project "Research on Medical Image Classification Algorithm Based on Deep Learning" (Qianjiaohe KY[2022]328).

References

[1] Chen j. Construction and application of an economic intelligent decision-making platform based on artificial intelligence technology. Informatica, 2024, 48(9): 89-106. Https://doi.Org/10.31449/inf.V48i9.5705

[2] Taipalus t. The effects of database complexity on SQL query formulation. Journal of systems and

software, 2020, 165: 110576. Https://doi.Org/10.1016/j.Jss.2020.110576

[3] Vellanki r b. SQL server administration and maintenance: best practices for modern environments[j]. Journal of computer science and technology studies, 2025, 7(6): 839-848. Https://doi.Org/10.32996/jcsts.2025.7.99

[4] Lauri j, dutta s, grassia m, ajwani d. Learning fine-grained search space pruning and heuristics for combinatorial optimization. Journal of heuristics. 2023, 29(2):313-347. Https://doi.Org/10.1007/s10732-023-09512-z

[5] Huang, p. (2025). Actor–critic deep reinforcement learning for multi-objective intelligent irrigation scheduling: algorithm and edge-cloud management system. Informatica, 49(14): 379-394. Https://doi.Org/10.31449/inf.V49i14.11138

[6] Rahman MM, islam s, kamruzzaman m, joy ZH. Advanced query optimization in SQL databases for real-time big data analytics. Academic journal on business administration, innovation & sustainability. 2024, 4(3):1-14. Https://doi.Org/10.69593/ajbais.V4i3.77

[7] Marcus r, negi p, mao h, tatbul n, alizadeh m, kraska t. Bao: making learned query optimization practical. Proceedings of the 2021 international conference on management of data. 2021:1275-1288. Https://doi.Org/10.1145/3448016.3452838

[8] Milicevic b, babovic z. A systematic review of deep learning applications in database query execution. Journal of big data. 2024, 11(1):173. Https://doi.Org/10.1186/s40537-024-01025-1

[9] Choi d, wee j, song s, lee h, lim j, bok k, et al. K-NN query optimization for high-dimensional index using machine learning. Electronics. 2023, 12(11):2375. Https://doi.Org/10.3390/electronics12112375

[10] Sassi n, jaziri w. Efficient AI-driven query optimization in large-scale databases: a reinforcement learning and graph-based approach. Mathematics. 2025, 13(11):1700. Https://doi.Org/10.3390/math13111700

[11] Khan a, ke x, wu y. Graph data management and graph machine learning: synergies and opportunities. ACM sigmod record. 2025, 54(2):28-42. Https://doi.Org/10.1145/3749116.3749122

[12] Yan z, uotila v, lu j. Join order selection with deep reinforcement learning: fundamentals, techniques, and challenges. Proceedings of the vldb endowment. 2023, 16(12):3882-3885. Https://doi.Org/10.14778/3611540.3611576

[13] Marcus r, papaemmanouil o. Deep reinforcement learning for join order enumeration. Proceedings of the first international workshop on exploiting artificial intelligence techniques for data management. 2018:1-4. Https://doi.Org/10.1145/3211954.3211957

[14] Bhopale AP, tiwari a. Transformer based contextual text representation framework for intelligent information retrieval. Expert systems with applications. 2024, 238:121629. Https://doi.Org/10.1016/j.Eswa.2023.121629

[15] Tan j, zhao k, li r, yu JX, piao c, cheng h, meng h, zhao d, rong y. Can large language models be query optimizer for relational databases? Proceedings of the ACM on management of data. 2025, 3(6):1-28. Https://doi.Org/10.1145/3769771

[16] Yao z, li h, zhang j, li c, chen h. A query optimization method utilizing large language models. Arxiv preprint arxiv:2503.06902. 2025. Https://doi.Org/10.48550/arxiv.2503.06902

[17] Oliynyk d, mayer r, rauber a. I know what you trained last summer: a survey on stealing machine learning models and defences. ACM computing surveys. 2023, 55(14s):1-41. Https://doi.Org/10.1145/3595292

[18] Haviv m, puterman m l. An improved algorithm for solving communicating average reward markov decision processes. Annals of operations research, 1991, 28(1): 229-242. Https://doi.Org/10.1007/BF02055583

[19] Bose k, das s. Can graph neural networks go deeper without over-smoothing? Yes, with a randomized path exploration! IEEE transactions on emerging topics in computational intelligence. 2023, 7(5):1595-1604. Https://doi.Org/10.1109/tetci.2023.3249255

[20] Schrittwieser j, antonoglou i, hubert t, et al. Mastering atari, go, chess and shogi by planning with a learned model. Nature, 2020, 588(7839): 604-609. Https://doi.Org/10.1038/s41586-020-03051-4

[21] Krishnan s, yang z, goldberg k, hellerstein j, stoica i. Learning to optimize join queries with deep reinforcement learning. Arxiv preprint arxiv:1808.03196. 2018. Https://doi.Org/10.48550/arxiv.1808.03196

[22] Sun j, li g. An end-to-end learning-based cost estimator. Arxiv preprint arxiv:1906.02560, 2019.Https://doi.Org/10.48550/arxiv.1906.02560

[23] Chai t, draxler r r. Root mean square error (rmse) or mean absolute error (MAE). Geoscientific model development discussions, 2014, 7(1): 1525-1534. Https://doi.Org/10.5194/gmdd-7-1525-2014

[24] Li s, li l, geng r, yang m, li b, yuan g, he w, yuan s, ma c, huang f, li y. Unifying structured data as graph for data-to-text pre-training. Transactions of the association for computational linguistics. 2024; 12:210-228. Https://doi.Org/10.1162/tacl_a_00641

[25] Akioyamen p, yi z, marcus r. The unreasonable effectiveness of llms for query optimization. Arxiv preprint arxiv:2411.02862. 2024. Https://doi.Org/10.48550/arxiv.2411.02862

[26] Sulimov p, lehmann c, stockinger k. Genjoin: conditional generative plan-to-plan query optimizer that learns from subplan hints. Proceedings of the ACM on management of data. 2025, 3(4):1-25. Https://doi.Org/10.1145/3749165

[27] Shang w, huang x. A survey of large language models on generative graph analytics: query, learning, and applications. IEEE transactions on knowledge and data engineering. 2025, 37(12): 6799-6819. Https://doi.Org/10.1109/tkde.2025.36098