الجمهوريــة الجزائريــة الديمقراطيــة الشعبيــة

République Algérienne Démocratique et Populaire

وزارة التعليـــم العالـي و البحـث العلمـي

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mustapha Stambouli
Mascara
Faculté des Sciences Exactes
Département d'Informatique

جـــامعـة مصـطفى اسطمبولي
معـسكـــر

THÈSE DE DOCTORAT

Spécialité : ***Informatique***

Option : ***Systèmes Informatiques***

Intitulée

Optimisation des requêtes analytiques dynamiques par des méthodes avancées

Présentée par : **SALMA Hanane**

Devant le jury :

Président	Mme Debbat Fatima	Professeure	U.Mustapha Stambouli-Mascara-Algérie
Directeur de thèse	Mme Yahyaoui Khadidja	Professeure	U.Mustapha Stambouli-Mascara-Algérie
Rapporteur	Mr Bellatrache Ladjel	Professeur	U. Poitiers - France
Examinateur	Mr Maaskri Moustafa	MCA	U. Ibn Khaldoun de Tiaret-Algérie
Examinateur	Mr Fekir Youcef	MC A	U.Mustapha Stambouli-Mascara-Algérie
Examinateur	Mme Mohammed Djaouti Soumia	MC A	U. Mustapha Stambouli de Mascara-Algérie

Année Universitaire : **2025 /2026**

الجمهوريـــة الجزائريـــة الديمقراطيـة الشعبيـة
People's Democratic Republic of Algeria
وزارة التعليـــم العالـي و البحـث العلمـي
Ministry of Higher Education and Scientific Research

Mustapha Stambouli University
Mascara
Exact Science University
Computer Science Department

جـــامعـة مصـطفى اسطمبولي
معـسكـــر

DOCTORAL THESIS

Specialty : *Computer Science*
Option : *Computer Systems*

Title

Optimization of Dynamic Analytical Queries Using Advanced Methods

Presented by : **SALMA Hanane**

In front of the Examination Committee:

President	Pr. Debbat Fatima	**U.Mustapha Stambouli Mascara-Algeria**
Thesis Director	Pr. Yahyaoui Khadidja	**U.Mustapha Stambouli Mascara-Algeria**
Co-Reporter	Pr.Bellatrache Ladjel	**U. Poitiers France**
Examiner	Dr.Maaskri Moustafa	**U. Ibn Khaldoun Tiaret- Algeria**
Examiner	Dr. Fekir Youcef	**U.Mustapha Stambouli Mascara-Algeria**
Examiner	Dr. Mohammed Djaouti Soumia	**U.Mustapha Stambouli Mascara-Algeria**

Academic Year: 2025–2026

Abstract

In modern data management systems, redundant computations and data duplication are the current challenges. This occurs because of analytical queries which often share common computational components called common subexpressions, significantly increasing processing time and costs. This phenomenon is present in various data processing paradigms, such as transactional (OLTP), analytical (OLAP), and real time processing (RTAP) systems.Recent studies demonstrated that over 18% of queries in cloud environments, such as Alibaba, contain computational duplication. Two queries are considered duplicates if they share the same subexpressions or have structural similarities. The challenge of discovering and exploiting these subexpressions is known as the common subexpression selection problem, a complex NP-complete problem.This problem is relevant to several key areas of database performance optimization, including multi-query optimization, materialized views, query rewriting, and big data and cloud computing.Physical scene selection is a key solution in this field, aiming to choose the best set of subexpressions that achieve performance optimization while considering storage, update, and reuse constraints. Although this area has been studied since the 1990s through more than 180 research papers, most traditional solutions remain limited to static environments and small query sizes.Today, however, modern applications require dynamic and large-scale processing. Systems like BIGSUBS[81] have proved their ability to detect more then thousands of subexpressions, reducing execution time and cost. Therefore, developing smarter and more flexible solutions has become essential. This thesis aims to propose solutions based on deep learning, using efficient data structures such as graphs, with cost-based models to estimate the benefit of common expression selection. The goal is to build scalable optimization strategies compatible with modern and complex query processing systems.

Keywords: Multi Query Optimization; Materialized View Selection; Machine Learning.

Résumé

Dans les systèmes modernes de gestion des données, les calculs redondants et la duplication des données constituent des défis majeurs. Cela provient des requêtes analytiques qui partagent souvent des sous-expressions communes, augmentant le temps de traitement et les coûts. Ce phénomène existe dans les systèmes OLTP, OLAP et RTAP. Des études récentes montrent que plus de 18% des requêtes dans les environnements cloud, tels qu'Alibaba, contiennent des duplications de calcul. Deux requêtes sont considérées comme similaires lorsqu'elles partagent des sous-expressions identiques ou des structures proches.La détection et l'exploitation de ces sous-expressions constituent le problème de sélection des sous-expressions communes, un problème NP-complet important pour l'optimisation des bases de données. Il intervient dans l'optimisation multi-requêtes, les vues matérialisées, la réécriture de requêtes et les environnements Big Data et cloud.

Bien que largement étudié depuis les années 1990, ce domaine reste limité par des approches statiques et adaptées à de petites charges de travail. Les systèmes modernes exigent des solutions dynamiques et scalables. Des approches récentes basées sur les graphes et l'apprentissage profond permettent d'identifier et d'exploiter efficacement ces sous-expressions, comme le système BigSub.

Cette thèse propose des méthodes d'optimisation basées sur l'apprentissage automatique et des modèles de coût afin d'estimer le gain des sous-expressions communes et d'améliorer le traitement des requêtes complexes.

Mots clés : Optimisation multi-requêtes, vues matérialisées, apprentissage automatique, apprentissage profond, bases de données.

الملخص

في أنظمة إدارة البيانات الحديثة، تُعد العمليات الحسابية المكررة وتكرار البيانات من أبرز التحديات التي تؤثر على أداء معالجة الاستعلامات التحليلية. ويعود ذلك إلى أن العديد من الاستعلامات تشترك في تعبيرات فرعية متشابهة أو متماثلة، مما يؤدي إلى زيادة زمن التنفيذ وارتفاع كلفة المعالجة. وتظهر هذه المشكلة في مختلف أنواع الأنظمة مثل OLTP و OLAPوRTAP، حيث تشير الدراسات الحديثة إلى أن أكثر من 18% من الاستعلامات في البيئات السحابية مثل Alibaba تحتوي على عمليات حسابية مكررة.

يعد اكتشاف هذه التعبيرات الفرعية واستغلالها جزءًا من مشكلة اختيار التعبيرات المشتركة، وهي مشكلة ذات تعقيد NP-Complete وتُعتبر من المسائل الأساسية في تحسين أداء قواعد البيانات. وتظهر هذه الإشكالية في مجالات متعددة مثل تحسين الاستعلامات المتعددة، اختيار العروض المادية، إعادة كتابة الاستعلامات، وأنظمة البيانات الضخمة والسحابة.

ورغم أن هذا المجال قد تمت دراسته منذ التسعينيات، إلا أن معظم الحلول التقليدية ما تزال محدودة وتعتمد على أساليب ثابتة لا تتناسب إلا مع أحمال عمل صغيرة. في المقابل، تتطلب الأنظمة الحديثة حلولًا أكثر ديناميكية وقابلية للتوسع. وقد ظهرت مؤخرًا مقاربات تعتمد على التعلم الآلي وتمثيل البيانات على شكل رسوم بيانية، وقد أثبتت فعاليتها في اكتشاف واستغلال التعبيرات المشتركة مثل نظام.BigSub

تقدم هذه الأطروحة منهجيات تحسين تعتمد على التعلم الآلي ونماذج تقدير الكلفة بهدف تقييم فائدة التعبيرات الفرعية المشتركة وتحسين معالجة الاستعلامات المعقدة.

الكلمات المفتاحية :تحسين الاستعلامات المتعددة، العروض المادية، التعلم الآلي، التعلم العميق، قواعد البيانات.

Acknowledgments

I would like to express my deepest and most sincere gratitude to my supervisors, Prof. Yahyaoui Khadidja and Prof. Bellatrache Ladjel, for their invaluable guidance, insightful advice, and unwavering support throughout the course of this doctoral research. Their expertise, patience, and dedication have been fundamental to the completion of this work.

I would like to extend my most profound and heartfelt appreciation to Prof. Yahyaoui Khadidja for her exceptional dedication and remarkable support. Despite the challenges she faced, she consistently stood by my side, offering encouragement, strength, and guidance at every stage of this journey. Her confidence in my abilities and her continuous motivation have been a true source of inspiration, and I will always remain deeply grateful for her unwavering support.

I would also like to express my sincere appreciation to the Faculty of Exact Sciences at the University of Mascara for providing a supportive and enriching academic environment that greatly contributed to the advancement of this research.

Finally, I would like to extend my heartfelt thanks to all my teachers and to everyone who offered me support along this journey, even through a simple word of encouragement. Their kindness and belief in me have played an essential role in my perseverance and achievement.

Dedication

In loving memory of my dear father, Salma Mohamed, may he rest in peace,

To my beloved mother, Bennedjadi Naïma,

To my dear sisters: Chahrazed and Linda,

To my dear brother: Noureddine,

And to my dear little companions, my beloved cats.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

General Introduction

In recent years, the volume and complexity of data have increased dramatically, particularly in domains that rely heavily on analytical processing, such as business intelligence, scientific research, and smart agriculture. Analytical queries typically are complex, long running, and data are deriving from large-scale datasets. However, these queries often process in dynamic environments where data, workloads, and user requirements evolve continuously. This has created a new problematic on query optimizers.

Traditional query optimization techniques are primarily static and rely on fixed assumptions about data distribution and workloads' nature. These methods, while effective in controlled settings on database management systems , often fail to deliver optimal performance in dynamic and unpredictable contexts. Consequently, dynamic analytical query optimization has become as a critical research area aimed at addressing these limitations.

1.1 Problem Statement

Query optimization in modern database management systems remains a critical and challenging problem due to the exponential number of possible execution plans for a given workload. Select an optimal execution strategy requires accurate cost estimation in terms of CPU, I/O, memory, and data transfer, which becomes increasingly complex in large-scale and distributed environments. Although traditional techniques such as Materialized Views can reduce redundant computations, they introduce additional issues of redundancy[16, 7],and it is related to storage overhead, maintenance cost, and data

redundancy . In the context of using Materialized Views for analytical processing, the problem becomes more complex because workloads consist of a large number of complex queries that often share common sub-expressions or similar sub-queries. While this shared structure offers opportunities for computation reuse, identifying non beneficial candidate Views for materialization can lead systematically to significant redundancy , increased execution cost, and higher fee of memory consumption .[77, 100]. This limitation is further reflected in Multi-Query Optimization (MQO) and Materialized View Selection (MVS) problems, both of which are known to be NP-hard problems and rely heavily on heuristics or cost-based approximations to capturing most beneficial and optimal computation dependencies across queries[154, 26]. The traditional approaches for queries and workloads data representation such as trees, directed acyclic graphs (DAGs), hypergraphs [7, 26], and bipartite graphs [81] suffer in achieving a proper balance between scalability, expressiveness, and adaptability, particularly in highly dynamic environments. Consequently, these representations are continuously evolving and characterized by high redundancy and significant similarity across queries. Therefore, the core problem addressed in this thesis is how to design an efficient and scalable query optimization framework that can effectively exploit shared computations across multiple queries, while remaining adaptive to dynamic workloads and resource constraints. This motivates a shift from static optimization techniques toward more intelligent, learning driven, and structure aware approaches capable of improving performance in dynamic environments .

1.2 Research Objectives and Contributions

This thesis aims to investigate and propose new hybrid approach for optimizing dynamic analytical queries. The primary objectives of this research are to capture and address the limitations of traditional query optimization methods in dynamic environments and design an adaptive optimization framework. The main objectives are as follows:

- In the first contribution, we address the problem of representing analytical SQL queries in database management systems. The main objective is to transform SQL queries into graph-structured representations that can be effectively exploited by deep learning models. To address this problem, we propose a Graph Neural Network (GNN)-based approach that captures both structural and semantic dependencies

between query components for query understanding and optimization.

- In the second contribution, we focus on the problem of query optimization using deep learning techniques. We propose a deep learning-based optimization framework that integrates several key stages, including workload parsing, identification of common subexpressions, candidate view generation, and neural-based optimization, followed by query rewriting. This pipeline improves optimization decisions by learning from historical workloads.

- In the third contribution, we address the problem of materialized view selection in large-scale query workloads. The objective is to reduce query execution cost by identifying the most beneficial views. We propose a workload-aware selection strategy that exploits query similarities and execution patterns to efficiently select high-impact materialized views.

- In the fourth contribution, we propose a GPU accelerated framework for materialized view selection, named NOVA. This approach uses General-Purpose computing on Graphics Processing Units (GPGPU) to accelerate the evaluation and selection process.

- In the fifth contribution, we introduce the Smart SQL framework, that generate text to SQL and introduce an adaptive query optimization system. This approach learns from workload to dynamically generate optimised processing plan.

All this objectives focus on bridging the gap between static query optimization methods and the need for adaptively using machine learning solutions that can ensure both scalability and efficiency in dynamic systems.

1.3 Research Methodology

The research methodology follows a systematic approach to identify, design, and implement our proposed techniques to optimize analytical queries, especially for dynamic environments. The proposed methodology is organized into the following steps:

First, a **modeling and data preprocessing** phase is performed, where analytical queries are transformed into structured graph representations . This step includes parsing

SQL queries, extracting features, and constructing graph representation that identify query semantics and execution dependencies.

Second, a **beneficial view selection for materialization** phase is introduced. In this stage, the system identifies the most useful materialized views from the query workload in order to reduce redundant computations and improve query execution efficiency.

Third, the optimization process is accelerated using **General-Purpose computing on Graphics Processing Units (GPGPU)**. This stage exploits parallel computing paradigm to speed up the evaluation and selection of candidate views, enabling scalable processing for large workloads.

Finally, a **Smart SQL-based optimization layer** is proposed as future direction of our thesis , where new optimized views and query rewriting are generated. This layer uses learned parts from the workload to dynamically adapt query execution strategies and further improve system performance.

1.4 Thesis Structure

The thesis is organized into several chapters, each contributing to the research objectives and providing a logical progression from theory to practice.

The structure of the thesis is:

- Chapter 1 presents the general introduction of the thesis, including the problem statement, research objectives, methodology, and an overview of the thesis organization. It orders the context and motivates the need for analytical query optimization at large scales .

- Chapter 2 provides the background and literature. It introduces the most fundamental concepts such as data warehouses and OLAP systems and reviews literature on query optimization, multi-query optimization, and materialized view selection. It also identifies challenges and research gaps .

- Chapter 3 focuses on modeling analytical queries using Graph Neural Networks (GNNs). It presents the transformation of SQL queries into graph representation and proposes a framework to present how extracted informations can be exploited to improve query understanding and optimization.

- Chapter 4 introduces a deep learning approach for query optimization. It talks about the proposed pipeline, which includes parsing workload, find common subexpressions, generate candidate views, and optimizing with deep neural networks model. Then it talks about a proposed model for rewriting queries.

- Chapter 5 presents NOVA, a GPU accelerated framework for fast view selection. This chapter address how parallel computing improves the efficiency of materialized view selection .

- Chapter 6 introduces the Smart SQL vision, a next-generation framework for intelligent query processing. It includes problem formulation, mathematical modeling of genrated optimal SQL queries.

- Finally, the thesis concludes with a summary of contributions and discusses potential directions for future research in modern smart query optimization systems.

This structure provides a coherent and comprehensive map flow of the research process, as shown in Figure 1.1, from identifying the problem and reviewing existing work to proposing and evaluating an innovative solution in the queries optimization domain.

Figure 1.1: Map of the Thesis Structure

The last chapter presents the general conclusion, and it concludes the thesis by providing a summary and an evaluation of the contributions made by this work. It also presents the perspectives and future directions of our research.

1.5 Research Activities and Scientific Contributions

1. Scientific Publications

1. Salma Hanane, Khadidja Yahyaoui, Bellatrache Ladjel, *Next-Level Dynamic Query Optimization: Smarter Joins, Faster Views*, ITEGAM Journal of Engineering and Technology for Industrial Applications, Vol. 12(58), pp. 1466–1474, March 2026. DOI: https://doi.org/10.5935/jetia.v12i58.3421

2. Panda, S., et al. including Salma Hanane, *Utilization of MMS and GPS in Road Surveying*, Proceedings of Data Analytics and Management, Springer Nature, October 2025. DOI: https://doi.org/10.1007/978-3-032-03740-4_44

- Currently we have submitted manuscripts to indexed journals.

2. International Conferences

- *Integration of Query Optimization in a Smart Manufacturing Environment*, ICIMT – International Conference on Integrated Manufacturing and Mechanics of Materials, Helwan University, Egypt, 26 February 2025.

- *Automatic Data-Driven System for Green Hydrogen Stations Using Deep Learning*, University of Maghnia, Algeria, 02 June 2024.

- *From Trees to Graphs: Evaluating ML Models for Query Execution Prediction on the JOB Benchmark*, ICDAM 2025, London Metropolitan University, UK 14 June 2025.

- *Query Optimization in PostgreSQL: Workload-Aware Model Selection Using Machine Learning*, ICDAM 2024, London Metropolitan University, UK 12 June 2024.

- *Advanced Optimization Techniques for Real-Time Query Processing in Streaming Data*, ICIAN 2024, University of Delhi, India, 29 September 2024.
- *Hybrid CPU–GPU Query Optimization in PostgreSQL: An OpenCL-Based Energy-Aware Approach*, NSET: QSAIC'25, University Center of Tipaza, Algeria.

3. Scientific Events, Workshops and Seminars

- Oral presentation: *The Impact of Big Data in Renewable Energies: Data-Driven Optimization*, Faculty of Exact Sciences, 11 December 2023.
- Scientific Day: *The Impact of Artificial Intelligence on Big Data*, Seminars and Workshops, 30 October 2023 ognized by Dr Khelil Abdellah.
- Participation in TheConference organized by Pr.Senouci Mohammed and Pr.Mourad Bouache, Algiers, on 29 December 2022.

4. Collaborations and Academic Activities

- Research collaboration: Intelligent Processing of Sensitive Data, Ministry of Energy and Mines, Directorate of Energy and Mines, 2024–2025.
- Academic cooperation between Directorate of Energy and Mines of Mascara and University of Mascara.

1.6 Conclusion

In this chapter, we have introduced the fundamental motivation and scope of this research, providing an overview of the challenges posed by dynamic environments for query optimization. The problem of optimizing queries in real time, especially under data volumes. In next chapter,we will explore the literature of query optimization we will highlight and analyze the limitations and gaps Multi query optimzation methods using materialized views and selection structure. By building on the gaps, this research aims to propose a novel solution that adapts to dynamic environments and improves query performance in modern database systems.

CHAPTER 2

Background and State of the Art

From Shadows to Light: A Journey in Query Optimization

"The goal of a database system is to provide efficient data access"

Silberschatz et al. [157]

Chapter Summary

2.1 Introduction

The large volume and complexity of data in modern information systems lead companies to begin to change the way of storing, managing, and analysing data. Traditional database systems are not sufficient to support big volumes of analytical workloads, especially in decision-support environments where cost-efficiency of query processing is important. As a result, data warehousing systems and Online Analytical Processing (OLAP) technologies have known as fundamental solutions to get advanced analytical capabilities over large and heterogeneous datasets. Data warehouse systems face significant challenges related to query performance, storage efficiency, and computational cost. In particular, analytical query processing is a complex hard problem. To address these issues, literature in the database domain and data science field describes a wide range of optimisation techniques that have been proposed, including query optimization strategies, multi-query optimisation (MQO), and materialised view selection approaches. This chapter presents the foundational concepts required for understanding the research problem addressed in this thesis. It begins with background notions related to data warehouses and OLAP systems, followed by a discussion of their main challenges and limitations. Then, it reviews research areas in query optimisation by focusing on analytical query processing, multi-query optimisation, dataset representation techniques, and materialised view selection approaches. Finally, a comparative analysis of existing approaches is presented to improve current research gaps and motivate the contributions of this thesis.

2.2 Background

Traditional database systems are primarily designed for real-time operational task management called 'Online Transaction Processing system' (OLTP). It is designed for fast processing of a large number of rapid online transactions. for example, such as point-of-sale (POS) systems, banking applications for money transfers, and customer relationship management (CRM) systems [7, 149]. On the other side, nowadays modern enterprises require intelligent analytical processing systems (OLAP) to execute large volumes of data, such as those stored in data warehouses, in order to support decision-making and predictive analytics. Data warehouses are among the most prominent practical applications

of these systems, which are used in diverse fields, such as forecasting, sales performance analysis, market behaviour analysis, and economic trend assessment [16, 75].

2.2.1 Basic conception of Data warehouses

In the context of an analytic processing of large scale of data and according to literature, several definitions of data warehouses have been proposed

— A data warehouse is defined as a *subject-oriented, integrated, non-volatile, and time-variant collection of data in support of management's decision-making process* [75].

— A data warehouse is a system that extracts, cleans, conforms, and delivers data from multiple sources into a dimensional data store for querying and analysis [89].

— A data warehouse as a centralised store that integrates data from multiple heterogeneous data sources to support reporting and decision-making [17].

— Most of the research in literature defines a data warehouse as a repository of information collected from multiple sources, stored under a unified schema, and used for analytical processing and data mining [17, 7, 131].

The data warehouse concept was formally defined and popularised by William H. Inmon [75, 41]. It is commonly described through two fundamental deployment paradigms. In a top-down approach introduced by Inmon[75], data is extracted from source systems and first processed in a staging area, where it is cleaned and validated. Then, it is stored in a centralized and normalized data warehouse to support BI [41] and analytics. The bottom-up approach introduced by Kimball[89, 90], where data are extracted from sources and modeled into a star schema design and then processed and loaded into data marts. The main difference between the two approaches lies in their starting point. Inmons top-down approach begins with a centralized data warehouse, while Kimballs bottom-up approach starts with data marts a specialised subset of a data warehouse designed for a specific business domain that are later integrated into a complete data warehouse.

In the evolution of data management systems[131] there are three major generations: data warehouse, data lake, and data lakehouse. Data warehouses[26, 43, 7, 20, 17], introduced in the 1980s1990s, are designed for structured data and optimized for business intelligence and reporting with strong data governance and schema-on-write design.

Data lakes[73, 43, 131, 133] emerged in the 2010s to address the need for storing massive volumes of structured, semi-structured, and unstructured data at low cost using a schema-on-read approach, but they often suffer from weak governance and data quality issues. To overcome these limitations, the data lakehouse [133, 9] architecture appeared around 2019, combining the flexibility of data lakes with the reliability and performance of data warehouses. It introduces strong data management features such as ACID transactions, schema enforcement, and versioning through technologies like Delta Lake[8, 73], enabling a unified platform that supports analytics, machine learning, and real-time data processing.

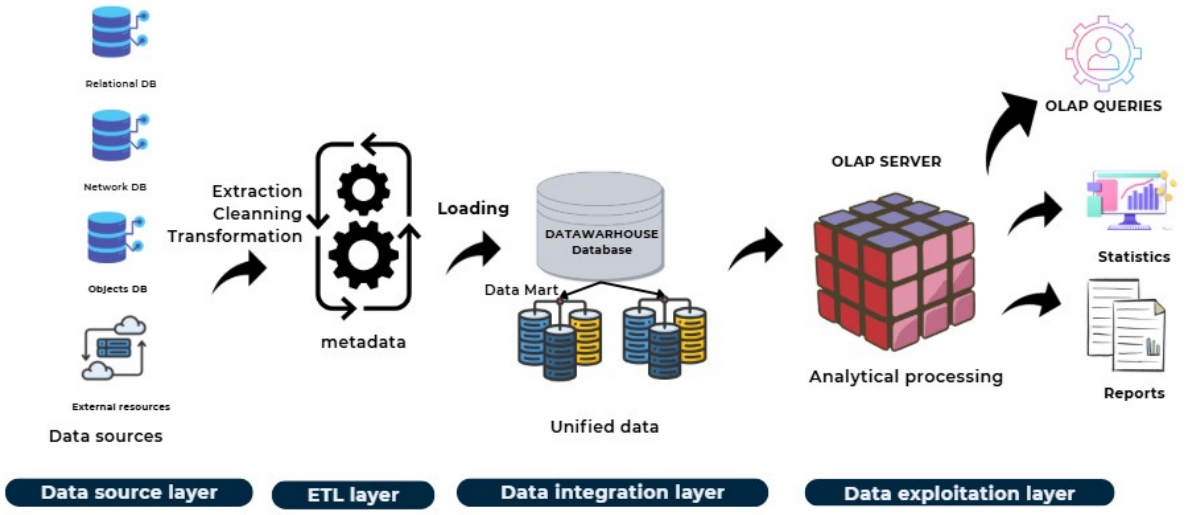

Figure 2.1: Architecture of a Data Warehouse

Figure 2.1 depicts the common structure of a data warehouse architecture, which consists of four main layers. The first is the layer of data sources through which data coming from different sources (databases, XML files, or any other type of sources) goes into the storage of data warehouse banks. The second layer is the ETL (Extraction, Transformation, Loading) layer that is responsible for cleansing, integrating, and transforming the raw data into the needed format for analyses (some advanced warehouses use ELT instead of ETL)[131]. The third one is the data integration layer that tries to integrate all data for storage with the aim of using them for further analysis using the OLAP system. Finally, the fourth one is the layer of data exploitation that serves as the front-end interface for report generation and analyzing business data.[16, 7, 185, 131].

Figure 2.2 illustrates the life cycle of a data warehouse, which is an iterative and structured process that improves the development, deployment, and continuous performance of a decision-support system [90, 87].

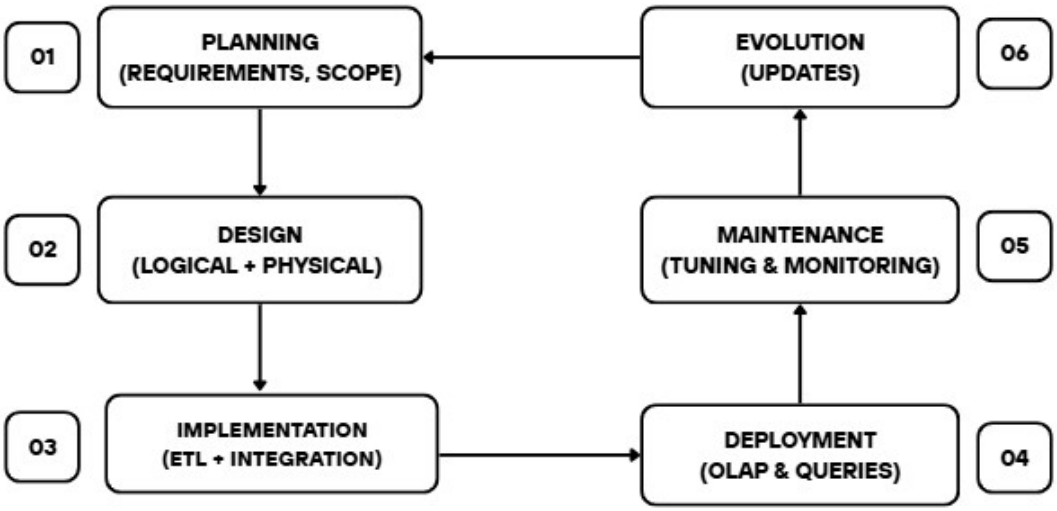

Figure 2.2: Life Cycle of a Data Warehouse

It typically begins with the planning phase, where organisational needs, business goals, and data sources are identified and specified. This is followed by logical design, which defines the overall conceptual architecture and structures, such as schemas (e.g., star or snowflake), independently of physical implementation. Next, the physical design phase focuses on how data is stored, indexed, and accessed to optimise performance. The process then moves to data acquisition and integration (ETL)[39, 20], where data is extracted from heterogeneous sources, transformed into a consistent format, and loaded

into the warehouse. After that, the implementation and deployment phase ensures the system is operational and accessible to users through analytical tools such as OLAP systems. Finally, the maintenance and evolution phase involves monitoring performance, updating data, and adapting the warehouse to new business requirements. This life cycle is not strictly linear but cyclical, allowing continuous refinement and scalability as organisational needs evolve [76, 89, 87, 20, 17, 7, 33, 131, 26]. In this thesis, we assume that the logical design phase and the process using ETL tools have already been completed, and we focus on the optimisation of OLAP queries and the physical design of the data warehouse, particularly the selection of materialised views. For this reason, special attention will be given to the query optimisation concept in the following sections.

2.2.2 Online Analytical Processing (OLAP) Systems

After visiting the data warehouse design phases, it is necessary to understand the role of OLAP (Online Analytical Processing), which supports multidimensional analysis for decision-making. Early OLAP architectures were based on ROLAP, MOLAP, and HOLAP models [29, 49, 26], focusing on pre-aggregation and relational optimisation. Figure 2.3 illustrate different OLAP systems

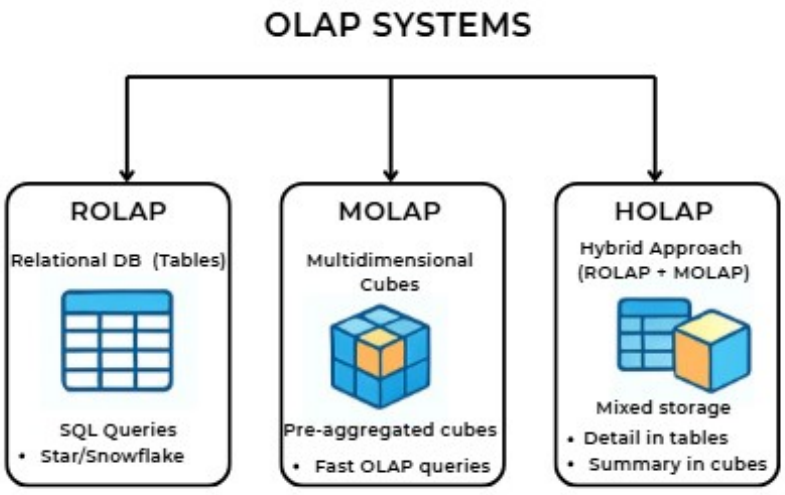

Figure 2.3: Types of OLAP Systems

Figure 2.4 shows the evolution of OLAP systems over time; driven by the rise of big data, OLAP systems evolved toward distributed and cloud-based architectures using Hadoop, Spark, and in-memory processing [132, 33, 26].Modern systems such as Apache

Kylin, Druid, and Google BigQuery support scalable and near real-time analytics[33, 26]. Recently, OLAP systems have been integrated with artificial intelligence (AI) and natural language processing (NLP), enabling intelligent query generation and advanced analytics [173, 112, 176].

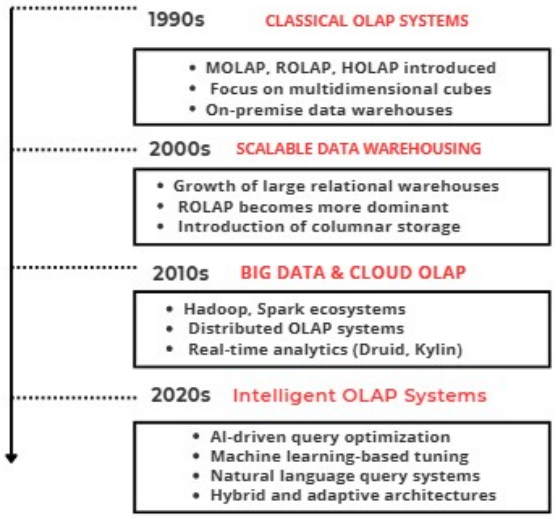

Figure 2.4: Evolution of OLAP Systems

The most widely used model is the star schema [148, 26].This model consists of a large central table, surrounded by smaller dimension tables that are independent from each other and directly connected to the fact table. The snowflake schema [148, 26]is an extension of the star schema, in which some dimension tables are normalised. It is generally used when the star schema is not sufficient to represent the complexity of the database. The resulting structure resembles a snowflake. The fact constellation schema is used to model more complex applications that require multiple fact tables sharing the same dimension tables. This type of schema can be considered as a collection of star schemas [43, 26].The MOLAP multidimensional model represents data as n-dimensional structures as shown in Figure2.5.

In this thesis, we use an adapted star schema based on the IMDb dataset[74], as shown in Figure 2.6.

2.2.3 Challenges and limitations

Classical data warehouses and traditional analytical systems are no longer enough because data now comes from many sources and in different forms such as structured and

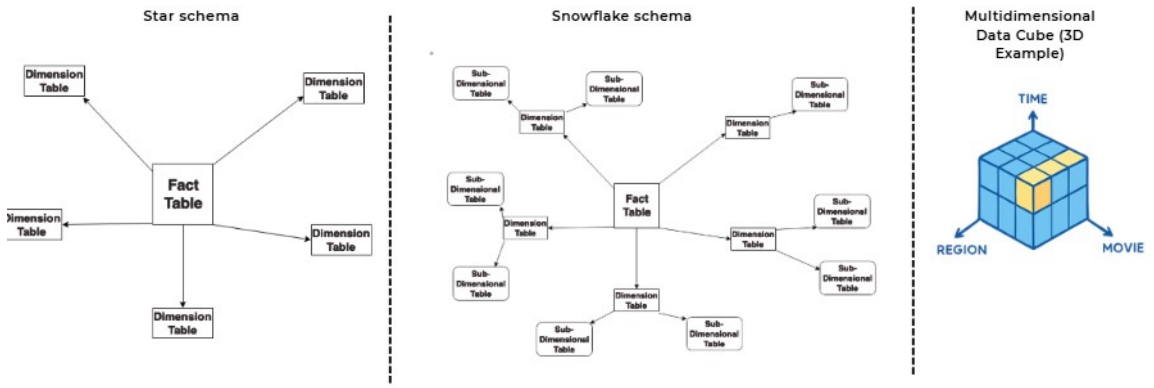

Figure 2.5: Examples of Star Schema , snowflake schema and multidimensional data cube

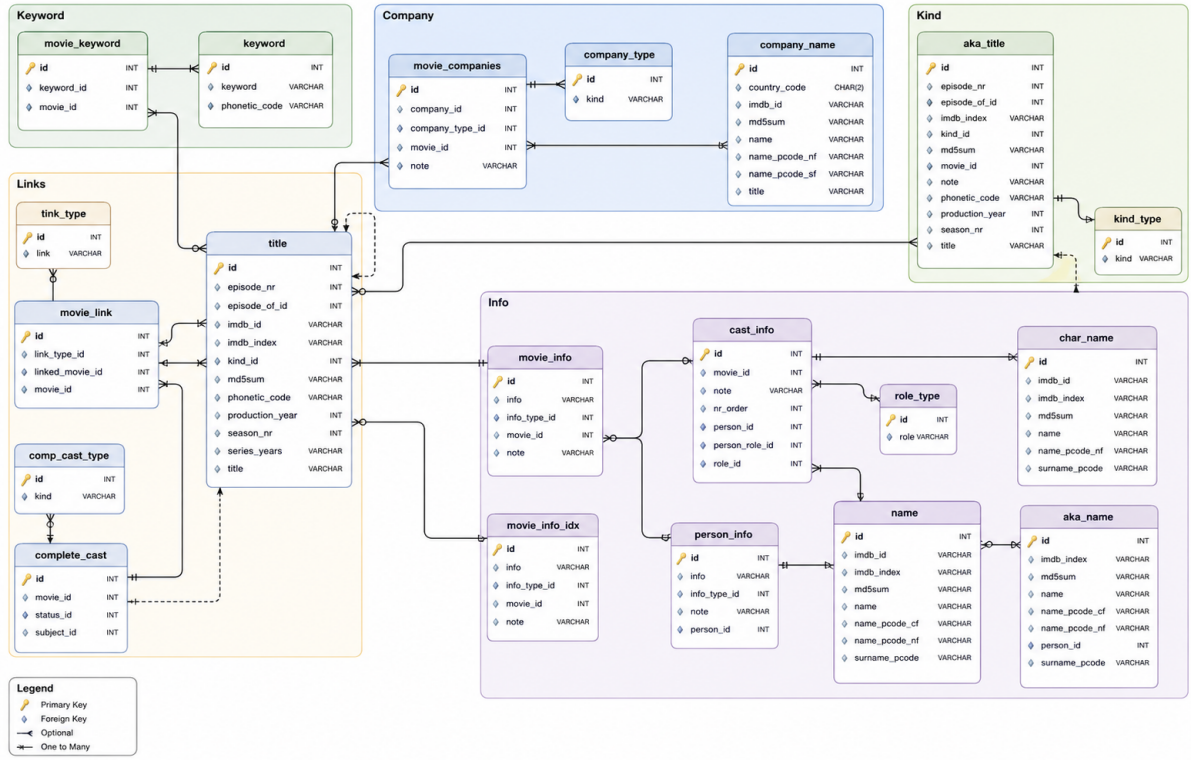

Figure 2.6: IMDB Star Schema Banchmark Exmple

unstructured data.Technologies like Hadoop [165]and MapReduce [40, 13] were introduced to process large data efficiently in a distributed way,MapReduce aims to split tasks into smaller parts, processes them in parallel, and then combines the results. It works with distributed systems like Hadoop- is an open-source system for storing and processing large and diverse data sets across distributed computers-. Later, cloud and hybrid systems improved flexibility and performance in data storage and processing. Recently, artificial intelligence and models like GPT have been added to help users analyse data more easily and interact with data warehouses in a smarter way. However, one of the major challenges that remains is query optimisation, especially when dealing with very large-scale and complex queries over distributed environments. Optimising query execution is difficult due to data volume, data distribution, and system heterogeneity, which can lead to high execution costs and delays. As a result, modern data warehouses are more powerful, flexible, and intelligent than before but still face important challenges in achieving efficient and optimal OLAP query processing .

2.3 Literature Review

Query processing consists of three main steps [17, 7, 20, 26]. First, the query is parsed and transformed into an expression tree using relational algebra rules. Second, the query is optimised by generating an efficient execution plan. Finally, the query is executed (in compiled or interpreted mode) to produce the final result. This section explains these three stages in detail and provides examples when necessary. Figure 2.7 illustrates the overall query processing workflow.It involves four main stages as detailed in [7]: first, the query is analyzed to check syntax correctness and verify that all referenced tables and attributes exist, producing a parse tree; second, it is transformed into a relational algebra expression using operators such as selection, projection, and join, while applying optimization rules like pushing selections downward and replacing Cartesian products with joins; third, an optimization phase generates logical and physical execution plans and selects the most efficient one based on cost estimation, considering CPU cost, I/O cost, and intermediate result sizes (cardinality); finally, the execution engine runs the chosen plan either in a materialized mode or in parallel execution (intra-query or inter-query)[26].

(a) **Materialized mode:** In this execution mode, the result of each operator (such as selection, projection, or join) is fully computed and stored as a temporary relation (in memory or on disk) before being passed to the next operator. Although this approach is simple and easy to manage, it can be costly due to the overhead of writing and reading intermediate results, which increases I/O and memory usage.

(b) **Parallel execution:** In this mode, query processing is distributed across multiple processors [32, 58, 15]to improve performance and scalability. It is divided into two main types:

- **Intra-query parallelism:** A single query is executed in parallel. This includes:
  - *Intra-operator parallelism:* a single operator is split into multiple independent tasks working on different data partitions.
  - *Inter-operator parallelism:* different operators of the same query are executed simultaneously in a pipeline or independently.
- **Inter-query parallelism:** multiple independent queries are executed simultaneously on different processors, improving system throughput, especially in transaction-heavy environments.

As business companies increasingly rely on data to support strategic decisions using, for example, OLAP systems, query optimisation has become a crucial part of database performance management. It involves selecting the most efficient way to execute a database query from several possible execution plans. The need for optimisation becomes critical as the volume of data increases, queries become more complex, and system resources (such as CPU, memory, and storage) are limited[7, 26].

For example, a simple SQL query like(Figure 2.7) can be executed using a full table scan or by using an index. The method chosen can significantly affect response time, especially when dealing with millions of records. Addressing these challenges requires advanced query optimisers that can estimate execution costs accurately and select optimal strategies based on system capabilities. Modern optimisers often use statistical models, cost-based algorithms, and machine learning techniques to improve performance and quality of processing. As data continues to grow in size and complexity, efficient

query optimisation remains a foundational task in guaranteeing responsive and scalable database systems and large workloads.

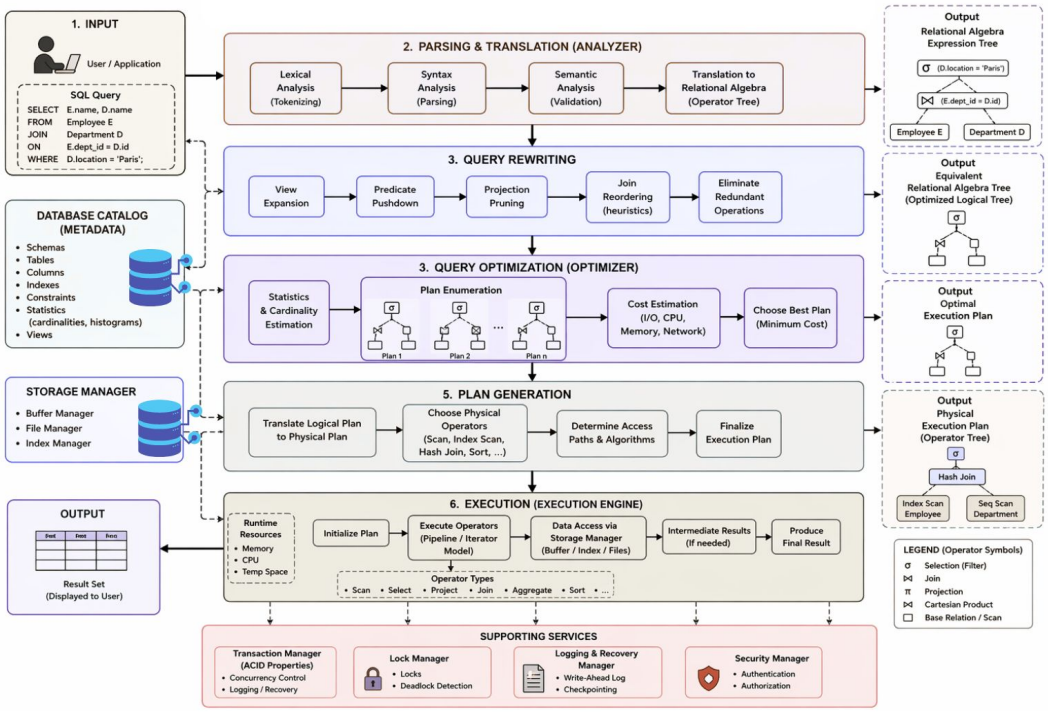

Figure 2.7: Basic Query Processing Architecture in Relational DBMS (Inspired by the work in [7]and [26])

2.3.1 Query Optimization Problems

Query optimization focuses on selecting the most efficient execution strategy for processing queries over large-scale data, complex workloads, and limited system resources. The main difficulty lies in the exponential number of possible execution plans, which makes cost estimation (CPU, I/O, and memory) a key factor in guiding optimization decisions. While traditional techniques such as materialized views and caching can improve performance by reducing redundant computations, they introduce additional challenges related to redundancy ,storage overhead, and maintenance cost [16, 7].

2.3.2 Optimization of Analytical queries

Nowadays,in critical situations, decision-support systems require fast execution of a large number of analytical queries, making performance optimization essential[77, 100]. These

queries share an important property: the presence of common sub-expressions, which allows reuse of computations and reduces overall processing cost. This property is closely related to several database problems, such as multi-query optimization problem and materialized view selection problem, and also extends to big data processing and cloud computing[120, 26]. Analytical queries are characterized by three main features: **large volume, dynamic behavior, and shared computations**, where the last one plays the most crucial role in performance improvement. Recent studies also show a significant amount of query redundancy in cloud environments, highlighting the importance of exploiting this sharing to improve efficiency and reduce cost[7, 85, 128]. However, in modern dynamic environments, OLAP queries are often not executed in isolation and it may share common sub-expressions and common results, also leading to significant redundancy in computation. This observation motivates more advanced optimization strategies, particularly multi-query optimization and view selection problems, which aim to exploit smartly and automatically shared computations and are discussed in the following sections.

2.3.3 Multi-Query Optimization (MQO) problem

Multi-query optimization (MQO) is a fundamental problem in database systems [115, 26] that aims to optimize a set of queries together by identifying and exploiting common subexpressions in order to reduce the total execution cost. Instead of optimizing each query separately, MQO builds a global execution plan by merging individual query plans, where each query may have multiple alternatives [125, 26]. This problem has been studied since the 1980s [153] and is proven to be NP-hard [154]due to the exponential growth of the search space [153, 115].

MQO has been applied to many database environments, including relational [197], distributed[86], XML[44], graph[102, 5, 123], data mining [177], and Hadoop [162] systems. It is also closely related to key optimization techniques such as caching, materialized view selection, indexing, and data reuse. In data warehouses, MQO is especially important because OLAP queries are highly interrelated and often executed together.

Most traditional approaches follow a two-step process: first generating execution plans for each query, and then merging them into a Unified Query Plan (UQP) represented as a Directed Acyclic Graph (DAG)[115], where leaf nodes represent base tables and internal

nodes represent shared operations such as joins and projections [7, 21].

Different techniques have been proposed, including greedy heuristics, pipelined execution strategies, A*-based search methods [50], and cost-based pruning to eliminate inefficient plans early. However, most of these methods assume static query workloads and do not consider modern constraints such as execution deadlines or distributed analytical queries in real time.

Overall, MQO improves performance by detecting and sharing common subqueries across multiple queries [26]. Its efficiency strongly depends on how the data and query interactions are represented, since good modeling enables better reuse of intermediate results and more efficient global query plans.

Example: Multi-Query Optimization with Real Queries

To demonstrate the effectiveness of multi-query optimisation in real-world scenarios, we consider two analytical queries over a 'movies' table from to IMDb data schema.

Query Q1: Movies by Genre and Rating

```
SELECT m.title, m.year, r.rating
FROM movies m
JOIN ratings r ON m.id = r.movie_id
JOIN genres g ON m.id = g.movie_id
WHERE g.genre = 'Action'
  AND r.rating > 8;
```

Query Q2: Movies by Director and Rating

```
SELECT m.title, m.year, d.director_name, r.rating
FROM movies m
JOIN ratings r ON m.id = r.movie_id
JOIN directors d ON m.id = d.movie_id
WHERE d.director_name = 'Christopher␣Nolan'
  AND r.rating > 8;
```

Both queries share a common sub operation: Join between `movies` and `ratings`, and Filtering condition: `rating > 8`

Algebraic Representation and Optimization Analysis: The queries Q1 and Q2 can be expressed in relational algebra as follows:

- Q1 = $(\sigma_{rating>8}(movies \bowtie ratings)) \bowtie genres$
- Q2 = $(\sigma_{rating>8}(movies \bowtie ratings)) \bowtie directors$.

Without optimisation, each query independently recomputes the join between *movies* and *ratings*, leading to redundant processing with costs 5000 for the join, 500 for filtering, and 2000 for additional joins, resulting in **total costs of 7500** for Q1 and 7800 for Q2 (overall 15300). **With multi-query optimisation**, the common intermediate result $common = \sigma_{rating>8}(movies \bowtie ratings)$ is **computed once and reused**, reducing the cost to 5500 for the shared part, 2000 for Q1, and 2300 for Q2, with a **total optimized cost of 9800**. This yields a reduction of approximately **36%**, demonstrating that sharing intermediate results significantly reduces redundant computations and improves query performance in analytical workloads.

On the engineering side, the choice of dataset representation techniques is very important and it plays a pivotal role in the multi-query optimisation process. Techniques such as query graph representation[7], automatic materialised views[128, 125, 61], intermediate result sharing[127, 128], and index-based optimisations form the backbone of MQO systems. These representations not only help in identifying common subexpressions but also support cost-based optimisations, enabling systems to make informed decisions on how to best combine and execute multiple queries. By providing an efficient way to model and manage the data, these techniques facilitate faster query execution, lower resource consumption, and ultimately, a more scalable database system [7, 85, 26].

In this context, in our thesis, understanding how to effectively represent datasets is fundamental to solving the MQO problems. The next section explores various dataset representation techniques employed in MQO systems and how they contribute to achieving optimal performance.

2.3.4 Dataset Representation Techniques in MQO

To better analyse the landscape of dataset representation techniques in Multi-Query Optimization (MQO), we categorise them into four major groups: (1) graph-based representations, (2) intermediate result and view management, (3) partitioning and indexing

approaches, and (4) rule-based, approximate, and probabilistic methods. This classification not only reflects historical developments but also highlights how different approaches address the challenges of redundancy elimination, scalability, and adaptability in large-scale query processing.

Graph-Based Representations

Graph-based models have long played a central role in representing queries and enabling multi-query optimization (MQO). In their simplest form, queries are expressed as **query trees**, where relational operators are nodes and edges capture data flow. While intuitive, tree representations struggle to efficiently capture shared subexpressions across multiple queries. To address this, Directed Acyclic Graphs (DAGs) became the dominant abstraction [153]. In DAGs, common subplans across queries can be merged into single nodes, thereby reducing redundant computation and supporting cost-based optimization frameworks [184, 198] .

Beyond DAGs, researchers explored more expressive models such as hypergraphs, in which nodes represent attributes or subqueries and hyperedges capture many-to-many relationships between them. Hypergraphs allow richer modeling of query interactions and overlapping computations. Some recent works, demonstrated how hypergraph-based view selection enables the detection of complex redundancies in large dynamic workloads [7, 21, 128]. However, hypergraphs incur significant overhead due to the cost of construction and maintenance at scale.

Another direction involves bipartite and query-plan graphs, where one partition represents queries and the other represents candidate views or intermediate results. Edges encode the usage relationships between queries and views, facilitating optimization strategies such as clustering and view selection.A recent industrial contribution in this line is the *BigSubs* framework developed at Microsoft [81]. Observing that workloads in large-scale shared analytics clusters contain massive overlaps across thousands of jobs, the authors formulate the problem of subexpression selection as a bipartite graph labeling task, where jobs and subexpressions form two distinct node sets. They then propose BigSubs[81, 127, 128, 125], a vertex-centric algorithm that iteratively selects which subexpressions to materialize and how to reuse them across jobs. Implemented on top of the SQL-like SCOPE engine, BigSubs demonstrated up to 40% savings in machine-hours

on production workloads, thus highlighting the scalability and practical effectiveness of bipartite graph representations in multi-query optimization.

More recently, the rise of Graph Neural Networks (GNNs) [69] has enabled learned representations of query structures. Instead of manually constructing DAGs or hypergraphs, GNN-based methods [107, 28, 159] learn embeddings directly from query graphs, capturing structural dependencies and latent workload patterns. This approach improves scalability and generalization to unseen queries, extending traditional symbolic graph-based representations into neural-based optimization frameworks.

Overall, the evolution from trees to DAGs, to hypergraphs, and finally to GNN-encoded graphs illustrates the continuous search for a balance between *expressiveness* and *scalability* in Multi query optimization domain.

Intermediate Result and View Management

Another major line of research focuses on the explicit management of intermediate results and materialized views. The central idea is to cache intermediate results such as joins and aggregates, making them reusable across queries, thereby avoiding redundant computation [125]. A more established technique is the use of materialized views, where precomputed results of frequent subqueries or OLAP-style aggregates are stored for rapid access. This approach is widely applied in data warehouses and OLAP systems. While materialized views offer significant speed-ups, they incur non-negligible storage and maintenance costs, making careful selection and refreshing strategies essential. This family of techniques demonstrates how computation reuse forms a cornerstone of efficient MQO, particularly in analytical settings.

Partitioning and Indexing Approaches

Partitioning and indexing techniques represent another class of dataset representation methods in MQO. Partitioning strategies (e.g., hash or range partitioning) divide data into disjoint subsets to enable parallel execution in distributed or shared-nothing environments [102, 140]. Although they substantially improve scalability, partitioning strategies may suffer from load imbalance (Data skew [53]) and expensive inter-partition communication. Complementarily, index-based techniques use auxiliary structures such as B-trees, hash indices, or bitmap indices to accelerate query execution [25, 178]. These structures

effectively reduce I/O and improve join and selection performance, but their maintenance overhead can become significant in dynamic workloads. Together, partitioning and indexing techniques bridge logical query optimization and physical execution efficiency.

Rule-Based, Approximate, and Probabilistic Techniques

Beyond structural methods, more adaptive approaches have emerged that focus on rewriting, approximation, clustering, and probabilistic modeling. Rule-based query rewriting relies on equivalence rules to transform queries into more efficient forms, as explored in ZigZag and cost-based optimization frameworks [85, 125]. Approximate Query Processing (AQP) trades exactness for efficiency by employing sampling and approximation methods, making it suitable for time-sensitive analytics [105, 129]. Query clustering groups queries based on structural or semantic similarity, reducing redundancy in large workloads [98]. In dynamic or uncertain environments, probabilistic and statistical models have been applied to capture execution uncertainty and adapt optimization strategies accordingly [37, 170]. Finally, classical cost-based formulations often treat MQO as a set cover problem to minimise shared computation costs, an approach that, while theoretically optimal, is computationally hard to scale [115, 99, 26, 199].

2.3.5 Discussion

The surveyed techniques reveal a diverse spectrum of strategies for representing and exploiting datasets in MQO. Graph-based models excel in structural overlap detection but scale poorly in massive workloads. Intermediate result and view management deliver substantial performance benefits but raise challenges of maintenance and consistency. Partitioning and indexing improve scalability and physical execution but face trade-offs in load balancing and update costs. Rule-based and probabilistic methods provide flexibility and adaptiveness, albeit at the expense of complexity and computational overhead. Overall, no single approach dominates; rather, hybrid techniques that combine structural, statistical, and adaptive elements are emerging as the most promising direction for next-generation MQO systems.

2.3.6 Materialized Views Selection Problem

Materialized views are defined as a solution to accelerate queries in DBMS by physically storing precomputed results[7, 21, 125]. In recent years, in the design of physical structures for data warehouses, materialized views have been known as a powerful mechanism for query optimisation. By reusing common sub-expressions across multiple queries, they transform redundant computations into efficient, reusable data structures. Materialising views may also introduce data redundancy and infect storage and maintenance costs, especially when the selected views provide limited benefit. To address this problem, the literature defines the Materialized View Selection (MVS) problem, which focuses on identifying the most beneficial views to materialise from a given query workload by effectively exploiting shared sub-expressions. Researchers focus in this field to optimise query performance [127, 61, 195, 26]by satisfying the following: (1) storage constraints, (2) maintenance cost and (3) query rewriting cost . The problem is NP-hard due to the exponential number of possible view combinations, improved in [125].

2.3.7 Materialized Views Selection Techniques

MVS techniques are broadly categorised based on workload typestatic or dynamicand use classical optimisation, heuristic techniques, or learning-based techniques. For static workloads with a fixed set of queries, classical techniques such as 0-1 Integer Linear Programming (ILP) detailed in [81, 54] formulate MV selection as an exact optimization problem, though they suffer from exponential complexity . To improve scalability, Metaheuristic approaches like Hybrid Genetic Algorithms ,This study improves large join query optimization by using query models to fine-tune genetic algorithms

ZigZag+ [85] approach which is an extension of ZiZag [125] , it is a dynamic strategy that iteratively explores multiple MVPPs, moving between them to improve the quality of selected views in order to provide faster, approximate solutions.

Learning-based approaches, such as AutoView [194, 60] and RLView [195], utilize Recurrent Neural Networks and Reinforcement Learning (with Wide and Deep models [34]) respectively to forecast MV utility and guide selection.

For dynamic workloads, where query patterns evolve, heuristics-based solutions like WATCHMAN [150] which aims to reduce query response time in data warehouses and

improve cache utilisation by making decisions about what data to cache and when to eliminate cached data, DynaMat [96] which is an dynamic view management for data warehouses which selects and retains materialized views at varying levels of detail autonomously based on query workload and system constraints including space and update time. Learning-based approaches like DQM [174] utilise reinforcement learning with runtime statistics to adapt MV selection over time, though the lack of query feature encoding can lead to unstable scoring. Recent advancements leverage Graph Neural Networks (GNNs) [59], in particular GraphSAGE [80], for encoding query graphs and learning latent workload patterns to improve generalisation to unseen queries. GCN-based methods have also been successfully applied to cardinality estimation, query cost estimation, and related tasks [80, 190].

2.4 Comparative studies

This section presents a systematic and critical analysis of existing research on Multi-Query Optimization (MQO), the View Selection Problem (VSP), and Materialized View Selection (MVS). The objective is to provide a unified taxonomy of prior work by comparing optimization objectives, data representation models, and algorithmic strategies. The synthesis is organized across four complementary perspectives, as summarized in Table 2.1, Table 2.2, and Table 2.3.

Figure 2.8 presents a conceptual abstraction of the Materialized View Selection (MVS) problem. It captures the interaction between workload characteristics, system constraints, optimization strategies, and cost models, while also illustrating the transition toward learning-driven optimization paradigms.

Table 2.1 provides a comparative analysis of MQO and VSP paradigms in terms of optimization strategy, data modeling, scalability, and learning capability. The results reveal a clear progression from rule-based query rewriting techniques [153] toward hybrid and learning-enhanced optimization frameworks [195].

Early MQO approaches rely on exhaustive or rule-based optimization strategies operating over query graphs. Although these methods provide structured query reuse mechanisms, they suffer from exponential complexity and limited scalability. Similarly, classical VSP approaches focus on workload-driven selection strategies [6, 48], but they exhibit

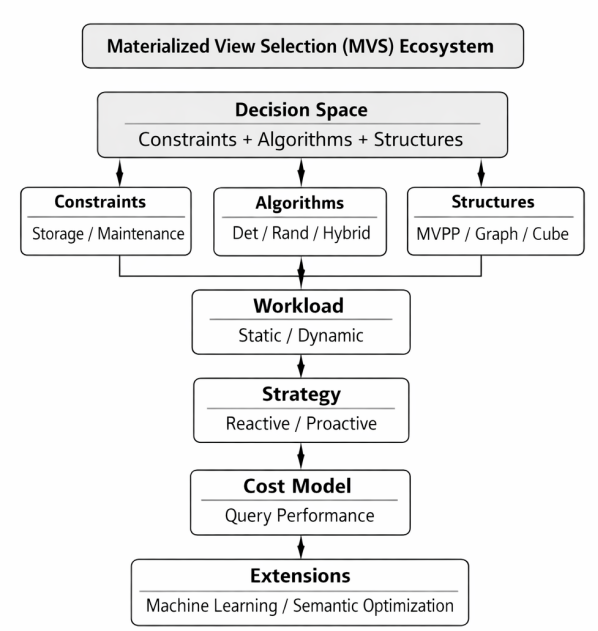

Figure 2.8: Conceptual framework of the Materialized View Selection (MVS) problem

poor generalization across heterogeneous workloads.

Hybrid approaches attempt to unify MQO and VSP under cost-based or queries interaction models, such as hypergraph representations [7]. However, these methods remain constrained by static assumptions regarding workload behaviour and limited adaptability to dynamic environments.

Recent learning based approaches such as [61] introduce adaptive optimization mechanisms. Nevertheless, as shown in Table 2.1, these methods are still limited by high computational complexity, partial automation, and dependency on training data quality.

Table 2.2 compares fundamental data representation models used in query optimization and materialized view selection. These representations define how query interactions and dependencies are structurally modelled to enable optimization.

AND-OR view graphs provide a complete representation of alternative execution plans, enabling exhaustive search and high expressiveness. However, they suffer from poor scalability due to combinatorial explosion.

MVPP-based models introduce cost-aware multi-query sharing mechanisms, improving optimization efficiency. Despite this advantage, their performance degrades significantly under large workloads due to computational overhead.

Data cube lattices offer efficient multidimensional aggregation and are widely used in OLAP systems. However, they are not suitable for dynamic query environments due to

their static structure.

Hypergraph models capture high-order relationships between queries and views, enabling expressive modeling of complex dependencies. Nevertheless, their optimization remains computationally expensive.

Finally, bipartite graph models provide a simplified abstraction of query and view relationships, as improved in [81]; they offer scalability at the expense of expressiveness, but it is an expensive computation fee at the workload processing.

Table 2.3 summarizes the evolution of Materialized View Selection (MVS) techniques across five major paradigms: classical optimization, cost-based heuristics, learning methods, hybrid strategies, and evolutionary approaches.

Classical optimization techniques, including integer linear programming and exhaustive search, provide optimal solutions under constrained settings but are computationally intractable for large-scale workloads. Greedy and dynamic programming approaches improve efficiency but remain limited to local optimality.

Cost-based heuristics such as DynaMat and Watchman introduce adaptive cost estimation mechanisms. While these methods improve runtime efficiency, they lack robust adaptability to evolving workloads.

Machine Learning approaches, including reinforcement learning and neural networks, represent a significant shift toward data-driven optimisation. These methods create adaptive decision-making based on observed query common subexpressions. However, they remain sensitive to training data distribution and hyperparameter tuning.

Overall, Table 2.3 demonstrates that no existing paradigm simultaneously achieves optimality, scalability, and adaptability in current MVS research through highly dynamic environments.

2.4.1 Research Synthesis and Gaps

The comparative analysis across Multi Query Optimization (MQO), View Selection Problem (VSP), data representation models, and materialised view selection (MVS) techniques reveals a clear evolutionary trajectory in query optimization research. The field has progressively shifted from static, rule-based systems toward learning-driven frameworks capable of processing complex and dynamic workloads.First, most classical and heuristic-based approaches fail to scale effectively under large-scale and highly dynamic query workloads.

Second, existing data representation models present a trade-off between expressiveness and efficiency, where highly expressive models are often computationally expensive, while efficient models tend to oversimplify query relationships. Third, although recent learning-based approaches show promising results, they remain limited by high training costs, data dependency, and poor generalisation across heterogeneous environments. These limitations suggest that query optimisation is no longer a purely algorithmic problem, but rather a multi-dimensional challenge involving structural modeling, cost estimation, and adaptive decision-making under uncertainty.

This thesis addresses these challenges by focusing on two directions: (i) syntactic and structural query optimization, and (ii) data-driven optimisation through materialized view selection and cost prediction. The objective is to move toward unified optimization frameworks that integrate structural awareness with learning capabilities.

In addition, under the progress in modern processing technologies such as GPUs Table 2.4 illustrated some critical research gaps in query processing platforms [22] . GPU-based approaches primarily focus on accelerating query execution without adequately addressing optimization strategies or workload adaptivity. Similarly, distributed systems provide scalability but often rely on heuristic or simplified optimization techniques.Furthermore, existing approaches do not integrate learning-based models with hardware-aware optimization, nor do they effectively support dynamic multi-query environments. This fragmentation lead to the need for unified frameworks that jointly combine query optimization, machine learning models, and modern hardware architectures.

Table 2.1: Gap Analysis of MQO and VSP Paradigms

Ref.	**Paradigm**	**Optimization Type**	**Data Model**	**Join Order Priority**	**Scalability**	**Learning**	**Research Gap**
[153]	MQO	Rule-based rewriting	Query graph	NO	NO	NO	Static optimization of queries and Exponential search space
[6, 48]	VSP	Workload-driven selection	Schema model	NO	PARTIAL	NO	High maintenance overhead and Poor generalization across workloads
[184]	MQO + VSP	Hybrid cost-based model	MVPP-based model	YES	YES	NO	High optimization complexity
[7]	MQO + VSP	Query interaction modeling	Hypergraph model	YES	YES	NO	Static workload
[26]	MQO + VSP	Query interaction modeling	Hypergraph model	YES	Partial	NO	Heuristic join order optimization and Requires training data
[194, 195]	MQO+VSP	Adaptive Learned model	Schema Learning model	YES	YES	Partial	Not fully autonomous optimisation. Very high complexity

Table 2.2: Data Representation Models: Comparative Analysis with Features, Strengths, and Weaknesses

Ref.	**Model**	**Idea**	**Expressive**	**Scalability**	**Cost**	**MultiQuery support**	**Strengths**	**Weaknesses**
[55, 56, 115]	AND-OR View Graph	All alternative query plans via AND/OR nodes	✓	×	×	✓	Exhaustive search, high expressiveness	High complexity, poor scalability
[115, 7, 125]	MVPP	Shared multi-query execution with cost integration	✓	×	✓	✓	Cost-aware, efficient MV selection	Expensive for large workloads
[62, 115]	Data Cube Lattice	Multidimensional aggregation and derivation	✓	✓	×	×	Fast OLAP, aggregation reuse	Limited for dynamic workloads
[21, 127]	Hypergraph Model	Captures complex multi-join relationships	✓	×	×	✓	Models high-order dependencies	Hard to optimize, complex
[81]	Bipartite Graph	Queryview relationship modeling (two sets)	×	✓	×	✓	Simple, scalable, intuitive	Limited expressiveness

Table 2.3: Evolution of Materialised View Selection (MVS) Techniques

Ref	Method	Approach	Static workload	Dynamic workload	Scalability.	Strengths	Limitations
Classical Optimization							
[93, 137]	0–1 ILP	Integer Programming	Yes	No	Low	Optimal solution guarantee	NP-hard, high cost
[153, 154]	Exact Search	Exhaustive / Heuristic	Yes	No	Medium	Near-optimal solutions	May miss global optimum
[31, 30]	Greedy	Heuristic	Yes	No	High	Fast and simple	Local optimum only
[101]	Dynamic Prog.	Exact Optimization	Yes	No	Low	Optimal (small scale)	High complexity
Cost-Based Heuristics							
[96]	DynaMat	Adaptive Cost Model	No	Yes	High	Self-tuning	Update overhead
[150]	WATCHMAN	Rule-based	No	Yes	High	Low latency	Static scoring
[192]	Hybrid Query Optimizer	Cost-based + Learning-based	No	Yes	Medium	Efficient (stable load)	Poor adaptability
Learning-Based							
[194, 51]	AutoView	RNN-based	Yes	No	Medium	Predictive modeling	supervised training cost
[195]	RLView	Reinforcement Learning	Yes	No	Medium	Adaptive learning	Sensitive tuning
[126]	DQM	RL + Feedback	No	Yes	Medium	Runtime adaptation	Instability
[27]	GraphSAGE	GNN-based	No	Yes	High	captures relations.	High cost
Hybrid Methods							
[125, 85]	MVPP	Heuristic + MVPP	No	Yes	High	Large-scale effective	Complex search
Evolutionary / Probabilistic							
[171]	Sim. Annealing	Probabilistic	Yes	No	High	Escapes local optima	Slow convergence
[47]	Genetic Algo.	Evolutionary	Yes	No	High	Large search space	High cost

Table 2.4: Hardware-Aware and Distributed Query Processing: Critical Comparative Analysis

Ref.	**Focus**	**Contribution**	**GPU**	**Distributed**	**Learning**	**Technical Gap**
[12]	GPU Query Execution	SQL operators accelerated using CUDA	Yes	No	No	Limited support for complex queries; no optimization strategy; static execution model
[63]	GPU Relational Engine	Full relational operators on GPU	Yes	No	No	Focus on execution only; lacks cost-based or adaptive optimization
[82]	GPU Join Processing	Efficient parallel join algorithms	Yes	Partial	No	Ignores workload dynamics; high data transfer overhead
[22]	GPU DBMS Survey	Analysis of GPU database architectures	Yes	Partial	No	No unified optimization framework; limited integration with query planning
[19]	Parallel CPU Processing	Optimized hash joins for multi-core systems	No	Yes	No	Not designed for heterogeneous (CPU+GPU) environments
[40]	Distributed Processing	Scalable batch data processing model	No	Yes	No	High latency; no fine-grained query optimization; no cost-based planning
[122]	Distributed Analytics	Columnar execution for interactive queries	No	Yes	No	Limited adaptability; simplified optimization strategies
[196]	Distributed Framework	In memory cluster computing	No	Yes	Partial	Relies on heuristic optimization; limited workload adaptivity
[136]	Hardware aware Optimization	Compilation based query execution	Partial	No	No	Focus on single node optimization; no distributed or learning integration

2.5 Conclusion

This chapter provided a comprehensive overview of most fundamental concepts and state-of-the-art techniques related to data warehousing and query optimization. It introduced principal concepts of large-scale analytical processing. The chapter also examined the most challenges associated with analytical query execution, including complexity and scalability issues. Furthermore, a literature review was conducted on optimization problems, particularly on analytical query optimization and its relation with Multi-Query Optimization (MQO), dataset representation techniques, and materialized view selection strategies. These approaches were powerful, but they are still limited when dealing with dynamic workloads, complex queries, and real-time processing requirements. The need of automatic optimization frameworks capable of learning from data parsing to query optimization and processing. It motivates the exploration of advanced techniques that integrate machine learning and deep learning models into the query optimization process. In this context, the next chapter introduces a novel approach for modeling queries using Graph Neural Networks (GNNs) in order to improve optimization performance in modern dynamic environments.

CHAPTER 3

FloraG:Modeling Analytical Queries Using GNN

FloraG: Graph Neural Network For Query Modeling and Formulation

"Graph Neural Networks are effective framework for representation learning of graphs"

Xu et al. [181]

Chapter Summary

3.1 Introduction

Recently, there has been a paradigm shift toward learning-based and structure-aware representations of queries. In particular, graph-based models have emerged as a powerful abstraction for capturing the structural dependencies inherent in relational queries. By representing queries as graphs, where nodes correspond to tables and edges represent join relationships, these approaches enable the exploitation of structural similarities across queries and improve generalization in optimization tasks.

Existing methods still face key limitations. Bipartite graph representations [81] are efficient but insufficient for modeling complex multi-way joins, while hypergraph-based approaches provide richer expressiveness

The development of data-intensive workloads has made query optimization even more complicated in current relational databases. Among all of the optimization techniques, join order optimization continues to be one of the most important and resource-intensive processes due to the vast number of possible options that arise from multi-join operations [125]. Optimization algorithms based on traditional cost models depend on manual integration of heuristics, cardinality estimation, and statistical assumptions to generate optimal query plans [115]. Even their popularity, they may be incapable of eliminate redundancy and minimize CPU fee, Memory use and storage cost [7, 85].To overcome these limitations, recent research has explored learning-based query optimization techniques. Machine learning methods have been applied to multiple optimization tasks such as cardinality estimation, join ordering, query rewriting, and cost prediction [94, 88, 118, 119]. These approaches improve adaptability by learning from historical workloads and executions. However, most existing learned optimizers treat queries independently and do not explicitly exploit structural similarities across query workloads, which leads to redundant computation and limited generalization across related queries. On the other hand, graph representations of queries are developed as a strong paradigm to capture common subqueries. Using graph representations allows capturing structural information in SQL queries by representing tables, joins, predicates, and dependencies as graphs. Initial efforts employed DAGs, bipartite graphs, and hypergraphs for Multi-Query Optimization (MQO) and common sub-expression detection [81, 7]. Although bipartite graphs offer efficient representation capabilities, they sometimes fail in expressive power regarding

dependency modeling. Hypergraph-based models improve expressiveness but introduce significant computational overhead. More recently, Graph Neural Networks (GNNs) have shown strong capability in learning representations from structured graph data and have been successfully applied to database-related tasks such as cardinality estimation, cost modeling, and workload representation [28, 159, 107, 93].Despite these advances, important challenges remain unresolved. Existing graph-based methods often face a trade-off between expressiveness and scalability. In addition, many learning-based approaches focus on single-query optimization and fail to consider workload-level redundancy. As a result, they do not fully exploit shared structural patterns across queries, which limits optimization efficiency in large analytical workloads. To address these challenges, we propose FloraG, a graph neural framework for join order optimization. FloraG represents SQL queries as structured graphs and use Graph Neural Network to learn expressive embedding that capture both local and global structural dependencies within queries. Moreover, FloraG introduces a reusable subgraph learning mechanism. The contribution of this work is threefold. First, we propose a unified workload-aware graph-learning framework for SQL query optimization. Second, we integrate structural query representation with GNN-based embedding learning to capture relational dependencies. Third, we introduce a subgraph reuse mechanism that exploits workload-level similarity to improve optimization efficiency. The proposed approach is implemented as a prototype, and it is integrated with PostgreSQL and evaluated on representative analytical workloads, demonstrating improvements in execution time and cost estimation compared to the baseline optimizer. This chapter will focus on the first and the second contribution of our thesis.

3.2 Related Work

The query optimization technique has moved from conventional rule-based and cost-based paradigms to graph-based and learning-based methods. The classic optimizer approaches, like Cascades [62] and industrial systems including including PrestoDB [155], SparkSQL [10], Orca [158], and Apache Calcite [14], depend extensively on statistics and cardinality estimation [57]during plan formation. Nevertheless, erroneous statistics, correlated predicates, and distributed computing environments usually result in inefficient

execution plans [30]. To address these limitations, heuristic strategies and query hints have been widely adopted [24, 1, 2, 3, 4].Recent studies increasingly integrate machine learning techniques into query optimization [202**?**]. Learned methods have been applied to cardinality estimation [88, 94, 134, 135, 160, 175, 186], join ordering [117, 193], query rewriting [201], and adaptive optimization frameworks such as Neo [118], Bao [119], and Balsa [187]. These approaches improve adaptability with large workloads but still suffer from training overhead, workload drift, and limited generalization. Literature represented the graph-based methods as an effective paradigm for modeling complex analytical queries and execution plans. Early approaches used Directed Acyclic Graphs (DAGs) for Multi-Query Optimization (MQO) to detect common subexpressions and reduce redundant computations [153, 184, 198]. More expressive representations such as hypergraphs were introduced to model complex joins to detect shared intermediate results [7, 21, 128, 26]. Other studies adopted bipartite graph representations to capture relationships between queries and used the conception of shared subexpressions and materialized view techniques [81, 127, 125]. Industrial frameworks such as BigSubs [81]demonstrated the effectiveness of graph-based workload optimization in reducing redundant computations. Recent advances in Graph Neural Networks (GNNs) [59] further extended graph based optimization by learning from structural representations directly or from parsing query sets. GNN-based [80] approaches have been applied to cardinality estimation, cost prediction, workload modeling, and join optimization [28, 159, 107]. Learned cardinality estimation models such as MSCN [91] improve estimation of accuracy for correlated joins using neural representations, while Neo [118] employs reinforcement learning for plan selection. GNN and graph embedding techniques have also been extended to knowledge graph cardinality estimation [151].Recent surveys [108] illustrate that graph neural networks are becoming a central paradigm in the database research field due to their ability to capture structural dependencies and workload semantics. Many studies in single query optimization investigated workload-level optimization problems, including Multi-Query Optimization (MQO) and Materialized View Selection (MVS). Existing MQO approaches include DAG-based unified plans [2, 40], A*-based optimization [50], hypergraph-based techniques [7], and workload clustering methods [98]. MVS approaches include heuristic and metaheuristic algorithms [85], reinforcement learning frameworks [195, 174], and adaptive systems such as AutoView [194, 60], WATCHMAN [150], and DynaMat [96]. Recent

studies also explored GNN-based workload representations using graph convolution and GraphSAGE techniques to model query dependencies more effectively [80, 190].To conclude, the literature demonstrates a transition from traditional optimization frameworks toward graph-based and learning-driven query optimization systems. Nevertheless, important challenges remain regarding scalability, workload generalization, representation quality, and the integration of semantic and structural information in dynamic analytical environments, motivating further research on efficient GNN-based query representations for next-generation query optimization systems. Overall, existing GNN-based optimizers learn query representations independently and fail to reuse structural sub graphs across workloads.

3.3 Problem Formulation

We consider a query workload $\mathcal{W} = \{Q_1, Q_2, ..., Q_n\}$, where each query Q_i is a join-only query represented as a graph $G(Q_i) = (T_i, E_i)$, in which T_i denotes the set of tables and E_i denotes the set of join relationships between these tables.

$$G(Q_i) = (V_i, E_i, F_i) \tag{3.1}$$

with V_i denoting the set of tables (relations), E_i the set of join relationships between tables, and F_i the associated node features such as predicates, attributes, and statistical information.

For each query Q_i, let $\Pi(Q_i)$ denote the set of all feasible execution plans (join orders). The optimal execution plan is defined as:

$$\pi_i^* = \arg\min_{\pi \in \Pi(Q_i)} \text{Cost}(Q_i, \pi) \tag{3.2}$$

where $\text{Cost}(Q_i, \pi)$ represents the estimated execution cost of query Q_i under plan π.

Since the number of possible join orders grows factorially with the number of joined relations, i.e.,

$$|\Pi(Q_i)| = O(m!) \tag{3.3}$$

where m is the number of relations involved in the query, exhaustive search becomes

computationally infeasible for large analytical workloads.

Moreover, queries within the workload may share common structural components, such that

$$V_i \cap V_j \neq \emptyset, E_i \cap E_j \neq \emptyset,$$

which indicates the existence of reusable substructures across queries.

To capture these structural patterns, each query graph is encoded into a dense vector representation through a Graph Neural Network (GNN):

$$h_{Q_i} = f(G(Q_i)) \tag{3.4}$$

where $f(\cdot)$ denotes the GNN encoder and h_{Q_i} is the learned embedding vector associated with query Q_i.

The objective of this work is therefore twofold: (1) to learn expressive graph-based representations of SQL queries that capture structural dependencies, and (2) to exploit workload-level similarities through shared sub-graph patterns to improve the efficiency and quality of join order optimization

Table 3.1: Notation Used in the Problem Formulation

Symbol	**Meaning**
Q_i	SQL query i
W	Set of SQL queries (workload)
$G(Q_i)$	Graph representation of query Q_i
V_i	Set of tables (relations) in query Q_i
E_i	Set of join relationships between tables
F_i	Node features (predicates, attributes, statistics)
$\Pi(Q_i)$	Set of all possible execution plans for query Q_i
π	Candidate execution plan (join order)
π_i^*	Optimal execution plan for query Q_i
$\text{Cost}(Q_i, \pi)$	Estimated execution cost of query Q_i using plan π
$f(\cdot)$	Graph Neural Network encoder
h_{Q_i}	Learned embedding vector of query Q_i

3.4 Methodology

We propose a novel framework called **FLORAG**,Figure 3.1 illustrates the end-to-end workflow of FLORAG, divided into an offline learning phase and an online inference phase. In the offline phase (left side), queries from the workload are transformed into graph representations, enriched with semantic features, and encoded into embeddings using a GNN model. These embeddings are clustered to identify structurally similar queries, from which common subgraph patterns are extracted and stored in a reusable library. In the online phase (right side), a new query is similarly encoded and matched against the learned clusters. If a match is found, previously extracted subgraphs are reused to rewrite the query, reducing redundant computations. The rewritten query is then passed to the join order optimization module, which generates candidate plans, estimates their costs using a learned cost model, and selects the optimal execution plan. The dashed connection between both phases represents knowledge transfer, enabling FLORAG to leverage past workload experience for efficient query optimization. This process is summarized in Algorithm 1.

Algorithm 1 Query Graph Construction

Require: SQL query Q_i

Ensure: Graph $G(Q_i) = (V, E, F)$

Parse Q_i to extract T, J, P
for each table $t \in T$ **do**
    Create node $v_t \in V$
end for
for each join $(t_i, t_j) \in J$ **do**
    Add edge $(v_{t_i}, v_{t_j}) \in E$
end for
for each predicate $p_k \in P$ **do**
    Attach embedding of p_k to node features F
end for
return $G(Q_i) = (V, E, F)$

In the FLORAG framework Figure3.2 show that each SQL query is first transformed

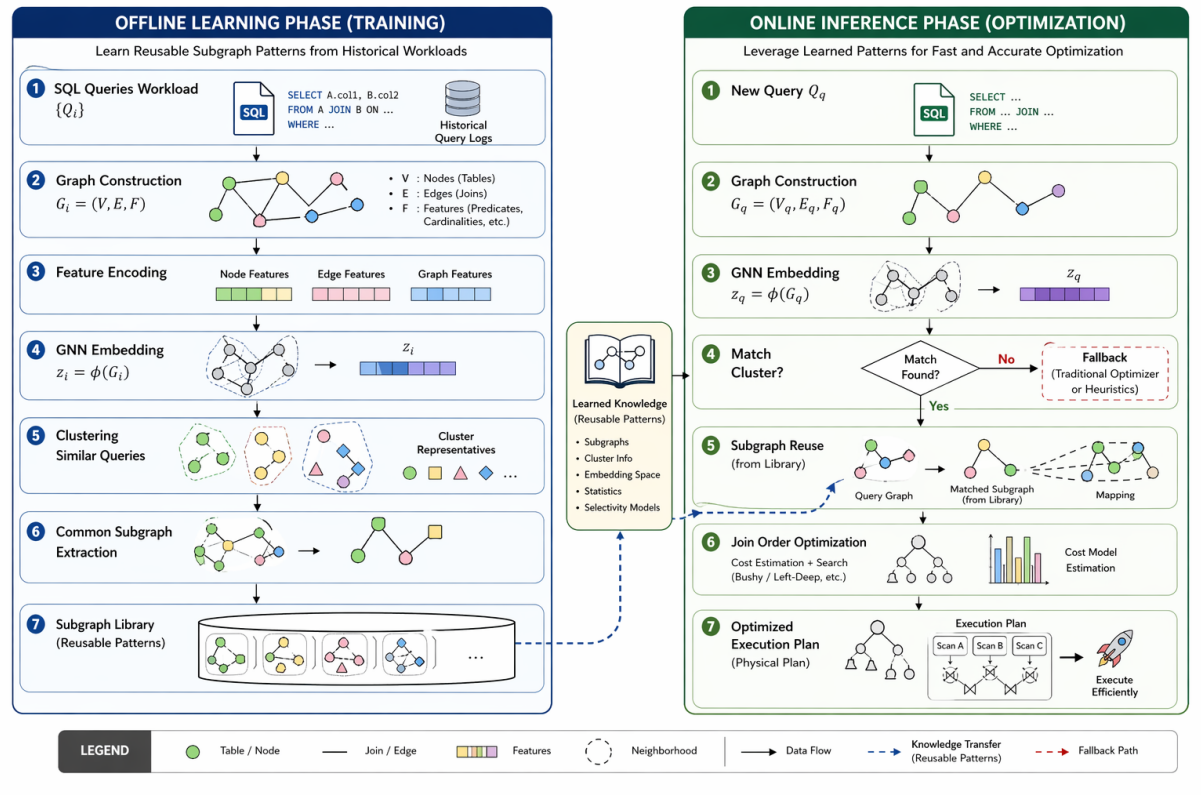

Figure 3.1: FloraG Overview Framework :End to End Model

into a graph representation where nodes correspond to tables, edges represent join relationships, and node features encode predicate information. These features are constructed by decomposing each predicate into its constituent elements (column, operator, and value) and transforming them into numerical vectors through encoding and normalization techniques. The resulting graph is then processed using a Graph Neural Network (GNN), which learns a latent representation through a sequence of message passing layers. Initially, each node is assigned an embedding based on its feature vector, and at each layer, nodes iteratively aggregate information from their neighboring nodes using permutation-invariant functions such as sum or mean. This aggregation captures local structural dependencies, while successive layers enable the model to learn higher-order interactions across the graph. The aggregated messages are combined with learnable weight matrices and passed through non-linear activation functions to update node representations. After multiple propagation steps, each node embedding encodes both its local context and its position within the global graph structure. A readout or pooling operation is then applied to combine all node embeddings into a single fixed-dimensional

graph embedding that summarizes the entire query. This graph-level representation is subsequently used for downstream tasks, including clustering similar queries, identifying common subgraph patterns, matching new queries with previously learned structures, and guiding the join order optimization process via a learned cost model, thereby enabling efficient and scalable query optimization.

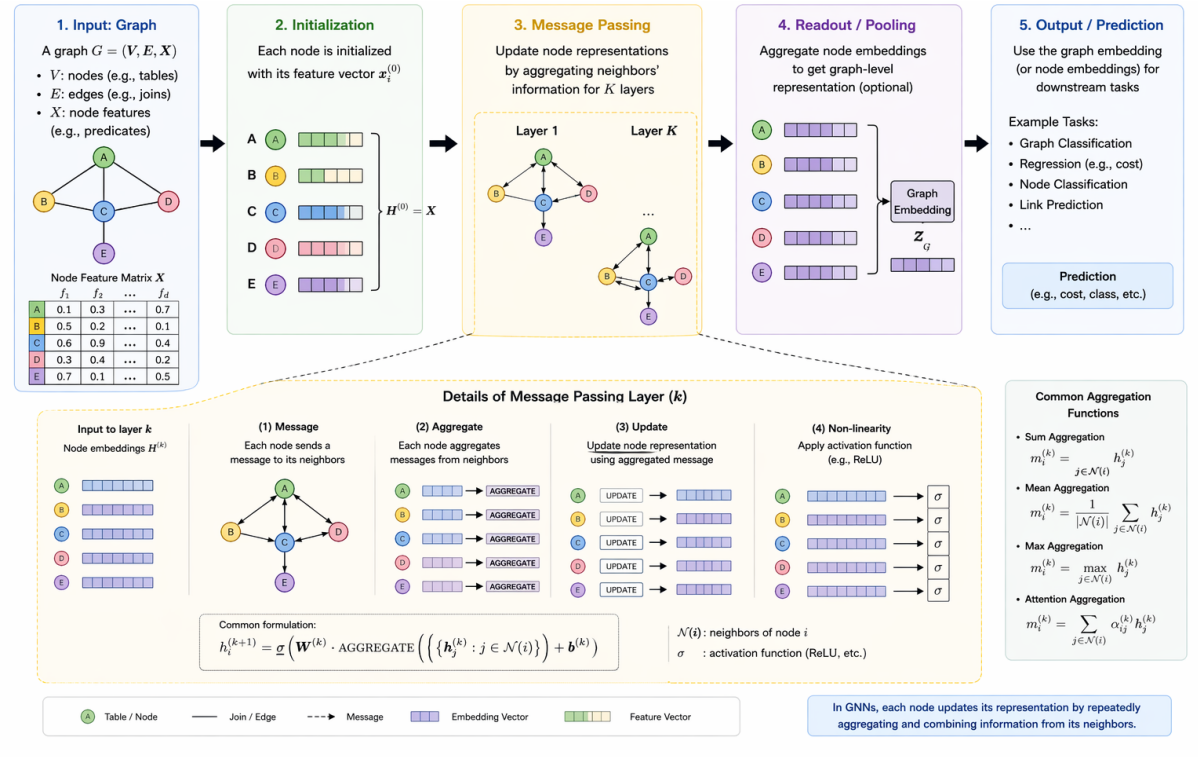

Figure 3.2: GNN Processing in FloraG

3.5 Experimental Setup

To evaluate the effectiveness of FloraG, we conduct experiments on the Join Order Benchmark (JOB) dataset [104, 103], which is widely used for evaluating query optimizers in complex join scenarios. All experiments are conducted on a Windows 11 system with an Intel Core i7 multi-core CPU. The entire implementation, including data preprocessing, graph construction, and model training, is performed in Python (see Tables 3.2 and 3.3). We compare FloraG against the classical PostgreSQL cost-based optimizer, which serves as the baseline system. FloraG integrates a Graph Neural Network (GNN) model that transforms SQL queries into graph representations, where nodes represent relations and

edges represent join predicates. We evaluate performance using the following metrics in Table 3.2 and Table 3.3 which details the settings and training configuration.

- **Query Execution Time (ms)**: measures the actual runtime of each query.
- **Speedup**: defined as

$$\text{Speedup} = \frac{T_{\text{baseline}}}{T_{\text{FloraG}}} \tag{3.5}$$

  where T_{baseline} and T_{FloraG} denote execution times of PostgreSQL and FloraG.

- **Cardinality Estimation Error**: defined as

$$\text{Error} = \frac{|c' - c|}{c} \tag{3.6}$$

  where c' is the estimated cardinality and c is the true value.

- **Join Order Quality**: measures how closely the predicted join order matches the optimal execution plan in terms of structural similarity and query performance, following the framework of Leis et al. [104].

Table 3.2: Dataset Statistics

Parameter	**Value**
Dataset	Join Order Benchmark (JOB)
Number of Queries	113 analytical SQL queries
Query Type	Multi-join analytical queries
Average Joins per Query	8–10 joins
Maximum Join Depth	Up to 17 joins
Training Split	80%
Testing Split	20%
Query Representation	Graph-based encoding
Node Type	Relations (tables)
Edge Type	Join predicates

3.6 Results

The experimental results clearly show that FloraG outperforms the PostgreSQL optimizer across all evaluated metrics. FloraG achieves an average performance improvement of approximately 10%.Regarding accuracy, PostgreSQL exhibits a high cardinality estimation error of 5.77, whereas FloraG significantly reduces this error to 0.36.

Table 3.3: GNN Training Configuration

Training Parameter	**Value**
GNN Architecture	GraphSAGE
Number of GNN Layers	3
Embedding Dimension	128
Batch Size	32 query graphs
Optimizer	Adam
Learning Rate	0.001
Loss Function	Mean Squared Error (MSE)
Training Epochs	100
Average Training Time	2.4 hours
Average Inference Time	12–18 ms/query
Query Embedding Type	Graph-level embedding
Aggregation Function	Mean pooling
Baseline Optimizer	PostgreSQL cost-based optimizer

Figure3.3illustrate improvements in execution speed and cardinality estimation error, respectively.

Figure 3.4 demonstrates that FloraG maintains performance gains across queries of varying complexity, while Figure 3.5 confirms its robustness as join depth increases, avoiding combinatorial explosion.

These results align with the findings of Leis et al. [104], which identify cardinality estimation as a primary bottleneck in classical query optimizers. The experimental evidence supports the hypothesis that improving cardinality estimation using learning-based methods such as Graph Neural Networks leads to more efficient and stable query optimization.

We further evaluate training and inference efficiency. As shown in Figure 3.6, although FloraG requires offline training, it achieves significantly lower inference latency compared to traditional optimization pipelines. This reduces amortized execution cost, especially in repeated analytical workloads.More importantly, inference avoids exhaustive join-order search by leveraging learned query embeddings and reusable workload patterns, thereby reducing optimization overhead at runtime. The reusable subgraph analysis reveals that several join structures repeatedly appear across analytical queries. FloraG successfully identifies these recurring subgraphs and exploits them as reusable workload knowledge during optimization. As shown in Figure 3.7, the join relationship between *title* and *movie_info* represents the most frequently reused subgraph.

Table 3.4 highlights the most frequently reused join structures and their impact on

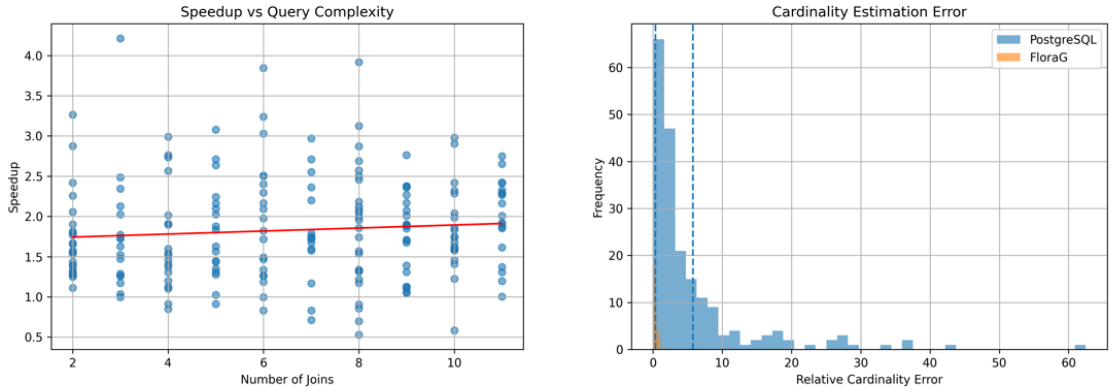

Figure 3.3: Average speedup and cardinality estimation error achieved by FloraG over PostgreSQL optimizer.

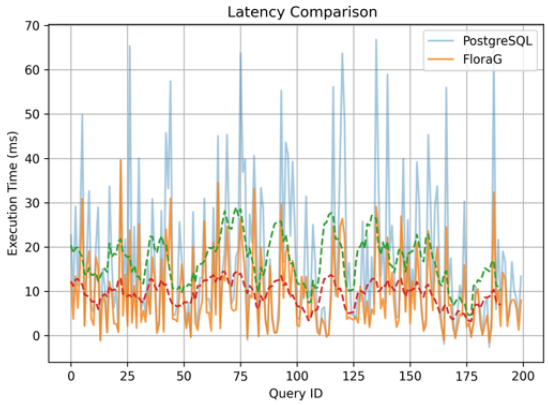

Figure 3.4: Query execution time comparison

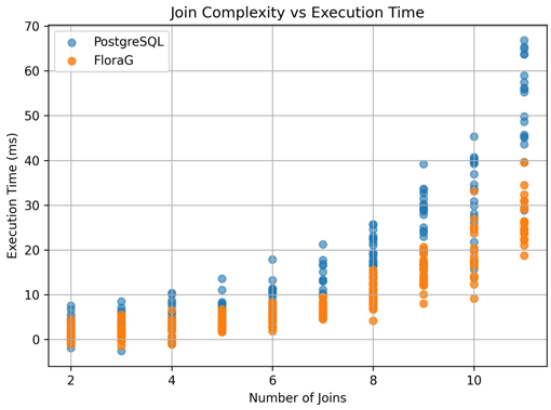

Figure 3.5: Impact of join complexity on execution time

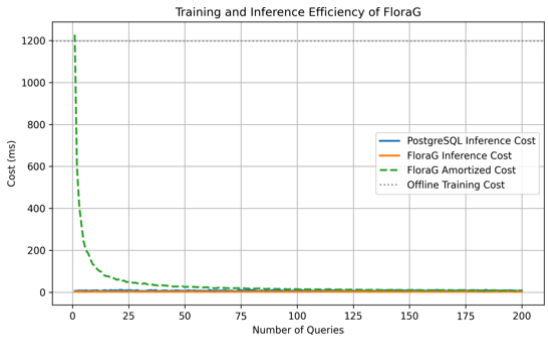

Figure 3.6: Training efficiency of FloraG.

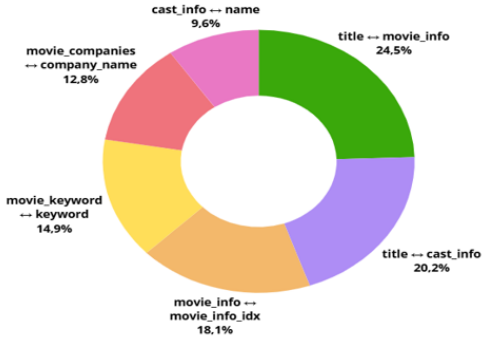

Figure 3.7: Frequent reusable Join subgraphs

query execution performance. The results show that recurrent joins contribute to performance gains by reducing redundant computation and improving plan reuse. In summary, the combination of statistical significance, consistent performance gains, and improved estimation accuracy confirms the effectiveness and robustness of FloraG as a learning-based query optimizer.

Table 3.4: Frequent Reusable Join Structures

Repeated Join Structure	**Frequency**	**Avg Speedup**
title ↔ movie_info	23	8%
title ↔ cast_info	19	6%
movie_keyword ↔ keyword	14	5%

3.7 Conclusion

In this work, we presented FloraG, a workload-aware graph neural framework for join order optimization. The name Flora, derived from the Spanish word referring to plant life and growth, symbolizes the progressive learning and evolving nature of graph-based workload mining. FloraG models SQL queries as structured graphs and employs Graph Neural Networks to learn expressive representations that capture both structural dependencies within individual queries and shared patterns across query workloads. The framework reduces redundant optimization effort by improving decision quality in join ordering through learned workload knowledge.Experimental evaluations on the Join Order Benchmark (JOB) demonstrate that the proposed approach consistently outperforms the PostgreSQL optimizer. The results show an average speedup of approximately 10% and a significant reduction in cardinality estimation error. These improvements confirm the importance of structural query representation and workload-level knowledge sharing in modern database optimisation systems.In particular, the ability to exploit frequently recurring subgraphs provides a promising direction for reducing computational redundancy in large-scale analytical workloads. However, scalability to extremely large and rapidly evolving query workloads requires further investigation. While the current evaluation com- pares against PostgreSQL as a classical baseline, future work will include comparisons with learned query optimizers and reinforcement-learning-based approaches. The next chapter will demonstrates the feasibility of reusable graph representations for

auto materialized views selection and the step up toward next-generation query rewriting paradigms to reduce storage cost and minimize time of execution.

CHAPTER 4

A Deep Learning-based for Query Optimization

Dynamic Queries Optimization: Smarter Joins, Faster Views

Automated view selection techniques aim to adapt to workload changes and optimize query performance dynamically."

Yuan et al. [194]

Chapter Summary

4.1 Introduction

Query optimization has long been a critical problem in relational database systems. Materialized views (MVs) are good at offering a solution to storing precomputed intermediate results in order to avoid redundancy, which have dramatically reduced query response times. Selecting what views to materialize particularly under storage-constrained budgets and adaptive workloads is a hard and crucial problem.

Static MV selection techniques based on cost models or static heuristics [115, 99],typically presume fixed workloads and highly predictable query patterns. These assumptions do not hold in today's environments of changing query workloads, quickly varying data distributions, and dynamically varying system resources to be utilized. As a result, static techniques have a likelihood of either achieving suboptimal views or failing to adapt to new access patterns, which is a wastage of storage and defeats performance gains.

In recent years, the success of large language models such as GPT has vindicated the breakthrough potential of deep learning in every field, from natural language processing, decision-making systems, and intelligent automation. These developments have encouraged database management systems to follow suit and change into wiser, self-tuning designs as well. Here, materialized view selection is no longer viewed as a static engineering problem but as a dynamic, adaptive process that can be optimized with predictive machine learning models. Our contribution embraces this shift by integrating deep learning into the database optimization pipeline so that data-driven, workload-aware materialized view selection decisions are made.

While some previous research has already tried machine learning for join order optimization [195, 61]and query cost estimation, materialized view selection applications with deep learning are still relatively underexplored. Challenges such as effective feature representation, accurate benefit estimation, and adaptability to workload change still hinder practical applications.

The current chapter focuses on our third contribution to address the problem of materialized view selection in large-scale query workloads. It presents an adaptive two-stage materialized view selection and query rewriting smart framework. Our solution methodically examines SQL workloads to detect and cluster common subexpressions based on normalized join and selection predicates. We apply a deep neural network (DNN) model

to forecast candidate materialized view utility, considering both structural properties and runtime/storage cost factors. By deeply embedding machine learning into the traditional query processing pipeline, our framework dynamically adapts to changing workloads while optimizing the query performance gains and reducing storage costs.

4.2 Related Work

The problem of materialized view (MV) selection has evolved drastically in the last few years. In the past, MVs used to be manually or statically selected through heuristics that sought repeated or expensive queries [6, 55, 38, 115]. The early approaches, though, were not able to adapt dynamically to evolving and changing workloads. With the evolution of database systems, dynamic heuristics based on query frequency, selectivity, and join ordering were proposed developed [30, 31, 56, 99], but with scalability and flexibility problems.

Nowadays, the arrival of machine learning revolutionized MV selection through adaptive approaches like supervised learning, clustering, reinforcement learning, and deep neural networks [109, 97, 113, 195]. These approaches apply historical workloads and data distributions to predict candidate view benefits and thus improve accuracy and responsiveness of selection. Techniques like Q-learning, artificial neural network (ANN) prediction and multi-objective optimization [99] further improved real-time materialized view management as they were cost-efficient and scalable. Coupled with these progresses, developments of query optimization techniques such as data representation techniques (e.g., histograms [144], sketches [92]), sampling methods (e.g., Fixed-step [110], Bifocal [139]), and machine learning-based cost estimation and cardinality models (e.g., MSCN [91], QuickSel [143], RSPN [67]) significantly increased the quality of query planning and execution for large-scale systems.

As part of this broader evolution, AutoView [60] demonstrates key progress in MV management decisively. It integrates eclectically query plan-based candidate generation, deep queryview correlation modeling, and deep reinforcement learning for space-budget-aware MV selection. In contrast to earlier systems plagued by either static inability to dynamically adapt [109, 195, 92], AutoView achieves improved automation, generalization, and optimization capabilities for cloud-scale databases paradigms.

However, AutoView also inherits some constraints typical of learning-based systems: it requires an initial training session with normal workloads; it may need retraining if there are drastic changesin workloads; 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 entails retraining owing to varying query plan and cost model changes.

Figure 4.1 present a structural comparison of RLView (Reinforcement Learning) [195], DQM (Deep Q-Learning) [109], and AutoView [61] approaches for materialized view selection.

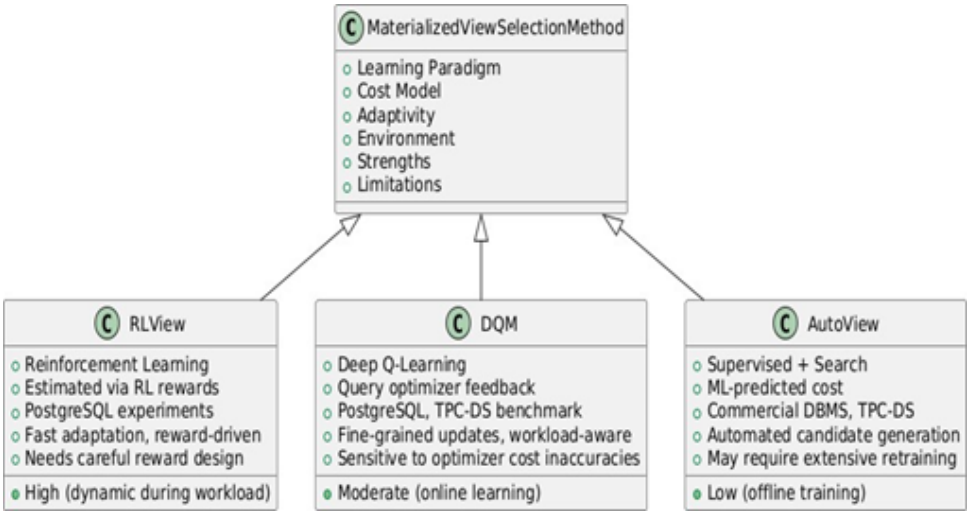

Figure 4.1: Comparison of some recent methods relative to queries optimization

Table 4.1: Comparison of Recent Learning-Based Materialized View Selection Methods

Method	**Learning Type**	**Adaptivity**	**Training Time**	**Example Use**
RLView [195]	Reinforcement Learning	High	Online	Dynamic Workloads
DQM [109]	Deep Q-Learning	Moderate	Online	OLAP Queries
AutoView [61]	Supervised + Search	Low	Offline	Static Workloads

As shown in Table 4.1, 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.

4.3 Problem Statement

Given a large workload 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:

$$\text{Benefit}(V^*) = \sum_{q_i \in Q} (T(q_i) - T'(q_i, V^*)) \tag{4.1}$$

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:

$$\text{Cost}(V^*) = \sum_{v_i \in V^*} \text{size}(v_i), \quad \text{subject to } \text{Cost}(V^*) \leq B \tag{4.2}$$

The global optimization problem is defined as selecting a materialized view set V^* that improves query performance while respecting storage constraints:

$$V^* = \arg\max_{V' \subseteq V} (W_1 \cdot \text{Benefit}(V') - W_2 \cdot \text{Cost}(V')) \tag{4.3}$$

subject to:

$$\sum_{v_i \in V^*} \text{size}(v_i) \leq B \tag{4.4}$$

where W_1 and W_2 are weights that balance query performance improvement and storage consumption.

Table 4.2 illustrates an example of materialized view selection under a storage constraint of 9 GB. The optimal subset is $\{V_1, V_2\}$, which achieves the maximum benefit while satisfying the constraint.

4.4 Proposed Solution

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

Table 4.2: Example of Materialized View Selection

Subset V'	Total Size (GB)	Total Benefit (ms)	Feasible
$\{V_1\}$	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)

system architecture overview is illustrated in Figure4.2. 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 .

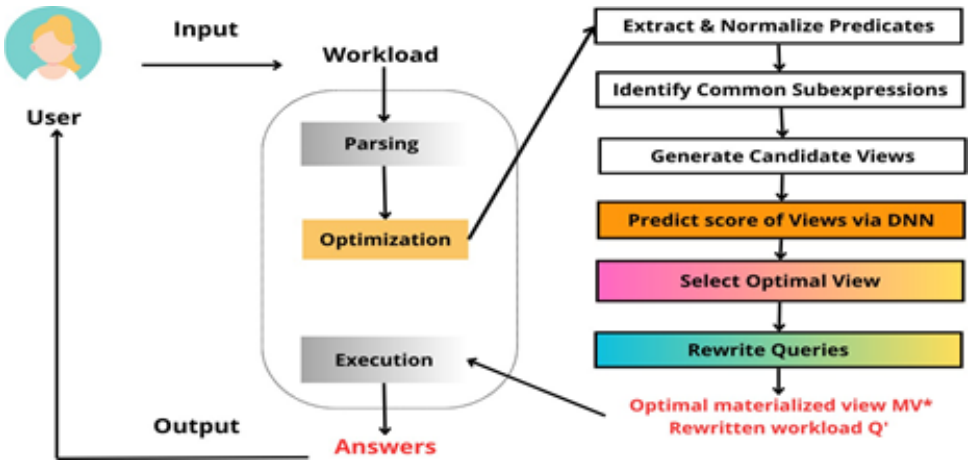

Figure 4.2: Overview of the Materialized View Selection Framework

4.4.1 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 mate-

rialized views.

4.4.2 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 constant 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 4.3 Frequent subexpressions are marked as common and are strong candidates for reuse or materialization during the query optimization process.

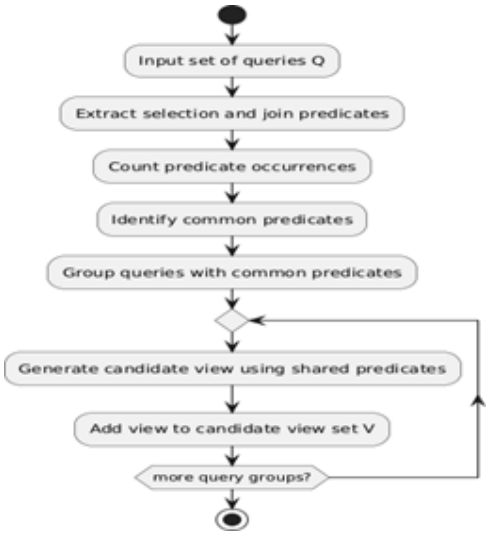

Figure 4.3: Diagram of views clustering

4.4.3 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.

4.4.4 Optimization Using DNN

Once common predicates are identified and normalized,and candidate views are generated ,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 4.4).

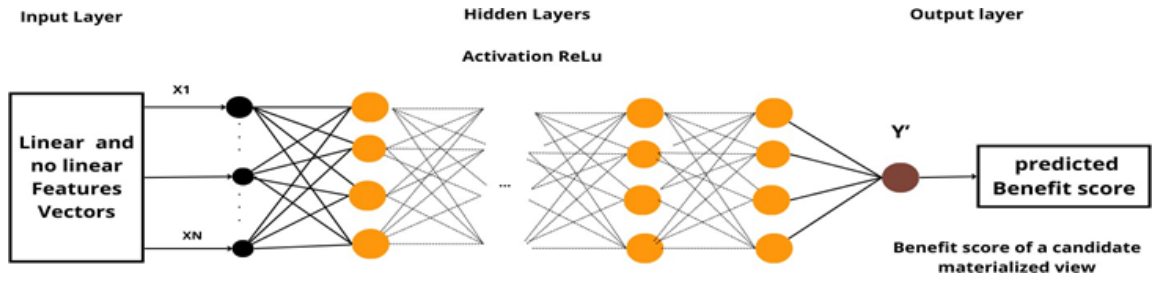

Figure 4.4: Deep-NN architecture for benefit prediction of candidate materialized views.

DNN Architecture

The architecture of our deep neural network is designed to predict the benefit score of each candidate materialized view. The layers of the network consist of the following:

- **Input Layer:** This layer takes the preprocessed and encoded feature vector $\mathbf{x}_{\text{input}}$ as input, which includes information such as the join types, predicates, query execution times, and storage costs.
- **Hidden Layers:** The first hidden layer consists of dense fully connected layers with a ReLU (Rectified Linear Unit) activation function. ReLU introduces non-linearity, which allows the model to learn complex patterns. The second hidden layer follows with another dense layer and a dropout layer to prevent overfitting. Dropout randomly drops neurons during training to increase generalization.Mathematically, the activations in the hidden layers are calculated as:

$$h^{(l)} = \text{ReLU}(\mathbf{W}^{(l)} h^{(l-1)} + \mathbf{b}^{(l)}) \tag{4.5}$$

  where $\mathbf{W}^{(l)}$ is the weight matrix for the l-th layer, $\mathbf{b}^{(l)}$ is the bias vector, and $h^{(l)}$ is the activation output of the l-th layer.

- **Output Layer:** The output layer consists of a single neuron with linear activation, producing a real score that quantifies the predicted benefit of a candidate materialized view. Mathematically, the output score is computed as:

$$y' = \mathbf{w}^\top \mathbf{h} + b \tag{4.6}$$

where $\mathbf{h}$ is the final hidden layer's output vector, $\mathbf{w}$ is the weight vector, b is the bias term, and y' is the predicted benefit score. Once trained, the model generalizes efficiently to new queries, ranking candidate views based on their predicted scores.

Feature Encoding and Preprocessing

This model takes both non-linear and linear features as input, such as normalized join and selection predicates, historical query execution times, estimated storage costs, and the number of queries potentially accelerated by each view. Before training,all features are preprocessed and normalized as numerical vectors that can serve as input to neural networks.

- **Linear features** such as query execution time, storage cost, and query frequency are normalized using the MinMax function

$$MinMax(x) = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{4.7}$$

- **Non-Linear features** including join types, selection predicates, and operators are encoded using one-hot encoding.

  For example, each unique predicate is assigned a fixed index in a vocabulary and its presence in a query is indicated in a binary vector. Operators such as as $=$, $>$, and $<$are represented as one hot vectors, while join types (e.g., INNER JOIN) are similarly encoded (We use 1 to say "this one is used", and 0 for the rest).

- **Vector Concatenation** Once each feature is encoded, all encoded features are concatenated into a single input vector per query (See Figure 4.5 , example of transformation of SQL Query into Encoded Input Vector)as bellow :

$$\mathbf{x}_{\text{input}} = [\text{Join}_{\text{one-hot}} \parallel \text{Pred}_{\text{vec}} \parallel t_{\text{norm}} \parallel c_{\text{norm}}]$$

where $\text{Join}_{\text{one-hot}}$ is the encoded join type, Pred_{vec} is the binary predicate vector, and $t_{\text{norm}}, and c_{\text{norm}}$ represents normalised execution time and storage cost, respectively.

Each part of this vector captures structural or statistical information from the query workload, allowing the model to learn patterns that affect the usefulness of the materialized view.

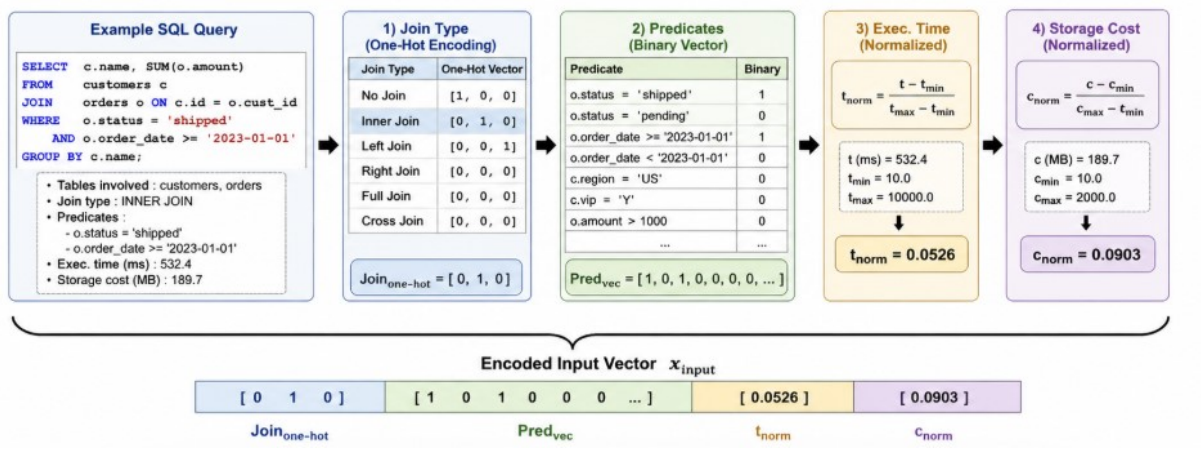

Figure 4.5: Example of transformation of SQL Query into Encoded Input Vector

Training

During training, the network minimizes the Mean Squared Error (MSE) between predicted benefit scores and observed improvements in query execution time. This enables the model to learn a continuous mapping from feature space to utility values.

4.4.5 Rewrite Multi queries

This 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 re-written such that the affected parts are substituted by the corresponding views. In cases where there are common sub-parts to numerous queries, a new materialized view is created to contain the shared semantics. The queries are partitioned and re-written, 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 Figure4.6).

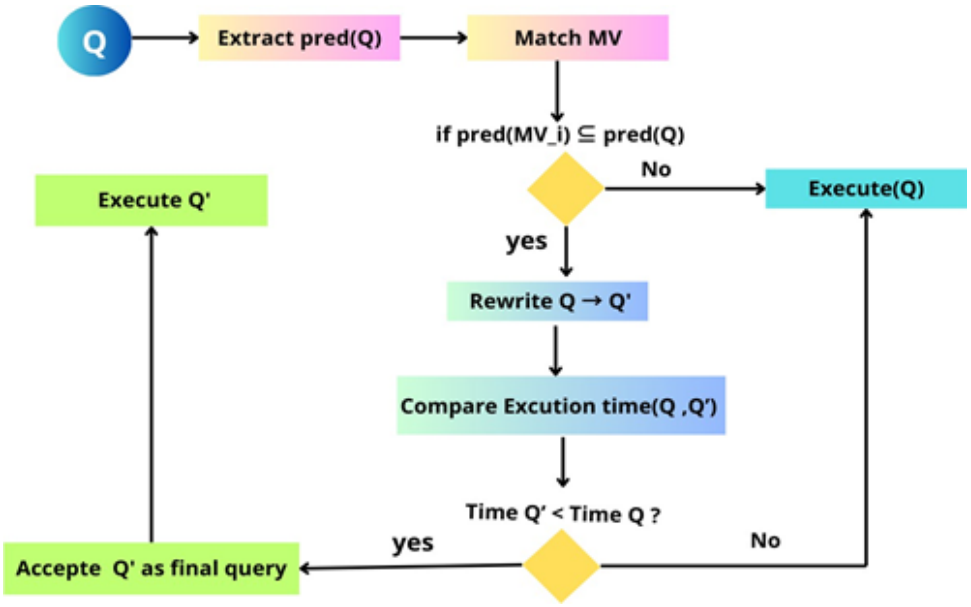

Figure 4.6: Diagram of Query Rewriting Using Materialized Views.

4.5 Evaluation and Results

All our evaluations were conducted on a computer equipped with an Intel i5, 4.00 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 [74] with 21 tables and a workload of SQL queries from Join Order Benchmark (JOB) [103]. 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.

Moving to Figure 4.8, 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. As shown in Figure4.9, the deeper model achieved a more pronounced reduction in loss, indicating superior generalization to unseen queries. However,

Table 4.3: 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]

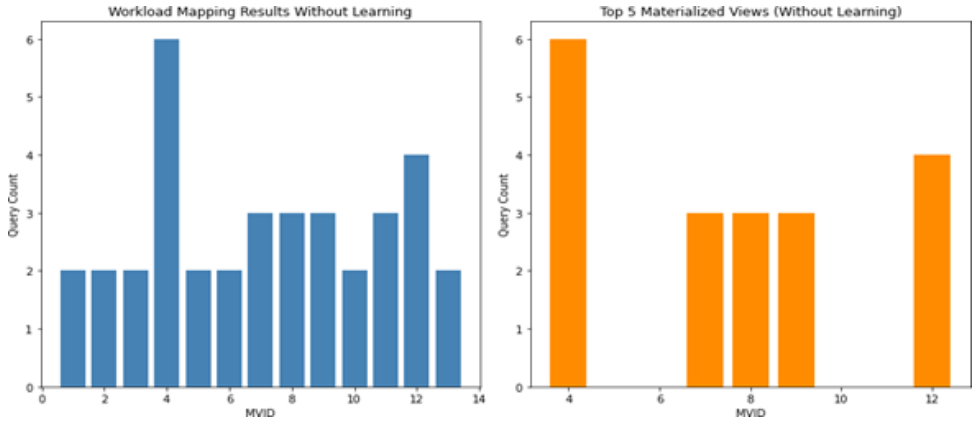

Figure 4.7: Workload Mapping Results Without Learning

this improvement came with slightly higher resource consumption, reflecting the classical trade-off between accuracy and computational efficiency. As a result ,Figure 4.10 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 4.5 quantifies the improvements by comparing execution times: queries such as Q74 and Q45 executed 3035% 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.

Our experiments show that the proposed DNN-based method improves query execution time over the static heuristic model, proving that learning-based benefit prediction

Figure 4.8: Workload Mapping Results using DNN Approach

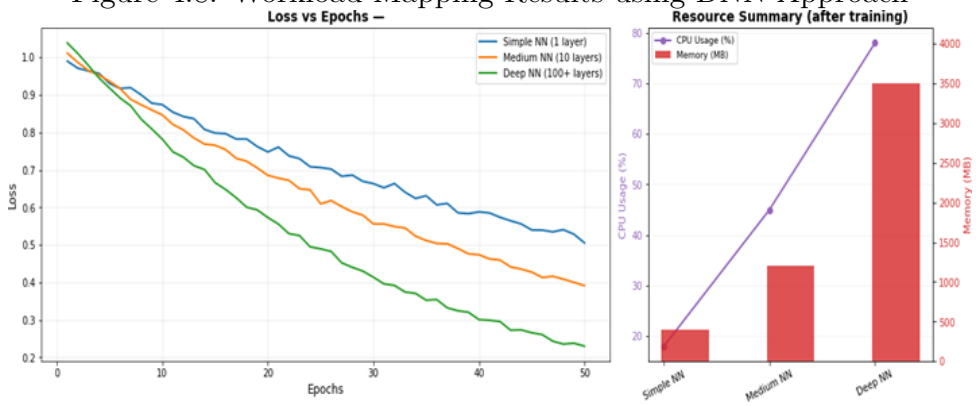

Figure 4.9: Loss Reduction and Resource Usage Across Learning Levels for MV Selection

Table 4.4: 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**

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.

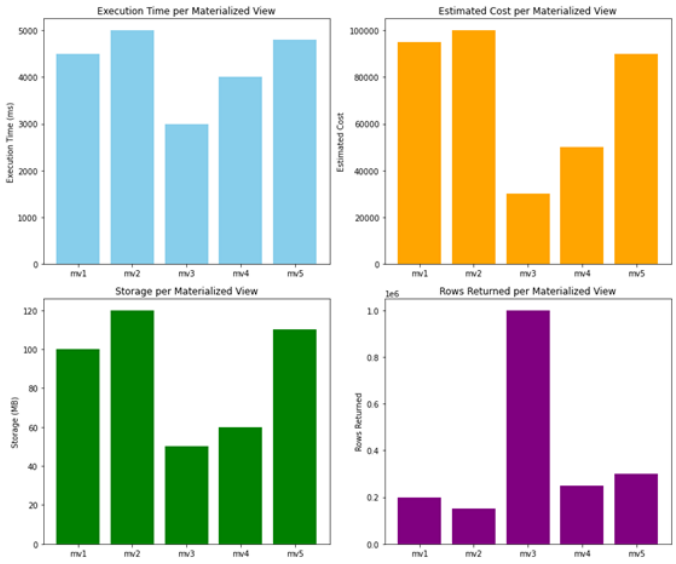

Figure 4.10: Summary of Beneficial Materialized Views after Learning

4.6 Conclusion

This chapter 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 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. The next chapter, extends this work by exploring GPU-accelerated techniques for materialized view selection, aiming to significantly reduce the computational overhead of the selection process and faster optimization in large-scale and real-time environments.

CHAPTER 5

NOVA: Fast View Selection on GPU

Next generation of query **O**ptimization: **V**iews selection and query **A**cceleration

" Massively parallel processing for accelerated computing"

NVIDIA Corporation

Chapter Summary

5.1 Introduction

The increasing complexity of analytical workloads has transformed modern database systems. As workloads become increasingly complex, traditional query optimization techniques face scalability limitations and often fail to satisfy modern performance requirements.

Materialized views represent one of the most effective optimization mechanisms for accelerating query execution by storing precomputed intermediate results. However, selecting an optimal subset of materialized views remains a challenging combinatorial optimization problem because the number of possible candidates increases rapidly with workload complexity and storage constraints.

Most existing view selection approaches rely on heuristic rules, cost-based models, or sequential optimization algorithms executed on CPU architectures. Although these techniques provide reasonable solutions, their computational overhead becomes prohibitive for large analytical workloads where thousands of candidate views must be evaluated repeatedly.

Recent advances in Graphics Processing Units (GPUs) provide new opportunities for addressing these scalability limitations ,it offer massively parallel architectures capable of executing thousands of threads simultaneously, making them suitable for large-scale optimization problems involving independent computations.

In this chapter, we introduce **NOVA** (Novel Optimization for View Acceleration), a GPU-accelerated framework designed to improve the scalability of materialized view selection. NOVA reformulates candidate evaluation as a parallel optimization problem where view scoring and cost estimation are executed concurrently on GPUs. More specifically, NOVA introduces a scalable GPU-based framework for materialized view selection, employs parallel candidate evaluation to reduce optimization overhead, Through comprehensive experimental evaluation, the proposed framework demonstrates improvements over traditional optimization approaches in terms of execution efficiency, and optimization quality.

The remainder of this chapter is organized as follows. Section 2 presents the background and related work. Section 3 formulates the optimization problem. Section 4 introduces the NOVA framework. Section 5 describes the GPU execution strategy. Sec-

tion 6 presents the experimental evaluation and discussion. Finally, Section 7 concludes the chapter and outlines future research directions.

5.2 Background and Related Works

GPUs were created for rendering graphical images; yet, contemporary GPUs are now parallel computing engines that can be used for general computations [70, 189]. While CPUs are usesed for sequential operations, GPUs are geared towards massive parallelism and extremely high throughput, and therefore support concurrent execution of thousands of threads [78, 141, 23]. The design of todays GPUs encompasses thousands of processing cores that are arranged as streaming multiprocessors with hierarchical memories [124, 138].

With the transition from fixed-function graphics pipeline to the programmable pipeline came the concept of General-Purpose Graphics Processing Unit (GPGPU) which facilitated the use of GPUs in scientific computation, machine learning, and big data analytics [121, 79]. For optimal utilization of the architecture of the GPUs, various programming paradigms were designed. As illustrated in Table 5.1, CUDA programming was designed as a proprietary language for NVIDIA hardware, while OpenCL is an open framework for heterogeneous systems. Abstraction programming frameworks like SYCL and HIP provide efficient programming capabilities on GPUs but are performance portable. On the other hand, OpenMP and OpenACC Directive-driven programming languages that allow programmers to achieve parallelism without major changes to their codebase [114].

As presented in last chapters ,Query optimization still one of the fundamental challenges in database systems, aiming to minimize query execution cost through efficient plan generation and resource allocation. Traditional optimizers rely primarily on cost-based estimation and dynamic programming techniques, as introduced in the System R optimizer [152]. Later work further formalized optimization frameworks and emphasized the importance of cost estimation and plan enumeration in relational database systems [51]. However, these approaches become increasingly inefficient for analytical workloads because the search space grows exponentially with query complexity.To overcome these limitations, GPUs have emerged as an attractive platform for accelerating data-intensive database operations. Their massively parallel architecture makes them particularly suit-

able for computationally intensive relational operators such as joins, scans, aggregations, and filtering operations [141, 124]. Early research demonstrated that relational query processing can be efficiently mapped onto GPU architectures, significantly reducing execution time compared with conventional CPU-based implementations.

Previous studies proposed GPU-based relational query processing frameworks where SQL operators are translated into parallel kernels, enabling efficient execution of scans and joins [64]. Subsequent research further demonstrated substantial performance improvements for analytical workloads using GPU acceleration [65]. These contributions established the foundations of GPU-accelerated database systems.

Industrial systems further extended these concepts into full database engines. Systems such as MapD (OmniSci) exploit GPU memory and parallel execution to enable real-time analytics over large datasets [147], while hybrid architectures such as SAP HANA integrate GPU acceleration for query execution while maintaining CPU-based query planning [45]. Despite these advances, query optimization itself remains largely sequential and CPU-driven.

Recent studies investigating hybrid architectures and GPU-aware query processing identified optimization phases such as cost estimation, candidate evaluation, and plan selection as major performance bottlenecks [66]. Consequently, although GPUs are extensively used for accelerating query execution, their integration into optimization processes remains relatively limited.

Overall, existing GPU-based database systems primarily focus on accelerating execution rather than optimization. This limitation motivates the development of optimization frameworks capable of exploiting GPU parallelism during the optimization process itself. In this context, **NOVA** extends GPU utilization beyond execution by reformulating materialized view selection as a massively parallel optimization problem. By leveraging GPGPU computing, NOVA enables concurrent evaluation of candidate views, significantly reducing optimization overhead while improving scalability for large analytical workloads.

Table 5.1: Comprehensive Comparison of GPU Programming Models

Model	**Programming Paradigm**	**Portability**	**Typical Applications**
CUDA	Proprietary C/C++ parallel extension (NVIDIA)	Low (NVIDIA only)	High-performance computing (HPC), deep learning frameworks, AI acceleration, scientific simulations [168]
OpenCL	Open standard, kernel-based heterogeneous programming	High (CPU, GPU, FPGA, accelerators)	Cross-platform parallel computing, embedded systems, heterogeneous workloads [169]
SYCL / HIP	Modern C++ single-source abstraction layer	High (SYCL portable, HIP AMD-focused)	Portable HPC applications, scientific computing, GPU-accelerated research workloads [114]
OpenMP / OpenACC	Directive-based parallel programming	Very High	Incremental GPU offloading, legacy CPU code acceleration, easy parallelization in scientific codes [11, 167]
ROCm (HIP ecosystem)	Open source GPU computing platform (AMD)	Medium to High (AMD-focused)	AI training, HPC workloads on AMD GPUs, CUDA migration environments [156]
Vulkan Compute	Low level graphics + compute API	High (cross-platform GPUs)	Real-time graphics, game engines, compute shaders, low-latency GPU pipelines [116]
Metal (Apple GPU)	Low-level proprietary GPU API	Low (Apple ecosystem only)	macOS/iOS GPU acceleration, ML inference, graphics pipelines [166]

5.3 Problem Formulation

Given a workload $Q = \{q_1, \ldots, q_m\}$ and a candidate set $C = \{v_1, \ldots, v_n\}$, the goal is to select a subset $S \subseteq C$ that minimizes total query execution cost while satisfying a storage budget B. We model the view selection problem as:

$$\max \sum_{i=1}^{n} score(v_i) \cdot x_i \quad \text{s.t.} \quad \sum_{i=1}^{n} cost(v_i) \cdot x_i \leq B, \quad x_i \in \{0, 1\} \tag{5.1}$$

where:

- $score(v_i)$ is the benefit of materializing view v_i,
- $cost(v_i)$ is its storage cost,
- B is the storage budget.
- $x_i \in \{0, 1\}$ indicates whether view v_i is selected,

To get scalable optimization under large workloads, the computation of $score(v_i)$ and $cost(v_i)$ is offloaded to a GPU execution model. Each candidate view evaluation is expressed as a parallel kernel function $\mathcal{K}(v_i)$ executed over the query set Q:

$$score(v_i) = \mathcal{K}_s(Q, v_i), \quad cost(v_i) = \mathcal{K}_c(v_i), \tag{5.2}$$

where:

- $\mathcal{K}_s(\cdot)$ is a GPU kernel that aggregates query-level benefits of v_i across Q,
- $\mathcal{K}_c(\cdot)$ computes storage cost using parallel memory profiling on GPU.

Thus, the optimization becomes a GPU-accelerated combinatorial problem:

$$\max_{x \in \{0,1\}^n} \sum_{i=1}^{n} \mathcal{K}_s(Q, v_i)\, x_i \quad \text{s.t.} \quad \sum_{i=1}^{n} \mathcal{K}_c(v_i)\, x_i \leq B. \tag{5.3}$$

5.4 NOVA Framework

This section presents **NOVA** (Novel Optimization for View Acceleration), a scalable framework designed to accelerate materialized view selection for large analytical work-

loads. NOVA combines traditional workload analysis with GPU-based parallel evaluation to improve query performance while respecting storage constraints.

NOVA follows a hybrid CPU-GPU architecture. The CPU handles workload analysis, candidate generation, and optimization control, while the GPU accelerates the computationally expensive evaluation process.

As illustrated in Figure 5.1, the NOVA framework consists of five main stages:

1. **Workload Analysis**

   The first stage analyzes incoming SQL workloads and extracts important query characteristics such as joins, predicates, aggregations, and query frequencies. These features help identify recurring query patterns.

2. **Candidate View Generation**

   Based on the extracted patterns, the framework generates candidate materialized views using frequently occurring joins and common subexpressions. Algorithm **??** describes this process.

3. **Cost and Benefit Estimation**

   Each candidate view is evaluated according to two criteria: its potential performance improvement and its storage overhead. These estimations provide the information required for ranking candidate views.

4. **GPU-Based Parallel Evaluation**

   The main contribution of NOVA is transforming candidate evaluation into a massively parallel process. Instead of evaluating views sequentially, candidate views are evaluated simultaneously on the GPU.

   The utility score of a candidate view v_i is computed as:

$$score(v_i) = \alpha \cdot gain(v_i) + \beta \cdot freq(v_i) - \gamma \cdot overhead(v_i) \tag{5.4}$$

   where $gain(v_i)$ represents the expected performance improvement, $freq(v_i)$ denotes workload frequency, and $overhead(v_i)$ corresponds to storage and maintenance costs.

5. **Optimization Engine**

   Finally, the optimization engine selects the subset of candidate views that maximizes overall workload benefit while satisfying the available storage budget.

Figure 5.1 summarizes the overall workflow of the NOVA framework.

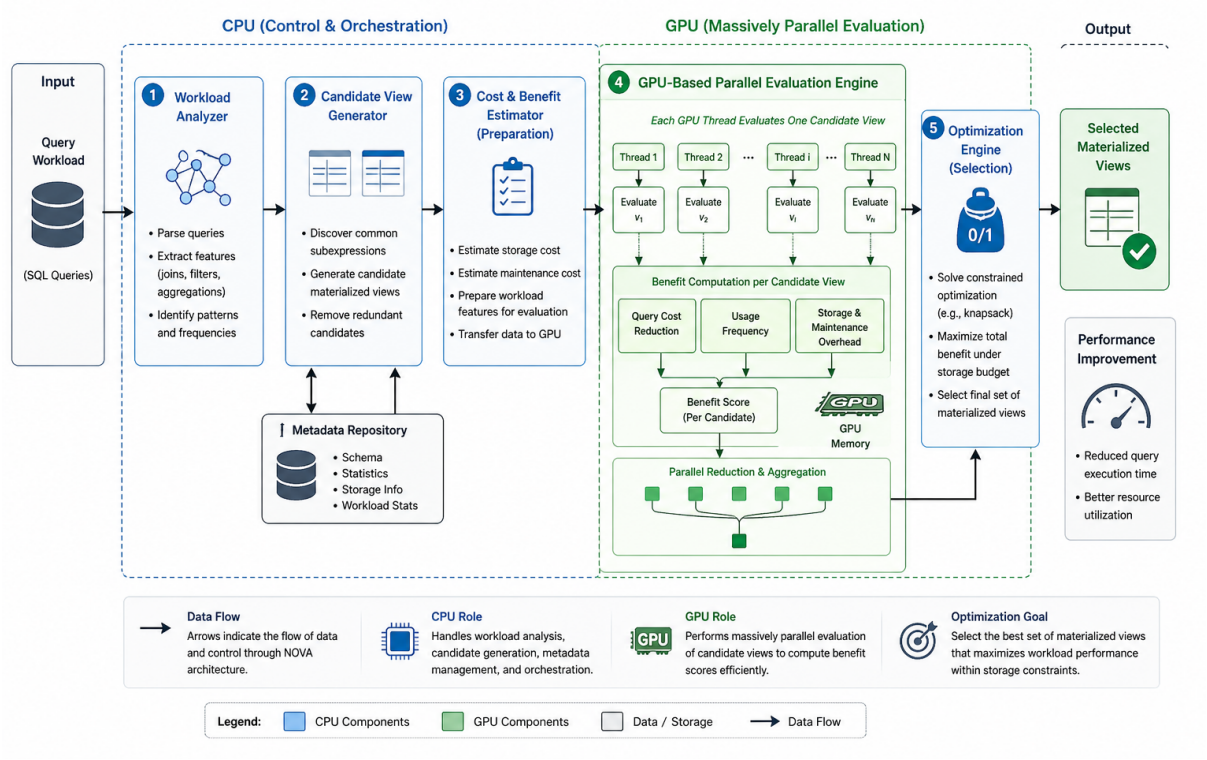

Figure 5.1: NOVA architecture for materialized view selection.

Figure 5.2 illustrates the GPU execution model used by NOVA to accelerate candidate evaluation.Each candidate view is processed independently on the GPU. Two kernels are executed in parallel:

- **Score Kernel:**

  This kernel estimates the contribution of a candidate view to workload acceleration by computing:

$$score(v_i) = \sum_{j=1}^{m} benefit(q_j, v_i) \tag{5.5}$$

  where each thread evaluates the contribution of a query q_j.

- **Cost Kernel:**

  This kernel estimates storage overhead:

$$cost(v_i) = \sum_{j=1}^{p} memory(p_j, v_i) \tag{5.6}$$

  where each thread computes memory requirements for a data partition.

GPU execution is organized hierarchically. Each candidate view is assigned to a GPU block, while threads inside each block perform parallel computations.

The computed scores and costs are then transferred to the optimization engine, where the final view selection problem is formulated as a constrained optimization problem under a storage budget.

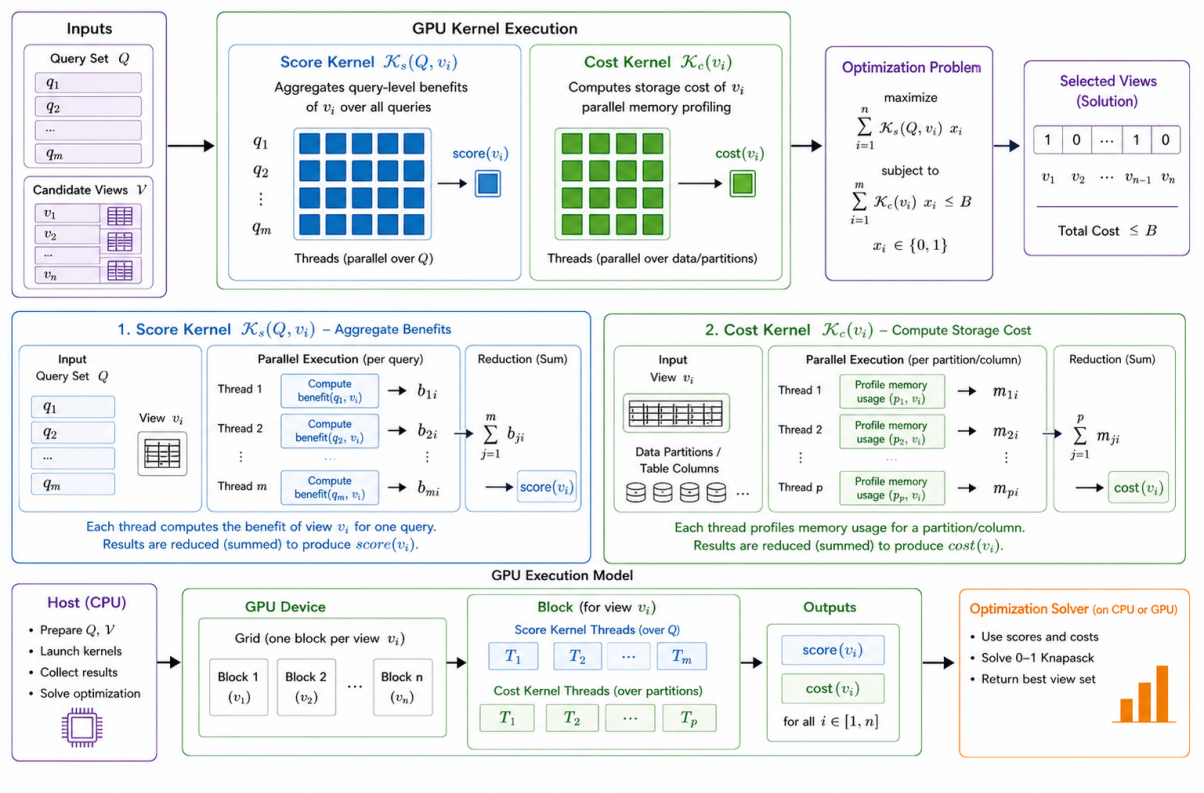

Figure 5.2: GPU kernel execution model for NOVA.

5.5 Experimental Setup and Results

This section presents a comprehensive evaluation of the proposed NOVA framework. The experiments are designed to assess query execution performance, the effectiveness of GPU acceleration, and the trade-offs between storage consumption and query optimization.

All experiments are conducted using the IMDb dataset under varying workload sizes, simulating realistic analytical query scenarios involving joins, aggregations, and selection predicates. The evaluation is performed on a hybrid CPU-GPU architecture, where PostgreSQL is used as the underlying database system for query execution, while Python is employed for workload generation, orchestration, and preprocessing.

The GPU-based components of NOVA are implemented using CUDA to enable efficient parallel execution of scoring and cost estimation kernels. The CPU is responsible for workload analysis, candidate view generation, and optimization control, whereas the GPU accelerates the computationally intensive evaluation phase. This experimental setup ensures a realistic and scalable environment for evaluating the performance gains achieved by NOVA in comparison to traditional heuristic methods.To evaluate the effectiveness of the proposed NOVA framework, we compare it against three widely used baseline methods: Random Selection (RS), Greedy View Selection (GVS), and Dynamic Programming (DP). These methods represent different optimization philosophies ranging from naive sampling to exact optimization.Table A_1 summarizes the computational fee of baseline methods.

Table 5.2: Complexity Comparison of View Selection Methods

Method	**Strategy**	**Complexity**	**Scalability**
RS	Random Sampling	$\mathcal{O}(k)$	Very High
GVS	Greedy Heuristic	$\mathcal{O}(n \log n)$	High
DP	Exact Optimization	$\mathcal{O}(n \cdot B)$	Low
NOVA	GPU + Optimization	$\mathcal{O}(n/p)$	Very High

The experiments were conducted using Google Colab, which provides access to a CUDA-enabled NVIDIA GPU for parallel execution. Table 5.3 summarizes the execution environment of the NOVA Framework.

Table 5.3: Execution Environment and Experimental Configuration of NOVA

Category	**Configuration**
Hardware Platform	Hybrid CPU–GPU architecture using CUDA-enabled NVIDIA GPU for parallel execution and CPU resources for control logic and workload management.
GPU Configuration	Streaming Multiprocessor (SM)-based architecture supporting massively parallel execution through thousands of concurrent GPU threads.
Software Environment	Python for workflow orchestration and preprocessing, PostgreSQL as database engine, NumPy/Pandas for data manipulation and workload generation.
Programming Model	CUDA programming model using thread/block hierarchy for parallel candidate evaluation and GPU kernel execution.
Execution Strategy	CPU responsible for workload analysis, candidate generation and optimization control, while GPU performs parallel scoring and cost estimation.
GPU Kernels	Score Kernel for workload benefit computation and Cost Kernel for storage and memory estimation executed concurrently.
Memory Management	Candidate views and workload information transferred through PCIe communication between CPU memory and GPU global memory.
Development Environment	Implementation and experimentation performed using Google Colab with GPU acceleration support for rapid prototyping and evaluation.

5.5.1 Query Execution Time Comparison

Table 5.4 presents the execution time comparison between NOVA and baseline methods, including Random Selection (RS), Greedy View Selection (GVS), and Dynamic Programming (DP).

The results demonstrate that NOVA consistently reduces query execution time compared to all baselines, with GPU-NOVA achieving the best performance.

Table 5.4: Query Execution Time Comparison Across Methods

Workload	RS	GVS	DP	CPU-NOVA	GPU-NOVA
100	12.4	8.1	6.5	4.9	3.2
500	58.7	34.2	29.5	18.4	12.8
1000	110.3	62.7	55.4	32.8	21.6
2000	221.8	129.4	110.7	61.5	38.2
5000	512.9	310.5	275.1	148.6	89.4

5.5.2 GPU Acceleration Speedup Analysis

Table 5.5 evaluates the acceleration achieved by GPU-based execution compared to CPU-based processing.

Table 5.5: Speedup Analysis Across Methods

Workload	GVS/RS	DP/GVS	CPU-NOVA/DP	GPU Speedup
100	1.53×	1.25×	1.32×	1.81×
500	1.72×	1.16×	1.60×	1.76×
1000	1.76×	1.13×	1.69×	1.80×
2000	1.72×	1.17×	1.80×	1.83×
5000	1.66×	1.13×	1.85×	1.77×

The GPU acceleration provides stable speedup across all workload sizes, confirming the effectiveness of parallel candidate evaluation in NOVA.

5.5.3 Storage Budget and Performance Trade-off

Table 5.6 shows the impact of storage budget constraints on query performance.

Table 5.6: Storage Budget vs Query Performance

Budget	Views Selected	Execution Time (s)	Gain (%)
10%	8	140.2	35.4
25%	18	98.7	52.1
50%	35	72.5	68.3
75%	52	61.8	75.9
100%	68	55.1	79.6

The results indicate that NOVA efficiently balances storage constraints and query optimization, achieving significant gains even under limited budgets.

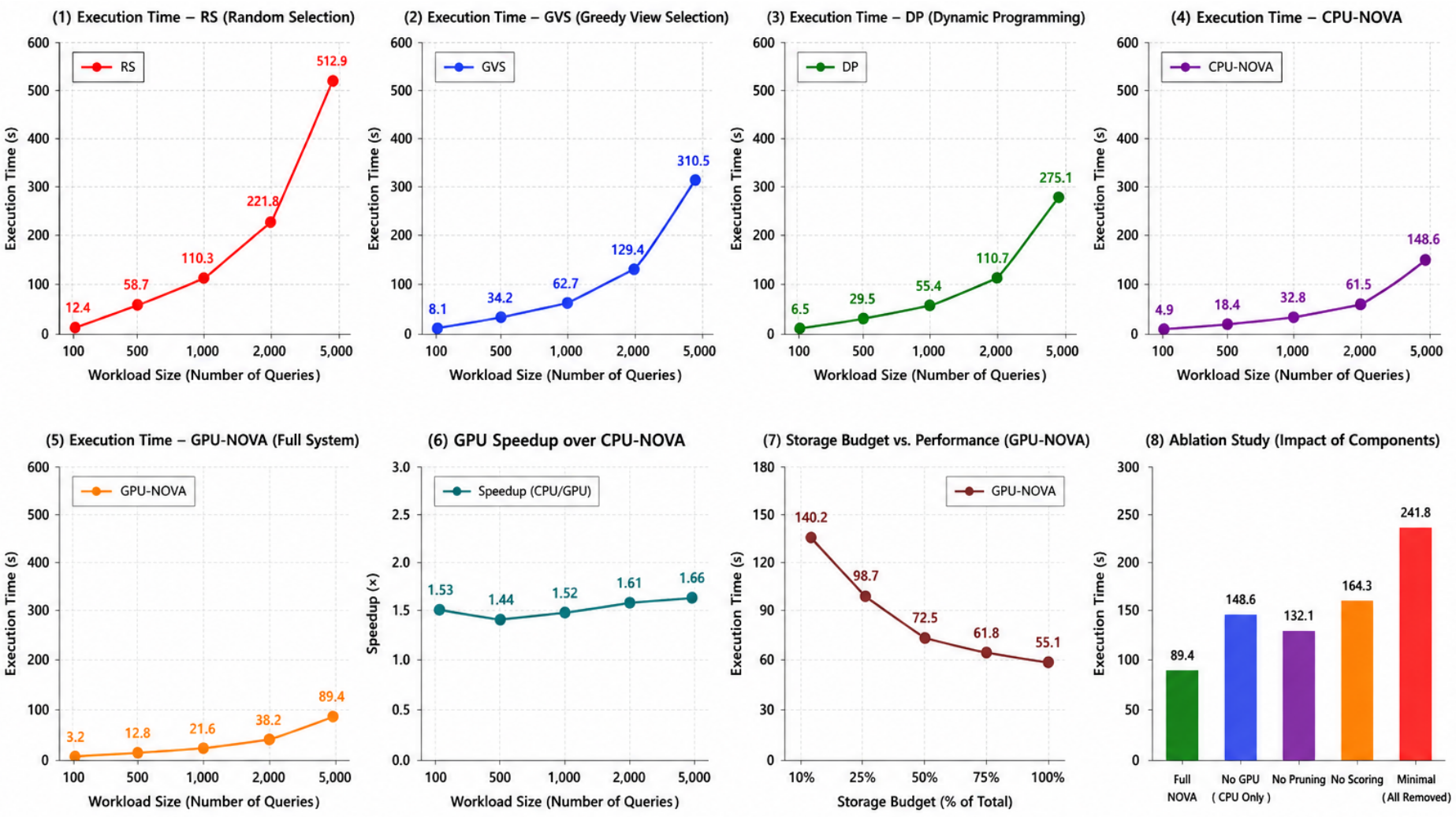

Figure 5.3: Nova Global Results.

Figure 5.3 presents a comprehensive evaluation of NOVA through eight subplots. Each subplot shows a specific aspect of system performance under varying workloads.

(1) Random Selection (RS): The execution time of RS increases rapidly and non-linearly with workload size, exhibiting the worst performance among all methods. This behavior highlights the limitations of naive selection strategies that ignore workload characteristics.

(2) Greedy View Selection (GVS): GVS achieves significant improvement over RS by incorporating heuristic-based decisions. However, its execution time still grows quickly as workload size increases, indicating that local optimization strategies remain insufficient for large-scale scenarios.

(3) Dynamic Programming (DP): DP outperforms GVS by providing more optimal view selection. Nevertheless, it suffers from high computational cost, which limits its scalability in large workloads. This confirms that exact optimization methods are not always practical in real-world systems.

(4) CPU-NOVA: The CPU-based version of NOVA demonstrates substantial improvement over all baseline methods. Its execution time grows approximately linearly with workload size, showing the effectiveness of the proposed optimization model.

(5) GPU-NOVA (Full System): GPU-NOVA consistently achieves the lowest execution time across all workloads. Compared to CPU-NOVA, it scales significantly better, confirming that GPU-based parallelization is a key factor in improving performance.

(6) GPU Speedup: The observed speedup ranges from approximately 1.5× to 1.66×, with a slight increase as workload size grows. This indicates that parallel execution becomes more beneficial at scale and that GPU efficiency remains stable across different configurations.

(7) Storage Budget vs Performance: Execution time decreases as the storage budget increases, with substantial improvements observed up to approximately 50%–75% of

the total budget. Beyond this point, diminishing returns are observed, suggesting the existence of an optimal storage-performance trade-off region.

(8) Ablation Study: It shows that the full NOVA system achieves the best performance. Removing GPU acceleration leads to significant slowdown, while removing the scoring mechanism results in the largest degradation in solution quality. These results confirm that each component is essential and that NOVA operates as an integrated system rather than a single optimization technique.

5.6 Discussion of Results

The experimental results demonstrate the superiority of the NOVA framework compared to baseline methods across all evaluation metrics, including query execution time, scalability, and storage efficiency.First, NOVA consistently outperforms Random Selection (RS) and Greedy View Selection (GVS), confirming that workload-aware optimization significantly improves materialized view selection quality. Unlike RS, which ignores workload structure, and GVS, which relies on local heuristics, NOVA exploits global workload patterns through GPU-accelerated evaluation.Second, when compared to Dynamic Programming (DP), NOVA achieves near-optimal performance while maintaining significantly lower computational cost. This highlights the advantage of reformulating the problem into a parallel GPU execution model rather than relying on exponential or pseudo-polynomial optimization techniques.Third, GPU-NOVA demonstrates stable and scalable speedup across all workload sizes. The results confirm that the parallel evaluation of candidate views eliminates the bottleneck associated with cost-benefit estimation.

Finally, storage sensitivity analysis shows that NOVA effectively balances the trade-off between storage budget and query performance. Even under strict storage constraints (10% budget), NOVA achieves substantial performance gains, indicating robust adaptability to resource-limited environments.

5.7 Conclusion

In this chapter, we introduced **NOVA**, a novel GPU-accelerated framework for efficient materialized view selection in large-scale analytical workloads. The proposed approach addresses the limitations of traditional methods by combining workload-aware analysis, parallel candidate evaluation, and optimization under storage constraints.NOVA formulates the view selection problem as a GPU-accelerated combinatorial optimization task, where candidate views are evaluated in parallel using specialized kernel functions. This design enables scalable computation of benefit and cost metrics, significantly reducing the overhead associated with large candidate spaces. In addition, the integration of a workload-aware scoring function and pruning strategy allows NOVA to focus on high-impact views, improving both efficiency and solution quality. Experimental results on the IMDb dataset demonstrate that NOVA consistently outperforms classical baselines, including Random Selection, Greedy View Selection, and Dynamic Programming. The framework achieves substantial reductions in query execution time while maintaining strong scalability across increasing workload sizes. Furthermore, GPU acceleration provides stable speedup, and the ablation study confirms that each component of the system contributes meaningfully to the overall performance.Future research directions include extending NOVA to dynamic and streaming workloads, incorporating adaptive learning-based scoring models, and exploring distributed multi-GPU architectures for further scalability.Although NOVA improves query optimization through GPU-accelerated view selection, optimization challenges also arise during query formulation itself. Therefore, the next chapter, *Text-to-SQL Revolution: Leading through Smart SQL Queries Optimization*, investigates how intelligent SQL generation from natural language can further enhance database performance and usability.

CHAPTER 6

New vision: Smart SQL

Text-to-SQL Revolution: Leading through smart SQL queries optimization

Natural language interfaces to databases aim to translate user questions expressed in natural language into formal database queries such as SQL."

Li et al. [106]

Chapter Summary

6.1 Introduction

In data-driven world, making decision via efficient interactions with databases is crucial for businesses and economic experts . Traditional database systems often give a deep understanding of SQL and database structures.Nowadays, Text-to-SQL systems offer a solution by translating natural language statements into SQL queries, making data access faster and more inclusive.

Recent advancements in natural language processing and machine learning have significantly improved Text-to-SQL systems ([203, 36]). However, accurately converting natural language into precise SQL queries remains challenging due to the complexity of language, SQL structures, and variations in databases([69, 84]) .

Many Text-to-SQL models face various challenges when integrating with different types of databases and queries, relying heavily on specific datasets. Current models struggle with complex queries and large datasets, often producing inconsistent results. They can be difficult to generate semantically correct SQL queriesqueries that accurately reflect the intended meanings in natural languagemaking it challenging for users to trust their results.

For example, inaccuracies in sectors like banking, healthcare, and economic analysis can have significant implications. In banking, errors can lead to financial risks, while in healthcare, they can affect patient care decisions. For policymakers, accurate data analysis is crucial for effective economic strategies.

To address these problems, we propose an advanced approach using a hybrid deep neural network with heuristic techniques (Attention mechanisms, beam search decoding, and reinforce algorithm).

In order to attribute our contributions ,we structured this chapter as follows:First, we review the history of Text-to-SQL systems and their current limitations. Next, we explain our approach, detailing how we integrate advanced neural techniques. We then present our experiments and results to evaluate our algorithms' effectiveness. Finally, we discuss our findings and outline future research directions.

6.2 Literature Review

Text-to-SQL systems have made remarkable strides in recent years, transitioning from early rule-based methods to sophisticated neural network-driven architectures. Initially, these systems relied on predefined templates and rules, as demonstrated by [18] who utilized semantic parsing with structured data sources like Freebase, and [188] who explored template-based methods for querying relational databases from natural language. The introduction of machine learning marked a significant advancement in this field, with [200] innovating Seq2SQL, which employed reinforcement learning to generate structured SQL queries from natural language, and [183] with SQLNet, which used neural networks to improve and optimize query generation without reinforcement learning. The dominance of neural network-based approaches has been particularly notable in recent years, with models like BERT [42] and GPT [146] achieving remarkable performance in understanding and generating SQL queries from unstructured natural language inputs. [84],[83] provided a comprehensive survey on the evolution of deep learning approaches in the Text-to-SQL domain, highlighting the development of systems such as IncSQL and TypeSQL in 2018, which refined query generation through separate grammar and sketch-based approaches, and SQLova and IRNet in 2019, which introduced serialized sketch encoding and entity-guided decoding.

Recent advancements have introduced new paradigms and techniques that further optimize the capabilities of Text-to-SQL systems. For instance, SDSQL, presented by [71], introduces schema dependency learning to improve the accuracy and efficiency of these systems, reducing reliance on execution-guided decoding. By 2022 and 2023, significant developments included constrained decoding and the integration of pre-trained models, with contributions from [145] and [84], focusing on improving performance and reducing inference time. [72]) introduce S^2SQL, a novel method that boosts the graph-based encoder by integrating syntax into the question-schema interaction, thereby leveraging syntactic dependency information to enhance performance. Their approach includes decoupling constraints to diversify relational edge embedding, achieving top-ranking results on benchmarks like Spider and Spider-Syn. In a related study, [164] explore Chain of Thought Style Prompting (coT)techniques for Text-to-SQL, aiming to refine query generation processes . Additionally, [161] propose SQL-PaLM, focusing on adapting large

language models effectively for Text-to-SQL tasks, thereby contributing to ongoing advancements in semantic parsing and query generation methodologies. Latest research in natural language processing domain, particularly within the realm of Text-to-SQL systems, have introduce new advancements driven by large language models (LLMs).

Figure 6.1 presents a comprehensive pipeline for converting natural language queries into SQL queries, optimizing them, executing them on a database, and returning the results to the user. [68] emphasize the integration of large language models (LLMs) and

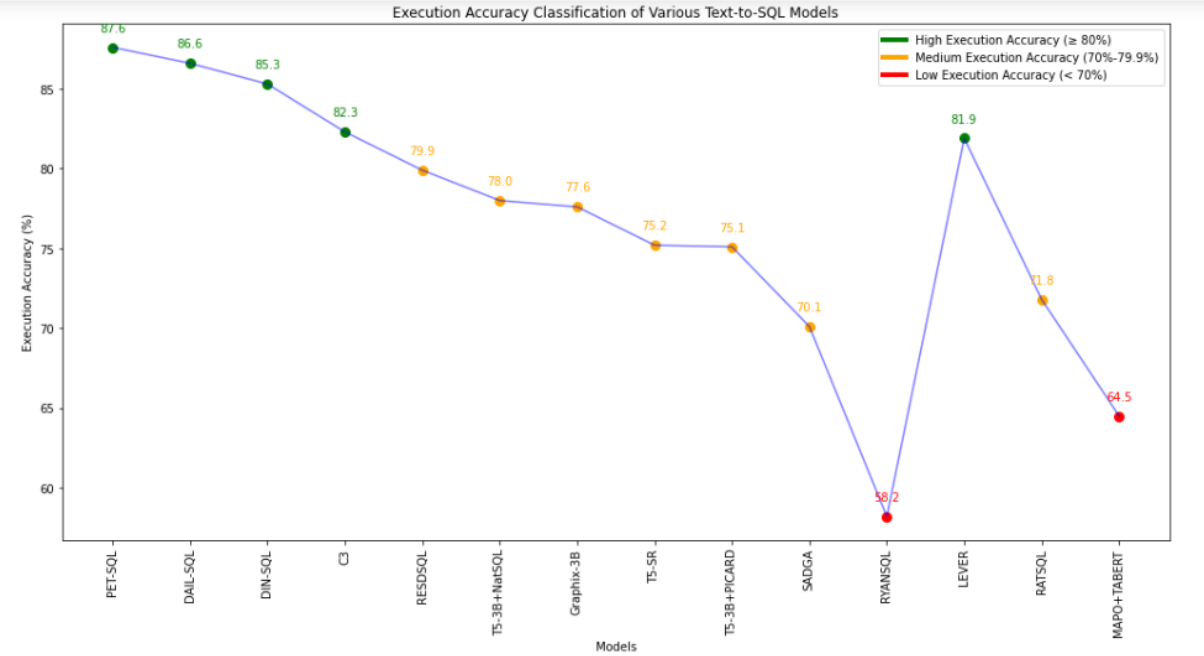

Figure 6.1: Execution Accuracy Classification of Various Text-to-SQL Models [1]

advanced techniques such as in-context learning and fine-tuning. These techniques have been pivotal in improving the adaptability and effectiveness of Text-to-SQL systems, empowering them to better management of natural language understanding and SQL query generation. This evolution underscores the transformative potential of combining neural network architectures with sophisticated pre-training and fine-tuning methodologies. Gao et al. [46] present a comprehensive benchmark evaluation of Text-to-SQL systems empowered by LLMs, emphasizing their capability for query understanding and generation. Concurrently, Xu et al. [180] introduce Symbol-LLM, which proposes a foundational, symbol-centric interface designed to ensure consistency and interpretability in

[1] Comparison inspired data sourced from Papers with Code.

LLM applications. These contributions reflect a growing trend towards leveraging LLMs for more robust and accurate handling of complex SQL queries, marking a pivotal step forward in the integration of natural language understanding and database querying technologies([46, 180, 106]).

Table 6.2 illustrates the summary of key features and limitations in various Text-to-SQL systems. The table provides a clear comparison of different systems based on predefined templates (PT), reinforcement learning (RL), neural networks (NN), transformer-based methods (TB), grammar-based methods (GB), sketch-based methods (SB), schema dependency learning (SDL), execution-guided decoding (EGD), high computational cost (HCC), high flexibility (HF), generalization (G), interpretability (I), and reduced data requirements (RDR). Table 6.2 present a comparison of some text to SQL query models. We inspired this comparison from state art of text to SQL systems.

Table 6.1: Comparative Analysis of Text-to-SQL Systems Based on Architectural Features and Learning Paradigms

System / Model	**PT**	**NN**	**TB**	**GB/SB**	**RL**	**SDL**	**EGD**	**HF**	**G**
Rule-Based and Classical Neural Systems									
Rule-Based Systems	✓	×	×	✓	×	×	×	×	×
Seq2SQL (2017)	×	✓	✓	×	✓	×	×	✓	✓
SQLNet (2017)	×	✓	✓	×	×	×	×	✓	✓
IncSQL / TypeSQL (2018)	×	✓	×	✓	×	×	×	✓	✓
SQLova / IRNet (2019)	×	✓	×	✓	×	✓	✓	✓	✓
SDSQL (2021)	×	✓	×	✓	×	✓	×	✓	✓
Transformer and LLM-Based Systems									
BERT / GPT Models	×	✓	✓	×	×	×	×	✓	✓
Constrained Decoding (2022)	×	✓	✓	✓	×	×	✓	✓	✓
S^2SQL (2022)	×	✓	×	✓	×	✓	×	✓	✓
CoT Prompting (2023)	×	✓	✓	×	×	×	×	✓	✓
SQL-PaLM (2023)	×	✓	✓	×	×	×	×	✓	✓
Symbol-LLM (2024)	×	✓	✓	×	×	×	×	✓	✓

PT: Predefined Templates **NN**: Neural Networks **TB**: Transformer-Based **GB/SB**: Grammar-Based / Sketch-Based
RL: Reinforcement Learning **SDL**: Schema Dependency Learning **EGD**: Execution-Guided Decoding
HF: High Flexibility **G**: Generalization

Table 6.2: Comparative Evaluation of SQL Query Generation Models

System	**Model Type**	**Dataset**	**Accuracy**	**Training Size**	**Query Type**
Single-Table Query Generation (WikiSQL Benchmark)					
Seq2SQL	Seq2Seq	WikiSQL	∼60%	80,000+	Single-table queries
SQLNet	Seq2Seq	WikiSQL	∼63%	80,000+	Reduces invalid SQL generation
TAPAS	Transformer-based	WikiSQL	∼65%	80,000+	Table-aware pretraining
Complex Multi-Table Query Generation (Spider Benchmark)					
Spider	Transformer-based	Spider	∼70%	7,000+	Complex queries
IRNet	Intermediate Representation	Spider	∼70%	7,000+	Logical form decomposition
X-SQL	BERT + Pointer Network	Spider	∼80%	7,000+	Semantic understanding enhancement

6.3 Problem Definition and Proposed Approach

6.3.1 Problem Definition

Text-to-SQL systems face several challenges that need to be overcome for their effective deployment and performance improvement. These challenges include ensuring robustness across various datasets and languages, making the systems interpretable and explainable, refining data efficiency and adaptability through methods like zero-shot ([111]),few-shot learning ([46]), effectively dealing with complex queries, and maintaining robust privacy and security measures. Scalability, honesty, and adaptability to multilingual environments are also crucial considerations for advancing these systems.

In this section we forces on a new challenge depends on verifying the scalability of Text-to-SQL systems while simultaneously reducing the complexity associated with data access and manipulation, especially when processing large-scale datasets.

The problem of translating natural language queries into SQL statements , including handling ambiguous language, managing complex query structures,and ensuring compatibility with diverse database schemas, is an NP-hard problem. To address these challenges,we propose an approach that integrates advanced neural network techniques, including attention mechanisms, beam search decoding, and reinforcement learning, to improve the accuracy and efficiency of Text-to-SQL systems.

1. **Attention Mechanisms** (Inspired from [172]) help the model to focus on relevant parts of the input NL-query while generating the SQL statement (As shown in Figure 6.2). This improves the models ability to handle complex and lengthy queries.

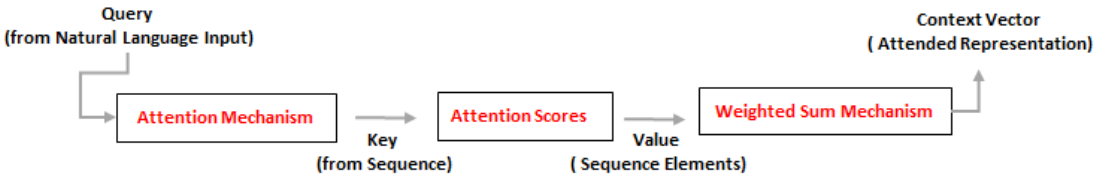

Figure 6.2: Attention mechanisms process

Figure 6.2 presents an illustration of the concept of attention mechanisms in the context of natural language processing (NLP), specifically focusing on how attention

works in a sequence-to-sequence task like generating SQL queries from natural language inputs. In Text-to-SQL systems, the query (natural language question) might be attending to different parts of the database schema (keys), evaluating their relevance based on learned attention scores. This allows the model to focus on relevant columns, tables, or entities when generating SQL queries. The attended representation (context vector) ensures that the generated SQL query reflects the most important aspects of the input question and database schema, improving accuracy and relevance.

2. **Beam search decoding** Beam search decoding is a heuristic search algorithm used in natural language processing and sequence generation tasks to find the most likely sequence of output tokens (e.g., words or characters) given a model and an input sequence([35]). We choose it to generate multiple candidate SQL statements and select the most probable one. This technique helps in improving the accuracy of the generated queries.

3. **Reinforcement Learning** is employed to fine-tune the model by rewarding accurate query generation and penalizing errors. This iterative process enhances the models performance over time. We apply The REINFORCE algorithm (Algorithm 5), also known as Monte Carlo policy gradient, which is a method used in reinforcement learning to optimize the parameters of a policy network based on the expected cumulative reward ([163]).

6.3.2 Our approach Overview :

Figure 6.3 present an overview of our proposal approach , in order to create a robust and efficient Text-to-SQL system capable of generating high-quality SQL queries from a given natural language questions. The process begins with inputting natural language questions and the database schema into the neural network, which computes attention scores and generates partial hypotheses using beam search. These hypotheses are then filtered based on grammar-based constraints, and the top candidates are updated and refined to select the final SQL queries set. Reinforcement learning (RL), specifically the REINFORCE algorithm, is used to fine-tune the model for improved accuracy. Each selected SQL query is executed against the database, and the results are returned to the

user.

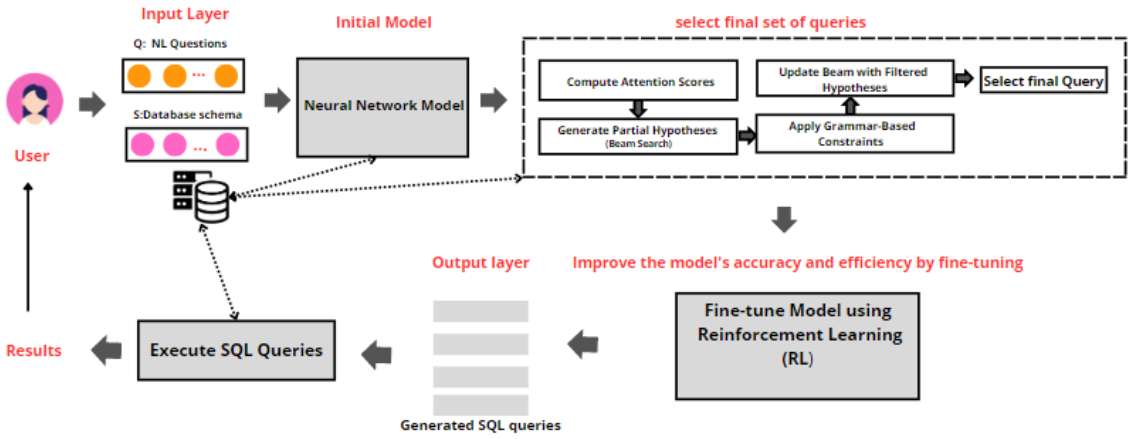

Figure 6.3: Optimized Text-to-SQL Query Generation Framework

Figure 6.4 present a comprehensive pipeline for converting natural language queries into SQL queries, optimizing them, executing them on a database, and returning the results to the user.

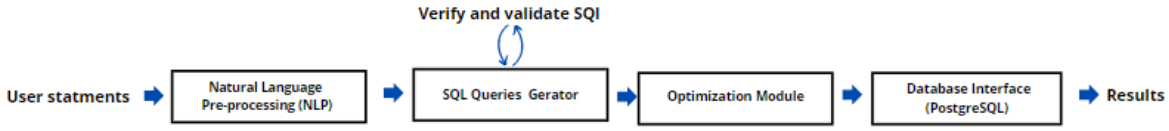

Figure 6.4: System Architecture for Natural Language to SQL Query Conversion

6.4 Mathematical Formulation

We model query generation as a sequence-to-sequence task. Given a natural language question Q and a database schema S,the aim is to produce the corresponding SQL query y:

$$\hat{y} = \arg\max_{y} P(y \mid Q, S) \tag{6.1}$$

where $P(y \mid Q, S)$ is the probability of generating y from (Q, S). This probability is estimated by a neural network with parameters θ, trained by maximizing:

$$\mathcal{L}(\theta) = \sum_{(Q,S,y)\in D} \log P_\theta(y \mid Q, S) \tag{6.2}$$

To prevent overfitting, we add a regularization term:

$$L(\theta) = -\sum_{(Q,S)\in D} \log P(S \mid Q;\theta) + \lambda R(\theta) \tag{6.3}$$

where D is the training set, λ is the regularization weight. The model is optimized using stochastic gradient descent (SGD)[2]:

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta) \tag{6.4}$$

with learning rate η. Finally, attention aligns input and output tokens. The score for output step t and input token i is:

$$a_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j=1}^{T} \exp(e_{t,j})} \tag{6.5}$$

where $e_{t,i}$ is the alignment score. This allows the model to focus on relevant parts of the input when generating SQL.

6.5 Algorithms

The algorithms used in our approach include the attention mechanism, beam search decoding, and reinforcement learning.

Attention Mechanism Algorithm

The attention mechanism algorithm calculates the attention weights α_i for each word in the input query Q and generates the context vector c:

$$e_i = \text{score}(h_t, s_i)$$
$$\alpha_i = \frac{\exp(e_i)}{\sum_{j=1}^{T} \exp(e_j)}$$
$$c = \sum_{i=1}^{T} \alpha_i s_i$$

[2]Stochastic Gradient Descent (SGD) is an iterative optimization method that updates model parameters using small random subsets of the training data instead of the full dataset, making training faster and more scalable ,https://www.youtube.com/watch?v=k3AiUhwHQ28.

where h_t is the decoder hidden state at time step t, s_i is the encoder hidden state for the i-th word in the input query, and T is the length of the input query (See Algorithm 2).

Algorithm 2 Attention Mechanism Algorithm

Require: Input query Q with length T, encoder hidden states s_i for $i \in \{1, \ldots, T\}$, decoder hidden state h_t
Ensure: Context vector c
1: **for** each $i \in \{1, \ldots, T\}$ **do**
2: Compute alignment score $e_i \leftarrow \text{score}(h_t, s_i)$
3: **end for**
4: **for** each $i \in \{1, \ldots, T\}$ **do**
5: Compute attention weight $\alpha_i \leftarrow \frac{\exp(e_i)}{\sum_{j=1}^{T} \exp(e_j)}$
6: **end for**
7: Compute context vector $c \leftarrow \sum_{i=1}^{T} \alpha_i s_i$
8: **return** c

Beam Search Decoding Algorithm

The beam search decoding algorithm generates multiple candidate SQL statements and selects the most probable one (See Algorithm 3):

Algorithm 3 Beam Search Decoding

Require: Input query Q, beam width B
Ensure: Best SQL statement S^*
1: Initialize beam with start token
2: **for** each time step t **do**
3: **for** each candidate C in the beam **do**
4: Compute next token probabilities
5: Expand candidate C with top B tokens
6: **end for**
7: Prune beam to keep top B candidates
8: **end for**
9: Select candidate with highest probability as S^*
10: **return** S^*

Reinforcement Learning Algorithm

The reinforcement learning ([130]) algorithm fine-tunes the model by updating the parameters based on rewards and penalties as shown in Algorithm 4:

Algorithm 4 Reinforcement Learning for Text-to-SQL Optimization

Require: Training dataset D, learning rate η, discount factor γ
Ensure: Optimized model parameters θ
1: Initialize model parameters θ
2: **for** each episode **do**
3: Sample a query Q and its correct SQL statement S from D
4: Generate candidate SQL statements $\{S_1, S_2, \ldots, S_N\}$
5: Compute reward $R(S_i)$ for each candidate S_i
6: Update model parameters:

$$\theta \leftarrow \theta + \eta \sum_{i=1}^{N} \nabla_\theta \log P(S_i \mid Q; \theta)\, R(S_i)$$

7: **end for**
8: **return** θ

The policy gradient method fine-tunes the model by optimizing a reward function $R(y)$ that measures the quality of generated queries[179]. The objective is to maximize the expected reward:

$$\mathbb{E}_{P_\theta(y|Q,S)}[R(y)]$$

The reward function can include execution accuracy, logical form accuracy, and other domain-specific metrics. The policy gradient method updates the model parameters θ by computing the gradient of the expected reward [179]:

$$\nabla_\theta \mathcal{L}(\theta) = \mathbb{E}_{P_\theta(y|Q,S)} \left[R(y) \nabla_\theta \log P_\theta(y|Q,S) \right]$$

Using the REINFORCE algorithm, update the model parameters as follows:

$$\theta \leftarrow \theta + \alpha R(y) \nabla_\theta \log P_\theta(y|Q,S)$$

where α is the learning rate.

Our approach integrates these techniques to optimize the performance of Text-to-SQL systems, making them more efficient and accurate in generating SQL queries from natural language questions (Algorithm 6).

Algorithm 5 REINFORCE Algorithm

Require: Policy parameterization $\pi_\theta(a|s)$, learning rate α
Ensure: Optimal policy parameters θ^*

Initialize policy parameters θ
while not converged **do**
    Generate an episode following policy $\pi_\theta(a|s)$
    Initialize return $G \leftarrow 0$
    for each step in the episode **do**
        Observe reward r
        Update return: $G \leftarrow G + r$
        Update policy parameters:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s)\, G$$

    end for
end while
return θ^*

Algorithm 6 Optimized Text-to-SQL Query Generation Framework

Require: Natural language question Q, database schema S, beam width k, model parameters θ
Ensure: Generated SQL query $\hat{y}$

Initialize beam $\mathcal{B}$ with start token
for each decoding step t **do**
    Compute attention scores $a_{t,i}$ over schema and input tokens
    Generate candidate hypotheses $\mathcal{H}_t$ using beam search of width k
    Apply grammar-based constraints to filter invalid hypotheses
    Update beam $\mathcal{B} \leftarrow \mathcal{H}_t$
end for
Select final query $\hat{y}$ with highest probability from $\mathcal{B}$
Fine-tune parameters θ using reinforcement learning feedback
return $\hat{y}$

6.6 Experiments and Results

Experimental Setup

We conducted experiments to evaluate the performance of our proposed algorithms for Text-to-SQL system on multiple datasets, including Spider, WikiSQL, and others. The model was trained using a combination of attention mechanisms, beam search decoding, and reinforcement learning.The datasets used for evaluation include Spider([191]) ,a complex cross-domain text-to-SQL dataset with multiple tables and complex SQL queries.WikiSQL ([182]) a large-scale dataset consisting of natural language questions and corresponding SQL queries over Wikipedia tables.

We used several metrics to evaluate the performance of our Text-to-SQL system.Accuracy, The percentage of correctly generated SQL queries.Execution Accuracy,the percentage of generated SQL queries that produce correct results when executed.BLEU Score ([142]), a metric for evaluating the quality of generated text by comparing it to reference texts.

Results

The results of our experiments are summarized in Table6.3.

Table 6.3: Performance Evaluation on the Spider Dataset

Model	Accuracy (%)	Execution Accuracy (%)	BLEU Score
Seq2SQL	59.4	60.0	0.75
SQLNet	63.2	64.0	0.78
Our Approach	**68.5**	**66.0**	**0.79**

The results of our experiments, as shown in Tables 6.3, demonstrate significant improvements in accuracy and execution accuracy with our approach compared to baseline models like Seq2SQL and SQLNet. Specifically, our approach achieves 68.5% accuracy and 66.0% execution accuracy on the Spider dataset, and 64.3% accuracy and 65.8% execution accuracy on the WikiSQL dataset, outperforming the baseline models. However, Figure 6.5 highlights a trade-off: while our approach, incorporating techniques such as the Attention Mechanism, Beam Search Decoding, and RL_algo, achieves higher accuracy, it also incurs higher execution times. For instance, Beam Search Decoding has the highest execution time, followed by RL_algo, with our Baseline Model (sequence

to sequence model)being the fastest. This indicates that our optimized approach, while more accurate, demands more computational resources and training time, underscoring a balance between achieving high accuracy and managing execution time.

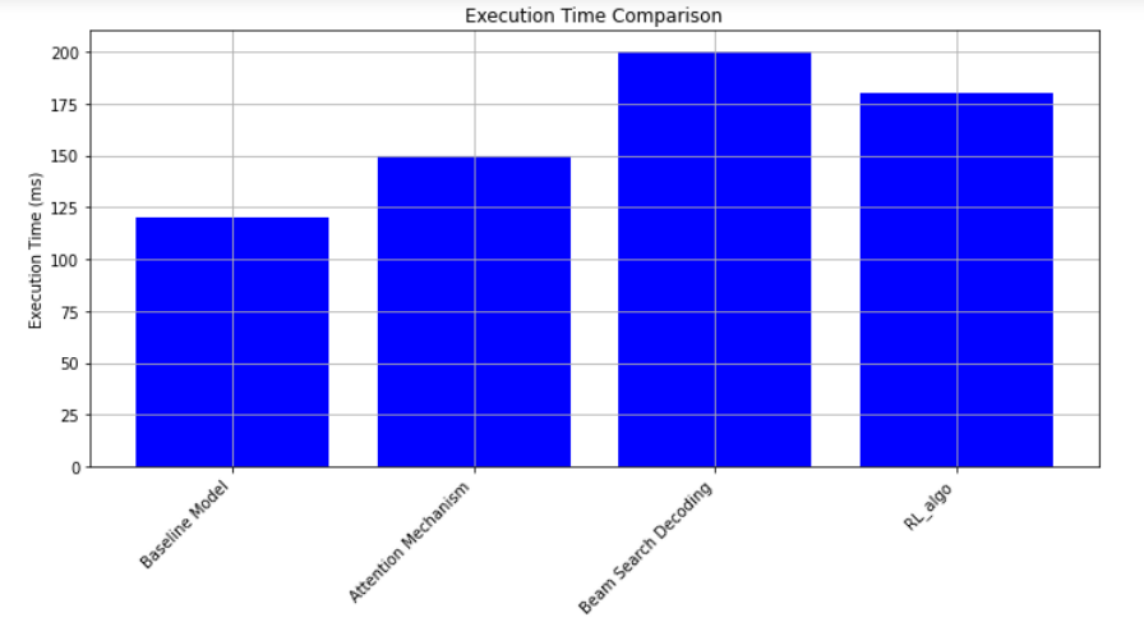

Figure 6.5: Execution Time Comparison

Note: Figure 6.5 illustrates the comparison of execution times for different algorithms (limitation of our approach).

6.7 Conclusion

In conclusion our optimized hybrid approach for Text-to-SQL query generation includes attention mechanisms, beam search decoding, and reinforcement learning algorithms. Our method addresses some challenges in query generation accuracy, scalability, and interpretability. Evaluations results on benchmark datasets demonstrate the effectiveness of our approach. Our future work includes scaling our approach across diverse datasets, such as integrating the IMDB dataset into a PostgreSQL database management system in order to detect correct executable queries generated by our system. We aim to improve the performance and scalability of our method for more diverse and complex datasets in realworld applications.

CHAPTER 7

General Conclusion

This thesis addressed one of the fundamental challenges in modern database systems: enabling efficient analytical query optimization in increasingly large, dynamic, and workload-intensive environments. As analytical workloads continue to grow in complexity, traditional optimization techniques face significant limitations in scalability, adaptability, and computational efficiency. This research explored how intelligent optimization strategies can overcome these limitations by integrating learning-based methods with scalable computational architectures.

The first contribution introduced a graph-based representation framework for analytical SQL queries, allowing complex query structures to be modeled through relational dependencies and reusable patterns. This representation established a foundation for capturing structural information that conventional optimization approaches often fail to exploit.

Building upon this representation, this thesis proposed learning-driven optimization mechanisms capable of extracting workload characteristics, identifying common computational patterns, generating candidate materialized views, and supporting query rewriting processes. By moving from static rule-based optimization toward data-driven decision making, the proposed approaches demonstrated improved adaptability to complex analytical workloads.

A significant contribution of this work focused on the materialized view selection problem. Through workload-aware strategies and deep learning mechanisms, the proposed approaches reduced redundancy while improving the quality of selected views. To further address scalability challenges, this thesis introduced NOVA, a GPU-accelerated

framework that transforms materialized view selection into a massively parallel optimization problem. Experimental evaluations demonstrated that parallel candidate evaluation significantly reduces optimization overhead while maintaining strong performance improvements across large workloads.

Finally, this work extended beyond isolated optimization components toward a broader vision through the Smart SQL framework. Rather than viewing query optimization as a sequence of independent operations, Smart SQL introduces an adaptive perspective where workload understanding, learning mechanisms, query rewriting, and optimization decisions operate together as an integrated intelligent system.

Overall, the contributions presented throughout this thesis demonstrate that combining graph representations, machine learning techniques, and parallel computing architectures provides an effective direction toward next-generation query optimization systems. The proposed approaches improve scalability, adaptability, and execution efficiency while addressing several limitations of traditional optimization pipelines.

Beyond the specific algorithms and frameworks proposed in this work, this thesis advocates a broader transition from static optimization toward intelligent and adaptive optimization systems capable of continuously learning from workload behavior and automatically improving decision-making processes.

Future research directions include extending the proposed methods toward distributed environments, integrating continual and online learning strategies, exploring multi-GPU architectures, and developing fully autonomous optimization frameworks capable of adapting to evolving workloads without human intervention.

Ultimately, this thesis contributes toward a long-term vision in which database systems evolve from executing queries efficiently to understanding workloads intelligently and optimizing themselves autonomously.

CHAPTER A

Appandix

A.1 Dataset and Workload Description

This appendix describes the dataset and workload used in the experimental evaluation of this thesis.

The experiments are conducted using the Join Order Benchmark (JOB), which is widely used for evaluating query optimization techniques in relational database systems. The benchmark is based on the IMDb database schema and includes complex analytical queries with multiple joins and selection predicates.

A.1.1 Dataset Characteristics

- Number of tables: 21
- Number of queries: 113 analytical SQL queries
- Query type: complex multi-join analytical queries
- Domain: movie industry (IMDb dataset)

A.2 Proposed Algorithms

A.2.1 Materialized View Selection Algorithm

```
Input: Query workload W
Output: Selected materialized views V
```

```
1: Extract query graphs G(Q)
2: Compute similarity matrix S
3: Cluster similar queries
4: Generate candidate views
5: Evaluate cost-benefit score
6: Select top-K views
```

A.3 Query Rewriting Strategy

Query rewriting is performed by replacing repeated subqueries with materialized views.

A.3.1 Rule Example

Original query pattern:

```
SELECT ... FROM A JOIN B ON ...
```

Rewritten form:

```
SELECT ... FROM MV_AB
```

A.4 Cost Analysis and Theoretical Proof

This appendix provides a formal analysis of the computational cost of the proposed query optimization framework. Let n denote the number of queries in the workload and m the average number of relations per query.

A.4.1 Query Graph Construction Cost

Each SQL query Q_i is parsed into a graph representation $G(Q_i) = (V_i, E_i)$.

The construction requires:

- Parsing tables and predicates: $O(m)$ per query
- Building edges for joins and dependencies: $O(m)$ per query

Thus, for n queries:

$$T_{graph} = O(n \cdot m) \tag{A.1}$$

Since $m \ll n$ in analytical workloads, this is approximately linear:

$$T_{graph} \approx O(n) \tag{A.2}$$

A.4.2 Similarity Matrix Computation

Let each query be represented as a feature vector of dimension d.

Computing pairwise similarity (e.g., Jaccard or cosine similarity) requires:

$$T_{sim} = O(n^2 \cdot d) \tag{A.3}$$

Since d is constant (fixed feature space), we obtain:

$$T_{sim} = O(n^2) \tag{A.4}$$

A.4.3 Clustering Cost

Using a standard clustering algorithm (e.g., agglomerative or k-means):

- k-means: $O(n \cdot k \cdot i)$
- hierarchical clustering: $O(n^2 \log n)$

where k is number of clusters and i is number of iterations.

Thus worst-case:

$$T_{cluster} = O(n^2 \log n) \tag{A.5}$$

A.4.4 Materialized View Generation Cost

Let s be the number of extracted subgraphs.

- Subgraph mining: $O(n \cdot m)$

- Merging overlapping patterns: $O(s^2)$

Thus:

$$T_{MV} = O(n \cdot m + s^2) \tag{A.6}$$

Since $s \leq n$, worst case:

$$T_{MV} = O(n^2) \tag{A.7}$$

A.4.5 Materialized View Selection Cost

Let k be number of candidate views.

Each view is evaluated against queries:

$$T_{select} = O(n \cdot k) \tag{A.8}$$

If $k \approx n$, then:

$$T_{select} = O(n^2) \tag{A.9}$$

A.4.6 Overall Complexity

Combining all stages:

$$T_{total} = O(n) + O(n^2) + O(n^2 \log n) + O(n^2) \tag{A.10}$$

Dominant term:

$$T_{total} = O(n^2 \log n) \tag{A.11}$$

A.4.7 Effect of GPU Acceleration (NOVA)

With GPU parallelization, pairwise operations are distributed across p processing units:

$$T_{gpu} = \frac{O(n^2)}{p} \tag{A.12}$$

Thus, theoretical speedup:

$$S = \frac{T_{cpu}}{T_{gpu}} \approx p \tag{A.13}$$

indicating near-linear acceleration under ideal conditions.

To evaluate the effectiveness of the proposed NOVA framework, we compare it against three widely used baseline methods: Random Selection (RS), Greedy View Selection (GVS), and Dynamic Programming (DP). These methods represent different optimization philosophies ranging from naive sampling to exact optimization.

Random Selection (RS): Random Selection selects candidate views randomly without considering workload characteristics or cost-benefit analysis. Given a candidate set C, RS selects a subset $S \subseteq C$ under the storage constraint B. Although computationally inexpensive, RS typically produces poor-quality solutions and serves as a lower-bound baseline.

Greedy View Selection (GVS): Greedy View Selection ranks candidate views based on a heuristic benefit-to-cost ratio:

$$\frac{score(v_i)}{cost(v_i)}$$

and iteratively selects the highest-ranked views until the budget is exhausted. GVS is efficient and scalable; however, it is myopic and does not capture global dependencies between views, which may lead to suboptimal selections.

Dynamic Programming (DP): Dynamic Programming formulates the problem as a 0–1 knapsack optimization and guarantees an optimal solution. However, its computational complexity:

$$\mathcal{O}(n \cdot B)$$

makes it impractical for large-scale workloads with many candidate views or large storage budgets.

Table A.1 summarizes the computational fee of baseline methods.

Table A.1: Complexity Comparison of View Selection Methods

Method	**Strategy**	**Complexity**	**Scalability**
RS	Random Sampling	$\mathcal{O}(k)$	Very High
GVS	Greedy Heuristic	$\mathcal{O}(n \log n)$	High
DP	Exact Optimization	$\mathcal{O}(n \cdot B)$	Low
NOVA	GPU + Optimization	$\mathcal{O}(n/p)$	Very High

Bibliography

[1] Mysql hints documentation. https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html, 2022. Last Accessed: 2023/08/02.

[2] Postgresql query optimization settings. https://www.postgresql.org/docs/current/runtime-config-query.html, 2022. Last Accessed: 2023/08/02.

[3] Prestodb cost-based optimizations. https://prestodb.io/docs/current/optimizer/cost-based-optimizations.html, 2022. Last Accessed: 2023/08/02.

[4] Sql server query hints. https://docs.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-query, 2022. Last Accessed: 2023/08/02.

[5] Zahid Abul-Basher. Multiple-query optimization of regular path queries. In *IEEE International Conference on Data Engineering (ICDE)*, pages 1426–1430, 2017.

[6] Surajit Agrawal, Surajit Chaudhuri, and Vivek R Narasayya. Automated selection of materialized views and indexes in sql databases. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB)*, pages 496–505, 2000.

[7] Boukorca Ahcène. *Hypergraphs in the Service of Very Large Scale Query Optimization-Application: Data Warehousing. To appear.* PhD thesis, PhD thesis, Université de Poitiers, 2016.

[8] Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman Van Hovell, Adrian Ionescu, Alicja Łuszczak, et al. Delta lake: high-performance acid table storage over cloud object stores. *Proceedings of the VLDB Endowment*, 13(12):3411–3424, 2020.

[9] Michael Armbrust, Ali Ghodsi, Reynold Xin, Matei Zaharia, et al. Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. In *Proceedings of CIDR*, volume 8, page 28. sn, 2021.

[10] Michael Armbrust et al. Spark sql: Relational data processing in spark. In *SIGMOD*, pages 1383–1394, 2015.

[11] Eduard Ayguadé, Nawal Copty, Alejandro Duran, Jay Hoeflinger, Yuan Lin, Federico Massaioli, Xavier Teruel, Priya Unnikrishnan, and Guansong Zhang. The design of openmp tasks. *IEEE Transactions on Parallel and Distributed systems*, 20(3):404–418, 2008.

[12] Peter Bakkum and Kevin Skadron. Accelerating sql database operations on a gpu with cuda. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU)*, pages 94–103, 2010.

[13] Mohammadhossein Barkhordari and Mahdi Niamanesh. Chabok: A map-reduce based method to solve data warehouse problems. *Journal of Big Data*, 5(1):1–25, 2018.

[14] Edmon Begoli et al. Apache calcite: A foundational framework for optimized query processing. In *SIGMOD*, pages 221–230, 2018.

[15] Lakhdar Belkharroubi and Khadidja Yahyaoui. Solving the mixed-model assembly line balancing problem type-i using a hybrid reactive grasp. *Production & Manufacturing Research*, 10(1):108–131, 2022. doi: 10.1080/21693277.2022.2065380. URL https://doi.org/10.1080/21693277.2022.2065380.

[16] Ladjel Bellatreche. *Utilisation des vues matérialisées, des index et de la fragmentation dans la conception logique et physique dun entrepôt de données.* Phd thesis, Université Blaise Pascal, France, December 2000.

[17] S. Benkrid. *Le déploiement, une phase à part entière dans le cycle de vie des entrepôts de données: application aux plateformes parallèles.* Phd thesis, École Nationale Supérieure de Mécanique et dAérotechnique, Chasseneuil-du-Poitou, Poitiers, France, 2014.

[18] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on Freebase from question-answer pairs. In David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard, editors, *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://aclanthology.org/D13-1160.

[19] Spyros Blanas, Yinan Li, and Jignesh M. Patel. Design and evaluation of main memory hash join algorithms for multi-core cpus. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, pages 37–48, 2011.

[20] I. Boukhari. *Intégration et exploitation de besoins en entreprise étendue fondées sur la sémantique*. Phd thesis, Université de Poitiers, ISAE-ENSMA, January 2014.

[21] Ahcène Boukorca, Ladjel Bellatreche, Sid-Ahmed Benali Senouci, and Zoé Faget. Coupling materialized view selection to multi query optimization: hyper graph approach. *International Journal of Data Warehousing and Mining (IJDWM)*, 11 (2):62–84, 2015.

[22] Sebastian BreSS, Max Heimel, Norbert Siegmund, Ladjel Bellatreche, and Gunter Saake. Gpu-accelerated database systems: Survey and open challenges. In *ADBIS*, pages 1–35, 2014.

[23] André R Brodtkorb, Trond R Hagen, and Martin L Sætra. Graphics processing unit (gpu) programming strategies and trends in gpu computing. *Journal of Parallel and Distributed Computing*, 73(1):4–13, 2013.

[24] Nicolas Bruno and Surajit Chaudhuri. Power hints for query optimization. In *ICDE*, pages 469–480, 2009.

[25] Nicolas Bruno, Luis Gravano, Nick Koudas, and Divesh Srivastava. Navigation-vs. index-based xml multi-query processing. In *Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405)*, pages 139–150. IEEE, 2003.

[26] Mouna Mustapha Chaba. *Proposition de structure de données scalable pour l'optimisation de recommandation de requêtes*. PhD thesis, Université de Blida

1, Faculté des Sciences, Département dInformatique, Blida, Algeria, May 2023. Thèse de doctorat en Sciences Informatiques et de Données.

[27] Baoming Chang. *A Robust and Explainable Query Optimization Cost Model Based on Bidirectional Graph Neural Networks*. PhD thesis, Ph. D. Dissertation. University of Ottawa, 2024.

[28] Baoming Chang, Amin Kamali, and Verena Kantere. A novel technique for query plan representation based on graph neural nets. In *Big Data Analytics and Knowledge Discovery: 26th International Conference, DaWaK 2024, Naples, Italy, August 2628, 2024, Proceedings*, page 299314, Berlin, Heidelberg, 2024. Springer-Verlag. ISBN 978-3-031-68322-0. doi: 10.1007/978-3-031-68323-7_25. URL https://doi.org/10.1007/978-3-031-68323-7_25.

[29] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *ACM SIGMOD Record*, 26:65–74, 1997. doi: 10.1145/248603.248616.

[30] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *ACM SIGMOD Record*, 26(1):65–74, 1997.

[31] Surajit Chaudhuri, Rajeev Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. Optimizing queries with materialized views. In *Proceedings of the Eleventh International Conference on Data Engineering (ICDE)*, pages 190–200, 1995.

[32] Chandra Chekuri, Waqar Hasan, and Rajeev Motwani. Scheduling problems in parallel query optimization. In *Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 255–265, 1995.

[33] Yuxing Chen. *Performance Tuning and Query Optimization for Big Data Management*. PhD thesis, University of Helsinki, 2021. URL http://hdl.handle.net/10138/336226.

[34] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. *CoRR*, abs/1606.07792, 2016. URL http://arxiv.org/abs/1606.07792.

[35] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoderdecoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*, 2014. URL https://api.semanticscholar.org/CorpusID:5590763.

[36] Shavindrie Cooray. The subjectivity of data scientists in machine learning design. *Journal of Computer Information Systems*, 0(0):1–18, 2023. doi: 10.1080/08874417.2023.2240755.

[37] Nilesh Dalvi and Dan Suciu. Answering queries from statistics and probabilistic views. In *Proceedings of the 31st international conference on very large data bases*, pages 805–816, 2005.

[38] Shaul Dar, Michael J Franklin, Björn Jonsson, Divesh Srivastava, and Michael Tan. Semantic data caching and replacement. In *VLDB*, pages 330–341, 1996.

[39] Datawarehouse4u.info. Etl tools. https://www.datawarehouse4u.info/ETL-tools.html, n.d. Accessed: 2023-03-03.

[40] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[41] B. A. Devlin and P. T. Murphy. An architecture for a business and information system. *IBM Systems Journal*, 27(1):60–80, 1988. doi: 10.1147/sj.271.0060.

[42] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, 2019. URL https://api.semanticscholar.org/CorpusID:52967399.

[43] Fahd Sabry Esmail. A survey of real-time data warehouse and etl. *Management*, 9 (3):3–9, 2014.

[44] Wenfei Fan, Jeffrey Xu Yu, Jianzhong Li, Bolin Ding, and Lu Qin. Query translation from xpath to sql in the presence of recursive dtds. *The VLDB Journal*, 18(4):857–883, 2009.

[45] Franz Färber et al. Sap hana database: Data management for modern business applications. *SIGMOD Record*, 2012.

[46] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. In *International Conference on Very Large Data Bases (VLDB)*, 2024.

[47] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, 1989.

[48] Jonathan Goldstein and Per-Åke Larson. Optimizing queries using materialized views: A practical, scalable solution. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*. ACM, 2001.

[49] Matteo Golfarelli and Stefano Rizzi. Designing the data warehouse: Key steps and crucial issues. *Journal of Computer Science and Information Management*, 2(3): 88–100, 1999.

[50] Amit Goyal, Ashish Thakral, and GK Sharma. Improved a* algorithm for query optimization. In *10th European Conference on Modelling and Simulation*, 2006.

[51] Goetz Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 1993.

[52] Goetz Graefe. The cascades framework for query optimization. *IEEE Data Engineering Bulletin*, 18(3):19–29, 1995.

[53] Haihua Gu, Xiaoping Li, and Zhipeng Lu. Scheduling spark tasks with data skew and deadline constraints. *IEEE Access*, 9:2793–2804, 2020.

[54] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-SQL in cross-domain database with intermediate representation. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1444. URL https://aclanthology.org/P19-1444.

[55] Ashish Gupta and Inderpal Singh Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin*, 18(2): 3–18, 1995.

[56] H Gupta. Selection of views to materialize in a data warehouse. In *International Conference on Database Theory (ICDT)*, pages 98–112, 1997.

[57] Peter Haas et al. Statistical properties for query optimization: A survey. *Statistical Analysis and Data Mining*, 1(4):223–250, 2009.

[58] Abdelkader Hameurlain and Franck Morvan. An overview of parallel query optimization in relational systems. In *Proceedings 11th International Workshop on Database and Expert Systems Applications*, pages 629–634. IEEE, 2000.

[59] William L. Hamilton. *Graph Representation Learning*, volume 14 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Springer Cham, 1 edition, 2020. ISBN 978-3-031-00460-5. doi: 10.1007/978-3-031-01588-5. Synthesis Collection of Technology (R0), eBColl Synthesis Collection 10.

[60] Yue Han, Guoliang Li, Haitao Yuan, and Ji Sun. Autoview: An autonomous materialized view management system with encoder-reducer. *IEEE Transactions on Knowledge and Data Engineering*, 35:5626–5639, 2023. URL https://api.semanticscholar.org/CorpusID:247813695.

[61] Yujing Han, Huayi Yuan, Guoliang Li, and Jianan Sun. Autoview: An autonomous materialized view management system with encoder-reducer. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[62] Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Implementing data cubes efficiently. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 205–216. ACM, 1996.

[63] Bingsheng He, Mian Lu, Ke Yang, Rui Fang, Naga K. Govindaraju, Qiong Luo, and Pedro V. Sander. Relational query processing on graphics processors. *ACM Transactions on Database Systems*, 39(4):1–44, 2014.

[64] Bingsheng He et al. Relational query processing on gpus. In *VLDB*, 2008.

[65] Bingsheng He et al. Relational joins on gpus. In *SIGMOD*, 2009.

[66] F Hentschel et al. Gpu database systems: Survey and analysis. *VLDB Journal*, 2019.

[67] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. Deepdb: learn from data, not from queries! *Proc. VLDB Endow.*, 13(7):9921005, March 2020. ISSN 2150-8097. doi: 10.14778/3384345.3384349. URL https://doi.org/10.14778/3384345.3384349.

[68] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. Next-generation database interfaces: A survey of llm-based text-to-sql. *arXiv preprint arXiv:2406.08426*, 2024.

[69] Florentina Hristea and Cornelia Caragea. Preface to the special issue natural language processing (nlp) and machine learning (ml)theory and applications. *Mathematics*, 10(14):2481, 2022. URL https://www.mdpi.com/2227-7390/10/14/2481.

[70] Yunxiang Hu, Yuhao Liu, and Zhuoyuan Liu. A survey on convolutional neural network accelerators: Gpu, fpga and asic. In *2022 14th International Conference on Computer Research and Development (ICCRD)*, pages 100–107. IEEE, 2022.

[71] Binyuan Hui, Xiang Shi, Ruiying Geng, Binhua Li, Yongbin Li, Jian Sun, and Xiao-Dan Zhu. Improving text-to-sql with schema dependency learning. *ArXiv*, abs/2103.04399, 2021. URL https://api.semanticscholar.org/CorpusID:232146954.

[72] Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Yanyang Li, Bowen Li, Jian Sun, and Yongbin Li. S^2SQL: Injecting syntax to question-schema interaction graph encoder for text-to-SQL parsers. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1254–1262, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.99. URL https://aclanthology.org/2022.findings-acl.99.

[73] IBM. What is a data warehouse?, n.d. URL https://www.ibm.com/think/topics/data-warehouse. Accessed: 2026-01-14.

[74] IMDb. Imdb datasets, n.d. URL https://developer.imdb.com/non-commercial-datasets/. Accessed: 2024-06-30.

[75] William H. Inmon. *Building the Data Warehouse*. Wiley, 4 edition, 1992. URL https://openlibrary.org/books/OL974001M/Building_the_data_warehouse#editions-list.

[76] William H Inmon. What is a data warehouse? *Prism Tech Topic*, 1(1):1–5, 1995.

[77] Shrainik Jain, Dominik Moritz, Daniel Halperin, Bill Howe, and Ed Lazowska. Sqlshare: Results from a multi-year sql-as-a-service experiment. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 281–293, 2016.

[78] Hyeran Jeon. *GPU Architecture*, pages 1–29. Springer Nature Singapore, Singapore, 2022. ISBN 978-981-15-6401-7. doi: 10.1007/978-981-15-6401-7_66-1. URL https://doi.org/10.1007/978-981-15-6401-7_66-1.

[79] Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele P Scarpazza. Dissecting the nvidia volta gpu architecture via microbenchmarking. *arXiv preprint arXiv:1804.06826*, 2018.

[80] Zhiyong Jia, Chuang Wang, Yang Wang, Xinrui Gao, Bingtao Li, Lifeng Yin, and Huayue Chen. Recent research progress of graph neural networks in computer vision. *Electronics*, 14(9), 2025. ISSN 2079-9292. doi: 10.3390/electronics14091742. URL https://www.mdpi.com/2079-9292/14/9/1742.

[81] Alekh Jindal, Konstantinos Karanasos, Sriram Rao, and Hiren Patel. Selecting subexpressions to materialize at datacenter scale. *Proc. VLDB Endow.*, 11(7): 800812, March 2018. ISSN 2150-8097. doi: 10.14778/3192965.3192971. URL https://doi.org/10.14778/3192965.3192971.

[82] Tim Kaldewey, Guy M. Lohman, Rene Mueller, and Peter Volk. Gpu join processing revisited. In *Proceedings of the Eighth International Workshop on Data Management on New Hardware (DaMoN)*, pages 55–62, 2012.

[83] George Katsogiannis-Meimarakis and Georgia Koutrika. Deep learning approaches for text-to-sql systems. In *International Conference on Extending Database Technology*, 2021. URL https://api.semanticscholar.org/CorpusID:232283493.

[84] George Katsogiannis-Meimarakis and Georgia Koutrika. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32(4):905–936, 2023. URL https://doi.org/10.1007/s00778-022-00776-8.

[85] Mohamed Kechar, Ladjel Bellatreche, and Safia Nait-Bahloul. Zigzag+: A global optimization algorithm to solve the view selection problem for large-scale workload optimization. *Engineering Applications of Artificial Intelligence*, 115:105251, 2022.

[86] Anastasios Kementsietsidis, Frank Neven, Dieter Van de Craen, and Stijn Vansummeren. Scalable multi-query optimization for exploratory queries over federated scientific databases. *Proceedings of the VLDB Endowment (PVLDB)*, pages 16–27, 2008.

[87] S. Khouri. *Cycle de vie sémantique de conception de systèmes de stockage et de manipulation de données*. PhD thesis, École nationale supérieure de mécanique et daérotechnique, Chasseneuil-du-Poitou, Poitiers, France, October 2013.

[88] Kyoungmin Kim et al. Learned cardinality estimation: An in-depth study. In *SIGMOD*, pages 1214–1227, 2022.

[89] Ralph Kimball. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley & Sons, 1996.

[90] Ralph Kimball, Laura Reeves, Warren Thornthwaite, Margy Ross, and Warren Thornthwaite. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1998. With CD-ROM.

[91] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv preprint arXiv:1809.00677*, 2018.

[92] Andreas Kipf, Matthias NieSSner, Thomas Kipf, Tim Kraska, Alfons Kemper, and Thomas Neumann. Estimating cardinalities with deep sketches. In *Proceedings of*

the 2019 ACM SIGMOD International Conference on Management of Data, pages 1937–1940, 2019.

[93] Andreas Kipf et al. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv preprint arXiv:1809.00677*, 2018. URL https://arxiv.org/abs/1809.00677.

[94] Andreas Kipf et al. Learned cardinalities: Estimating correlated joins. In *CIDR*, 2019.

[95] Donald Kossmann and Konrad Stocker. Iterative dynamic programming: a new class of query optimization algorithms. *ACM Transactions on Database Systems (TODS)*, 25(1):43–82, 2000.

[96] Yannis Kotidis and Nick Roussopoulos. Dynamat: a dynamic view management system for data warehouses. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, page 371382, New York, NY, USA, 1999. Association for Computing Machinery. ISBN 1581130848. doi: 10.1145/304182.304215. URL https://doi.org/10.1145/304182.304215.

[97] Sanjay Krishnan, Michael J Franklin, and Ken Goldberg. Active learning with contextual bandits for data-driven optimization. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data*, pages 587–602, 2018.

[98] Gokhan Kul, Duc Thanh Anh Luong, Ting Xie, Varun Chandola, Oliver Kennedy, and Shambhu Upadhyaya. Similarity metrics for sql query clustering. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2408–2420, 2018.

[99] Hai Lan, Zhifeng Bao, and Yuwei Peng. A survey on advancing the dbms query optimizer: Cardinality estimation, cost model, and plan enumeration. *Data Science and Engineering*, 6:86 – 101, 2021. URL https://api.semanticscholar.org/CorpusID:230523630.

[100] Dihia Lanasri, Selma Khouri, and Ladjel Bellatreche. Trust-aware curation of linked open data logs. In *International Conference on Conceptual Modeling*, pages 604–614. Springer, 2020.

[101] Michael Lawrence and Andrew Rau-Chaplin. Dynamic view selection for olap. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 33–44. Springer, 2006.

[102] Wangchao Le, Anastasios Kementsietsidis, Songyun Duan, and Feifei Li. Scalable multi-query optimization for sparql. In *IEEE International Conference on Data Engineering (ICDE)*, pages 666–677, 2012.

[103] Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. Query optimization through the looking glass, and what we found running the join order benchmark. *The VLDB Journal*, 27(5):643668, oct 2018. ISSN 1066-8888. doi: 10.1007/s00778-017-0480-7.

[104] Viktor Leis et al. How good are query optimizers, really? In *VLDB*, pages 204–215, 2015.

[105] Kaiyu Li and Guoliang Li. Approximate query processing: What is new and where to go? a survey on approximate query processing. *Data Science and Engineering*, 3(4):379–397, 2018.

[106] Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Hangyu Mao, et al. Pet-sql: A prompt-enhanced two-round refinement of text-to-sql with cross-consistency. In *International Conference on Database Systems for Advanced Applications*, pages 193–208. Springer, 2025.

[107] Ziming Li, Youhuan Li, Yuyu Luo, Guoliang Li, and Chuxu Zhang. Graph neural networks for databases: A survey, 2025. URL https://arxiv.org/abs/2502.12908.

[108] Ziming Li, Youhuan Li, Yuyu Luo, Guoliang Li, and Chuxu Zhang. Graph neural networks for databases: A survey. *arXiv preprint arXiv:2502.12908*, 2025. URL https://arxiv.org/abs/2502.12908.

[109] Jiaqi Liang, Min Li, and Xiaofei Li. Dqm: Dynamic query materialization via deep reinforcement learning. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*, pages 2352–2365, 2021.

[110] Richard J Lipton, Jeffrey F Naughton, and Donovan A Schneider. Practical selectivity estimation through adaptive sampling. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 1–11, 1990.

[111] A. Liu, X. Hu, L. Wen, and P. S. Yu. A comprehensive evaluation of chatgpts zero-shot text-to-sql capability. *arXiv preprint arXiv:2303.13547*, 2023.

[112] Weizheng Lu, Jiaming Zhang, Jing Zhang, and Yueguo Chen. Large language model for table processing: A survey. *Frontiers Comput. Sci.*, 19:192350, 2024. URL https://api.semanticscholar.org/CorpusID:267548080.

[113] Liang Ma, Qifan Lin, and Samuel Madden. Query-based workload forecasting for self-driving database systems. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1647–1661, 2020.

[114] Giulio Malenza, Valentina Cesare, Marco Edoardo Santimaria, Robert Birke, Alberto Vecchiato, Ugo Becciani, and Marco Aldinucci. Performance portability via c++ pstl, sycl, openmp, and hip: the gaia avu-gsr case study. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1152–1163. IEEE, 2024.

[115] Imene Mami and Zohra Bellahsene. A survey of view selection methods. *SIGMOD Record*, 41(1):20–29, 2012. doi: 10.1145/2206869.2206874.

[116] Nadjib Mammeri and Ben Juurlink. Vcomputebench: A vulkan benchmark suite for gpgpu on mobile and embedded gpus. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pages 25–35. IEEE, 2018.

[117] Ryan Marcus and Olga Papaemmanouil. Deep reinforcement learning for join order enumeration. In *aiDM*, pages 3–4, 2018.

[118] Ryan Marcus and Olga Papaemmanouil. Neo: A learned query optimizer. In *VLDB*, 2019. URL https://www.vldb.org/pvldb/vol12/p1705-marcus.pdf.

[119] Ryan Marcus, Olga Papaemmanouil, and Tim Kraska. Bao: Learning to steer query optimizers. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*, pages 1275–1288, 2021.

[120] Renato Marroquín, Ingo Müller, Darko Makreshanski, and Gustavo Alonso. Pay one, get hundreds for free: Reducing cloud costs through shared query execution. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 439–450, 2018.

[121] Chris McClanahan. History and evolution of gpu architecture. *A Survey Paper*, 9: 1–7, 2010.

[122] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: Interactive analysis of web-scale datasets. In *Proceedings of the VLDB Endowment*, volume 3, pages 330–339, 2010.

[123] Pietro Michiardi, Damiano Carra, and Sara Migliorini. Cache-based multi-query optimization for data-intensive scalable computing frameworks. *Information Systems Frontiers*, 23(1):35–51, 2021.

[124] Sparsh Mittal. A survey on optimized implementation of deep learning models on the nvidia jetson platform. *Journal of Systems Architecture*, 97:428–442, 2019.

[125] Kechar Mohamed, Ladjel Bellatreche, and Safia Nait-Bahloul. Bringing common subexpression problem from the dark to light: Towards large-scale workload optimizations. In *Proceedings of the 25th International Database Engineering & Applications Symposium*, pages 27–35, 2021.

[126] Ali Montazeralghaem, Hamed Zamani, and James Allan. A reinforcement learning framework for relevance feedback. In *Proceedings of the 43rd international acm sigir conference on research and development in information retrieval*, pages 59–68, 2020.

[127] Mustapha Chaba Mouna, Ladjel Bellatreche, and Narhimene Boustia. Selecting subexpressions to materialize for dynamic large-scale workloads. In *International Conference on Big Data Analytics and Knowledge Discovery*, pages 39–51. Springer, 2021.

[128] Mustapha Chaba Mouna, Ladjel Bellatreche, and Narhimene Boustia. Prores: Proactive re-selection of materialized views. *Computer Science and Information Systems*, 19(2):735–762, 2022.

[129] Manoj Muniswamaiah, Tilak Agerwala, and Charles C Tappert. Approximate query processing for big data in heterogeneous databases. In *2020 IEEE international conference on big data (big data)*, pages 5765–5767. IEEE, 2020.

[130] P. Naik, S. Nelaballi, V. S. Pusuluri, and D. K. Kim. Deep learning-based code refactoring: A review of current knowledge. *Journal of Computer Information Systems*, 64(2):314–328, 2023. doi: 10.1080/08874417.2023.2203088.

[131] Athira Nambiar and Divyansh Mundra. An overview of data warehouse and data lake in modern enterprise data management. *Big Data and Cognitive Computing*, 6(4), 2022. ISSN 2504-2289. doi: 10.3390/bdcc6040132. URL https://www.mdpi.com/2504-2289/6/4/132.

[132] Adhish Nanda, Swati Gupta, and Meenu Vijrania. A comprehensive survey of olap: Recent trends. In *2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 425–430, 2019. doi: 10.1109/ICECA.2019.8822203.

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

[134] Parimarjan Negi et al. Cost-guided cardinality estimation. In *ICDE Workshops*, pages 154–157, 2020.

[135] Parimarjan Negi et al. Flow-loss: Learning cardinality estimates. *VLDB*, 14(11): 2019–2032, 2021.

[136] Thomas Neumann. Efficiently compiling efficient query plans for modern hardware. In *Proceedings of the VLDB Endowment*, volume 4, pages 539–550, 2011.

[137] Thomas Neumann and Bernhard Radke. Adaptive optimization of very large join queries. In *Proceedings of the 2018 International Conference on Management of Data*, pages 677–692, 2018.

[138] NVIDIA Corporation. Nvidia gpu computing, 2024. Available: https://www.nvidia.com.

[139] Frank Olken and Doron Rotem. Random sampling from databases. In *International Conference on Data Engineering*, pages 92–100, 1993.

[140] Matthaios Olma, Manos Karpathiotakis, Ioannis Alagiannis, Manos Athanassoulis, and Anastasia Ailamaki. Adaptive partitioning and indexing for in situ query processing. *The VLDB Journal*, 569(1), 2020.

[141] John D Owens, Mike Houston, David Luebke, Simon Green, John E Stone, and James C Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[142] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318, Philadelphia, PA, USA, 2002. Association for Computational Linguistics, IBM T. J. Watson Research Center. Yorktown Heights, NY 10598, USA. {papineni, roukos, toddward, weijing}@us.ibm.com.

[143] Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. Quicksel: Quick selectivity learning with mixture models. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD/PODS 20, page 10171033. ACM, May 2020. doi: 10.1145/3318464.3389727. URL http://dx.doi.org/10.1145/3318464.3389727.

[144] Viswanath Poosala and Yannis E Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB)*, pages 486–495, 1997.

[145] Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. A survey on text-to-sql parsing: Concepts, methods, and future directions. *ArXiv*, abs/2208.13629, 2022. URL https://api.semanticscholar.org/CorpusID:251903737.

[146] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *OpenAI Technical Report*, 2018. URL https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf.

[147] Christopher Root and Todd Mostak. Mapd: A gpu-powered big data analytics and visualization platform. In *ACM SIGGRAPH 2016 Talks*, pages 1–2. springer, 2016.

[148] Jimi Sanchez. A review of star schema benchmark. *arXiv preprint arXiv:1606.00295*, 2016. URL https://arxiv.org/abs/1606.00295.

[149] Leo Willyanto Santoso and Yulia. Data warehouse with big data technology for higher education. *Procedia Computer Science*, 124:93–99, 2017. doi: 10.1016/j.procs.2017.12.134. URL https://doi.org/10.1016/j.procs.2017.12.134.

[150] Peter Scheuermann, Junho Shim, and Radek Vingralek. Watchman: A data warehouse intelligent cache manager. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, page 5162, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc. ISBN 1558603824.

[151] Tim Schwabe and Maribel Acosta. Cardinality estimation over knowledge graphs with embeddings and graph neural networks. *arXiv preprint arXiv:2303.01140*, 2023. URL https://arxiv.org/abs/2303.01140.

[152] Patricia G Selinger, Morton M Astrahan, and Donald D Chamberlin. Access path selection in a relational database management system. *ACM SIGMOD*, 1979.

[153] Timos K Sellis. Multiple-query optimization. *ACM Transactions on Database Systems (TODS)*, 13(1):23–52, 1988.

[154] Timos K. Sellis. On the multiple query optimization problem. *IEEE Transactions on Knowledge and Data Engineering*, pages 262–266, 1990.

[155] Raghav Sethi et al. Presto: Sql on everything. In *ICDE*, pages 1802–1813, 2019.

[156] Kawthar Shafie Khorassani, Jahanzeb Hashmi, Ching-Hsiang Chu, Chen-Chun Chen, Hari Subramoni, and Dhabaleswar K Panda. Designing a rocm-aware mpi library for amd gpus: early experiences. In *International Conference on High Performance Computing*, pages 118–136. Springer, 2021.

[157] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill Education, 7th edition, 2019. ISBN 978-1260084504.

[158] Mohamed Soliman et al. Orca: A modular query optimizer architecture. In *SIGMOD*, pages 337–348, 2014.

[159] Yuanfeng Song, Yuqiang Li, Shuhuan Fan, Dongsheng He, and Jianming Liao. A new graph neural network-based join optimization algorithm. In *2022 International Conference on Algorithms, Data Mining, and Information Technology (ADMIT)*, pages 20–24, 2022. doi: 10.1109/ADMIT57209.2022.00012.

[160] Ji Sun et al. Learned cardinality estimation: Design space exploration. *VLDB*, 15 (1):85–97, 2021.

[161] R. Sun, S. O. Arik, H. Nakhost, H. Dai, R. Sinha, P. Yin, and T. Pfister. SQL-PaLM: Improved large language model adaptation for text-to-sql. *arXiv preprint arXiv:2306.00739*, 2023.

[162] Sadanand Sundaray. Optimizing sql-on-hadoop engines for scalable data analytics: A hybrid approach to resource allocation and performance tuning. In Audrius Lopata, Daina Gudonienė, and Jonas Čeponis, editors, *Information and Software Technologies*, pages 112–123, Cham, 2026. Springer Nature Switzerland. ISBN 978-3-032-16808-5.

[163] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018.

[164] C.-Y. Tai, Z. Chen, T. Zhang, X. Deng, and H. Sun. Exploring chain of thought style prompting for text-to-sql. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[165] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu, and Raghotham Murthy. Hive - a petabyte scale data warehouse using hadoop. In *Proceedings of the 2010 IEEE 26th International Conference on Data Engineering (ICDE)*, pages 996–1005. IEEE, 2010.

[166] Amund Tveit, Torbjørn Morland, and Thomas Brox Røst. Deeplearningkit-an gpu optimized deep learning framework for apple's ios, os x and tvos developed in metal and swift. *arXiv preprint arXiv:1605.04614*, 2016.

[167] Unknown. Openacc directive-based parallel programming for accelerators. https://www.openacc.org, . Consulté le 21 novembre 2025.

[168] Unknown. pycuda:python wrapper for nvidia cuda. https://pypi.org/project/pycuda/, . Consulté le 21/11/2025.

[169] Unknown. Pyopencl documentation. https://pypi.org/project/pyopencl/, . Consulté le 21/11/2025.

[170] Guy Van den Broeck and Dan Suciu. Query processing on probabilistic data: A survey. *Foundations and Trends in Databases*, 7(3-4):197–341, 2017.

[171] Peter J. M. van Laarhoven and Emile H. L. Aarts. Simulated annealing. In *Simulated Annealing: Theory and Applications*, pages 7–15. Springer, 1987.

[172] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, 2017. URL https://api.semanticscholar.org/CorpusID:13756489.

[173] Guoren Wang, Yue Zeng, Rong-Hua Li, Hongchao Qin, Xuanhua Shi, Yubin Xia, Xuequn Shang, and Liang Hong. Temporal graph cube. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):13015–13030, 2023. doi: 10.1109/TKDE.2023.3270460.

[174] Tianhao Wang, Yi Zeng, Ming Jin, and R. Jia. A unified framework for task-driven data quality management. *ArXiv*, abs/2106.05484, 2021. URL https://api.semanticscholar.org/CorpusID:235390485.

[175] Xiaoying Wang et al. Are we ready for learned cardinality estimation? *VLDB*, 14 (9):1640–1655, 2021.

[176] Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *ArXiv*, abs/2401.04398, 2024. URL https://api.semanticscholar.org/CorpusID:266899992.

[177] Marek Wojciechowski and Monika Rokosik. Efficient processing of streams of frequent itemset queries. In *European Conference on Advances in Databases and Information Systems (ADBIS)*, pages 15–26, 2014.

[178] Yang Wu, Xuanhe Zhou, Yong Zhang, and Guoliang Li. Automatic index tuning: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 36(12):7657–7676, 2024.

[179] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68 (2):121101, 2025.

[180] F. Xu, Z. Wu, Q. Sun, S. Ren, F. Yuan, S. Yuan, Q. Lin, Y. Qiao, and J. Liu. Symbol-LLM: Towards foundational symbol-centric interface for large language models. *arXiv preprint arXiv:2311.09278*, 2024.

[181] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019. URL https://arxiv.org/abs/1810.00826.

[182] Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, and Vadim Sheinin. SQL-to-text generation with graph-to-sequence model. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 931–936, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1112. URL https://aclanthology.org/D18-1112.

[183] Xiaojun Xu, Chang Liu, and Dawn Xiaodong Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *ArXiv*, abs/1711.04436, 2017. URL https://api.semanticscholar.org/CorpusID:10746949.

[184] Jian Yang, Kamalakar Karlapalem, and Qing Li. Algorithms for materialized view design in data warehousing environment. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, page 136145, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1558604707.

[185] Zongheng Yang. *Machine Learning for Query Optimization*. PhD thesis, EECS Department, University of California, Berkeley, Aug 2022. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-194.html.

[186] Zongheng Yang et al. Neurocard: One cardinality estimator for all tables. *VLDB*, 14(1):61–73, 2020.

[187] Zongheng Yang et al. Balsa: Learning a query optimizer without expert demonstrations. In *SIGMOD*, pages 931–944, 2022.

[188] Xuchen Yao and Benjamin Van Durme. Information extraction over structured data: Question answering with freebase. In *Annual Meeting of the Association for Computational Linguistics*, 2014. URL https://api.semanticscholar.org/CorpusID:2131938.

[189] Zhisheng Ye, Wei Gao, Qinghao Hu, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen. Deep learning workload scheduling in gpu datacenters: A survey. *ACM Computing Surveys*, 56(6):1–38, 2024.

[190] Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. Tabert: Pretraining for joint understanding of textual and tabular data, 2020. URL https://arxiv.org/abs/2005.08314.

[191] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[192] Xiang Yu, Chengliang Chai, Guoliang Li, and Jiabin Liu. Cost-based or learning-based? a hybrid query optimizer for query plan selection. *Proc. VLDB Endow.*, 15 (13):3924–3936, 2022.

[193] Xiang Yu et al. A hybrid query optimizer for query plan selection. *VLDB*, 15(13): 3924–3936, 2022.

[194] Haitao Yuan, Guoliang Li, Ling Feng, Ji Sun, and Yue Han. Automatic view generation with deep learning and reinforcement learning. In *2020 IEEE 36th*

International Conference on Data Engineering (ICDE), pages 1501–1512. IEEE, 2020.

[195] Huayi Yuan, Yujing Han, Guoliang Li, and Jianan Sun. Rlview: Reinforcement learning based materialized view selection. In *arXiv preprint arXiv:2004.04741*, 2020.

[196] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud)*, 2010.

[197] Stanley B. Zdonik and David Maier, editors. *Readings in Object-Oriented Database Systems*. Morgan Kaufmann, 1990.

[198] Chuan Zhang, Xin Yao, and Jian Yang. An evolutionary approach to materialized views selection in a data warehouse environment. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 31(3):282–294, 2001.

[199] Zhigen Zhao, Shuo Cheng, Yan Ding, Ziyi Zhou, Shiqi Zhang, Danfei Xu, and Ye Zhao. A survey of optimization-based task and motion planning: From classical to learning approaches. *IEEE/ASME Transactions On Mechatronics*, 30(4):2799–2825, 2024.

[200] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *ArXiv*, abs/1709.00103, 2017. URL https://api.semanticscholar.org/CorpusID:25156106.

[201] Rong Zhu et al. A learned query rewrite system using monte carlo tree search. *VLDB*, 15(1):46–50, 2021.

[202] Xuanhe Zhu et al. Learned query optimizer: At the forefront of ai-driven databases. In *EDBT*, 2022.

[203] Moti Zwilling. Big data challenges in social sciences: An nlp analysis. *Journal of Computer Information Systems*, 63(3):537–554, 2023. doi: 10.1080/08874417.2022.2085211.