Database Optimizers in the Era of Learning

Dimitris Tsesmelis
Athena Research Center, Greece
dimitris.tsesmelis@athenarc.gr

Alkis Simitsis
Athena Research Center, Greece
alkis@athenarc.gr

***Abstract*—In this tutorial, we review advances made recently in a decades-old problem, namely query optimization. Along with the traditional optimization techniques many of which are still being used successfully in production, various techniques inspired by AI, such as genetic algorithms, had been explored as early as in the '90s as potential solutions, without gaining at that time much traction, especially in commercial offerings. More recently, with the rapid progress on learning, several approaches have brought this technology within the core of a database management system (DBMS) aiming at developing scalable, learning solutions to all challenging components of the system optimizer. We present the early efforts in this area, describe advancements, limitations and open issues, and discuss future research directions.**

I. Introduction and Motivation

Query optimization has been studied for more than four decades and continues to be an active area of research. The problem's combinatorial complexity does not help present a unified solution and the current research highlights lead to heuristic and cost-based approaches, which often work well only in the presence of limiting assumptions. Several techniques have been proposed including pruning, genetic, and randomized algorithms, but in corner cases they tend to fail and, thus, to produce poor execution plans.

With the advent of machine learning (ML) technology and the recent, rapid advances in this area, a new trend has emerged in data management research that explores potential opportunities of enriching the currently programmed heuristics with learned ones, uses dynamic, self-learning approaches to cost modeling, and enriches traditional plan generation strategies using learning to learn from the outcomes of previous planning instances and dramatically reduce search time for future planning. Most approaches use deep neural network-based learning techniques, broadly classified as query-based and data-based. The former are typically modeled as supervised learning, with models trained on running queries and using statistics (e.g., cardinalities) measured as labels. The latter are modeled as unsupervised learning and capture distributions and correlations with the joint probability density functions of the underlying data.

Although the idea of a learning, self-serving optimizer is very appealing, reality currently presents significant limitations. Model training is extremely data-intensive and time-consuming. Model maintenance, especially in the presence of data update and schema evolution, imposes practical limitations and calls for schema agnostic approaches. The robustness of the solutions (i.e., query plans) proposed is yet to be shown.

However, as a considerable body of knowledge has already been accumulated and this area of research seems to be full of potential, we believe that it is time to collectively present the state-of-the-art approaches, with their positive findings and their known limitations, and discuss future directions towards building the new generation of data management optimizers.

Recent tutorials touched upon aspects related to ML/AI and DBs [22], [23], [48], [51] and provide a great motivation and a broad overview of using learning in database systems. These studies are an excellent introduction to works focusing on specific areas and delving into them in detail, as ours.

II. Tutorial scope and coverage

Our tutorial focuses on the optimization problem from a systemic perspective and aims at presenting a comprehensive review of query optimization approaches using learning techniques, answering questions such as: (a) how far along have we went so far with learning query optimization, what have we learnt and what is still missing, (b) whether learning provides us with a competitive advantage over or along with traditional query optimization, and (c) whether learning query optimization has the potential to impact systems in production.

Duration and Outline. We propose a 90-min tutorial:

1) Introduction, motivation, and brief overview of the evolution of query optimization techniques [∼10']
2) Learning query optimization [∼20']
3) Learning cost estimation [∼20']
4) Learning runtime prediction [∼20']
5) Benchmarks, data preparation, and statistics [∼10']
6) Open issues and Research Directions [∼10']

Our presentation will cover the works listed next, with analytical examples and multidimensional comparisons of their offerings and limitations (Fig. 1-2 show abridged examples).

A. Traditional query optimization

Query optimization largely depends on cardinality and selectivity estimation, and in particular, on having reasonably good estimates for intermediate result sizes. Related approaches employ a variety of techniques (e.g., histograms, entropy, probabilistic models, sketches, etc.), and work with assumptions (e.g., attribute value independence, uniformity, data independence) and when these cannot be met the techniques usually fall back to an educated guess. We will revisit briefly some popular techniques to set the context and motivate the opportunity that arises for learning techniques.

Optimizer	Method	D/S[1]	E[2]	E/E[3]	Gain	Feat.	Train.	Re-Train.	Pract.[4]
DQ [20]	MDP w/ deep Q-learn.	✗/✗	✗	Rand. Sampl.	JO	1-h on columns, flat	Native plans	Partial	
ReJOIN [32]	MDP w/ policy grad.	✗/✗	✗	Policy Sampl.	JO	1-h on relations, flat	Cold start	No	
RTOS [58]	MDP w/ deep Q-learn.	✓/✓	✗	Rand. Sampl.	JO	1-h, flat + Forest enc.	Native cost	Yes	
SkinnerDB [45]	UCT w/ incr. proc.	✓/✓	✗	UCT	JO	n/a	No	Contin.	
AlphaJoin [59]	MCTS w/ UCT	✗/✗	✗	UCT	JO	1-h on relations, flat	Hist. runs	No	
Neo [33]	MDP w/ TCNN	✓/✗	✗	Rand. Sampl.	JO,PO	1-h plan enc., tree	Native plans	Yes	
Bao [31]	CMABP w/ TCNN	✓/✓	✓	Thoms. Sampl.	JO,PO	1-h phys. op. etc., tree	Native plans	Yes	
Robopt [16]	Plan Enum. w/ ML	✓/✓	✗	n/a	PP,PO	1-hot on 4 elements	Data gen.	No	
Microlearner [14]	Micromodels	✓/✓	✓	n/a	n/a	Oper. feat., flat	SCOPE jobs	Yes	

Fig. 1. Comparison of learning optimizers [[1]Data/Sch. agn., [2]Explainable, [3]Explor./Exploit., [4]Practicality, Join Order Sel. (JO), Physical Operator Sel. (PO), Platform-specific Plan (PP), ✓, ✓, ✗: full, partial, and no support]

B. Learning query optimization

System Design. Many argue that the advances in ML can be used to build better, smarter, and easier to use (as in maintain, tune, optimize) DBMSs and learned components can fully replace core components of a DBMS, such as data structures, indices, sorting algorithms, and query execution [6], [19], [51]. Example system frameworks that work towards this direction include *SageDB* [18] and *openGauss* [24].

Join ordering. Reinforcement learning (RL) has been used to improve join ordering. *DQ* [20] and *ReJOIN* [32] combine RL with a human-engineered cost model to automatically learn search strategies to navigate the space of possible join orderings. *RTOS* [58] extends these by employing a Tree-LSTM to compute query cost. *SkinnerDB* [45] uses RL to learn optimal join orders during query execution and enables fast join order switching. *AlphaJoin* [59] uses MCTS for join ordering and ADN to choose between plans produced by AlphaJoin (for long queries) and PostgreSQL (for short queries).

Optimizers. *Elfino* [4] is a self-driving query optimizer, which tracks queries in their entire lifespan and uses an AI algorithm to learn from mistakes made by the Spark Optimizer. *Neo* [33] replaces most traditional optimizer components with ML models and deep neural networks (NN). *Bao* [31] follows a schema and data agnostic approach to learning optimization. It runs a query with a predefined set of hints and keeps the plan ranked first by a tree convolutional NN (TCNN). Periodic retrain balances exploration and exploitation. *Microlearner* [14] divides a complex cloud workload into a hierarchy of subsets, using them to learn micro-models independently and in parallel. *Robopt* [16] uses ML to prune the exponentially growing search space of plans in cross-platform query optimization.

Comparison. How mature and practical an optimizer is could be assessed through various dimensions like method used, data/schema agnostic, explainability, exploration vs. exploitation, source of performance gain, featurization, requiring training/retraining, realistic baselines, datasets, scalability, distributed execution, overhead, performance, robustness, evaluation details (benchmark, size, workload, execution engine, reproducibility), and method details (e.g., optimizer, loss function, reward, NN layers/units). Figure 1 shows a simplified and abridged (due to space constraints) comparison of learning optimizers including a subjective measure 'practicality' that combines all the dimensions included in our analysis in evaluating how feasible would be to use an approach in production.

C. Learning cost estimation

Cardinality. [17] formulates query features as sets and feeds them to a MSCN. [52] builds local models, each focusing on a part of the db schema. [36] evaluates deep learning techniques w.r.t. trade-offs among model size, training time, and prediction accuracy. *E2E* [42] uses tree-structured NNs to predict query cardinality and cost. *CRN* [12] uses deep learning to estimate cardinalities using the containment rates among queries. *Fauce* [25] incorporates uncertainty information for cardinality estimation into a deep learning model. *DeepDB* [13] proposes a data-driven approach that supports workload and data changes, fast retraining, and reasonable accuracy for unseen queries. *FLAT* [61] builds on top of DeepDB [13], considering the level of dependency between attributes. *Naru* [56] and *NeuroCard* [57] use autoregressive models to learn the joint distribution of relations, with the latter focusing on learning the full outer join of all tables. *UAE* [53] uses data and queries to accurately estimate cardinalities, even when data or workload shifts. [43], [50] learn estimating the cardinality for similarity search.

Selectivity. *LW* [7] employs NNs and tree-based ensembles to improve selectivity estimation of multi-dimensional range predicates. [11] employs neural density estimation to build a selectivity estimator of the joint probability distribution from data samples. *ASTRID* [39] builds neural language models for selectivity estimation of prefix and substring queries. *LATEST* [37] uses classification and decision trees to choose estimators for time windows on streams.

Cost models. [29] partitions the feature space with decision trees and builds a regression model for each split. [18] makes traditional cost models differentiable to support ad hoc queries. *CherryPick* [2] builds performance models by finding deployments that satisfy a performance target using Bayesian optimization. *CLEO* [40] learns cost models from production workloads and integrates them within a query optimizer.

Comparison. Figure 2 shows a classification of learning cost estimators. Supervised methods work better in static environments, where they avoid re-training. Several data-driven approaches adopt incremental training, thus becoming more dynamic. Most approaches use q-error as the de facto loss function or quality metric. *Flow-Loss* [35] presents a novel loss function that considers the quality of the estimated cardinalities in terms of query runtime. [49] argues that the proposed solutions are not yet production ready due to issues like high

Approach		Method	D/S[1]	Adaptability	Featurization	Loss	CC[2]	Unif. Bench.
Query-driven	[7], [11], [12], [17], [25], [36], [37] [39], [42], [43], [50], [52]	Regression	✗/✗	Re-Train	1-h, embeddings	Q-error variants	Limited	✗
Data-driven	[13], [53], [56], [57], [61]	Joint Prob. Distr.	✓/✗	Incremental	1-h, embeddings	Cross entropy	Limited	✗

Fig. 2. Classification of learning cost estimators (table shows the predominant values per category) [[1]Data/Sch. agn., [2]Commercial Comparison]

training/inference time, black-box decisions, and difficulty to recover after data updates. Most approaches lack a comparison against mature commercial systems or a unified benchmark.

D. Learning runtime prediction

Early approaches applied learning techniques to runtime prediction, e.g., [8], [10], [46]. [1] shows that using linear regression to map PostgreSQL's estimate to actual execution time is not effective. [54] argues that the optimizer estimates can be useful if the optimizer's cost model is tuned just before making an estimate. *Q-Cop* [44] and *ActiveSLA* [55] exploit query runtime prediction to perform *admission control*.

Recent efforts show promising results in runtime prediction using learning, treating a DBMS as a black-box and building efficient query runtime prediction models, e.g., [26], [34], [60]. *Prestroid* [15] uses TCNN to predict resource consumption of large scale queries at a lower cost. *RouLette* [41] uses RL to identify shareable work among SPJ queries and perform multi-query optimization and runtime adaptation. *MB2* [28] employs ML to create and maintain prediction models for self-driving databases. *Seagull* [38] leverages runtime prediction to optimize resource allocation on the cloud.

In the presentation, we will elaborate on these works and will provide a multidimensional comparison.

E. Benchmarks, data preparation, and statistics

Benchmarks. Traditional benchmarks like the TPC suite are not suitable for learning approaches. Recent benchmarks focus on specific problems like join ordering (JOB [21]) and learned indexes [30]. Still, we need richer benchmarks based on realistic datasets (synthetic data is either random (too difficult), or follows a known data distribution (too easy)), supporting dynamic workloads and schemas, descriptive metrics about the specialization of the systems tested, adaptability to data/schema changes, similarity between workloads and data distributions, and cost to train/maintain the models [5], [30].

Data preparation. DataFarm [47] uses small, user provided workload patterns to generate a labeled dataset. It employs adaptive learning to iteratively execute sample jobs and produce labels for building a ML runtime prediction model for unlabeled jobs. *Robopt* [16] offers a training data generator that creates a query workload with query runtime estimates and uses polynomial interpolation to label new queries from already acquired labels. *HAL* [27] generalizes a learned database component across different datasets and workloads.

Statistics. Learning optimizers can scale beyond typical database statistics and tap on a much richer pool of raw information, including metrics related to execution containers, processes, services, and applications. This allows using learning techniques to develop profiling models, identify the root cause of failures, generate recommendations for improving performance or resource allocation, etc. [3], [9].

F. Open Problems and Future Directions

Limitations. Pain points include: (1) cost and complexity of *model training* and *maintenance*, (2) *robustness* of query plans due to the stochastic nature of learning, (3) *catastrophic forgetting* when data distribution shifts over time, (4) lack of *standard benchmarks* as ML approaches can be biased by the underlying data and workload, (5) *fair comparison* with industrial strength practices (e.g., compressed indexes, complex data formats like compressed/encoded/column-store/parquet, provision for disk spills), and (6) consideration of *realistic environments* supporting features like recovery and concurrency.

Opportunities. There are several directions worth pursuing: (1) *Remove* human from the loop and replace today's needs for human tweaks with ML based tweaks: operations primarily affected include optimization, caching, and workload management/scheduling. (2) *Identify* techniques that work consistently well: as the ML toolkit is arguably large, the proposed solutions span a large variety of methods, without having yet a clear winner. (3) *Rethink* the end-to-end system architecture: (a) consider a brand new runtime, (b) decompose the monolithic database system into mini-services and think of instance-optimized database systems, and (c) instead of replacing traditional database boxes with ML boxes, we may need to think of a totally new set of boxes all together. (4) *Expand* the analysis to: (a) complex data (e.g., graph, spatial, streaming), (b) distributed and/or federated architectures involving more than one execution engine, (c) a multitude of optimization objectives (not just cost). In the presentation, we will elaborate on these ideas and will provide a few examples.

III. Target Audience and Learning Output

Material. The tutorial will be example-driven showcasing the strengths and limitations of the state of the art. The tutorial material will become publicly available.

Audience. It comprises students, academics, researchers, and practitioners who want to understand the current state of the art and the future directions of learning based optimization. No prior knowledge is needed on learning techniques, but we assume basic understanding of basic database concepts.

Output. The learning output includes: (a) Understanding use cases and existing learning approaches to query optimization, cost estimation, and runtime prediction. (b) Understanding the technical limitations and the trade-offs between design choices and achieved goals. And (c) exposure to the new challenges with learning as a core architectural component, which may also combine techniques across the optimization pipeline using transfer learning and multi-task learning.

IV. Presenters

Dimitris Tsesmelis is a Marie Skłodowska-Curie early stage researcher at Athena RC and UPC. He obtained his BSc from University of Athens (graduated 2nd in his class) and a joint MSc from ULB in Brussels, UPC in Barcelona, and CS in Paris. In the past, he worked in Accenture building large-scale data management solutions. His current research interests span data management, distributed systems, and machine learning.

Alkis Simitsis is a Research Director at Athena Research Center. Previously, he held various positions with HP/HPE Labs, Micro Focus, Unravel Data, and IBM Research, including Chief/Principal Scientist. He has 18+ years of experience building innovative solutions for scalable big data infrastructure, data-intensive analytics, distributed databases, and systems optimization. He holds 41 patents, has published 110+ papers (6300+ citations, h-index: 42), and frequently serves in various roles in PC's of top-tier int'l scientific conferences.

Acknowledgment

This work has received funding from the EU's Horizon 2020 MSCA-ITN programme (grant agreement No 955895).

References

[1] M. Akdere *et al.*, "Learning-based query performance modeling and prediction," in *ICDE*, 2012.
[2] O. Alipourfard *et al.*, "Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics," in *NSDI*, 2017.
[3] A. Arvanitis *et al.*, "Automated performance management for the big data stack," in *CIDR*, 2019.
[4] S. Babu and A. Popescu, "Using ai to build a self-driving query optimizer," in *Spark+AI Summit*, 2018.
[5] L. Bindschaedler *et al.*, "Towards a benchmark for learned systems," in *ICDE Workshops*, 2021.
[6] J. Ding *et al.*, "ALEX: an updatable adaptive learned index," in *SIGMOD*, 2020.
[7] A. Dutt *et al.*, "Selectivity estimation for range predicates using lightweight models," in *PVLDB*, 2019.
[8] A. Ganapathi *et al.*, "Predicting multiple metrics for queries: Better decisions enabled by machine learning," in *ICDE*, 2009.
[9] H. Grushka-Cohen *et al.*, "Diversifying database activity monitoring with bandits," *CoRR:abs/1910.10777*, 2019.
[10] C. Gupta *et al.*, "Pqr: Predicting query execution times for autonomous workload management," in *ICAC*, 2008.
[11] S. Hasan *et al.*, "Deep learning models for selectivity estimation of multi-attribute queries," in *SIGMOD*, 2020.
[12] R. Hayek and O. Shmueli, "Improved cardinality estimation by learning queries containment rates," in *EDBT*, 2020, pp. 157–168.
[13] B. Hilprecht *et al.*, "DeepDB: Learn from data, not from queries!" *PVLDB 13(7)*, 2020.
[14] A. Jindal *et al.*, "Microlearner: A fine-grained learning optimizer for big data workloads at microsoft," in *ICDE*, 2021.
[15] J. K. Z. Kang *et al.*, "Efficient deep learning pipelines for accurate cost estimations over large scale query workload," in *SIGMOD*, 2021.
[16] Z. Kaoudi *et al.*, "Ml-based cross-platform query optimization," in *ICDE*. IEEE, 2020.
[17] A. Kipf *et al.*, "Learned cardinalities: Estimating correlated joins with deep learning," in *CIDR*, 2019.
[18] T. Kraska *et al.*, "Sagedb: A learned database system," in *CIDR*, 2019.
[19] T. Kraska *et al.*, "The Case for Learned Index Structures," in *SIGMOD*, 2018.
[20] S. Krishnan *et al.*, "Learning to optimize join queries with deep reinforcement learning," 2018, CoRR:abs/1808.03196.
[21] V. Leis *et al.*, "How good are query optimizers, really?" in *PVLDB 9(3)*, 2015.
[22] G. Li *et al.*, "Machine learning for databases," *PVLDB 14(12)*, 2021.
[23] G. Li *et al.*, "AI Meets Database: AI4DB and DB4AI," in *SIGMOD*, 2021.
[24] G. Li *et al.*, "openGauss: An autonomous database system," in *PVLDB 14(12)*, 2021.
[25] J. Liu *et al.*, "Fauce: Fast and accurate deep ensembles with uncertainty for cardinality estimation," *PVLDB 14(11)*, 2021.
[26] L. Ma *et al.*, "Query-based workload forecasting for self-driving database management systems," in *SIGMOD*, 2018.
[27] L. Ma *et al.*, "Active learning for ML enhanced database systems," in *SIGMOD*, 2020.
[28] L. Ma *et al.*, "MB2: decomposed behavior modeling for self-driving database management systems," in *SIGMOD*, 2021.
[29] T. Malik *et al.*, "A black-box approach to query cardinality estimation," in *CIDR*, 2007.
[30] R. Marcus *et al.*, "Benchmarking Learned Indexes," in *PVLDB 14(1)*,2021.
[31] R. Marcus *et al.*, "Bao: Making learned query optimization practical," in *SIGMOD*, 2021.
[32] R. Marcus and O. Papaemmanouil, "Deep reinforcement learning for join order enumeration," in *aiDM@SIGMOD*, 2018.
[33] R. C. Marcus *et al.*, "Neo: A learned query optimizer," in *PVLDB 12(11)*, 2019.
[34] R. C. Marcus and O. Papaemmanouil, "Plan-structured deep neural network models for query performance prediction," in*PVLDB 12(11)*, 2019.
[35] P. Negi *et al.*, "Flow-loss: Learning cardinality estimates that matter," *PVLDB 14(11)*, 2021.
[36] J. Ortiz *et al.*, "Learning state representations for query optimization with deep reinforcement learning," in *DEEM@SIGMOD*, 2018.
[37] M. Patil and A. Magdy, "LATEST: learning-assisted selectivity estimation over spatio-textual streams," in *ICDE*, 2021.
[38] O. Poppe *et al.*, "Seagull: An infrastructure for load prediction and optimized resource allocation," in *PVLDB 14(2)*, 2021.
[39] S. Shetiya *et al.*, "Astrid: Accurate selectivity estimation for string predicates using deep learning," in *PVLDB 14(4)*, 2021.
[40] T. Siddiqui *et al.*, "Cost models for big data query processing: Learning, retrofitting, and our findings," in *SIGMOD*, 2020.
[41] P. Sioulas and A. Ailamaki, "Scalable multi-query execution using reinforcement learning," in *SIGMOD*, 2021.
[42] J. Sun and G. Li, "An end-to-end learning-based cost estimator," *VLDB 13(3)*, 2019.
[43] J. Sun *et al.*, "Learned cardinality estimation for similarity queries," in *SIGMOD*, 2021.
[44] S. Tozer *et al.*, "Q-cop: Avoiding bad query mixes to minimize client timeouts under heavy loads," in *ICDE*, 2010.
[45] I. Trummer *et al.*, "Skinnerdb: Regret-bounded query evaluation via reinforcement learning," in *SIGMOD*, 2019.
[46] S. Venkataraman *et al.*, "Ernest: Efficient performance prediction for large-scale advanced analytics," in *NSDI*, 2016.
[47] F. Ventura *et al.*, "Expand your training limits! generating training data for ml-based data management," in *SIGMOD*, 2021.
[48] W. Wang *et al.*, "Database meets deep learning: Challenges and opportunities," 2019, CoRR:abs/1906.08986.
[49] X. Wang *et al.*, "Are we ready for learned cardinality estimation?" *PVLDB 14(9)*, 2021.
[50] Y. Wang *et al.*, "Monotonic cardinality estimation of similarity selection: A deep learning approach," in *SIGMOD*, 2020.
[51] A. Wasay *et al.*, "Deep learning: Systems and responsibility," in *SIGMOD*, 2021.
[52] L. Woltmann *et al.*, "Cardinality estimation with local deep learning models," in *aiDM@SIGMOD*, 2019.
[53] P. Wu and G. Cong, "A unified deep model of learning from both data and queries for cardinality estimation," in *SIGMOD*, 2021.
[54] W. Wu *et al.*, "Predicting query execution time: Are optimizer cost models really unusable?" in *ICDE*, 2013.
[55] P. Xiong *et al.*, "Activesla: a profit-oriented admission control framework for database-as-a-service providers," in *SoCC*, 2011.
[56] Z. Yang *et al.*, "Deep unsupervised cardinality estimation," in *PVLDB 13(3)*, 2019.
[57] Z. Yang *et al.*, "Neurocard: One cardinality estimator for all tables," *PVLDB 14(1)*, 2020.
[58] X. Yu *et al.*, "Reinforcement learning with tree-lstm for join order selection," in *ICDE*, 2020.
[59] J. Zhang, "Alphajoin: Join order selection à la alphago," in *PVLDB-PhD*, 2020.
[60] X. Zhou *et al.*, "Query performance prediction for concurrent queries using graph embedding," in *PVLDB 13(9)*, 2020.
[61] R. Zhu *et al.*, "FLAT: fast, lightweight and accurate method for cardinality estimation," *PVLDB*, vol. 14, no. 9, pp. 1489–1502, 2021.