Journal of Engineering and Technology for Industrial Applications

ISSN ONLINE: 2447-0228

ITEGAM-JETIA

Manaus, v.12 n.58, p.1466-1474. March/April, 2026.
DOI: https://doi.org/10.5935/jetia.v12i58.3421

RESEARCH ARTICLE OPEN ACCESS

NEXT-LEVEL DYNAMIC QUERIES OPTIMIZATION: SMARTER JOINS, FASTER VIEWS

Salma Hanane*[1], Yahyaoui Khadidja[2] and Bellatrache Ladjel[3]

[1, 2]Computer Science Department, Mascara University, Mascara, Algeria.
[3]ISAE ENSMA, Poitier, France.

[1]https://orcid.org/0009-0002-1812-6237, [2]http://orcid.org/0000-0001-9541-4879, [3]http://orcid.org/0000-0001-9968-0066

Email:*salma.hanane@univ-mascara.dz

ARTICLE INFO

Article History
Received: January 26, 2026
Reviewed: March 2, 2026
Accepted: April 9, 2026
Published: April 30, 2026

Keywords:
Materialized View Selection, Multi-Query Optimization, Deep Neural Networks , SQL Workload Analysis, Dynamic Query Planning.

ABSTRACT

In modern database systems, query workload optimization through materialized views is crucial for achieving high performance. This paper introduces a novel intelligent framework that identifies frequent subexpressions from SQL workloads, selects candidate materialized views, and predicts their potential benefits using a Deep Neural Network (DNN) model.Unlike existing static heuristic methods, our approach adopts a dynamically learning mechanism with proper determination of properties required for effective views in line with distributed workloads and predicate normalization. By excluding an extra step needed in choosing views or query rewriting, query execution is reduced with the proposed framework, as illustrated in existing experimental results of proposed methods in comparison with existing ones.

I. INTRODUCTION

Query optimization has been one of the crucial challenges in relational database systems. Materialized views store precomputed outcomes of intermediate results to avoid redundancies and consequently give responses to queries considerably faster. However, the problem of selecting which views to materialize-especially under storage-constrained budgets and dynamic workloads-remains complex and crucial. Traditional static MV selection techniques, including those based on cost models or fixed heuristics [1], [2], inherently assume a stable workload and predictable query patterns. These assumptions hold hardly in modern environments, where workloads, data distributions, and system resources change dynamically.As a result, most of the static approaches yield suboptimal views or are not able to adapt to newly emerging access patterns, which would have been wasted in storage and undermined the potential performance gains.

Recent breakthroughs in deep learning methods, represented by large language models like GPT, have pointed out remarkable transformative potential over natural language processing, decision-making, and intelligent automation. Influenced by this trend, database systems are now becoming self-tuning, adaptive systems. Materialized view selection can then be framed as a dynamic, data-driven optimization problem using predictive machine learning models. Our contribution embraces this shift by integrating deep learning into the database optimization pipeline to enable workload-aware, adaptive MV selection. Prior research has explored machine learning for join order optimization [3], [4] and query cost estimation; application of deep learning for materialized view selection remains underexplored.

Key challenges - effective feature representation, accurate benefit estimation, and adaptability to workload changes - make these approaches challenging to adopt in practice.This paper proposes a two-tier adaptive approach to the selection and rewriting of materialized views. The proposed strategy systematically analyzes SQL workloads to detect and group frequently occurring subexpressions using normalized join and selection predicates. A DNN model predicts the utility of the candidate views, along with structural properties combined with runtime and storage cost considerations. Given the embedding of machine learning into the query-processing pipeline, our framework dynamically adapts to the continuously evolving workloads while it optimizes query performance at a minimal storage overhead.

II. LITERATURE REVIEW

The problem of materialized view (MV) selection has evolved drastically in recent years. In the past, MVs were manually or statically selected through heuristics that targeted repeated or expensive queries [5-7], [2]. However, these early approaches were unable to adapt dynamically to evolving workloads. With the advancement of database systems, dynamic heuristics based on query frequency, selectivity, and join ordering were developed [8-10], [1] though they still faced scalability and flexibility challenges.

Today, the advent of machine learning has revolutionized MV selection through adaptive approaches such as supervised learning, clustering, reinforcement learning, and deep neural networks [11-13], [4]. These approaches leverage historical workloads and data distributions to predict candidate view benefits, thereby improving the accuracy and responsiveness of MV selection.

Techniques like Q-learning, artificial neural network (ANN) prediction, and multi-objective optimization [1] have further enhanced real-time materialized view management by being both cost-efficient and scalable. Alongside these advances, developments in query optimization techniques—such as data representation methods (e.g., histograms [14], sketches [15]), sampling methods (e.g., Fixed-step [16], Bifocal [17]), and machine learning-based cost estimation and cardinality models (e.g., MSCN 18], QuickSel 19], RSPN - [20])—have significantly increased the quality of query planning and execution in large-scale systems. As part of this broader evolution, AutoView [21] demonstrates significant progress in MV management. It integrates query plan-based candidate generation, deep query–view correlation modeling, and deep reinforcement learning for space-budget-aware MV selection.

Unlike earlier systems, which struggled with static, non-adaptive approaches [11], [4], [15], AutoView achieves improved automation, generalization, and optimization capabilities for cloud-scale database paradigms. However, AutoView also inherits some limitations typical of learning-based systems: it requires an initial training session with standard workloads; it may require retraining if workloads change drastically; the deep model's decision-making process may not be as interpretable as that of heuristic-driven approaches; and migrating the model between different database management systems typically necessitates retraining due to variations in query plans and cost models. Figure 1 presents a structural comparison of RLView (Reinforcement Learning) [4], DQM (Deep Q-Learning) [11], and AutoView [3] approaches for materialized view selection.

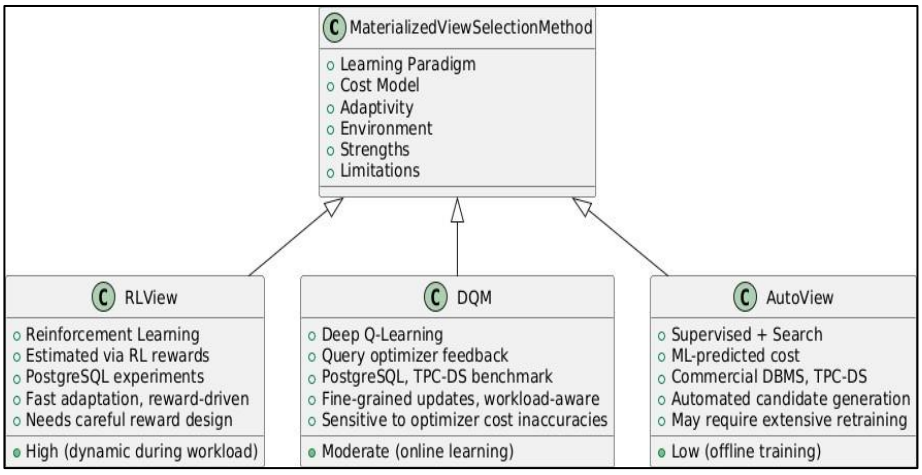

Figure 1: Comparison of some recent methods relative to queries optimization.
Source: Authors, (2026).

Table 1: Comparison of Some recent Learning-Based MVS Methods.

Method	Learning Type	Adaptivity	Training Time	Example Use
RLView [4]	Reinforcement Learning	High	Online	Dynamic workloads
DQM [11]	Deep Q-Learning	Moderate	Online	OLAP queries Static
AutoView [3]	Supervised + Search	Low	Offline	workloads

Source: Authors, (2026).

As shown in Table 1, RLView, DQM, and AutoView represent distinct approaches to materialized view selection. These methods are designed to handle different kinds of workloads, with RLView and DQM being more adaptive to dynamic workloads, while AutoView works effectively on static workloads with offline training. In summary, the evolution of MV selection research across different periods has steadily contributed to the development of increasingly efficient, scalable, and adaptive solutions, addressing the complexities of large, dynamic query environments.

III. PROBLEM STATEMENT

Given a large workload set of SQL queries $Q = \{q_1, q_2, \ldots, q_n\}$, and a set of candidate materialized views $V = \{v_1, v_2, \ldots, v_m\}$ generated from common subexpressions and shared predicates. The goal is to select a subset $V^* \subseteq V$ that maximizes the overall query performance improvement defined as:

$$\text{Benefit}(V^*) = \sum_{q_i \in Q} (T(q_i) - T'(q_i, V^*)) \quad (1)$$

Where $T(q_i)$ is the execution time of the original query and $T'(q_i, V^*)$ is the execution time after rewriting the query using the selected materialized views. The selection must also satisfy a storage budget constraint, where B denotes the available storage space.

$$\text{Cost}(V^*) = \sum_{V_i \in V^*} (size\ (v_i) \quad where\ \ \text{Cost}(V^*) \leq \text{B} \tag{2}$$

The global optimization problem goal is to choose a materialized view set V^* that improves query performance without using more than storage capacity (minimizing resource consumption). Maximizing Benefit (V^*) works towards minimizing query execution time by choosing views that decrease query execution time. On the other hand, minimizing Cost(V^*) ensures that the views chosen do not use too much storage fee.

$$V^* = \text{Arg} \max_{V^* \subseteq V'} ((W_1 \cdot \text{Benefit}(V') - W_2 \cdot \text{Cost}(V')) \ subject\ to \sum_{V_i \in V^*} (size\ (v_i) \leq \text{B} \tag{3}$$

The weights w_1 and w_2 aim to balance the trade between query speed optimization and storage management. The weights can be optimized depending on whether query speed or storage is more important in the specific situation. The objective is to develop a system that intelligently selects the most beneficial materialized views under storage limitations and speed up workload execution. Table 2 presents an example to clarify the functionality of our formulation. We assume that the system selects three candidate materialized views and a storage budget of 9 GB. The optimal subset of condidate materialized views set under this storage budget is $\{ V_1, V_2 \}$,which provides the maximum total benefit of 40 ms while satisfying the storage constraint.

Table 2: Example of Materialized View Selection.

Subset V′	Total Size (GB)	Total Benefit (ms)	Feasible for selection as condidate for Materialization?
$\{ V_1 \}$	5	20	Yes
$\{ V_2 \}$	4	18	Yes
$\{ V_3 \}$	6	25	Yes
$\{ V_1, V_2 \}$	9	40	Yes
$\{ V_1, V_3 \}$	11	45	No (over budget)
$\{ V_2, V_3 \}$	10	43	No (over budget)

Source: Authors, (2026).

IV. PROPOSED APPROACH

In this work, we present an intelligent two-stage strategy for materialized view (MV) selection and query rewriting to optimize the performance of large-scale SQL workloads.The system architecture overview is illustrated in Figure 2. The pipeline of our system inspired by the traditional query processing : Parsing, optimization, and execution,with the integration of machine learning techniques in optimization stage. Specifically, an artificial neural network (DNN) is employed to predict the utility of candidate views to optimize large workload of queries.

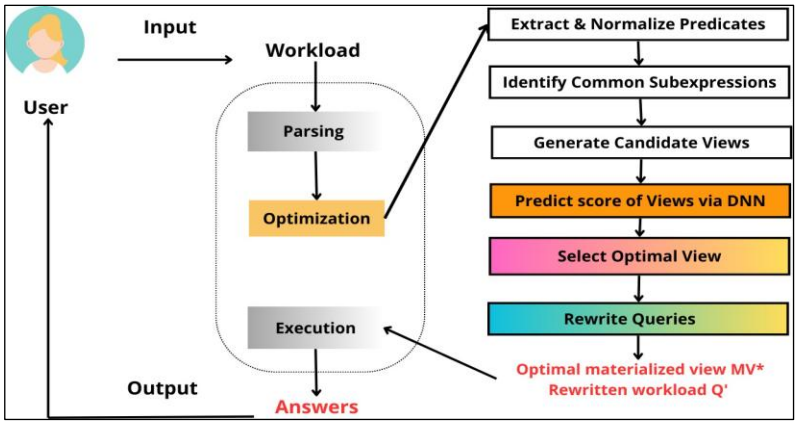

Figure 2: Overview of the Materialized View Selection Framework.
Source: Authors, (2026).

IV.1 WORKLOAD PARSING

Our system starts by parsing the SQL workload input and processing each file separately. For each query, it gives a unique ID and then parses and analyzes the query to identify its structure. Specifically, it extracts the tables involved, the join conditions between them, and the selection predicates from the WHERE clause. These extracted elements are placed in global collections such as *All_Join_Predicates* and *All_Selection_Predicates*. At the same time, the system keeps track of which predicates and tables appear in each query. This information is important for future steps where we need to identify common subexpressions across queries to suggest useful materialized views.

IV.2 IDENTIFYING COMMON SUBEXPRESSIONS

After extracting tables, the join predicates and the selection predicates are identified, which are then normalized and grouped globally. Systematically extracts subexpressions from each query by focusing on join conditions, predicates, and selected attributes. We normalize these entities to enable apples-to-apples comparison by eliminating aliases, normalizing attribute order, and abstracting con- stant values. This enables us to structurally compare subexpressions, rather than syntactically. We employ a hashing mechanism to count the frequency of each unique subexpression across the entire workload ,as shown in Figure 3 Frequent subexpressions are marked as common and are strong candidates for reuse or materialization during the query optimization process.

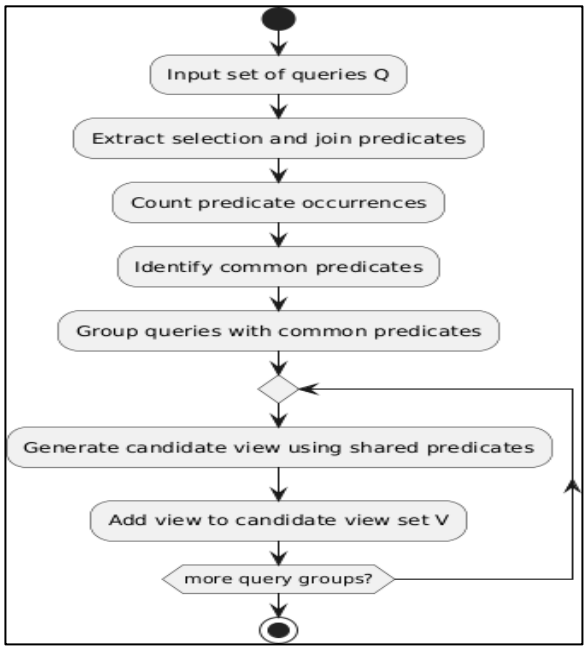

Figure 3: Diagram of views clustering.
Source: Authors, (2026).

IV.3 GENERATING CANDIDATE VIEWS BASED ON PREDICATES

To generate views from grouped queries with the common predicates, the process starts by combining common selection predicates into a single WHERE clause, eliminating redundant filtering. Next, common join predicates are merged into the JOIN condition of the query in an effort to reduce redundant join operations. Columns utilized in the view are based on fields queried most frequently in the group so that the view has only the most critical data. The outcome of this step is a set of candidate views.

IV.4 GENERATING CANDIDATE VIEWS BASED ON PREDICA OPTIMIZATION USING DNN

Once common predicates are identified and normalized,and candidate views are generated , in the first stage of our system we employ a Deep neural network (DNN) architecture to predict a benefit score of views, indicating how much a materialized view would improve query performance (See Figure 4).

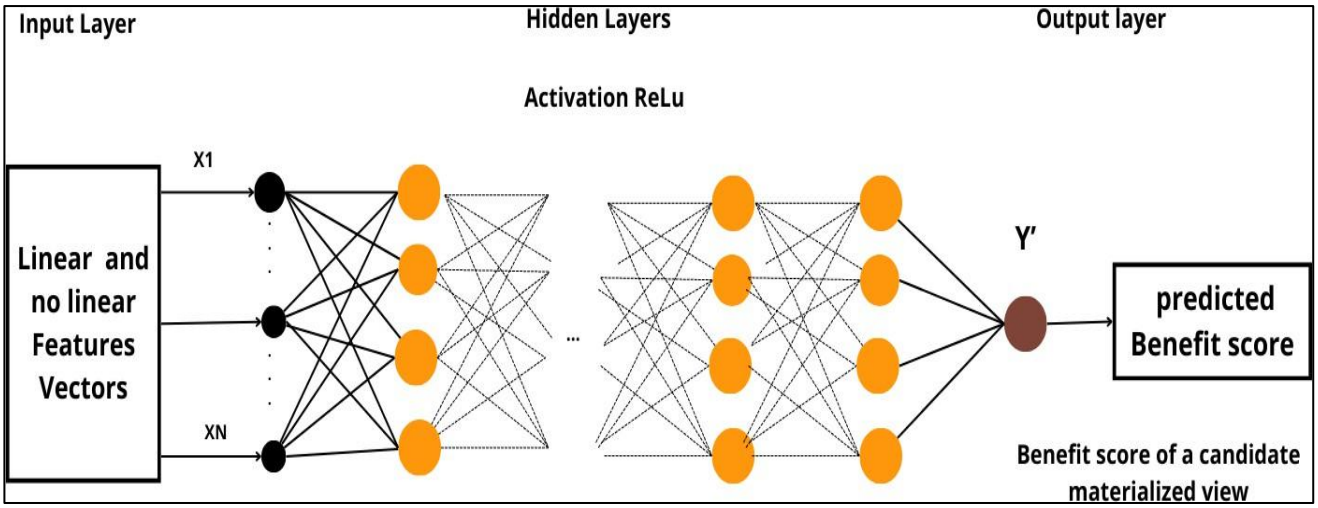

Figure 4: Deep Neural Network architecture for benefit score prediction of candidate materialized views.
Source: Authors, (2026).

The proposed deep neural network (DNN) is intended to predict the benefit score corresponding to each candidate materialized view by learning from both structural and statistical properties of the query workload. The input layer takes a preprocessed feature vector x_input, which conveys information about join types, selection predicates, past query execution times, storage costs, and query frequency. Linear features like execution time, storage cost, and query frequency are normalized using Min-Max normalization to maintain numerical stability, and non-linear features, such as join types, predicates, and operators, are represented using one-hot encoding.

After encoding, all features are stacked together into a unified input vector that conveys comprehensive workload information. The network design involves multiple fully connected hidden layers. The first hidden layer uses dense neurons with a Rectified Linear Unit (ReLU) activation function to inject non-linearity and facilitate the learning of complex patterns. A second hidden layer is used, along with a dropout layer to combat overfitting and enhance generalization. The activation functions of the hidden layers are calculated as:

Activation function

$$h^l = \mathrm{ReLU}\left(W^l h^{l-1} + b^l\right) \tag{4}$$

Where W^l and b^l are the weight matrix and bias vector of the l-th layer, respectively.The output layer consists of a single neuron with linear activation, producing a continuous value that represents the predicted benefit score of a candidate materialized view. This score is computed as :

Predicted benefit score

$$y' = \sum_{i=1}^{d} (W_i h_i + b) \tag{5}$$

Where h_d denotes the output of the final hidden layer and i is the number of hidden neurons feeding into the output neuron. During training, the model minimizes the Mean Squared Error (MSE) between the predicted benefit scores and the observed improvements in query execution time. This training strategy enables the model to generalize effectively to unseen queries and to accurately rank candidate-materialized views based on their expected performance gains.

IV.5 GENERATING CANDIDATE VIEWS BASED ON PREDICA OPTIMIZATION USING DNN

In the second stage of our system, the operation begins by checking newly arriving SQL queries to determine if their join or selection predicates may be optimized by candidate materialized views. If it is found that such predicates are found in materialized views already in place, a procedure known as predicate matching, the original queries are rewritten such that the affected parts are substituted by the correspond created to contain the shared semantics. The queries are partitioned and rewritten, each to use the new common materialized view. The performance of the rewritten queries is compared with the original queries to measure the improvement. This evaluation feedback is subsequently used to refine and optimize materialized view selection algorithms, generating a cycle of adaptive performance tuning (See Figure 5).

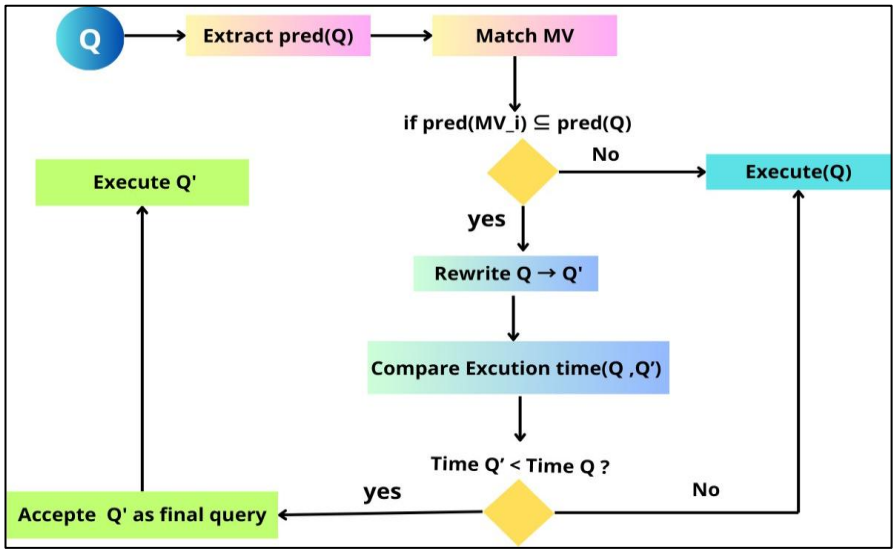

Figure 5: Diagram of Query Rewriting Using Materialized Views.
Source: Authors, (2026).

V. EALUATION AND RESULS

All our evaluations were conducted on a computer equipped with an Intel i7, 16.00 GB of RAM, and a 64-bit Windows operating system. To evaluate the performance of the proposed method, we used PostgreSQL as DBMS, and the implementation was carried out using Python 3. In our experiments, we used the schema of the relational database IMDb dataset [22] with 21 tables and a workload of SQL queries from Join Order Benchmark (JOB) [23].

IMDb has over 21 million rows, ranging in size from a few thousand to a few million rows per table. The JOB workload consists of 113 SQL queries that are designed to test combinations of join and selection predicate. The results can be followed as a progressive walkthrough of our experiments. Table 3 and its visualization in Figure 6 show the initial heuristic mapping of queries into candidate materialized views, where structurally similar queries such as Q22, Q29, and Q72–Q75 were grouped under MV4. While this clustering revealed repeated subexpressions, it often failed to capture the most beneficial views.

Table 3: Workload Mapping results without DNN Learning.

MVID	MV Related Queries ID
1	[Q10, Q13]
2	[Q11, Q12]
3	[Q21, Q24]
4	[Q22, Q29, Q72, Q73, Q74, Q75]
5	[Q52, Q54]
6	[Q48, Q49]
7	[Q87, Q88, Q89]
8	[Q90, Q91, Q92]
9	[Q96, Q98, Q100]
10	[Q97, Q99]
11	[Q45, Q46, Q47]
12	[Q26, Q27, Q28, Q30]
13	[Q82, Q83]

Source: Authors, (2026).

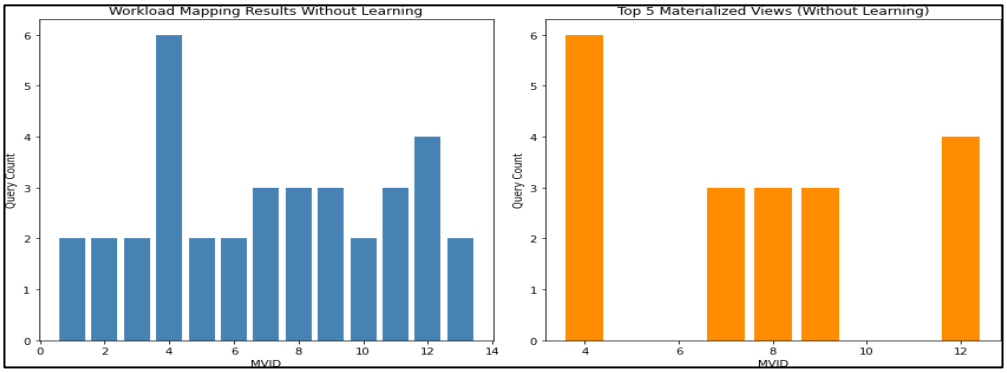

Figure 6: Workload Mapping Results Without Learning (Heuristic mapping method).
Source: Authors, (2026).

Moving to Figure 7, the DNN-based mapping demonstrates a more refined grouping, as the model adaptively selected views that provided higher performance gains while reducing redundancy, and also built a mapping dictionary that included statistical information such as tables occurrence and common subexpressions including join predicates. During evaluation, we compared a medium- sized neural network against a deeper neural network architecture.

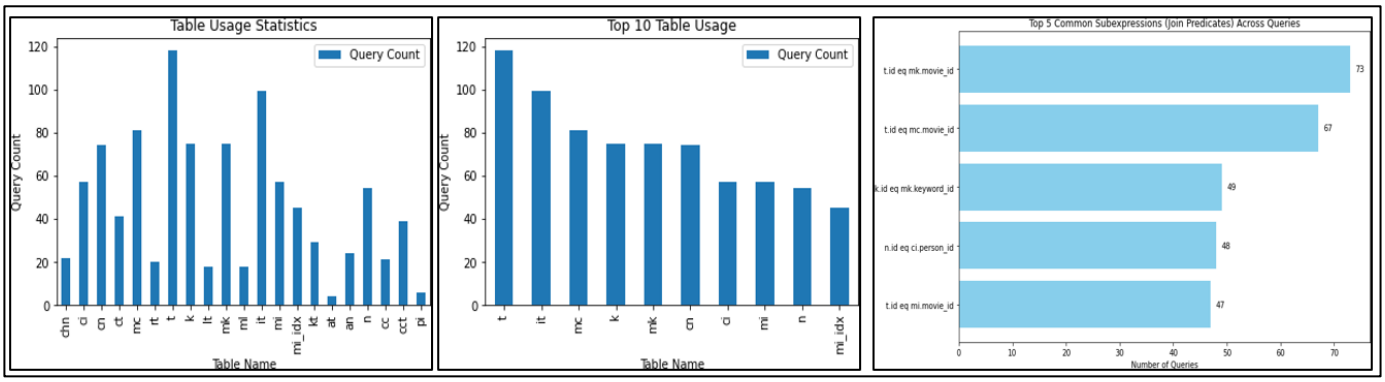

Figure 7: Workload Mapping Results using DNN Approach.
Source: Authors(2026).

As shown in Figure 8, the deeper model achieved a more pronounced reduction in loss, indicating superior generalization to unseen queries. However, this improvement came with slightly higher resource consumption, reflecting the classical trade-off between accuracy and computational efficiency.

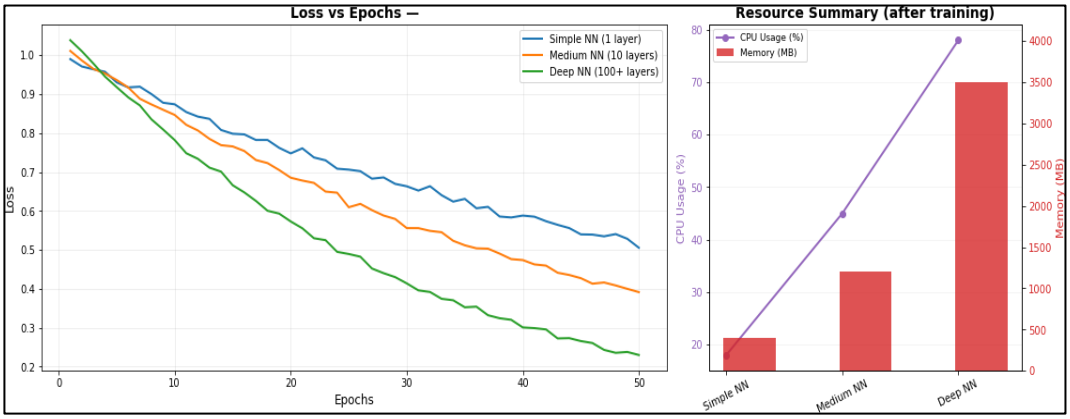

Figure 8: Loss Reduction Resource Usage summary Across Learning Levels for MV selection.
Source: Authors, (2026).

As a result, Figure 9 highlights the final set of materialized views chosen after DNN learning, with MV3 and MV4 emerging as particularly beneficial due to their balance of query coverage and storage efficiency. Finally, Table 4 quantifies the improvements by comparing execution times: queries such as Q74 and Q45 executed **30–35% faster** with DNN-based views than with heuristic methods. This sequence of results illustrates how our framework evolves from static heuristic clustering to intelligent, learning-driven optimization that consistently accelerates query performance under storage constraints.

Table 4: Comparison of Query Performance with Different MV Selection Strategies.

Query ID	Heuristic static method (ms)	DNN (ms)
Q22	600	**420**
Q29	810	**590**
Q45	500	**360**
Q74	1050	**960**
Q98	1120	**820**

Source: Authors, (2026).

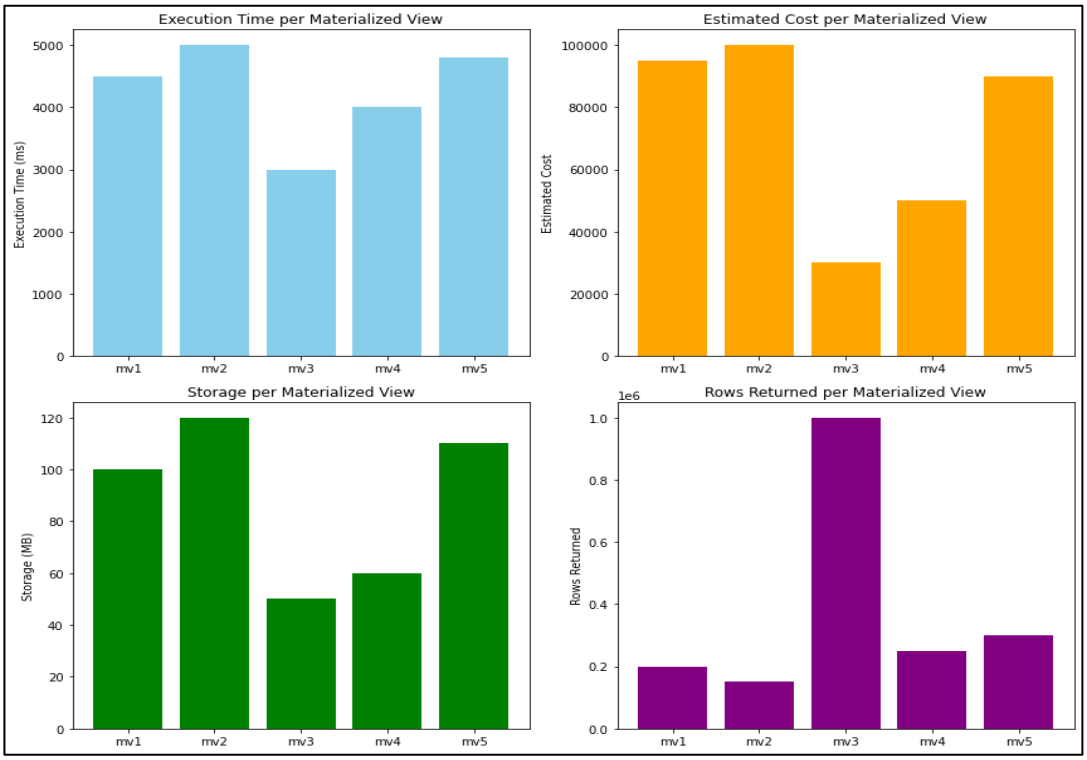

Figure 9: Summary of beneficial Materialized Views after DNN Learning.
Source: Authors, (2026).

Our experiments show that the proposed DNN-based method improves query execution time over the static heuristic model, proving that learning-based benefit prediction is more effective than heuristic rules. While actual methods like RLView, DQM and the AutoView system have advanced adaptive view management, they often require complex retraining. Our contribution is simpler but complementary a lightweight DNN regression model that balances performance and storage efficiency, and can be easily applied in real systems such as PostgreSQL.

V. CONCLUSION

This paper proposes an approach to selecting materialized views (MVs) and query optimization for improving big SQL workload performance. Using a deep neural network (Deep-NN), our approach correctly predicts the benefit of candidate MVs, optimizing query performance with storage management. Evaluations of our algorithms on the IMDb schema and JOB benchmark queries showed that the DNN-based approach performed better than static traditional methods by reducing query execution time and improving storage efficiency. MVs like MV3 and MV4, selected according to our smart model, demonstrated significant performance with low storage consumption.

Our findings also indicate that shorter execution times do not necessarily translate to lower storage costs. Some perspectives, even though they processed more rows, still had good performance, indicating that our model successfully balances speed and storage. These results validate the utility of learning in optimizing MV selection for dynamic query environments. Future development will be aimed at improving the DNN model, incorporating additional query features, and making the system capable of dealing with more complex dynamic workloads.

VI. AUTHOR'S CONTRIBUTION

Conceptualization: Salma Hanane, Yahyaoui Khadidja and Bellatrache Ladjel.
Methodology: Bellatrache Ladjel.
Investigation: Salma Hanane.
Discussion of results: Salma Hanane, Yahyaoui Khadidja and Bellatrache Ladjel.
Writing – Original Draft: Yahyaoui Khadidja.
Writing – Review and Editing: Salma Hanane.
Resources: Salma Hanane.
Supervision: Yahyaoui Khadidja and Bellatrache Ladjel.
Approval of the final text: Salma Hanane, Yahyaoui Khadidja and Bellatrache Ladjel.

VII. REFERENCES

[1] H. Lan, Z. Bao, and Y. Peng, "A survey on advancing the DBMS query optimizer: Cardinality estimation, cost model, and plan enumeration," Data Sci. Eng., vol. 6, pp. 86–101, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:230523630.

[2] I. Mami and Z. Bellahsene, "A survey of view selection methods," SIGMOD Record, vol. 41, no. 1, pp. 20–29, 2012. https://doi.org/10.1145/2206869.2206874.

[3] Y. Han, H. Yuan, G. Li, and J. Sun, "Autoview: An autonomous materialized view management system with encoder-reducer," IEEE Trans. Knowl. Data Eng., 2022.

[4] H. Yuan, Y. Han, G. Li, and J. Sun, "RLView: Reinforcement learning based materialized view selection," arXiv preprint arXiv:2004.04741, 2020. [1] S. Agrawal, S. Chaudhuri, and V. R. Narasayya, "Automated selection of materialized views and indexes in SQL databases," in *Proc. 26th Int. Conf. Very Large Data Bases (VLDB)*, 2000, pp. 496–505.

[5] S. Agrawal, S. Chaudhuri, and V. R. Narasayya, "Automated selection of materialized views and indexes in SQL databases," in Proc. 26th Int. Conf. Very Large Data Bases (VLDB), 2000, pp. 496–505.

[6] A. Gupta and I. S. Mumick, "Maintenance of materialized views: Problems, techniques, and applications," IEEE Data Eng. Bull., vol. 18, no. 2, pp. 3–18, 1995.

[7] S. Dar, M. J. Franklin, B. Jonsson, D. Srivastava, and M. Tan, "Semantic data caching and replacement," in VLDB, 1996, pp. 330–341.

[8] S. Chaudhuri and U. Dayal, "An overview of data warehousing and OLAP technology," ACM SIGMOD Record, vol. 26, no. 1, pp. 65–74, 1997.

[9] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim, "Optimizing queries with materialized views," in Proc. 11th Int. Conf. Data Engineering (ICDE), 1995, pp. 190–200.

[10] H. Gupta, "Selection of views to materialize in a data warehouse," in Int. Conf. Database Theory (ICDT), 1997, pp. 98–112.

[11] J. Liang, M. Li, and X. Li, "DQM: Dynamic query materialization via deep reinforcement learning," in Proc. 2021 ACM SIGMOD Int. Conf. Management of Data, 2021, pp. 2352–2365.

[12] S. Krishnan, M. J. Franklin, and K. Goldberg, "Active learning with contextual bandits for data-driven optimization," in Proc. 2018 ACM SIGMOD Int. Conf. Management of Data, 2018, pp. 587–602.

[13] L. Ma, Q. Lin, and S. Madden, "Query-based workload forecasting for self-driving database systems," in Proc. 2020 ACM SIGMOD Int. Conf. Management of Data, 2020, pp. 1647–1661.

[14] V. Poosala and Y. E. Ioannidis, "Selectivity estimation without the attribute value independence assumption," in Proc. 23rd Int. Conf. Very Large Data Bases (VLDB), 1997, pp. 486–495.

[15] A. Kipf, M. Nießner, T. Kipf, T. Kraska, A. Kemper, and T. Neumann, "Estimating cardinalities with deep sketches," in Proc. 2019 ACM SIGMOD Int. Conf. Management of Data, 2019, pp. 1937–1940.

[16] R. J. Lipton, J. F. Naughton, and D. A. Schneider, "Practical selectivity estimation through adaptive sampling," in Proc. 1990 ACM SIGMOD Int. Conf. Management of Data, 1990, pp. 1–11.

[17] F. Olken and D. Rotem, "Random sampling from databases," in Int. Conf. Data Engineering, 1993, pp. 92–100.

[18] A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz, and A. Kemper, "Learned cardinalities: Estimating correlated joins with deep learning," arXiv preprint arXiv:1809.00677, 2018.

[19] Y. Park, S. Zhong, and B. Mozafari, "QuickSel: Quick selectivity learning with mixture models," in Proc. 2020 ACM SIGMOD Int. Conf. Management of Data, 2020, pp. 1017–1033. https://doi.org/10.1145/3318464.3389727.

[20] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig, "DeepDB: Learn from data, not from queries!" Proc. VLDB Endow., vol. 13, no. 7, pp. 992–1005, Mar. 2020. https://doi.org/10.14778/3384345.3384349.

[21] Y. Han, G. Li, H. Yuan, and J. Sun, "${AutoView}$: An autonomous materialized view management system with encoder-reducer," IEEE Trans. Knowl. Data Eng., vol. 35, pp. 5626–5639, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:247813695.

[22] IMDb, "IMDb datasets," [Online]. Available: https://developer.imdb.com/non-commercial-datasets/, accessed: Jun. 30, 2024.

[23] V. Leis, B. Radke, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann, "Query optimization through the looking glass, and what we found running the join order benchmark," VLDB J., vol. 27, no. 5, pp. 643–668, Oct. 2018. https://doi.org/10.1007/s00778-017-0480-7.