Optimizing Tensor Computations: From Applications to Compilation and Runtime Techniques

Matthias Boehm
TU Berlin
Berlin, Germany
matthias.boehm@tu-berlin.de

Matteo Interlandi
Microsoft GSL
Los Angeles, CA, USA
matteo.interlandi@microsoft.com

Chris Jermaine
Rice University
Houston, TX, USA
cmj4@rice.edu

ABSTRACT

Machine learning (ML) training and scoring fundamentally relies on linear algebra programs and more general tensor computations. Most ML systems utilize distributed parameter servers and similar distribution strategies for mini-batch stochastic gradient descent training. However, many more tasks in the data science and engineering lifecycle can benefit from efficient tensor computations. Examples include primitives for data cleaning, data and model debugging, data augmentation, query processing, numerical simulations, as well as a wide variety of training and scoring algorithms. In this survey tutorial, we first make a case for the importance of optimizing more general tensor computations, and then provide an in-depth survey of existing applications, optimizing compilation techniques, and underlying runtime strategies. Interestingly, there are close connections to data-intensive applications, query rewriting and optimization, as well as query processing and physical design. Our goal for the tutorial is to structure existing work, create common terminology, and identify open research challenges.

CCS CONCEPTS

• **Information systems** → **Data management systems**; **Data mining**; **Computing platforms**; • **Computing methodologies** → **Machine learning**; **Distributed computing methodologies**; **Parallel computing methodologies**; • **Mathematics of computing** → **Mathematical software**; • **Theory of computation** → **Design and analysis of algorithms**.

KEYWORDS

Tensor Computations, Linear Algebra, Declarative Machine Learning, Data Science, Data Engineering, Large-scale Machine Learning

ACM Reference Format:
Matthias Boehm, Matteo Interlandi, and Chris Jermaine. 2023. Optimizing Tensor Computations: From Applications to Compilation and Runtime Techniques. In *Companion of the 2023 International Conference on Management of Data (SIGMOD-Companion '23), June 18–23, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3555041.3589407

1 INTRODUCTION

Over the past decade(s), a wide variety of machine learning (ML) systems emerged from the ML/stats, data management, and high-performance computing communities. Early work focused on statistical computing platforms like R and ML algorithm libraries like scikit-learn [98], as well as specialized systems for clustering, matrix factorization [40], graph processing [126], and others. Later work also provided infrastructure for scalable, distributed computation [3, 104, 131] and hardware accelerators. Nowadays, most general-purpose ML systems focus exclusively on distributed parameter servers [1, 30, 57, 77, 115] and similar distribution strategies [17, 64, 101] for mini-batch training via stochastic gradient descent (SGD). Many more tasks in data science and engineering rely on linear algebra and numerical computation, but they often exhibit different characteristics and are implemented separately [9].

General Tensor Computations: In contrast to this narrowing focus on mini-batch SGD, in this tutorial, we make a case for supporting general linear algebra programs and tensor computations for a wide variety of applications and workload characteristics. Besides diverse ML algorithms and statistical learning [39], there are new compelling use cases. First, state-of-the-art data integration [31], feature and semantic type detection [50, 113, 132], and data cleaning [78, 121] all rely on ML. Second, there is work on data imputation, cleaning, and ML tightly interwoven with query processing [19], which requires integrated systems support [28, 32, 38, 44, 129]. Third, also data augmentation and simulation cleanly map to numerical computation. Recent work applies machine learning for more cost-effective weather forecasting [2] as well as simulations of fluid dynamics and material deformation [99]. Interestingly, both data augmentation and simulation allow for generating unlimited datasets. Fourth, even complex, enumeration-based algorithms for model debugging [106] as well as tree-based models [86] can be expressed and efficiently executed in linear algebra.

Optimizing Tensor Computations: Efficient and scalable system infrastructure for such use cases relies—due to complex, hierarchically composed primitives—on optimizing compilers and generating scalable runtime plans. Given increasing specialization, hand-crafting plans for different characteristics and deployments becomes infeasible. Automatic plan generation allows to seamlessly adapt to diverse workloads and data characteristics. In this context, a rapidly growing set of compilation and runtime techniques emerges. Our goal for this tutorial is to structure the space, create common terminology, and identify open research challenges. Common terminology and well-defined sub-areas would serve our community well by focusing efforts and simplifying reuse.

Tutorial Scope: Drawing from our experience building systems for linear algebra and tensor computations (e.g., SystemML [12],

SIGMOD-Companion '23, June 18–23, 2023, Seattle, WA, USA

ACM ISBN 978-1-4503-9507-6/23/06…$15.00
https://doi.org/10.1145/3555041.3589407

SystemDS [11], Hummingbird [86], TQP [44], SimSQL [18, 39, 52, 82]), we aim to survey the state-of-the-art from applications to compilation and runtime techniques. The technical background is summarized in Sections 2-4, and the tutorial format is as follows:

- *Tutorial Type:* Survey,
- *Preferred Duration:* 3 hours (1.5 hours would also be possible but only in reduced breadth and depth),
- *Target Audience:* Systems researchers and practitioners with basic applied ML background (we do not expect prior knowledge of state-of-the-art algorithms or system internals), and
- *Hands-on Tutorial:* no HW/SW requirements.

2 TENSOR COMPUTATIONS

In data-centric ML pipelines, there are multiple compelling use cases for more general tensor computations and respective system infrastructure in terms of compilation and runtime techniques.

2.1 Data Preparation and Cleaning

The inspiring tutorial by Dong and Rekatsinas [31] made a great case for the natural symbiosis of data integration and machine learning. Example tasks that heavily rely on ML—and thus, tensor computations—are data extraction, schema alignment, entity resolution, and data fusion. Interestingly, the same observation applies to other data preparation tasks such as data validation [110], data cleaning [45, 78], outlier/anomaly detection [127], missing value imputation [121], semantic type detection [50, 132], feature selection [123], feature engineering [113], and feature transformations [100]. For example, missing value imputation via chained equations (mice) [121] repeatedly extracts features with missing values, trains models (classifiers for categorical, regressors for numerical) using observed feature values as labels, and utilizes the models for missing value imputation. For this reason, implementing cleaning primitives in linear algebra is very compelling because it avoids unnecessary boundary crossing among systems and libraries.

2.2 Data Augmentation and Simulation

Data augmentation takes a small labeled dataset and generates many more synthetic examples via transformations and the original labels. Common transformations include reflections, translations, shearing, rotations, cropping, and mixup, which increase data coverage for improved generalization. The seminal AlexNet [72] paper heavily relied on data augmentation (by 2048x the original datasize), and since then it has become common practice. Recently, additional work also tunes data augmentation pipelines and their parameters [26], and pushes data augmentation as specialized kernels into model training to avoid data materialization [29]. Furthermore, machine learning is also applied for more cost-effective weather forecasting [2] as well as simulations of fluid dynamics and material deformation [99]. In this context, very simple MLP models are utilized and the simulation characteristics yield a wide variety of workload characteristics. Prior related work also focused on Monte Carlo sampling [51], and Markov chain simulation [18, 39]. Both data augmentation and simulation allow for generating unlimited datasets with interesting opportunities of directed sampling according to model accuracy as well as fusion.

2.3 Query Processing

Recently, tensor computations have been proposed for executing relational operators and even full queries. TCUDB [48] maps join operations into matrix multiplications for efficient execution on Tensor Cores [27, 91]. Raven [66] co-optimizes classical ML models and relational queries. During optimization, Raven can push relational operations such as projection and filters into the ML model as tensor operations. TQP [37, 44] maps Spark SQL queries into tensor computations. TQP supports the full TPC-H benchmark. TQP implements several relational operators (join, aggregation, group by, etc) as PyTorch tensor programs, and chains them together to form query plans which are executable on any hardware supported by PyTorch (e.g., CPU, GPU, TPU [61, 62], etc). Beyond using tensor computations for allowing queries to leverage hardware acceleration, TQP has also proposed tensor computations for query processing over unstructured data such as images, as well taking advantage of the auto-differentiation infrastructure in PyTorch for enabling differentiable queries [38]. While many preliminary results are very promising, other work has also pointed out remaining challenges of mapping queries to TPUs [47].

2.4 ML Algorithms and Debugging

Finally, there is rich literature on first- and second-order optimization, statistical learning, a variety of ML models, and more specialized fields such as robust optimization. Besides such algorithms—including tree-based models for both inference [86] and training[1]—that naturally map to tensor computations, recently also model debugging, explanations, and fairness constraints have been elegantly expressed in linear algebra. Examples include linear-algebra-based slice finding [106], learning curve prediction for different slices [120], explanations via linear approximations [81, 85, 103], as well as constrained and unconstrained optimization for fairness and accuracy [89, 107, 133, 133].

3 COMPILATION TECHNIQUES

Compilation techniques in ML systems are inspired by programming language compilers, query optimization in DBMS, and optimizing HPC compilers. Some of the covered material overlaps with a previous SIGMOD 2017 tutorial [73] but existing work evolved significantly in the past six years.

3.1 Simplification Rewrites

Size Propagation: As a basis for advanced compilation techniques, many systems first propagate size information [16] (e.g., dimensions and sparsity) and subsequently use this information for memory and cost estimation. A central challenge is sparsity estimation of intermediates, which is addressed via naïve metadata estimators [16], naïve bitset estimators [80, 118], density maps [67, 68], biased sampling [128], layered graphs [22, 23], and sketches [116]. Besides sparsity, there has been work on propagating other properties such as symmetry, constants, and storage formats [105].

[1] For example, see the veectorized decisionTree() and randomForest() built-in functions—as well as their corresponding predict functions—in Apache SystemDS [11].

Rewrites: Applied rewrites then include traditional programming language rewrites—such as common subexpression elimination, constant folding, branch removal, and loop hoisting [4, 24]—as well as simplification rewrites for linear algebra expressions [16, 74, 119], dedicated dynamic programming approaches for matrix multiplication chains, and sparsity exploitation [12, 49]. Examples systems that apply such rewrites are SystemDS, TensorFlow, and PyTorch. Graph substitutions are also applied to expressions in deep neural networks [35, 56]. Other rewrites include loop vectorization, and incremental computations [90, 108, 109]. Recent work aims to overcome the need for hand-crafting simplification rewrites by automatic rewrite generation and sum-product optimization based on meta-properties of operations [33, 55, 69, 125].

3.2 Operator Fusion and Code Generation

Operator fusion and code generation are used during ahead-of-time and just-in-time compilation in order to eliminate unnecessary intermediates, apply scan sharing, exploit sparsity, and specialize runtime plans to the underlying, increasingly specialized hardware and runtime strategies. Example systems with dedicated code generators include BTO [10], Tupleware [25], Kasen [135], SystemDS [14, 33], Weld [94, 95], TACO [70], PlinyCompute [137], Julia [60], TensorFlow XLA, PyTorch, Tensor Comprehensions [122], TVM [20], NIVIDA TensorRT [92], DAPHNE [28], and Tuplex [117]. Recently, several systems were built on top of existing code generators (e.g., JAX on TensorFlow XLA, and TQP on TVM). MLIR [75] aims to avoid redundancy by providing compiler-infrastructure as a library with clearly defined dialects, which gains popularity because hardware vendors can provide specific dialects for their hardware devices, easing their adoption. Remaining challenges include increasing the fusion potential for integrated query processing and linear algebra / tensor operations with dynamic tensor shapes.

3.3 Operator Selection and Placement

Beyond dedicated systems for local and distributed computation, there are several ML systems with multiple backends. Examples include PyTorch [97], TensorFlow [1], SystemDS [11], Samsara [111], DaskML [104], and code generators like TF XLA and TVM [20]. Predominantly though, operator selection and placement is still done via heuristics and manual placement. SystemDS also automatically chooses local and distributed operations depending on memory estimates and budgets, as well as different physical operators based on data characteristics. Reinforcement learning has been successfully used to place neural network layers onto multiple heterogeneous hardware devices [83]. In addition to placing entire operators on devices, DTensors [43] in TensorFlow and PyTorch as well as federated matrices/frames in DAPHNE [28] further allow for more fine-grained placement of shards of tensors on heterogeneous devices. Recent work on compiler infrastructure for ASICs also include the spatial-temporal mapping of data flow graphs to die space [93] and "hardware islands" of multiple devices [6].

4 RUNTIME STRATEGIES

Underneath the evolving compilation techniques, there are important runtime strategies, especially regarding data representations, parallelization strategies, and dedicated runtime backends.

4.1 Data Representations

Overall, we observe an increasing specialization in terms of partitioning and tiling strategies of matrices/frames; as well as dense, sparse, and compressed tile representations. First, in terms of overall partitioning there are local tensors, distributed collections of tiles, as well as federated or sharded tensors with implicit/or explicit sharding information. Distributed collections of tiles originate from ML systems on data-parallel computing frameworks like Spark [131], Flink [3], or Dask [104]. Later, such abstractions have also been adopted for in-DBMS machine learning [52, 82, 112]. This representation has been proven to be very versatile, and recent work on tensor relational algebra makes a case for adopting it as a logical abstraction [54, 129]. Second, at the level of individual tiles, we see more and more specialized sparse [21, 91] and compressed [7, 34, 59, 65, 76, 124, 134] matrix representations as well as specialized data types [71, 91]. Unfortunately, the selection of such representations is still largely a manual trial and error process.

4.2 Parallelization Strategies

Over the last decade, a wide variety of parallelization strategies has been devised, often designed to exploit the characteristics of the underlying hardware and compute infrastructure. First, data-parallel operations follow an SPMD (single-program, multiple-data) model on distributed collections. A variety of physical operators for broadcast-based, shuffle-based, and specialized operations has been proposed and integrated into ML systems [12, 53, 111]. Second, for use cases like hyper-parameter tuning, cross-validation, and embarrassingly-parallel programs, task parallelism (e.g., via parallel for loops) and hybrid parallelism (e.g., concurrent data-parallel jobs on large-scale datasets) has been adopted as well [15, 63, 114]. Third, irregular workloads—as used in reinforcement learning—are addressed with task-dependency-graphs and future-based scheduling [79, 84] as well as tightly integrated architectures of CPU drivers and hardware accelerators [46]. Fourth, distributed mini-batch training relies on parameter servers [1, 30, 57, 77, 115] and similar distribution strategies [41]. Here, we often differentiate data- and model-parallel parameter servers, which hold data and model partitions, respectively. Given the challenges of efficient data exchange and synchronization barriers, compared to tuned single-node implementations, recent work added dedicated parallelization for training multiple models [87, 88], sampled, independent subnet training [130], as well as sparsity and locality exploitation for sparse and skewed parameter access (e.g., matrix factorization) [101, 102].

4.3 Alternative Backends

Although many ML systems comprise custom runtime backends—with the help of communication libraries such as gRPC, MPI, and NCCL [36]—there is commonly used infrastructure. The first generation of parameter servers [30, 115] often relied on parameter management on Key/Value-stores. Similarly, Function-as-a-Service (FaaS) ML systems—which aim for low start-up costs and automatic elasticity—communicate over rather slow Key-Value Stores and even object stores like S3 [58]. A very popular infrastructure are general-purpose data-parallel computation frameworks like Spark [131], Flink [3], and Dask [104]. Other backends include Ray [84] for irregular task-parallelism, SQL with dedicated matrix/vector types and recursive computation [52], and federated learning backends (e.g., SystemDS federated [8] and TensorFlow federated [42, 64]) with additional integration of privacy enhancing technologies. The individual ML system backends also implement specific techniques for providing efficient data access. Examples include buffer-pool-like eviction of live variables from GPU to CPU memory or from CPU memory to disk [13], dedicated tile layouts (aka page layouts) with reordered and padded rows [5], as well as in-memory and disk-based index structures for out-of-core data [68, 96, 136].

5 BIOGRAPHIES

The tutorial presenters have backgrounds from industry and academia, and have built various systems with different architectures.

Matthias Boehm: Matthias Boehm is a full professor for large-scale data engineering at Technische Universität Berlin and the BIFOLD research center. His research group focuses on high-level, data science-centric abstractions as well as systems and tools to execute these tasks in an efficient and scalable manner. From 2018 through 2022, Matthias was a BMK-endowed professor for data management at Graz University of Technology, Austria, and a research area manager for data management at the co-located Know-Center GmbH. Prior to 2018, he was a postdoc and research staff member at IBM Research - Almaden, CA, USA, with a major focus on compilation and runtime techniques for declarative, large-scale machine learning in Apache SystemML. Matthias received his Ph.D. from Dresden University of Technology, Germany in 2011 with a dissertation on cost-based optimization of integration flows.

Matteo Interlandi: Matteo Interlandi is a Principal Scientist in the Gray Systems Lab (GSL) at Microsoft, working at the intersection between Machine Learning and Database Systems. Before Microsoft, he was a Postdoctoral Scholar at the University of California, Los Angeles. Prior to joining UCLA, he was Research Associate at the Qatar Computing Research Institute and at the Institute for Human and Machine Cognition. Matteo received his Ph.D. from the University of Modena and Reggio Emilia. Matteo's work has received a best demo award at VLDB 2022, an honorable mention at SIGMOD 2021 and was featured in the "Best of VLDB 2016".

Christopher Jermaine: Chris Jermaine is a J.S. Abercrombie Professor of Engineering and Chair, of the CS department at Rice University. He studies data analytics: how to analyze, store, retrieve, and manipulate large and heterogeneous data sets. Within this problem space, most of his work focuses on: (1) The systems-oriented problems that arise when building software to manage and process large and diverse data sets, especially systems for machine learning; and (2) The difficulties that arise when applying statistical methods to such data sets. Chris received a BA from the Mathematics Department at UCSD, an MSc from the Computer Science and Engineering Department at OSU, and a PhD from the College of Computing at Georgia Tech. He is the recipient of a 2008 Alfred P. Sloan Foundation Research Fellowship, a National Science Foundation CAREER award, a 2007 ACM SIGMOD Best Paper Award, a 2009 ACM SIGKDD Best Paper Runner-Up, a 2017 ICDE Best Paper Award, and a 2019 VLDB Best Paper Runner-Up.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI*. 265–283.

[2] Shreya Agrawal, Luke Barrington, Carla Bromberg, John Burge, Cenk Gazen, and Jason Hickey. 2019. Machine Learning for Precipitation Now- casting from Radar Images. *CoRR* abs/1912.12132 (2019).

[3] Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, Felix Naumann, Mathias Peters, Astrid Rheinländer, Matthias J. Sax, Sebastian Schelter, Mareike Höger, Kostas Tzoumas, and Daniel Warneke. 2014. The Stratosphere platform for big data analytics. *VLDB J.* 23, 6 (2014), 939–964. https://doi.org/10.1007/s00778-014-0357-y

[4] Randy Allen and Ken Kennedy. 2001. *Optimizing Compilers for Modern Architectures: A Dependence-based Approach.* Morgan Kaufmann.

[5] Arash Ashari, Naser Sedaghati, John Eisenlohr, and P. Sadayappan. 2014. An efficient two-dimensional blocking strategy for sparse matrix-vector multiplication on GPUs. In *ICS*. 273–282. https://doi.org/10.1145/2597652.2597678

[6] Paul Barham, Aakanksha Chowdhery, Jeff Dean, Sanjay Ghemawat, Steven Hand, Dan Hurt, Michael Isard, Hyeontaek Lim, Ruoming Pang, Sudip Roy, Brennan Saeta, Parker Schuh, Ryan Sepassi, Laurent El Shafey, Chandramohan A. Thekkath, and Yonghui Wu. 2022. Pathways: Asynchronous Distributed Dataflow for ML. In *MLSys*. https://proceedings.mlsys.org/paper/2022/hash/

[7] Sebastian Baunsgaard and Matthias Boehm. 2023. AWARE: Workload-aware, Redundancy-exploiting Linear Algebra. In *SIGMOD*. https://doi.org/10.1145/3588682

[8] Sebastian Baunsgaard, Matthias Boehm, Ankit Chaudhary, Behrouz Derakhshan, Stefan Geißelsöder, Philipp M. Grulich, Michael Hildebrand, Kevin Innerebner, Volker Markl, Claus Neubauer, Sarah Osterburg, Olga Ovcharenko, Sergey Redyuk, Tobias Rieger, Alireza Rezaei Mahdiraji, Sebastian Benjamin Wrede, and Steffen Zeuch. 2021. ExDRa: Exploratory Data Science on Federated Raw Data. In *SIGMOD*. 2450–2463. https://doi.org/10.1145/3448016.3457549

[9] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *SIGKDD*. 1387–1395.

[10] Geoffrey Belter, Elizabeth R. Jessup, Ian Karlin, and Jeremy G. Siek. 2009. Automating the generation of composed linear algebra kernels. In *SC*. https://doi.org/10.1145/1654059.1654119

[11] Matthias Boehm, Iulian Antonov, Sebastian Baunsgaard, Mark Dokter, Robert Ginthör, Kevin Innerebner, Florijan Klezin, Stefanie N. Lindstaedt, Arnab Phani, Benjamin Rath, Berthold Reinwald, Shafaq Siddiqui, and Sebastian Benjamin Wrede. 2020. SystemDS: A Declarative Machine Learning System for the End-to-End Data Science Lifecycle. In *CIDR*.

[12] Matthias Boehm, Michael Dusenberry, Deron Eriksson, Alexandre V. Evfimievski, Faraz Makari Manshadi, Niketan Pansare, Berthold Reinwald, Frederick Reiss, Prithviraj Sen, Arvind Surve, and Shirish Tatikonda. 2016. SystemML: Declarative Machine Learning on Spark. *Proc. VLDB Endow.* 9, 13 (2016), 1425–1436. https://doi.org/10.14778/3007263.3007279

[13] Matthias Boehm, Arun Kumar, and Jun Yang. 2019. *Data Management in Machine Learning Systems.* Morgan & Claypool Publishers. https://doi.org/10.2200/S00895ED1V01Y201901DTM057

[14] Matthias Boehm, Berthold Reinwald, Dylan Hutchison, Prithviraj Sen, Alexandre V. Evfimievski, and Niketan Pansare. 2018. On Optimizing Operator Fusion Plans for Large-Scale Machine Learning in SystemML. *Proc. VLDB Endow.* 11, 12 (2018), 1755–1768. https://doi.org/10.14778/3229863.3229865

[15] Matthias Boehm, Shirish Tatikonda, Berthold Reinwald, Prithviraj Sen, Yuanyuan Tian, Douglas Burdick, and Shivakumar Vaithyanathan. 2014. Hybrid Parallelization Strategies for Large-Scale Machine Learning in SystemML. *Proc. VLDB Endow.* 7, 7 (2014), 553–564. https://doi.org/10.14778/2732286.2732292

[16] Matthias Böhm, Douglas R. Burdick, Alexandre V. Evfimievski, Berthold Reinwald, Frederick R. Reiss, Prithviraj Sen, Shirish Tatikonda, and Yuanyuan Tian. 2014. SystemML's Optimizer: Plan Generation for Large-Scale Machine Learning Programs. *IEEE Data Eng. Bull.* 37, 3 (2014), 52–62. http://sites.computer.org/debull/A14sept/p52.pdf

[17] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konecný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. In *MLSys*.

[18] Zhuhua Cai, Zografoula Vagena, Luis Leopoldo Perez, Subramanian Arumugam, Peter J. Haas, and Christopher M. Jermaine. 2013. Simulation of database-valued markov chains using SimSQL. In *SIGMOD*. 637–648. https://doi.org/10.1145/2463676.2465283

[19] José Cambronero, John K. Feser, Micah J. Smith, and Samuel Madden. 2017. Query Optimization for Dynamic Imputation. *Proc. VLDB Endow.* 10, 11 (2017),

1310–1321.

[20] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Q. Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *OSDI*. 578–594. https://www.usenix.org/conference/osdi18/presentation/chen

[21] Stephen Chou, Fredrik Kjolstad, and Saman P. Amarasinghe. 2018. Format abstraction for sparse tensor algebra compilers. *Proc. ACM Program. Lang.* 2, OOPSLA (2018), 123:1–123:30. https://doi.org/10.1145/3276493

[22] Edith Cohen. 1997. Size-Estimation Framework with Applications to Transitive Closure and Reachability. *J. Comput. Syst. Sci.* 55, 3 (1997), 441–453. https://doi.org/10.1006/jcss.1997.1534

[23] Edith Cohen. 1998. Structure Prediction and Computation of Sparse Matrix Products. *J. Comb. Optim.* 2, 4 (1998), 307–332.

[24] Keith D. Cooper and Linda Torczon. 2004. *Engineering a Compiler.* Morgan Kaufmann.

[25] Andrew Crotty, Alex Galakatos, Kayhan Dursun, Tim Kraska, Carsten Binnig, Ugur Çetintemel, and Stan Zdonik. 2015. An Architecture for Compiling UDF-centric Workflows. *Proc. VLDB Endow.* 8, 12 (2015), 1466–1477. https://doi.org/10.14778/2824032.2824045

[26] Ekin D. Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. 2019. AutoAugment: Learning Augmentation Strategies From Data. In *CVPR*. 113–123. https://doi.org/10.1109/CVPR.2019.00020

[27] William J. Dally. 2018. Hardware for Deep Learning. https://youtu.be/zDBF0xwQW-0 MLSys Keynote.

[28] Patrick Damme et al. 2022. DAPHNE: An Open and Extensible System Infrastructure for Integrated Data Analysis Pipelines. In *CIDR*. https://www.cidrdb.org/cidr2022/papers/p4-damme.pdf

[29] Tri Dao, Albert Gu, Alexander Ratner, Virginia Smith, Christopher De Sa, and Christopher Ré. 2019. A Kernel Theory of Modern Data Augmentation. In *ICML (Proceedings of Machine Learning Research, Vol. 97)*. 1528–1537. http://proceedings.mlr.press/v97/dao19b.html

[30] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large Scale Distributed Deep Networks. In *NeurIPS*. 1232–1240.

[31] Xin Luna Dong and Theodoros Rekatsinas. 2018. Data Integration and Machine Learning: A Natural Synergy. In *SIGMOD*. 1645–1650. https://doi.org/10.1145/3183713.3197387

[32] Joseph Vinish D'silva, Florestan De Moor, and Bettina Kemme. 2018. AIDA - Abstraction for Advanced In-Database Analytics. *Proc. VLDB Endow.* 11, 11 (2018), 1400–1413.

[33] Tarek Elgamal, Shangyu Luo, Matthias Boehm, Alexandre V. Evfimievski, Shirish Tatikonda, Berthold Reinwald, and Prithviraj Sen. 2017. SPOOF: Sum-Product Optimization and Operator Fusion for Large-Scale Machine Learning. In *CIDR*. http://cidrdb.org/cidr2017/papers/p3-elgamal-cidr17.pdf

[34] Ahmed Elgohary, Matthias Boehm, Peter J. Haas, Frederick R. Reiss, and Berthold Reinwald. 2018. Compressed linear algebra for large-scale machine learning. *VLDB J.* 27, 5 (2018), 719–744.

[35] Jingzhi Fang, Yanyan Shen, Yue Wang, and Lei Chen. 2020. Optimizing DNN Computation Graph using Graph Substitutions. *Proc. VLDB Endow.* 13, 11 (2020), 2734–2746. http://www.vldb.org/pvldb/vol13/p2734-fang.pdf

[36] Shaoduo Gan, Xiangru Lian, Rui Wang, Jianbin Chang, Chengjun Liu, Hongmei Shi, Shengzhuo Zhang, Xianghong Li, Tengxu Sun, Jiawei Jiang, Binhang Yuan, Sen Yang, Ji Liu, and Ce Zhang. 2021. BAGUA: Scaling up Distributed Learning with System Relaxations. *Proc. VLDB Endow.* 15, 4 (2021), 804–813. https://doi.org/10.14778/3503585.3503590

[37] Apurva Gandhi, Yuki Asada, Victor Fu, Advitya Gemawat, Lihao Zhang, Rathijit Sen, Carlo Curino, Jesús Camacho-Rodríguez, and Matteo Interlandi. 2022. The Tensor Data Platform: Towards an AI-centric Database System. *CoRR* abs/2211.02753 (2022). https://doi.org/10.48550/arXiv.2211.02753

[38] Apurva Gandhi, Yuki Asada, Victor Fu, Advitya Gemawat, Lihao Zhang, Rathijit Sen, Carlo Curino, Jesus Camacho-Rodriguez, and Matteo Interlandi. 2023. The Tensor Data Platform: Towards an AI-centric Database System. In *CIDR*. https://www.cidrdb.org/cidr2023/papers/p68-gandhi.pdf

[39] Zekai J. Gao, Shangyu Luo, Luis Leopoldo Perez, and Chris Jermaine. 2017. The BUDS Language for Distributed Bayesian Machine Learning. In *SIGMOD*. https://doi.org/10.1145/3035918.3035937

[40] Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis. 2011. Large-scale matrix factorization with distributed stochastic gradient descent. In *SIGKDD*. 69–77. https://doi.org/10.1145/2020408.2020426

[41] Google. 2019. Inside TensorFlow: tf.distribute.Strategy. https://www.youtube.com/watch?v=jKV53r9-H14

[42] Google. 2020. TensorFlow Federated: Machine Learning on Decentralized Data . https://www.tensorflow.org/federated

[43] Google. 2022. *DTensor Concepts.* https://www.tensorflow.org/guide/dtensor_overview

[44] Dong He, Supun Chathuranga Nakandala, Dalitso Banda, Rathijit Sen, Karla Saur, Kwanghyun Park, Carlo Curino, Jesús Camacho-Rodríguez, Konstantinos Karanasos, and Matteo Interlandi. 2022. Query Processing on Tensor Computation Runtimes. *Proc. VLDB Endow.* 15, 11 (2022), 2811–2825. https://www.vldb.org/pvldb/vol15/p2811-he.pdf

[45] Alireza Heidari, Joshua McGrath, Ihab F. Ilyas, and Theodoros Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *SIGMOD*. 829–846. https://doi.org/10.1145/3299869.3319888

[46] Matteo Hessel, Manuel Kroiss, Aidan Clark, Iurii Kemaev, John Quan, Thomas Keck, Fabio Viola, and Hado van Hasselt. 2021. Podracer architectures for scalable Reinforcement Learning. *CoRR* abs/2104.06272 (2021). https://arxiv.org/abs/2104.06272

[47] Pedro Holanda and Hannes Mühleisen. 2019. Relational Queries with a Tensor Processing Unit. In *DaMoN*. 19:1–19:3. https://doi.org/10.1145/3329785.3329932

[48] Yu-Ching Hu, Yuliang Li, and Hung-Wei Tseng. 2022. TCUDB: Accelerating Database with Tensor Processors. In *SIGMOD*. 1360–1374. https://doi.org/10.1145/3514221.3517869

[49] Botong Huang, Shivnath Babu, and Jun Yang. 2013. Cumulon: optimizing statistical data analysis in the cloud. In *SIGMOD*. 1–12. https://doi.org/10.1145/2463676.2465273

[50] Madelon Hulsebos, Kevin Zeng Hu, Michiel A. Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César A. Hidalgo. 2019. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. In *SIGKDD*. 1500–1508. https://doi.org/10.1145/3292500.3330993

[51] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher M. Jermaine, and Peter J. Haas. 2008. MCDB: a monte carlo approach to managing uncertain data. In *SIGMOD*. 687–700. https://doi.org/10.1145/1376616.1376686

[52] Dimitrije Jankov, Shangyu Luo, Binhang Yuan, Zhuhua Cai, Jia Zou, Chris Jermaine, and Zekai J. Gao. 2019. Declarative Recursive Computation on an RDBMS. *Proc. VLDB Endow.* 12, 7 (2019), 822–835.

[53] Dimitrije Jankov, Binhang Yuan, Shangyu Luo, and Chris Jermaine. 2021. Distributed Numerical and Machine Learning Computations via Two-Phase Execution of Aggregated Join Trees. *Proc. VLDB Endow.* 14, 7 (2021), 1228–1240. https://doi.org/10.14778/3450980.3450991

[54] Chris Jermaine. 2021. The Tensor-Relational Algebra, and Other Ideas in Machine Learning System Design. In *SSDBM*. 270. https://doi.org/10.1145/3468791.3472262

[55] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. 2019. TASO: optimizing deep learning computation with automatic generation of graph substitutions. In *SOSP*. 47–62. https://doi.org/10.1145/3341301.3359630

[56] Zhihao Jia, James Thomas, Todd Warszawski, Mingyu Gao, Matei Zaharia, and Alex Aiken. 2019. Optimizing DNN Computation with Relaxed Graph Substitutions. In *MLSys*. https://proceedings.mlsys.org/book/276.pdf

[57] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. 2017. Heterogeneity-aware Distributed Parameter Servers. In *SIGMOD*. 463–478.

[58] Jiawei Jiang, Shaoduo Gan, Yue Liu, Fanlin Wang, Gustavo Alonso, Ana Klimovic, Ankit Singla, Wentao Wu, and Ce Zhang. 2021. Towards Demystifying Serverless Machine Learning Training. In *SIGMOD*. 857–871. https://doi.org/10.1145/3448016.3459240

[59] Sian Jin, Chengming Zhang, Xintong Jiang, Yunhe Feng, Hui Guan, Guanpeng Li, Shuaiwen Song, and Dingwen Tao. 2021. COMET: A Novel Memory-Efficient Deep Learning Training Framework by Using Error-Bounded Lossy Compression. *Proc. VLDB Endow.* 15, 4 (2021), 886–899. https://doi.org/10.14778/3503585.3503597

[60] Steven G. Johnson. 2017. More Dots: Syntactic Loop Fusion in Julia. https://julialang.org/blog/2017/01/moredots/

[61] Norman P. Jouppi and otjers. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *ISCA*. 1–12. https://doi.org/10.1145/3079856.3080246

[62] Norman P. Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, James Laudon, Cliff Young, and David A. Patterson. 2020. A domain-specific supercomputer for training deep neural networks. *Commun. ACM* 63, 7 (2020), 67–78. https://doi.org/10.1145/3360307

[63] Julia. 2022. Parallel Computing. https://docs.julialang.org/en/v1/manual/parallel-computing/

[64] Peter Kairouz, Brendan McMahan, and Virginia Smith. 2020. Federated Learning Tutorial. In *NeurIPS*. https://slideslive.com/38935813/federated-learning-tutorial

[65] Vasileios Karakasis, Theodoros Gkountouvas, Kornilios Kourtis, Georgios I. Goumas, and Nectarios Koziris. 2013. An Extended Compression Format for the Optimization of Sparse Matrix-Vector Multiplication. *IEEE Trans. Parallel Distributed Syst.* 24, 10 (2013), 1930–1940. https://doi.org/10.1109/TPDS.2012.290

[66] Konstantinos Karanasos, Matteo Interlandi, Fotis Psallidas, Rathijit Sen, Kwanghyun Park, Ivan Popivanov, Doris Xin, Supun Nakandala, Subru Krishnan, Markus Weimer, Yuan Yu, Raghu Ramakrishnan, and Carlo Curino. 2020. Extending Relational Query Processing with ML Inference. In *CIDR*. http://cidrdb.org/cidr2020/papers/p24-karanasos-cidr20.pdf

[67] David Kernert, Frank Köhler, and Wolfgang Lehner. 2015. SpMacho - Optimizing Sparse Linear Algebra Expressions with Probabilistic Density Estimation. In

EDBT. 289–300.

[68] David Kernert, Wolfgang Lehner, and Frank Köhler. 2016. Topology-Aware Optimization of Big Sparse Matrices and Matrix Multiplications on Main-Memory Systems. In *ICDE*. 823–834.

[69] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. 2016. FAQ: Questions Asked Frequently. In *PODS*. 13–28. https://doi.org/10.1145/2902251.2902280

[70] Fredrik Kjolstad, Shoaib Kamil, Stephen Chou, David Lugato, and Saman P. Amarasinghe. 2017. The tensor algebra compiler. *Proc. ACM Program. Lang.* 1, OOPSLA (2017), 77:1–77:29. https://doi.org/10.1145/3133901

[71] Urs Köster, Tristan Webb, Xin Wang, Marcel Nassar, Arjun K. Bansal, William Constable, Oguz Elibol, Stewart Hall, Luke Hornof, Amir Khosrowshahi, Carey Kloss, Ruby J. Pai, and Naveen Rao. 2017. Flexpoint: An Adaptive Numerical Format for Efficient Training of Deep Neural Networks. In *NeurIPS*. 1742–1752. https://proceedings.neurips.cc/paper/2017/hash/a0160709701140704575d499c997b6ca-Abstract.html

[72] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NeurIPS*. 1106–1114. https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[73] Arun Kumar, Matthias Boehm, and Jun Yang. 2017. Data Management in Machine Learning: Challenges, Techniques, and Systems. In *SIGMOD*. 1717–1722. https://doi.org/10.1145/3035918.3054775

[74] Rasmus Munk Larsen and Tatiana Shpeisman. 2019. TensorFlow Graph Optimizations. https://web.stanford.edu/class/cs245/slides/TFGraphOptimizationsStanford.pdf Guest Lecture Stanford.

[75] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques A. Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *CGO*. IEEE, 2–14. https://doi.org/10.1109/CGO51591.2021.9370308

[76] Fengan Li, Lingjiao Chen, Yijing Zeng, Arun Kumar, Xi Wu, Jeffrey F. Naughton, and Jignesh M. Patel. 2019. Tuple-oriented Compression for Large-scale Mini-batch Stochastic Gradient Descent. In *SIGMOD*. 1517–1534.

[77] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *OSDI*. 583–598.

[78] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2021. CleanML: A Study for Evaluating the Impact of Data Cleaning on ML Classification Tasks. In *ICDE*. 13–24. https://doi.org/10.1109/ICDE51399.2021.00009

[79] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael I. Jordan, and Ion Stoica. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. In *ICML*, Vol. 80. 3059–3068. http://proceedings.mlr.press/v80/liang18b.html

[80] Weifeng Liu and Brian Vinter. 2014. An Efficient GPU General Sparse Matrix-Matrix Multiplication for Irregular Data. In *IPDPS*. 370–381.

[81] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *NeurIPS*. 4765–4774. https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html

[82] Shangyu Luo, Zekai J. Gao, Michael N. Gubanov, Luis Leopoldo Perez, and Christopher M. Jermaine. 2017. Scalable Linear Algebra on a Relational Database System. In *ICDE*. 523–534.

[83] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. 2017. Device Placement Optimization with Reinforcement Learning. In *ICML*, Vol. 70. 2430–2439.

[84] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *OSDI*. 561–577. https://www.usenix.org/conference/osdi18/presentation/nishihara

[85] Supun Nakandala, Arun Kumar, and Yannis Papakonstantinou. 2019. Incremental and Approximate Inference for Faster Occlusion-based Deep CNN Explanations. In *SIGMOD*. 1589–1606. https://doi.org/10.1145/3299869.3319874

[86] Supun Nakandala, Karla Saur, Gyeong-In Yu, Konstantinos Karanasos, Carlo Curino, Markus Weimer, and Matteo Interlandi. 2020. A Tensor Compiler for Unified Machine Learning Prediction Serving. In *OSDI*. 899–917. https://www.usenix.org/conference/osdi20/presentation/nakandala

[87] Supun Nakandala, Yuhao Zhang, and Arun Kumar. 2019. Cerebro: Efficient and Reproducible Model Selection on Deep Learning Systems. In *DEEM@SIGMOD Workshop*. 6:1–6:4. https://doi.org/10.1145/3329486.3329496

[88] Supun Nakandala, Yuhao Zhang, and Arun Kumar. 2020. Cerebro: A Data System for Optimized Deep Learning Model Selection. *Proc. VLDB Endow.* 13, 11 (2020), 2159–2173. http://www.vldb.org/pvldb/vol13/p2159-nakandala.pdf

[89] Felix Neutatz, Felix Biessmann, and Ziawasch Abedjan. 2021. Enforcing Constraints for Machine Learning Systems via Declarative Feature Selection: An Experimental Study. In *SIGMOD*. 1345–1358. https://doi.org/10.1145/3448016.3457295

[90] Milos Nikolic, Mohammed Elseidy, and Christoph Koch. 2014. LINVIEW: incremental view maintenance for complex analytical queries. In *SIGMOD*. 253–264.

[91] NVIDIA. 2020. A100 Tensor Core GPU Architecture.

[92] NVIDIA. 2022. TensorRT Developer Guide. https://docs.nvidia.com/deeplearning/tensorrt/pdf/TensorRT-Developer-Guide.pdf

[93] Kunle Olukotun. 2021. "Let the Data Flow!". In *CIDR*.

[94] Shoumik Palkar, James Thomas, Deepak Narayanan, Pratiksha Thaker, Rahul Palamuttam, Parimarjan Negi, Anil Shanbhag, Malte Schwarzkopf, Holger Pirk, Saman P. Amarasinghe, Samuel Madden, and Matei Zaharia. 2018. Evaluating End-to-End Optimization for Data Analytics Applications in Weld. *Proc. VLDB Endow.* 11, 9 (2018), 1002–1015. https://doi.org/10.14778/3213880.3213890

[95] Shoumik Palkar, James Thomas, Anil Shanbhag, Malte Schwarzkopf, Saman P. Amarasinghe, and Matei Zaharia. 2017. A Common Runtime for High Performance Data Analysis. In *CIDR*. http://cidrdb.org/cidr2017/papers/p127-palkar-cidr17.pdf

[96] Stavros Papadopoulos, Kushal Datta, Samuel Madden, and Timothy G. Mattson. 2016. The TileDB Array Data Storage Manager. *Proc. VLDB Endow.* 10, 4 (2016), 349–360. https://doi.org/10.14778/3025111.3025117

[97] Adam Paszke et al. 2019. PyTorch: An Imperative Style, High- Performance Deep Learning Library. In *NeurIPS*.

[98] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12 (2011), 2825–2830.

[99] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. 2021. Learning Mesh-Based Simulation with Graph Networks. *ICLR* (2021).

[100] Arnab Phani, Lukas Erlbacher, and Matthias Boehm. 2022. UPLIFT: Parallelization Strategies for Feature Transformations in Machine Learning Workloads. *Proc. VLDB Endow.* 15, 11 (2022), 2929–2938. https://www.vldb.org/pvldb/vol15/p2929-phani.pdf

[101] Alexander Renz-Wieland, Rainer Gemulla, Zoi Kaoudi, and Volker Markl. 2022. NuPS: A Parameter Server for Machine Learning with Non-Uniform Parameter Access. In *SIGMOD*. 481–495. https://doi.org/10.1145/3514221.3517860

[102] Alexander Renz-Wieland, Rainer Gemulla, Steffen Zeuch, and Volker Markl. 2020. Dynamic Parameter Allocation in Parameter Servers. *Proc. VLDB Endow.* 13, 11 (2020), 1877–1890. http://www.vldb.org/pvldb/vol13/p1877-renz-wieland.pdf

[103] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *SIGKDD*. 1135–1144. https://doi.org/10.1145/2939672.2939778

[104] Matthew Rocklin. 2015. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In *SCIPY*.

[105] Hongbo Rong, Jongsoo Park, Lingxiang Xiang, Todd A. Anderson, and Mikhail Smelyanskiy. 2016. Sparso: Context-driven Optimizations of Sparse Linear Algebra. In *PACT*. 247–259. https://doi.org/10.1145/2967938.2967943

[106] Svetlana Sagadeeva and Matthias Boehm. 2021. SliceLine: Fast, Linear-Algebra-based Slice Finding for ML Model Debugging. In *SIGMOD*. 2290–2299. https://doi.org/10.1145/3448016.3457323

[107] Ricardo Salazar, Felix Neutatz, and Ziawasch Abedjan. 2021. Automated Feature Engineering for Algorithmic Fairness. *Proc. VLDB Endow.* 14, 9 (2021), 1694–1702. https://doi.org/10.14778/3461535.3463474

[108] Sebastian Schelter. 2020. "Amnesia" - Machine Learning Models That Can Forget User Data Very Fast. In *CIDR*. http://cidrdb.org/cidr2020/papers/p32-schelter-cidr20.pdf

[109] Sebastian Schelter, Stefan Grafberger, and Ted Dunning. 2021. HedgeCut: Maintaining Randomised Trees for Low-Latency Machine Unlearning. In *SIGMOD*. 1545–1557. https://doi.org/10.1145/3448016.3457239

[110] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Bießmann, and Andreas Grafberger. 2018. Automating Large-Scale Data Quality Verification. *Proc. VLDB Endow.* 11, 12 (2018), 1781–1794. https://doi.org/10.14778/3229863.3229867

[111] Sebastian Schelter, Andrew Palumbo, Shannon Quinn, Suneel Marthi, and Andrew Musselman. 2016. Samsara: Declarative Machine Learning on Distributed Dataflow Systems.

[112] Maximilian E. Schüle, Tobias Götz, Alfons Kemper, and Thomas Neumann. 2022. ArrayQL Integration into Code-Generating Database Systems. In *EDBT*. https://doi.org/10.5441/002/edbt.2022.04

[113] Vraj Shah, Jonathan Lacanlale, Premanand Kumar, Kevin Yang, and Arun Kumar. 2021. Towards Benchmarking Feature Type Inference for AutoML Platforms. In *SIGMOD*. 1584–1596. https://doi.org/10.1145/3448016.3457274

[114] Gaurav Sharma and Jos Martin. 2009. MATLAB$^{®}$: A Language for Parallel Computing. *Int. J. Parallel Program.* 37, 1 (2009), 3–36. https://doi.org/10.1007/s10766-008-0082-5

[115] Alexander J. Smola and Shravan M. Narayanamurthy. 2010. An Architecture for Parallel Topic Models. *Proc. VLDB Endow.* 3, 1 (2010), 703–710.

[116] Johanna Sommer, Matthias Boehm, Alexandre V. Evfimievski, Berthold Reinwald, and Peter J. Haas. 2019. MNC: Structure-Exploiting Sparsity Estimation for Matrix Expressions. In *SIGMOD*. 1607–1623. https://doi.org/10.1145/3299869.3319854

[117] Leonhard F. Spiegelberg, Rahul Yesantharao, Malte Schwarzkopf, and Tim Kraska. 2021. Tuplex: Data Science in Python at Native Code Speed. In *SIGMOD*. 1718–1731. https://doi.org/10.1145/3448016.3457244

[118] Michael Stonebraker, Paul Brown, Alex Poliakov, and Suchi Raman. 2011. The Architecture of SciDB. In *SSDBM*. 1–16.

[119] Arvind K. Sujeeth, HyoukJoong Lee, Kevin J. Brown, Tiark Rompf, Hassan Chafi, Michael Wu, Anand R. Atreya, Martin Odersky, and Kunle Olukotun. 2011. OptiML: An Implicitly Parallel Domain-Specific Language for Machine Learning. In *ICML*. 609–616. https://icml.cc/2011/papers/373_icmlpaper.pdf

[120] Ki Hyun Tae and Steven Euijong Whang. 2021. Slice Tuner: A Selective Data Acquisition Framework for Accurate and Fair Machine Learning Models. In *SIGMOD*. 1771–1783. https://doi.org/10.1145/3448016.3452792

[121] Stef van Buuren and Karin Groothuis-Oudshoorn. 2011. mice: Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software, Articles* 45, 3 (2011), 1–67.

[122] Nicolas Vasilache, Oleksandr Zinenko, Theodoros Theodoridis, Priya Goyal, Zachary DeVito, William S. Moses, Sven Verdoolaege, Andrew Adams, and Albert Cohen. 2018. Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions. *CoRR* abs/1802.04730 (2018). http://arxiv.org/abs/1802.04730

[123] William N. Venables and Brian D. Ripley. 2002. *Modern Applied Statistics with S, 4th Ed.* Springer.

[124] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. 2018. Training Deep Neural Networks with 8-bit Floating Point Numbers. In *NeurIPS*. 7686–7695. https://proceedings.neurips.cc/paper/2018/hash/335d3d1cd7ef05ec77714a215134914c-Abstract.html

[125] Yisu Remy Wang, Shana Hutchison, Dan Suciu, Bill Howe, and Jonathan Leang. 2020. SPORES: Sum-Product Optimization via Relational Equality Saturation for Large Scale Linear Algebra. *Proc. VLDB Endow.* 13, 11 (2020), 1919–1932. http://www.vldb.org/pvldb/vol13/p1919-wang.pdf

[126] Da Yan, Yingyi Bu, Yuanyuan Tian, Amol Deshpande, and James Cheng. 2016. Big Graph Analytics Systems. In *SIGMOD*. 2241–2243. https://doi.org/10.1145/2882903.2912566

[127] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn J. Keogh. 2016. Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets. In *ICDM*. 1317–1322. https://doi.org/10.1109/ICDM.2016.0179

[128] Yongyang Yu, MingJie Tang, Walid G. Aref, Qutaibah M. Malluhi, Mostafa M. Abbas, and Mourad Ouzzani. 2017. In-Memory Distributed Matrix Computation Processing and Optimization. In *ICDE*.

[129] Binhang Yuan, Dimitrije Jankov, Jia Zou, Yuxin Tang, Daniel Bourgeois, and Chris Jermaine. 2021. Tensor Relational Algebra for Distributed Machine Learning System Design. *Proc. VLDB Endow.* 14, 8 (2021), 1338–1350. https://doi.org/10.14778/3457390.3457399

[130] Binhang Yuan, Cameron R. Wolfe, Chen Dun, Yuxin Tang, Anastasios Kyrillidis, and Chris Jermaine. 2022. Distributed Learning of Fully Connected Neural Networks using Independent Subnet Training. *Proc. VLDB Endow.* 15, 8 (2022), 1581–1590. https://www.vldb.org/pvldb/vol15/p1581-wolfe.pdf

[131] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *NSDI*. 15–28.

[132] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çagatay Demiralp, and Wang-Chiew Tan. 2020. Sato: Contextual Semantic Type Detection in Tables. *Proc. VLDB Endow.* 13, 11 (2020), 1835–1848. http://www.vldb.org/pvldb/vol13/p1835-zhang.pdf

[133] Hantian Zhang, Xu Chu, Abolfazl Asudeh, and Shamkant B. Navathe. 2021. OmniFair: A Declarative System for Model-Agnostic Group Fairness in Machine Learning. In *SIGMOD*. 2076–2088. https://doi.org/10.1145/3448016.3452787

[134] Hantian Zhang, Jerry Li, Kaan Kara, Dan Alistarh, Ji Liu, and Ce Zhang. 2017. ZipML: Training Linear Models with End-to-End Low Precision, and a Little Bit of Deep Learning. In *ICML*, Vol. 70. 4035–4043. http://proceedings.mlr.press/v70/zhang17e.html

[135] Mingxing Zhang, Yongwei Wu, Kang Chen, Teng Ma, and Weimin Zheng. 2016. Measuring and Optimizing Distributed Array Programs. *Proc. VLDB Endow.* 9, 12 (2016), 912–923. https://doi.org/10.14778/2994509.2994511

[136] Yi Zhang, Kamesh Munagala, and Jun Yang. 2011. Storing Matrices on Disk: Theory and Practice Revisited. *Proc. VLDB Endow.* 4, 11 (2011), 1075–1086. http://www.vldb.org/pvldb/vol4/p1075-zhang.pdf

[137] Jia Zou, R. Matthew Barnett, Tania Lorido-Botran, Shangyu Luo, Carlos Monroy, Sourav Sikdar, Kia Teymourian, Binhang Yuan, and Chris Jermaine. 2018. PlinyCompute: A Platform for High-Performance, Distributed, Data-Intensive Tool Development. In *SIGMOD*. https://doi.org/10.1145/3183713.3196933