MACE: A Mamba-based Database-Agnostic Cost Estimator

Pohsun An
Keio University
Kanagawa, Japan
anpohsun13211@keio.jp

Shumpei Shiina
Toyota Motor Corporation
Tokyo, Japan
shumpei_shiina@mail.toyota.co.jp

Tatsuhiro Nakamori
Keio University
Kanagawa, Japan
tatsuhironm@keio.jp

Hideyuki Kawashima
Keio University
Kanagawa, Japan
river@sfc.keio.ac.jp

***Abstract*—Accurate query execution time prediction is crucial for scheduling and resource management. A state-of-the-art method, DACE, relies on a Transformer with a tree-structured mask derived from the query plan. While this design effectively encodes logical hierarchy, it limits information flow to local subtrees. Consequently, the model is blind to critical dependencies that transcend these boundaries—ranging from top-down context to lateral interactions (e.g., sideways information passing) and physical factors such as shared resource contention—which are key determinants of the actual execution time. To address this, we propose MACE, which applies Mamba's selective state-space mechanism to linearized query plans. By removing the tree mask, MACE dynamically captures both inherent tree structures and complex global interactions without the quadratic cost of self-attention. On the Zero-Shot benchmark, MACE reduces the 95th percentile and median Q-Error by 19.2% and 4.9%, respectively, while achieving up to a 1.66× speedup on GPU, demonstrating that enforcing tree connectivity is not a prerequisite for accurate prediction.**

***Index Terms*—Query execution time prediction, machine learning, query optimization**

I. Introduction

A. Motivation

Accurate prediction of query execution time is critical for resource allocation and scheduling in cloud environments [1]. Conventional optimizer estimates often fail on complex queries [2], motivating ML-based approaches [3]–[7]. However, achieving high accuracy alone is insufficient for practical deployments. Existing approaches often fall short in terms of throughput for cloud-scale cost estimation, portability across diverse database engines, and practical feasibility due to reliance on unrealistic assumptions.

Existing models often struggle to meet these requirements: recursive methods like Zero-Shot [6] suffer from inference overhead, while T3 [4] relies on unrealistic oracle cardinalities to improve robustness. We therefore choose DACE [3] as our primary baseline, as it effectively addresses these limitations. DACE replaces recursion with a Transformer [8] using a tree-structured mask (tree mask) derived from the query plan, enabling efficient batching while preserving dependencies. It ensures robustness by learning the error distribution of the optimizer's estimated cost (EDQO), rather than relying on raw data features, following the insight that correcting estimates is more reliable [9].

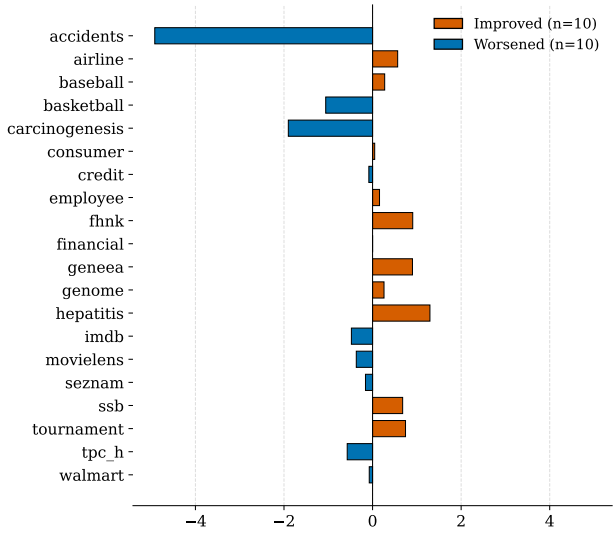

Fig. 1. Per-workload change in DACE's 95th percentile Q-Error on the Zero-Shot [6] benchmark (e.g., IMDB, TPC-H). The horizontal axis denotes the difference in Q-Error; negative values indicate that removing the tree mask improves (reduces) the tail error for that workload.

B. Reconsidering Structural Constraints

DACE enforces a tree mask that restricts a node's attention to itself and its descendants. While this design effectively encodes the logical hierarchy, it assumes that blocking information flow from outside the subtree is always beneficial. We question whether this structural constraint is necessary, or if it might inadvertently block global information. To investigate the impact of this design choice, we compared DACE's performance with and without the tree mask.

Fig. 1 shows the change in the 95th percentile Q-Error (Eq. (1)) when removing the tree mask from DACE across the Zero-Shot [6] benchmark workloads. In the figure, negative values indicate that removing this mask reduces the error, while positive values favor the mask. These results demonstrate that the mask's impact is *workload-dependent*.

In cases where the mask degrades performance, it acts as a barrier to global context, creating blind spots for dependencies that transcend parent-child links. Specifically, the mask blocks top-down context; a node cannot perceive an upstream `LIMIT`

clause that truncates its execution. It also obscures lateral interactions such as sideways information passing (e.g., Bloom filters), where one branch dynamically restricts a sibling's scan size. These logical dependencies, and physical factors like shared buffer contention, are invisible to the restricted subtree view. This aligns with findings that restricted attention creates information bottlenecks requiring deeper layers to compensate [10], [11]—a requirement DACE's single-layer architecture cannot meet. This leads to a hypothesis: **enforcing tree-structured connectivity is not a prerequisite for accurate performance prediction.**

C. Our Approach and Contributions

Guided by this hypothesis, we seek a model that propagates global context without hard-coded constraints, yet avoids the quadratic complexity of fully unconstrained self-attention. Simply removing the tree mask in a Transformer implies quadratic cost. Alternatively, while the information bottleneck caused by restricted attention could be mitigated by stacking more layers, doing so increases inference latency. This motivates us to adopt Mamba [12], a linear-time selective state-space model. Instead of enforcing a fixed topology, we represent parent-child relationships loosely through plan linearization, relying on Mamba's selective mechanism to dynamically retain or ignore information based on input content.

To implement this, we introduce **MACE** (Mamba-based database-Agnostic Cost Estimator). MACE linearizes query plans into sequences using bidirectional depth-first search (DFS) orders (top-down and bottom-up) augmented with structural features. By treating structure as *input features* rather than *architectural constraints*, MACE allows information to flow across structurally distant parts of the linearized sequence. This enables the model to capture dependencies between related operators that are invisible to DACE's tree mask.

Our work highlights the following contributions:

- We propose MACE, which adopts Mamba's selective state-space mechanism to estimate query cost without a tree mask, relaxing structural rigidity while capturing both logical hierarchy and hidden physical interactions.
- We introduce a bidirectional linearization strategy; while bottom-up ordering aligns with cost accumulation, combining it with top-down traversal improves stability and context propagation over unidirectional approaches.
- We empirically show that MACE outperforms DACE on the Zero-Shot benchmark, reducing p95 and p50 Q-Error by 19.2% and 4.9%, respectively, and achieving up to a 1.66× GPU speedup at large batch sizes.

The source code is available [16].

D. Organization

The rest of this paper is organized as follows. Section II gives the background, Section III presents MACE, Section IV reports the experiments, Section V discusses the results, and Section VI concludes.

II. Background: DACE

DACE [3] is a state-of-the-art method for query execution time prediction. It linearizes a plan tree via DFS, embeds node features (e.g., operator type, cost), and predicts execution time using a Transformer [8] encoder followed by an MLP. To preserve tree structure after linearization, DACE builds a tree mask based on ancestor-descendant relations in the query plan. In effect, a node can attend only to itself and nodes in its subtree, encouraging bottom-up aggregation of descendant information. DACE enables parallel processing of all plan nodes on the linearized sequence, avoiding the sequential dependency bottlenecks inherent in recursive message-passing architectures (e.g., Tree-LSTMs or GNNs).

Despite its advantages, DACE has limitations stemming from this architecture.

A. Limitation 1: Tree Mask Rigidity.

The tree mask hard-codes attention connectivity based on the plan tree structure. This rigidity introduces two issues:
(i) Assumption of Subtree Independence. The tree mask enforces a hierarchical aggregation, effectively treating sibling subplans as independent units. This prevents the model from capturing *lateral interactions* between structurally distant operators—ranging from logical dependencies like sideways information passing (e.g., Bloom filters) to physical factors such as resource contention (e.g., shared buffer pool usage)—which influence the runtime behavior.
(ii) Lack of Top-Down Context. The bottom-up nature of the mask prevents nodes from attending to their ancestors. Consequently, operators cannot incorporate *global context* from upstream nodes, such as a `LIMIT` clause that drastically reduces the actual cardinality processed by a subtree.

To isolate the impact of the tree mask, we compare DACE with and without it on the benchmark proposed by Zero-Shot [6]. We use Q-Error (Eq. (1)) and report the per-workload change in p95 Q-Error averaged over three seeds.

Fig. 1 shows the per-workload change in p95 Q-Error when removing the tree mask, i.e., $\Delta = \text{WithoutMask} - \text{WithMask}$, across the 20 workloads.

$$Q\text{-Error} = \max\left(\frac{\text{prediction}}{\text{actual}}, \frac{\text{actual}}{\text{prediction}}\right) \tag{1}$$

Removing the tree mask yields mixed effects: some workloads improve, while others regress. This variability suggests that while the tree mask can be restrictive, simply discarding it is not a universal solution. This highlights that neither strictly enforcing connectivity nor fully ignoring it is optimal across all scenarios.

B. Limitation 2: Insufficiency of Static Structural Features.

The mixed results above imply that neither enforcing connectivity (With Mask) nor ignoring structure (Without Mask) is sufficient. Motivated by the fact that query execution is inherently hierarchical [5]–[7], [13], we hypothesize that

TABLE I
STRUCTURAL FEATURES ADDED TO THE INPUT.

#	Feature	Description
1	Height	Node depth in the tree (root is 0)
2	Subtree Size	Node count in its subtree
3	Sibling Position	Position among sibling nodes (0-based)
4	Num Children	Number of child nodes
5	Subtree Height	Max distance to a leaf

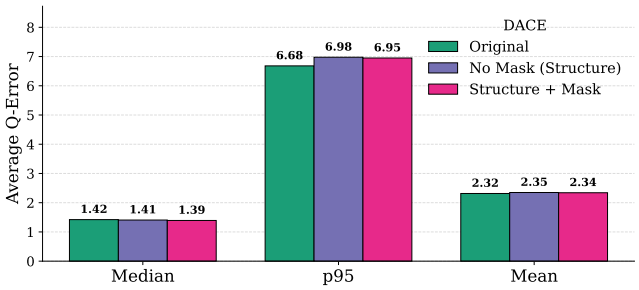

Fig. 2. Comparison of DACE Q-Error (p50, p95, mean) under combinations of the tree mask and structural features.

providing an explicit structural context—without enforcing connectivity—can strike an optimal balance. We augment DACE's input with the structural features listed in Table I and evaluate whether the model can learn to use structure softly.

Fig. 2 presents the Q-Errors of DACE under different configurations of the tree mask and structural features. Although structural features sometimes reduce the p50 Q-Error, they offer limited benefit in lowering the tail Q-Error (p95). This suggests that merely injecting static structural information is insufficient for robustness. Such features cannot capture how the *importance* of structural dependencies changes depending on the query content. Therefore, we argue that structural cues and long-range dependencies should be captured *adaptively* through the model itself, rather than solely relying on static rules or engineered features.

III. MACE

Mamba [12] is a state-space model (SSM) that achieves linear scaling in sequence length via parallel scans. Notably, unlike prior SSMs, it employs an input-dependent *selection* mechanism. This allows the model to selectively propagate or forget information based on the current input content.

We hypothesize that this architecture provides the necessary mechanisms to address the limitations of DACE identified in Section II. First, addressing the tree mask (Limitation 1) requires a mechanism that can capture global context with linear complexity. Second, correcting the insufficiency of static structural features (Limitation 2) necessitates a content-aware process to dynamically modulate feature importance. Based on this hypothesis, we propose MACE (Mamba-based database-Agnostic Cost Estimator). By leveraging the selection mechanism, MACE is *designed* to function as a learnable filter: it aims to retain relevant global context without relying on rigid masks (targeting L1) while adaptively modulating structural signals to approximate a soft topology (targeting L2).

MACE retains DACE's encoder-regressor template but replaces the masked Transformer with a bidirectional Mamba encoder for linear-time, mask-free context aggregation. Fig. 3 demonstrates the workflow using an example plan: the plan is linearized into a sequence, encoded by two Mamba streams (parent→child for constraints, child→parent for costs), and finally fused for the MLP to predict the execution time. Specifically, the architecture comprises four main parts: (i) input processing with the structural features in Section II, (ii) the bidirectional Mamba encoder, and (iii) an MLP that predicts cost from the fused representation, and (iv) the loss function used for optimization.

A. Input Processing

For each plan node, we form a single feature vector by concatenating three groups of features: (1) an operator embedding that represents the operator type; (2) optimizer estimates, namely `Total Cost`, `Plan Rows`, `Card Prod`, and `Plan Width`; and (3) structural features listed in Table I. These three groups are complementary: the operator embedding captures discrete operator semantics using a compact 7-d embedding; optimizer estimates provide strong local signals about resource use; and structural features encode the node's topological position to compensate for linearization. Specifically, among the optimizer estimates, `Card Prod` (the log-scaled product of child `Plan Rows`) summarizes fan-in and selectivity at multi-input operators, while `Plan Width` models the per-tuple payload affecting memory and I/O. All numeric features are robust-scaled using statistics from training workloads, with `Card Prod` using a log-sum fallback to avoid overflow. We pass the structural features through a small structure encoder (Linear + ReLU) and optionally apply feature-wise dropout. Finally, we concatenate all groups, project them to a fixed hidden size (e.g., 16), apply LayerNorm, and reshape the result into the sequence format required by Mamba.

B. Bidirectional Mamba Encoder

We use Mamba to encode the linearized plan sequence. First, we linearize each plan via depth-first search (DFS). Crucially, Mamba's selection mechanism allows the model to effectively skip over irrelevant intermediate nodes in the linearized sequence. This capability enables it to propagate signals between distant relatives (e.g., an ancestor and a descendant separated by other branches) as if they were directly connected, thereby recovering the tree topology without a tree mask. Since Mamba is unidirectional, a single pass cannot capture the full context of the tree. To address this, we employ a bidirectional strategy with distinct semantic roles.

Forward Pass (Parent→Child) This pass follows the control flow of the query engine. It captures top-down context, such as `LIMIT` constraints imposed by ancestors or pipeline blocking

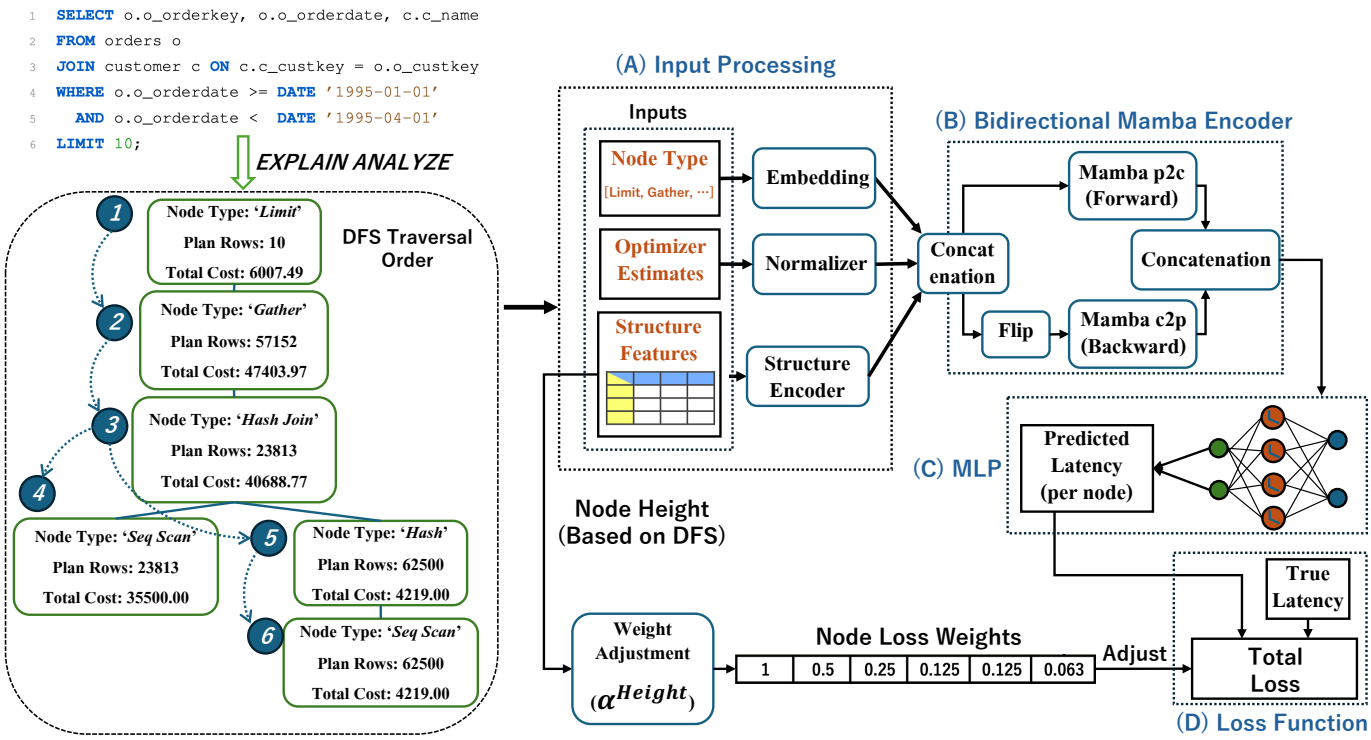

Fig. 3. Overview of MACE and an example workflow.

states (e.g., the build side of a Hash Join), which significantly alter the execution behavior of descendant operators.

Backward Pass (Child→Parent) This pass follows the data flow. It captures bottom-up signals, effectively accumulating resource costs (CPU cycles, I/O reads) and aggregating intermediate result sizes from leaf nodes up to the root, mirroring the actual cost accumulation process of a database optimizer.

We instantiate two Mamba blocks with identical architectures but independent weights to specialize in each direction, utilizing the official implementation [14]. We deliberately avoid weight tying because the information characteristics—propagating control constraints versus accumulating data costs—are fundamentally asymmetric. Finally, we fuse the outputs of these two passes via concatenation. This design provides the subsequent MLP with a comprehensive view of both the execution context (ancestors) and the data volume (descendants), effectively replacing the global attention mechanism of DACE with linear-time recurrence.

C. Multi Layer Perceptron (MLP)

The fused representation is fed to an MLP that outputs a scalar execution time prediction for each plan node. The head is a compact 3-layer MLP with ReLU activations and dropout, designed to keep the parameter count low. Our default configuration uses concatenation to preserve maximum information flow from both directions. Although this doubles the input dimension, the lightweight MLP head keeps the overall parameter budget comparable to the baseline. We generate predictions for all nodes to enable fine-grained supervision during training, although only the root prediction is used for the final query cost inference.

D. Loss Function

Following DACE, we minimize a weighted sum of node-wise errors. For GPU-efficient mini-batch training, we pad variable-length node sequences to a fixed length. This padding is unrelated to DACE's plan-derived tree mask; it is used only for batching. Since Mamba processes all positions in the padded sequence, we apply a loss mask so that padded tokens do not contribute to the objective.

For a plan p, the loss is

$$\text{loss}_p = \sum_{i=1}^{n} \left(L_{p,i} \cdot \left(\log(\text{QError}(\widehat{c}_{p,i}, c_{p,i})) \right)^2 \right), \qquad (2)$$

where $\widehat{c}_{p,i}$ and $c_{p,i}$ are the predicted and observed execution times for node i, and QError is defined in Eq. 1. DACE decays the node loss weight by tree depth using a hyperparameter α. Let $H_{p,i}$ be the depth of node i (root $= 0$); then

$$L_{p,i} = \alpha^{H_{p,i}}. \qquad (3)$$

We follow DACE and set $\alpha = 0.5$ (Node Loss Weights in Fig. 3). Finally, we square the log-Q-Error to stabilize heavy tails while emphasizing large errors.

IV. Experiments

A. Implementation

We implemented MACE and DACE using the official `mamba-ssm` [14] package and the official DACE implementation [15]. Query plans are linearized into per-node tensors,

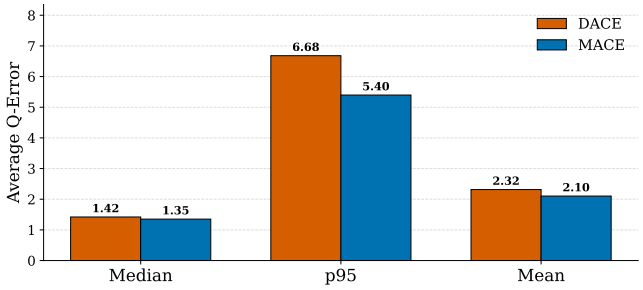

Fig. 4. Comparison of Q-Error (p50, p95, mean) between MACE and DACE (averaged over 20 datasets).

structural features, and loss masks, which are cached and reused across runs. Our implementation is available [16].

B. Experiment Setting

Workloads. We use the benchmark proposed by Zero-Shot [6], comprising 20 heterogeneous datasets with substantially diverse schemas.

Training Data. Following Zero-Shot, we generated 10,000 query plans per dataset (complex queries in Zero-Shot) and obtained their ground truth latencies using the `EXPLAIN ANALYZE` command. For training and evaluation, we treat one dataset as the test target and use its query plans for testing, while using the remaining 19 datasets for training (Leave-One-Out Validation).

Baselines. We compare MACE with DACE [3]. We do not re-run all prior baselines (e.g., Zero-Shot [6]) since DACE already provides extensive comparisons. Instead, we focus on DACE for a controlled evaluation and to isolate the impact of the tree mask on prediction accuracy.

Parameter Settings. The inputs are 16 dimensions (7-d embedding + 4-d numerical + 5-d structural). Mamba's parameters are $d_{\text{state}} = 32$, $d_{\text{conv}} = 4$, and expansion factor $E = 2$. The 3-layer MLP uses hidden sizes $W_1 = 128$ and $W_2 = 64$, $W_3 = 1$, and LoRA [17] is applied to all MLP layers with ranks $(16, 8, 4)$. Optimization uses AdamW with learning rate 1×10^{-3} and weight decay 1×10^{-4}. Results are averaged over three random seeds.

Hardware and Platform. PostgreSQL 17.5 was used as the DBMS. The database server for data collection was equipped with dual Intel Xeon Gold 6448Y CPUs (64 cores, 128 threads, 512 GB memory). Model training was performed on an NVIDIA H100 GPU (80 GB HBM3), and the implementation used Python 3.10.12 and PyTorch 2.3.1.

C. Prediction Accuracy

We report plan-level p50/p95/mean Q-Error averaged over the 20 leave-one-out runs (Fig. 4).

First, in the overall average (Fig. 4), compared to DACE, MACE reduces p50 from 1.42 to 1.35 (approximately 4.9%), p95 from 6.68 to 5.40 (approximately 19.2%), and mean from 2.32 to 2.10 (approximately 9.2%), with a large improvement in p95. This is consistent with the observation in the previous section that the tree mask can act as a restrictive constraint. That is, Mamba's sequential state updates and input-dependent selection mechanism can make it easier to incorporate the necessary information depending on the plan shape and operator types, without relying on a tree mask.

Next, in a per-dataset analysis, MACE improves p50 on 17 out of 20 datasets, indicating consistent gains across workloads. Improvements in p95 are dataset-dependent, but degradations are limited: p95 Q-Error decreases by about 2.8 on average in improved cases (13/20), whereas in degraded cases, it increases by only about 0.6 on average (7/20).

Overall, these results show that MACE is promising, with consistent p50 gains and a lower overall average p95 Q-Error.

D. Traversal Direction Study

Although MACE uses bidirectional Mamba, this subsection isolates the effect of traversal direction. After matching the input features to those used by DACE (one-hot vector, Plan Rows, Total Cost), we replace the encoder with unidirectional Mamba and compare two directions: (i) c2p (child→parent) and (ii) p2c (parent→child). We linearize each plan by DFS and feed the resulting node list; c2p simply uses the reversed order so that descendant information is incorporated first.

As shown in Fig. 5, c2p outperforms DACE, whereas p2c degrades substantially. This gap reflects a limitation of unidirectional sequence models: p2c encodes parents before their descendants, preventing bottom-up aggregation of subtree cues, while c2p accumulates descendant-side statistics and cost signals first, enabling more informative parent representations. We further discuss this in Section V.

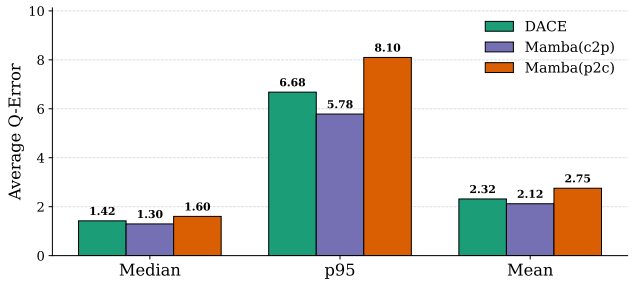

Fig. 5. Comparison of Q-Error (p50, p95, mean) across traversal directions: DACE vs. unidirectional Mamba (c2p/p2c).

E. Ablation Study

To assess the impact of the structural features in Table I, we compare MACE with structural features (With Structure) against a variant that removes all structural features except Height (Without Structure). We keep Height because it is required to compute the loss weight $L_{p,i} = \alpha^{H_{p,i}}$; thus this ablation isolates the remaining features (Subtree Size, Sibling Position, Num Children, and Subtree Height). We show p50/p95/mean Q-Error averaged over three seeds.

As shown in Fig. 6, removing structural features leads to a performance drop, confirming their utility. However, MACE

remains robust without them. This indicates that explicit structural features are a beneficial enhancement rather than a prerequisite, as the model successfully learns the core topology from the traversal order alone.

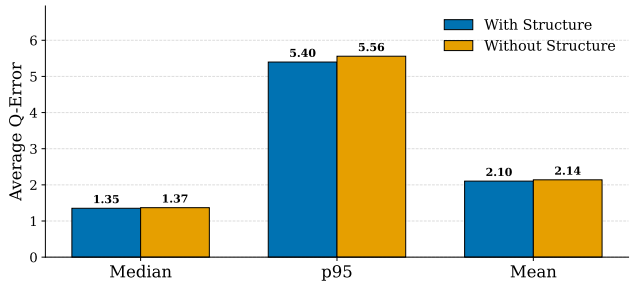

Fig. 6. Ablation study: comparison of Q-Error (p50, p95, mean) with vs. without structural features.

F. Inference Speed

We compare GPU inference latency of DACE and MACE over batch size $B \in \{64, 256, 512\}$ and query sequence length $L \in \{128, 256, 512\}$ (Table II). All reported latencies are averaged over 100 runs, following 50 warmup runs. We emphasize performance at larger batch sizes, as cloud-scale services typically aggregate requests to maximize throughput. While DACE is faster at small batches (e.g., $B = 64$), MACE becomes faster as B and/or L increase, reaching $1.66\times$ speedup at $B = 512, L = 512$. This efficiency stems from Mamba's linear-time parallel scan, which scales better than the quadratic attention mechanism.

TABLE II
INFERENCE LATENCY ON GPU. SPEEDUP = DACE/MACE.

Batch Size	L	DACE [ms]	MACE [ms]	Speedup
64	128	0.347	0.633	0.55
	256	0.454	0.628	0.72
	512	0.774	0.698	**1.11**
256	128	0.674	0.643	**1.04**
	256	1.231	0.904	**1.36**
	512	2.644	1.694	**1.56**
512	128	1.159	0.852	**1.36**
	256	2.375	1.633	**1.45**
	512	5.172	3.110	**1.66**

V. DISCUSSION

A. Mechanisms of Performance Gain

Our results show that the c2p stream alone outperforms DACE. We attribute this advantage to two factors: order alignment and dynamic gating.

Bottom-up Alignment and Structural Bottlenecks: Since query costs accumulate from leaves to the root [13], c2p processing allows parents to naturally encode complete subtree summaries. In contrast, DACE's tree mask forces single-step aggregation, which creates information bottlenecks in shallow networks [10], [11]. Since DACE relies on a single-layer Transformer [8], it is susceptible to these limitations.

Dynamic Gating Hypothesis: Unlike the tree mask, Mamba's selective mechanism dynamically filters information based on input content. This input-dependent compression likely retains relevant signals while discarding noise more effectively than a tree mask.

B. MACE Design Rationale and Robustness:

While the c2p stream alone captures the primary cost signal effectively, the p2c stream injects global context (e.g., root-level constraints). Our results show that MACE achieves the lowest tail error (p95), suggesting that difficult queries benefit from this global perspective. Although fusion slightly underperforms the pure c2p approach on p50 error, we adopt the bidirectional architecture as our primary method because mitigating extreme prediction errors is important for reliable query scheduling and risk management.

VI. CONCLUSION

We proposed MACE, a Mamba-based query execution time predictor that eliminates DACE's tree mask. On the Zero-Shot benchmark, MACE reduces the 95th percentile and median Q-Error by 19.2% and 4.9%, respectively, while achieving up to a $1.66\times$ speedup at large batch sizes on GPU. Traversal analysis further indicates that bottom-up aggregation is crucial. Even unidirectional c2p Mamba can outperform DACE, suggesting that, with appropriate linearization order, Mamba can effectively capture structural cues. Future work will explore more sophisticated fusion strategies to combine c2p's accuracy with bidirectional robustness, and extend MACE to jointly learn cardinality and cost.

REFERENCES

[1] R. Marcus et al. *Bao: Making learned query optimization practical*. SIGMOD, pp. 1275–1288, 2021.
[2] V. Leis et al. *How Good Are Query Optimizers, Really?* PVLDB, 9(3): 204–215, 2015.
[3] Z. Liang et al. *DACE: A Database-Agnostic Cost Estimator*. ICDE, pp. 4925–4937, 2024.
[4] M. Rieger et al. *T3: Accurate and Fast Performance Prediction for Relational Database Systems With Compiled Decision Trees*. PACMMOD, 3(3), Article 227, 2025.
[5] R. Marcus et al. *Plan-structured deep neural network models for query performance prediction*. PVLDB, 12(11): 1733–1746, 2019.
[6] B. Hilprecht et al. *Zero-shot cost models for out-of-the-box learned cost prediction*. PVLDB, 15(11): 2361–2374, 2022.
[7] Y. Zhao et al. *Queryformer: A tree transformer model for query plan representation*. PVLDB, 15(8): 1658–1670, 2022.
[8] A. Vaswani et al. *Attention is all you need*. NeurIPS, 2017.
[9] W. Wu et al. *Predicting query execution time: Are optimizer cost models really unusable?* ICDE, pp. 1081–1092, 2013.
[10] C. Yun et al. *O(n) Connections are Expressive Enough: Universal Approximability of Sparse Transformers*. NeurIPS, 2020.
[11] F. Barbero et al. *Transformers need glasses! Information over-squashing in language tasks*. NeurIPS, 2024.
[12] A. Gu and T. Dao. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. arXiv:2312.00752, 2023.
[13] J. Sun et al. *An end-to-end learning-based cost estimator*. PVLDB, 13(3): 307–319, 2019.
[14] Mamba Implementation. https://github.com/state-spaces/mamba, 2024.
[15] DACE Implementation. https://github.com/liang-zibo/DACE, 2024.
[16] MACE Implementation. https://github.com/Anpo13211/MACE, 2026.
[17] E. J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv:2106.09685, 2021.