Vision Paper

Serving Deep Learning Models from Relational Databases

Lixi Zhou[a], Qi Lin[a], Kanchan Chowdhury[a], Saif Masood[a],

Alexandre Eichenberger[c*], Hong Min[c*], Alexander Sim[e*], Jie Wang[b*],

Yida Wang[b*], Kesheng Wu[e*], Binhang Yuan[d*], Jia Zou[a]

[a] Arizona State University, [b] Amazon, [c] IBM T. J. Watson Research Center,

[d] Hong Kong University of Science and Technology, [e] Lawrence Berkeley National Lab

ABSTRACT

Serving deep learning (DL) models on relational data has become a critical requirement across diverse commercial and scientific domains, sparking growing interest recently. In this visionary paper, we embark on a comprehensive exploration of representative architectures to address the requirement. We highlight three pivotal paradigms: The state-of-the-art *DL-centric* architecture offloads DL computations to dedicated DL frameworks. The potential *UDF-centric* architecture encapsulates one or more tensor computations into User Defined Functions (UDFs) within the relational database management system (RDBMS). The potential *relation-centric* architecture aims to represent a large-scale tensor computation through relational operators. While each of these architectures demonstrates promise in specific use scenarios, we identify urgent requirements for seamless integration of these architectures and the middle ground in-between these architectures. We delve into the gaps that impede the integration and explore innovative strategies to close them. We present a pathway to establish a novel RDBMS for enabling a broad class of data-intensive DL inference applications.

1 INTRODUCTION

Recently, key applications emerged from the desire to nest SQL queries with deep learning inferences, including but not limited to the following categories:

• **Commercial applications** such as credit card fraud detection, personalized recommendation, customer service chatbots, and anti-money laundering (AML). These use cases increasingly rely on deep learning [13, 17, 27, 38, 51, 53, 75], and are latency-critical. In these cases, transaction data, order data, and customer profiles are usually managed by a relational database management system (RDBMS), and subject to SQL queries for updates and analytics.

• **Scientific applications** also benefit from the intriguing capabilities of integrating deep learning and RDBMS for efficient surrogate models and self-driving lab experiments[62]. For example, real-time control coupled with material modeling requires fast access to pre-computed results to trigger model inferences. [49, 62].

• **AI-enhanced autonomous RDBMS** increasingly uses Deep Learning (DL) [39, 40, 78, 82, 90]. In order to minimize the runtime overheads incurred by AI techniques, it is critical to reduce the latency of DL inferences for these techniques.

The DL models in these workloads range from simple models such as feed-forward neural networks (FFNN) and convolutional neural networks (CNN) to complex models such as Transformer [30] and recommendation models [53], as well as the foundation models [12], including large language models (LLMs) [59].

The state-of-the-art architecture for supporting inference queries, termed **the DL-centric architecture**, offloads the inference computations to decoupled DL runtimes, as illustrated in Fig. 1a. For example, Amazon Redshift offloads the inference to SageMaker, Microsoft Raven [37, 58] offloads the inference from Microsoft SQLServer to the ONNX runtime [22], Google BigQuery offloads the inference to TensorFlow [8], PostgresML [6] offloads the inference of language models to HuggingFace [74]. In addition, the emerging EvaDB [23] also runs the inference computations nested in SQL queries in decoupled ML systems such as Hugging Face, OpenAI, YOLO, Stable Diffusion, etc.

In addition, there are two potential architectures as alternatives, which have not gained enough attention yet.

• The potential **UDF-centric architecture** encapsulates inference logic as UDFs within RDBMS, as illustrated in Fig. 1b. However, existing systems such as VerticaML [24] and PostgresML [6] mostly use UDF-encapsulation for traditional ML such as XGBoost and logistic regression. There are few existing implementations of UDF-centric systems for DL, although DL offers diverse libraries on CPU/GPU [2, 3, 56] for implementing the UDFs.

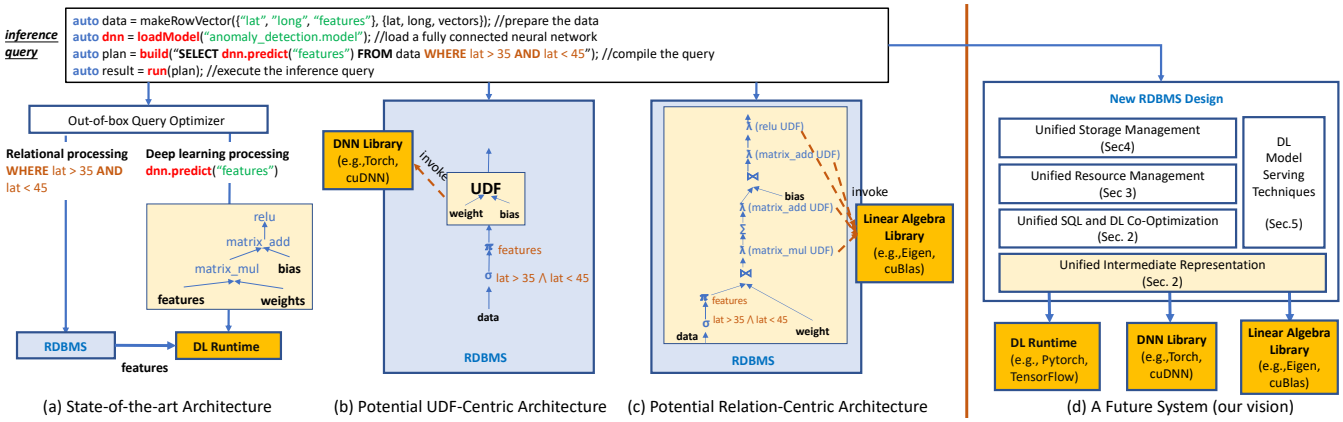
• The potential **relation-centric architecture**, as illustrated in Fig. 1c, represents a model parameter tensor as a relation, i.e., a collection of tensor blocks, and breaks down a tensor operator into multiple relational operators that nest with fine-grained UDFs. This approach has been advocated by SystemsML [11] and Tensor Relational Algebra [31, 47, 77] for scaling to massive tensor operations. However, this architecture hasn't been widely adopted for serving state-of-the-art DL models.

Why do we need a new architecture? In this vision paper, we argue for a new architecture because of the significant shortcomings that we observed in all existing architectures:

• The DL-centric architecture leads to significant cross-system overheads since features that were prepared by the data processing system need to be transferred to separate ML systems for inferences, which renders them inappropriate for latency-critical applications. In addition, the decoupled architecture prevents the inference computations and the relational query processing from being fully co-optimized. Moreover, handling large tensors often incurs out-of-memory (OOM) errors in resource-constraint environments. Finally, managing multiple systems incurs high operational costs.

• The UDF-centric architecture highly relies on the efficiency of the underlying libraries. It lacks the flexibility in optimizing

* These authors are ordered alphabetically; Jia Zou (jia.zou@asu.edu) is the corresponding author.



(a) State-of-the-art Architecture (b) Potential UDF-Centric Architecture (c) Potential Relation-Centric Architecture (d) A Future System (our vision)

Figure 1: Overview of the three pivotal architectures and our vision

and scheduling the operations within the UDF in a fine-grained style. As a result, it shares many problems with the DL-centric architecture, such as the difficulty in co-optimizing the relational processing and UDF processing and in handling large-scale tensors.

• The relation-centric architecture facilitates co-optimization of inference computations and relational processing by converting the former to the latter. In addition, it views a tensor as a collection of tensor blocks, which can resolve the OOM errors in resource-constrained CPU/GPU environments leveraging RDBMS's disk spilling and caching capabilities. However, not all DL operations can benefit from the conversion into relational processing. It significantly increases latency by converting small model inference computations that fit into the CPU/GPU caches from UDF processing into relational processing.

On one hand, each existing architecture has significant limitations to meet the latency and resource requirements of various applications. On the other hand, it is challenging for users to identify the optimal architectural design and it is also not ideal to develop a domain-specific solution for each type of application, which requires significant systems expertise and expensive development costs. Therefore, there exists an urgent requirement for a new architecture with a user-friendly interface and an efficient and intelligent kernel that is adaptive to a diverse array of applications.

Our Vision: In this work, we advocate for a novel RDBMS that seamlessly integrates the DL-centric, UDF-centric, and relation-centric architectures as well as various middle grounds in between these architectures. The envisioned system consists of the following novel components, as illustrated in Fig. 1:

• A novel unified intermediate representation (IR) to enable a novel query optimizer that not only dynamically selects one of the DL-centric, UDF-centric, and relation-centric representations for each operator, but also co-optimizes the SQL processing and the model inferences by designing novel transformation rules (Sec. 2).

• A novel unified resource management framework for tuning the threading and memory allocation for DB operations, DL runtime operations, DNN library operations, and linear algebra operations, and dispatching these operations to devices (Sec. 3).

• A novel storage co-optimization framework for tensor blocks and relational data that facilitates new techniques, such as the novel accuracy-aware data/model deduplication and the novel physics database design that considers data/model co-partitioning (Sec. 4).

• Model serving techniques newly adapted for RDBMS, such as using RDBMS indexing to cache inference results in an application-aware style (Sec. 5).

Expected Benefits. The proposed novel RDBMS design has numerous advantages over the state-of-the-art architecture and other potential architectures. First, it enhances the productivity of developing applications that desire DB functionalities such as SQL, transaction management, and access control and shortens the time to market. Second, it delivers superior performance for a broad class of inference applications at different scales through unified optimization for computation, resource management, and data storage. Third, it effectively avoids the cross-system overheads for applications bottlenecked by data transfer. We discussed more benefits such as extensions to training in Sec. 6. We further validates some of the ideas in Sec. 7.

2 QUERY COMPILATION AND OPTIMIZATION

Challenge 1. How to design a query compilation framework to dynamically generate the DL-centric, UDF-centric, and relation-centric representations for each operator and facilitate co-optimization of relational processing and DL inference computations?

(1) Unified intermediate representations(IR). We consider the following options for designing a unified IR to address the challenge.

• Multi-level IR. Multi-Level Intermediate Representation (MLIR) [42, 43] is developed by Google and LLVM for integrating multiple levels of IR dialects for machine learning processing. Recently, LingoDB [36] and Daphne IR [20] provide an MLIR dialect for relational processing, which can serve as a nice foundation for integrating with existing dialects for AI/ML computations developed by the MLIR community to facilitate cross-optimizations within the MLIR framework. However, a challenging problem is how to co-optimize the DB dialect and ML dialects, and how to glue MLIR and RDBMS runtime for parallel query execution, caching, and indexing [54, 55]. These are not discussed in existing works.

• **A Novel Adaptive In-database IR**. An alternative is to extend existing relational algebra IR to represent linear algebra computations as UDFs or relational operators adaptively. This approach may seamlessly integrate with the RDBMS runtime. For example, we consider an IR, where each node is a traditional relational operator or a model UDF operator. The latter corresponds to a DL model inference computation. Such computation can be lowered

to a graph IR, where each node represents a linear algebra operator such as matrix multiplication, matrix addition, relu, softmax, conv2d, etc.

Both options can support flexible IR transformations required by the envisioned system. First, each linear algebra operator can be flexibly converted into relational operators when a tensor exceeds available memory. For example, a large-scale matrix multiplication can be replaced by a join followed by an aggregation as illustrated in Fig. 1c [77]. Second, one or more operators can be offloaded to external DL runtimes, e.g., for pipelining across multiple GPU devices. Third, multiple simple linear algebra operators can be merged into one coarse-grained UDF by invoking libraries, such as CuDNN [56], and IBM zDNN [3], or using code generation [15, 60, 71], to reduce data transfer latency.

(2) Co-Optimization. A unified IR will open opportunities for co-optimizing SQL processing and DL models, such as novel query transformation rules. For example, consider a pipeline that first joins two separate wide feature datasets D_1 and D_2 to obtain the entire features and run a FFNN model on top of the features. If the first layer of the model is an embedding layer or a fully connected layer that will reduce the feature dimensions significantly, we may decompose the corresponding weight matrix W into two submatrices W_1 and W_2, each of which only processes the features belonging to D_1 or D_2 respectively, so that $W \times (D_1 \bowtie D_2) = (W_1 \times D_1) \bowtie (W_2 \times D_2)$. Then, a novel query transformation rule is to push down the submatrix multiplication $W_1 \times D_1$ and $W_2 \times D_2$ before we perform the join. It will reduce the intermediate data size if the number of features in the outputs from $W_1 \times D_1$ and $W_2 \times D_2$ are significantly smaller than the total number of features in D_1 and D_2.

Considering the tremendous flexibility, optimizing for complicated inference queries could be expensive. In addition to employing standard techniques such as learning-based optimization [39], Bayesian optimization [26], and query graph partitioning [31], a promising technique is ahead-of-time (AoT) compilation [44, 80, 81]. For example, when loading a model into RDBMS, the system will generate multiple execution plans at compilation time and select the best plan at runtime.

3 UNIFIED RESOURCE MANAGEMENT

Challenge 2. How to unify the resource management for RDBMS, DL runtimes, and DL libraries invoked from UDFs?

(1) Hyper-Parameter Tuning. Tuning parameters is important in coordinating resources between the RDBMS runtime and external DL runtimes colocating on the same machine. For example, while configuring the RDBMS buffer pool sizes, we shall also consider the memory requirements of the DL runtimes. Similarly, in the UDF-centric approach, the UDF, e.g., invoking OpenBLAS libraries, may rely on OpenMP for parallelism. We must carefully configure the number of threads for the SQL query processing and OpenMP. Otherwise, significant context switch overheads may occur. For example, multiple RDBMS threads execute the same pipeline stage that contains linear algebra operators in data-parallel style. However, each linear algebra operator may run with a different number of OpenMP threads.

Existing RDBMS and ML hyper-parameter tuning works [14, 45, 46, 48, 69, 79, 86] did not consider the heterogeneous nature of these threading configurations of UDF-based linear algebra operators. The heterogeneity in the tasks executed by the RDBMS threads and ML threads made the formulation of the hyper-parameter co-optimization problem challenging. In addition, each ML thread may have a different lifetime, which further complicates the problem. In addition, hyper-parameter search usually requires either significant search latency at the online stage or significant offline overheads for training surrogate models, reinforcement learning, etc. Motivated by these challenges, it is promising to explore novel data-efficient hyper-parameter learning techniques, such as meta-learning, zero-shot, and few-shot learning with generative models [52]. In addition, it may retrieve heuristic or historical knowledge as contexts through nearest-neighbor indexing to augment the learning process.

(2) Device Allocation. Whether DL inference computation may benefit from hardware accelerators such as GPU depends on many factors. According to our study of decision forest inference [28], we found that for inference queries that involve simple models and small datasets, the overheads of moving data from host memory to GPU memory could outweigh the performance benefits brought by GPU acceleration. The resource management of the envisioned system should intelligently allocate devices like CPU and GPU to different inference queries.

To resolve the problem, one option is to extend the physical query optimizer to model the running of each UDF that encapsulates DL operator(s) as a producer-transfer-consumer process [67] and estimate the overall latency accordingly depending on how the CPU-GPU data transfers are overlapped with CPU/GPU processing. Such an approach will be novel because existing UDF optimization works [19, 57, 90] focus on logical optimization.

4 STORAGE CO-OPTIMIZATION

Challenge 3. How to manage the storage and caching of relational, tensor, and vector data in a unified way?

The inference queries need to manipulate both relational data and tensor/vector data and we focus on the following opportunities.

(1) Accuracy-Aware Deduplication and Compression. Different from relational data that must be stored exactly, some errors/approximations in the tensor/vector data may not significantly affect the downstream application accuracy. In addition, similar tensor/vector data may be deduplicated approximately to reduce storage costs and memory footprint [63, 83].

Therefore, in RDBMS, the caching of model data should facilitate error-bounded compression and deduplication. For example, the weights that are approximately shared by multiple tensors or the embedding vectors that are frequently accessed or are insignificant to the inference results should be prioritized in the caching hierarchy.

Moreover, the RDBMS should facilitate accuracy-aware query optimization. For example, the storage optimizer may automatically employ compression, such as pruning and quantization, to create multiple versions of the same model with different size, efficiency, and accuracy trade-offs. Then, the query optimizer may dynamically select one version of the model for runtime query processing based on the latency and accuracy requirements defined in the Service Level Agreement (SLA). In addition, managing DL models in RDBMS will facilitate the binding of each model and its training datasets (e.g., through the catalog component), which will facilitate model selection based on data similarity [84].

(2) Novel Physical Database Design. Physical database design [9, 61] in RDBMS determines the partitioning of data and selection of indexing and materialized view. In our envisioned

system, this component should be refactored to consolidate existing tensor partitioning, distributing, and offloading in existing DL libraries [15, 66, 76] with the RDBMS storage management for tensor relations (i.e., as in the relation-centric approach). For example, the automatic tiling of the tensors in TVM [15] and the distribution of the tensor blocks to multiple devices in DISTAL [76] could be merged with the physical database design.

In addition, it is promising to co-partition relational data and tensor data to facilitate relation-centric processing. For example, the first linear algebra computation in a feed-forward neural network is to multiply the feature tensor and the weight tensor, which can be converted into a join of the feature tensor and the weight tensor followed by an aggregation [32, 33, 83, 88]. If the feature tensor is the output of relational processing on relational data, co-partitioning the relational data and the weight tensor will avoid the shuffling process to conduct a distributed join, as we demonstrated in our prior work [90].

5 DL SERVING TECHNIQUES IN RDBMS

Challenge 1. Which techniques are compliant with and should be incorporated into RDBMS to accelerate the Relation-Centric and UDF-Centric approaches, and how to incorporate these techniques?

(1) Caching of Inference Results and Contexts. Although there exist solutions that cache inference results [18, 25, 41], none of these solutions consider the integration with RDBMS. One option is to leverage RDBMS indexing to facilitate an inference result cache. The idea is to create a table that records the features or intermediate representations (e.g., embedding vectors) of frequent inference requests and the corresponding prediction results or contexts that are auxiliary to the prediction. We then construct nearest neighbor indexing over the features so that an inference query may efficiently retrieve results or contexts from the cache.

It is promising to leverage nearest neighbor indexing widely used in vector databases [4, 5, 7, 35, 72], such as hierarchical navigable small world (HNSW) [50], locality sensitive hashing (LSH) [87], inverted file indexing (IVF) [68], and product quantization [34], within RDBMS to facilitate the fast retrieval of inference results.

Such integration also triggers many research opportunities. First, the approximate caching will make a trade-off between the accuracy and latency, and may not benefit many accuracy-critical applications. Therefore, the proposed caching mechanism and the query optimizer should be application-aware, and should dynamically recommend whether to cache data for a specific application and whether to use the cache for a specific query based on the service level agreements (SLAs) and user configurations. Second, it is important to derive error bounds for the inference result caching. One option is considering a probabilistic error bound based on Monte Carlo sampling [64]. Another option is to use the exact inference result caching leveraging the hashing indexing or multi-dimensional indexing. Third, the buffer pool page replacement policy also needs to be improved to coordinate the disparate access patterns of the vector data, the relational data, and various indexes.

(2). Pipelining of DL Computations. In DL serving systems, if a model exceeds available memory, it will be partitioned into multiple operators/layers, which will be dispatched to multiple devices based on the costs of the operators and the available resources of the devices [10, 16, 65]. Those devices work in parallel, composing a pipeline. A pipeline stage at each device works in a streaming style. It continuously accepts inputs from its upstreaming stage, processes the inputs, and passes the output to the downstream stage.

However, although the "pipelining" concepts in RDBMS also exist, it mostly refers to operator fusion, and the query processing in RDBMS mainly relies on data parallelism.

These two approaches represent different trade-offs. The pipelining in DL frameworks is subject to the operator size limitation, where an operator or a layer must fit the resource on the corresponding device. The Relation-Centric processing in RDBMS has no such restriction because a large-scale tensor is represented as a collection of blocks that can spill to disks or be distributed to multiple devices. However, the pipelining in DL frameworks tends to be more efficient than the RDBMS parallel processing. First, there is no need to shuffle data globally as required by database join and aggregation. Second, computations can be reused across queries without re-compilation/re-scheduling overheads.

Implementing the DL pipelining mechanism in the UDF-centric approach is feasible by breaking the model UDF into multiple fine-grained operator UDFs and deploying those UDFs on different devices following the stream processing paradigm. However, it is challenging to accommodate both batch and stream processing in one system [21]. Another option is to offload large-scale models to DL frameworks, provided that each operator fits memory and the data transfer between the RDBMS and the DL framework is insignificant.

6 FURTHER DISCUSSION

6.1 Extension to Deep Learning Training

DL training also consists of linear algebra operations and may also benefit from the optimization techniques we proposed. While we focus on real-time deep learning inference queries, an interesting question is whether it is feasible to extend a deep learning inference system for the corresponding training job within the same infrastructure. For the DL-centric architecture, the extension is straightforward — the underlying DL system (e.g. PyTorch) is equipped with the automatic differentiation engine that can construct the backward propagation computation graph automatically, and then the DL system can execute it for the SGD-based training computation on the hardware runtime. For the UDF-centric architecture, the extension relies on the implementation of the UDF that should be able to integrate the functionality of the corresponding backward computation and the SGD-based optimizers. For the relational-centric architecture, how the extension should be designed and implemented is still an open question — one potential solution is to implement the corresponding backward computation as a set of separated fine-grained UDFs corresponding to each of the fine-grained UDFs representing the forward/inference computations, and then leverage a relational view to construct the query execution plan to schedule the computation over the execution environment [70].

6.2 From A Monolithic System to A Loosely Coupled Ecosystem

We argue for a system to unify the different types of DL processing that lies in-between a monolithic system and a loosely coupled ecosystem to integrate all representations for diverse

workloads. Most processing, including relational-centric, UDF-centric, and even simple DL-centric processing, could be seamlessly integrated with the RDBMS. The system will only offload complicated ML/DL processing that bottlenecks the pipeline processing with a time constraint to existing ML/DL systems. Compared to a loosely coupled ecosystem [23, 58], where an out-of-box optimizer analyzes user code and dispatches computations to RDBMS, vector databases, and ML systems, our envisioned system can achieve better end-to-end performance for a broad class of workloads by avoiding cross-system overheads and maximizing co-optimization capabilities. Compared to a monolithic system, our envisioned system avoids reinventing the wheels for DL and tensor computation optimization, reduces the implementation cost by leveraging existing investments in DL systems, and achieves better encapsulation and transparency between RDBMS and ML systems.

6.3 Limitations

Potential drawbacks of the proposed system include the limitation of the size and complexity of the models that can be natively supported by the RDBMS with a significant performance advantage. More complicated models, such as large language models (LLMs), may achieve better training and inference latency in specialized systems. However, RDBMS can reduce memory costs and provide better security and privacy protections, regardless of the model size and complexity. Such unique features could be attractive for many applications, such as those hosted by small businesses. In addition, although it will be challenging for the envisioned system to provide native LLM support, it can serve as a high-performance retrieving engine by allowing efficient inference queries to retrieve data for augmenting LLM inferences [29, 85].

7 VALIDATION AND EVIDENCE

A path to implement the envisioned problem should involve two steps. First, we will enhance existing RDBMS to unify the three representations. Second, we will enhance the unified system to implement proposed techniques. We validate each step in the following sections.

7.1 Unified Inference Query Processing

To validate the potential benefits of the proposed unified query optimization strategy, we developed a naive rule-based inference query optimizer, which adaptively selects the in-database representation for each operator based on the required memory size of the operator. If the operator's memory requirement exceeds a configurable memory limit threshold, it will choose the relation-centric representation, otherwise, it will choose the UDF-centric representation. An operator's memory requirement is estimated as the sum of the operator's input size and the output size (e.g., for a matrix multiplication operator, if the input tensors have shapes $m \times k$ and $k \times n$, the memory requirement is simply estimated as $m \times k + k \times n + m \times n$). We prototyped the simple optimizer on top of netsDB [83] which is our object-oriented RDBMS implemented in C++ [89–92], to leverage its capability in representing a model in analyzable UDFs [90]. Such capability enables our optimizer to traverse through each operator and estimate its memory requirement.

We loaded multiple feed-forward neural network (FFNN) models and convolutional neural network (CNN) models into netsDB, using the optimizer to select the in-database representation. The detailed descriptions of these models are listed in Tab. 1 and Tab. 2.

Table 1: Fully Connected (FC) Models (one hidden layer)

Model	Number of Features/Number of Neurons (hidden layer size)/Outputs
Fraud-FC-256	28/256/2
Fraud-FC-512	28/512/2
Encoder-FC	76/3,072/768
Amazon-14k-FC	597,540/1,024/14,588

Table 2: Convolutional Models (Stride size = 1 and Padding size = 0)

Model	Image/Input Shape	Kernel Shape
DeepBench-CONV1	$112 \times 112 \times 64$	$64 \times 64 \times 1 \times 1$
LandCover	$2500 \times 2500 \times 3$	$2048 \times 3 \times 1 \times 1$

We compared the in-database processing of these models to popular ML systems including TensorFlow 2.5 and Pytorch 2.1.0. All of the systems run in an AWS r4.2xlarge instance, which has eight CPU cores and 61 gigabytes memory, and 500 GB SSD. For our prototype, the samples to be inferred are loaded to netsDB before the testing. For TensorFlow and Pytorch, the samples are loaded from PostgreSQL using a high-performance connector called ConnectorX [73]. In the baselines, all hyper-parameters, such as the batch size, are fine-tuned to be optimal. In all experiments, we set the memory threshold to 2 gigabytes, so that if the operator's memory requirements exceed this size, the relation-centric representation will be used for the operator.

Latency Reduction for Small-Scale Models For small-scale models that fit into the memory threshold, our optimizer chose the UDF-centric representation that encapsulates all model inference operations in a single UDF. As illustrated in Fig. 2 and Fig. 3, the proposed architecture is able to reduce the latency for inference queries that involve small-scale models.

Because the model inference complexity is low in these workloads, the cross-system data transfer becomes the bottleneck of the DL-centric architecture. However, the proposed in-database model serving architecture can effectively alleviate such bottlenecks by running the inference computations directly on top of the data.

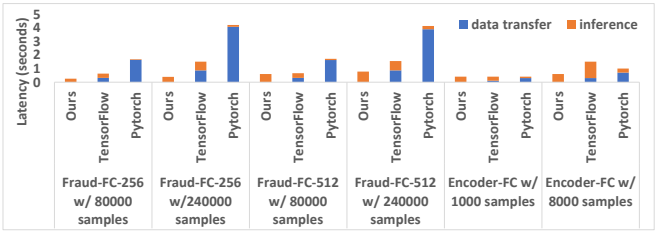


Figure 2: Latency reduction using our rule-based optimizer for FFNN model inference over data managed by RDBMS.

OOM Error Avoidance for Large-Scale Models For large-scale ML operators in which the memory requirements exceed the memory limit threshold, our optimizer chose the relation-centric representation. For example, in the AWS-14K-FC model, the tensor of the input features X has a shape of $batch\_size \times 597,540$, and the weight matrix that connects the input features and the hidden layer W has a shape of $1,024 \times 597,540$. Simply the weight matrix exceeds the 2 gigabytes threshold, therefore, the operator of $X \times W^T$ would be represented in relation-centric. First, the weight matrix will be chunked into matrix blocks, and then the

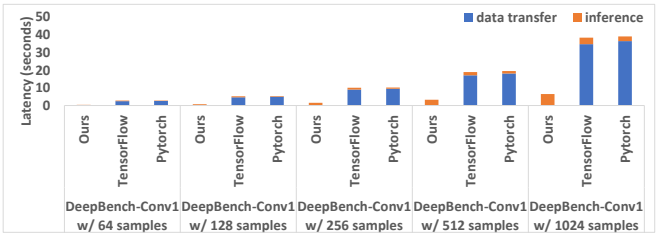


Figure 3: Latency reduction using our rule-based optimizer for CNN model inference over data managed by RDBMS.

matrix multiplication will be converted into a join followed by an aggregation. Similarly, for the convolutional operation in LandCover, the input tensor has a shape of $batch\_size \times 2500 \times 2500 \times 3$, and the output feature map has a shape of $batch\_size \times 2500 \times 2500 \times 2048$. The convolution operation will also be converted into relational operators. First, following the spatial rewriting algorithm for convolutional computation [1], each image will be flattened into a matrix F of the shape $6,250,000 \times (3+1)$, and the kernel will be flattened into a matrix K of the shape $2048 \times (3+1)$. Second, the convolution computation will be converted into $F \times K^T$, which will also be represented as a join followed by an aggregation.

As illustrated in Tab. 3, while the large-scale ML operator does not fit into the available memory, DL-centric architectures will throw Out-of-Memory (OOM) errors. However, the relation-centric inference processing running in the RDBMS will run at the block level, which effectively reduces the memory footprint from the scale of the tensors to the scale of the blocks. At the same time, although the entire collection of blocks cannot fit into the memory, a portion of the blocks are spilled to disk and loaded into the buffer pool (set to 20 gigabytes) when needed. Leveraging the RDBMS buffer pool management capability, the relation-centric representation can work with resource-constrained devices. In Tab. 3, we also noticed that when the ML operators of the model fit into the available memory, the DL-centric implementation relying on TensorFlow and Pytorch has a performance advantage. That's because the inference computation rather than the cross-system data transfer becomes the bottleneck in such DL-centric architecture, while our relation-centric implementation involves the additional overheads of chunking model parameter tensors into tensor blocks.

Table 3: Latency comparison for large-scale model inferences over data managed by RDBMS

Model	Batch Size	Ours	UDF centric	TensorFlow	Pytorch
Amazon-14k-FC	1000	58.6	60.4	34.6	**22.6**
	8000	**407.2**	OOM	OOM	OOM
LandCover	1	36.8	OOM	**9.9**	OOM
	2	**45.2**	OOM	OOM	OOM

7.2 Other Representative Techniques

7.2.1 Model Decomposition and Push-Down. We vertically partition the Bosch dataset [28] that has 1.18 millions of tuples with 968 features into two datasets D_1 and D_2, each having 484 features. We consider an inference pipeline that first joins D_1 and D_2 into an augmented feature dataset, denoted as D, through a similarity join based on the similarity of values in two columns from D_1 and D_2 respectively. These two columns are selected as having the highest correlation among pairs of columns from $D1$ and $D2$. We then run FFNN model over the augmented features. The model has a hidden layer of 256 neurons and an output layer of 2 neurons. (Therefore, the weight matrix at the first layer W has a shape of 256×968.) The results showed that if we decompose the matrix multiplication of $D \times W^T$ into $D_1 \times W_1^T \bowtie D_2 \times W_2^T$ as described in Sec. 2, it will achieve 5.7× speedup.

7.2.2 Caching of Inference Results. In our preliminary experiments, we've found using Faiss' HNSW as indexing to cache the inference result, we can accelerate the inference latency for a simple CNN model with two convolutional layers (the first layer has 32 3×3 kernels, and the second layer has 16 3×3 kernels) and two fully connected layers with 64 and 10 neurons respectively, by 10.3× speedup, with a significant accuracy reduction from 98.75% to 93.65%. We also evaluated this method using a simple FFNN model that has four fully connected layers with 128, 1024, 2048, and 64 neurons respectively, on the MNIST dataset, and observed a speedup by 7.3× with accuracy dropped from 97.74% to 95.26%. These results motivate an adaptive inference result caching strategy which decides whether to cache the inference results by estimating a probabilistic error bound using techniques such as Monte Carlo sampling and checking whether the error bound is acceptable to the application's SLA.

8 CONCLUSIONS

In this work, we discuss how the RDBMS should be upgraded to support various model-serving applications that rely on RDBMS for data management. First, we identify three pivotal architectures: DL-centric, UDF-centric, and relation-centric, and we argue that these approaches as well as the middle ground among them should be unified to facilitate the serving of deep learning models on relational data at different scales. We further identify challenges and opportunities in existing RDBMS systems for achieving unified optimization. In summary, we argue for:

- A novel unified IR and novel adaptive optimizer for co-optimizing inference computations and relational computations at a fine-grained style so that any subgraph in the inference query IR has the flexibility to be scheduled as DL-centric, UDF-centric, or relation-centric. Such an optimizer will open opportunities for identifying new query graph transformation rules, such as model decomposition and push-down.
- Coordination of computational and memory resources for DB, DL runtimes, and lower-level runtimes that support DL libraries to unify the resource management for the future RDBMS ecosystem.
- Renovating the storage, caching, and indexing of tensors/vectors to reduce storage costs, considering data placement in GPU, and facilitating accuracy-aware query optimization.
- Integrating compliant model serving techniques such as caching of inference results into RDBMS and offloading non-compliant techniques to DL runtimes.

ACKNOWLEDGMENTS

The work is sponsored by the National Science Foundation (NSF) CAREER Award (Number 2144923), the IBM Academic Research Award, the Amazon Research Award, and a U.S. Department of Homeland Security (DHS) Award (Number 17STQAC00001-03-03). Kanchan Chowdhury's work is supported by Prof. Mohamed Sarwat.

REFERENCES

[1] [n. d.]. Optimizing TensorFlow Convolution Performance on Aarch64. ([n. d.]). https://www.linaro.org/blog/optimizing-tensorflow-convolution-performance-on-aarch64/
[2] 2020. Eigen. http://eigen.tuxfamily.org/index.php?title=Main_Page

[3] 2022. IBM Z Deep Neural Network Library (zDNN). https://github.com/IBM/zDNN.

[4] 2022. Qdrant. https://qdrant.tech/.

[5] 2023. Pinecone: Vector Database for Vector Search. https://www.pinecone.io/.

[6] 2023. postgresML. https://postgresml.org/docs/guides/transformers/setup.

[7] 2023. Vespa: The big data serving engine. https://vespa.ai/.

[8] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[9] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R Narasayya. 2000. Automated selection of materialized views and indexes in SQL databases. In *VLDB*, Vol. 2000. 496–505.

[10] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 2022. DeepSpeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[11] Matthias Boehm, Michael W Dusenberry, Deron Eriksson, Alexandre V Evfimievski, Faraz Makari Manshadi, Niketan Pansare, Berthold Reinwald, Frederick R Reiss, Prithviraj Sen, Arvind C Surve, et al. 2016. Systemml: Declarative machine learning on spark. *Proceedings of the VLDB Endowment* 9, 13 (2016), 1425–1436.

[12] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* (2021).

[13] Stuart Breslow, Mikael Hagstroem, Daniel Mikkelsen, and Kate Robu. 2017. The new frontier in anti–money laundering. *McKinsey & Company, November* (2017).

[14] Matthew Butrovich, Wan Shen Lim, Lin Ma, John Rollinson, William Zhang, Yu Xia, and Andrew Pavlo. 2022. Tastes Great! Less Filling! High Performance and Accurate Training Data Collection for Self-Driving Database Management Systems. ACM| Proceedings of the 2022 International Conference on Management of Data.

[15] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. {TVM}: An automated end-to-end optimizing compiler for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 578–594.

[16] Zhi Chen, Cody Hao Yu, Trevor Morris, Jorn Tuyls, Yi-Hsiang Lai, Jared Roesch, Elliott Delaye, Vin Sharma, and Yida Wang. 2021. Bring your own codegen to deep learning compiler. *arXiv preprint arXiv:2105.03215* (2021).

[17] Dawei Cheng, Sheng Xiang, Chencheng Shang, Yiyi Zhang, Fangzhou Yang, and Liqing Zhang. 2020. Spatio-temporal attention-based neural network for credit card fraud detection. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 362–369.

[18] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. 2017. Clipper: A low-latency online prediction serving system. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 613–627.

[19] Andrew Crotty, Alex Galakatos, Kayhan Dursun, Tim Kraska, Ugur Cetintemel, and Stanley B Zdonik. 2015. Tupleware:" Big" Data, Big Analytics, Small Clusters.. In *CIDR*.

[20] Patrick Damme, Marius Birkenbach, Constantinos Bitsakos, Matthias Boehm, Philippe Bonnet, Florina Ciorba, Mark Dokter, Pawl Dowgiallo, Ahmed Eleliemy, Christian Faerber, et al. 2022. DAPHNE: An Open and Extensible System Infrastructure for Integrated Data Analysis Pipelines. In *Conference on Innovative Data Systems Research*.

[21] Tathagata Das, Yuan Zhong, Ion Stoica, and Scott Shenker. 2014. Adaptive stream processing using dynamic batch sizing. In *Proceedings of the ACM Symposium on Cloud Computing*. 1–13.

[22] ONNX Runtime developers. 2021. Open neural network exchange format (ONNX). https://github.com/onnx/onnx

[23] EvaDB. 2023. *EvaDB: Database system for AI-powered apps.* Technical Report. https://github.com/georgia-tech-db/evadb

[24] Arash Fard, Anh Le, George Larionov, Waqas Dhillon, and Chuck Bear. 2020. Vertica-ml: Distributed machine learning in vertica database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 755–768.

[25] Alessandro Finamore, James Roberts, Massimo Gallo, and Dario Rossi. 2022. Accelerating deep learning classification with error-controlled approximate-key caching. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2118–2127.

[26] Peter I Frazier. 2018. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811* (2018).

[27] Jianfeng Gao, Michel Galley, and Lihong Li. 2018. Neural approaches to conversational AI. In *The 41st international ACM SIGIR conference on research & development in information retrieval*. 1371–1374.

[28] Hong Guan, Mahidhar Reddy Dwarampudi, Venkatesh Gunda, Hong Min, Lei Yu, and Jia Zou. 2023. A Comparison of Decision Forest Inference Platforms from A Database Perspective. *arXiv preprint arXiv:2302.04430* (2023).

[29] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*. PMLR, 3929–3938.

[30] S. Huang and et al. 2010. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *ICDEW*. 41–51.

[31] Dimitrije Jankov, Shangyu Luo, Binhang Yuan, Zhuhua Cai, Jia Zou, Chris Jermaine, and Zekai J Gao. 2019. Declarative Recursive Computation on an RDBMS. *Proceedings of the VLDB Endowment* 12, 7 (2019).

[32] Dimitrije Jankov, Shangyu Luo, Binhang Yuan, Zhuhua Cai, Jia Zou, Chris Jermaine, and Zekai J Gao. 2019. Declarative recursive computation on an RDBMS: or, why you should use a database for distributed machine learning. *Proceedings of the VLDB Endowment* 12, 7 (2019), 822–835.

[33] Dimitrije Jankov, Shangyu Luo, Binhang Yuan, Zhuhua Cai, Jia Zou, Chris Jermaine, and Zekai J Gao. 2020. Declarative recursive computation on an RDBMS: or, why you should use a database for distributed machine learning. *ACM SIGMOD Record* 49, 1 (2020), 43–50.

[34] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[35] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[36] Michael Jungmair, André Kohn, and Jana Giceva. 2022. Designing an open framework for query optimization and compilation. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2389–2401.

[37] Konstantinos Karanasos, Matteo Interlandi, Doris Xin, Fotis Psallidas, Rathijit Sen, Kwanghyun Park, Ivan Popivanov, Supun Nakandal, Subru Krishnan, Markus Weimer, et al. 2019. Extending relational query processing with ML inference. *arXiv preprint arXiv:1911.00231* (2019).

[38] Neil Katkov. 2022. OPERATIONALIZING FRAUD PREVENTION ON IBM Z16. https://www.ibm.com/downloads/cas/DOXY3Q94

[39] Kyoungmin Kim, Jisung Jung, In Seo, Wook-Shin Han, Kangwoo Choi, and Jaehyok Chong. 2022. Learned cardinality estimation: An in-depth study. In *Proceedings of the 2022 International Conference on Management of Data*. 1214–1227.

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

[41] Adarsh Kumar, Arjun Balasubramanian, Shivaram Venkataraman, and Aditya Akella. 2019. Accelerating Deep Learning Inference via Freezing.. In *HotCloud*.

[42] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2–14.

[43] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. Mlir: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2–14.

[44] Yunseong Lee, Alberto Scolari, Byung-Gon Chun, Marco Domenico Santambrogio, Markus Weimer, and Matteo Interlandi. 2018. {PRETZEL}: Opening the Black Box of Machine Learning Prediction Serving Systems. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 611–626.

[45] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2118–2130.

[46] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The journal of machine learning research* 18, 1 (2017), 6765–6816.

[47] Shangyu Luo, Zekai J Gao, Michael Gubanov, Luis L Perez, and Christopher Jermaine. 2018. Scalable linear algebra on a relational database system. *IEEE Transactions on Knowledge and Data Engineering* 31, 7 (2018), 1224–1238.

[48] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J Gordon. 2018. Query-based workload forecasting for self-driving database management systems. In *Proceedings of the 2018 International Conference on Management of Data*. 631–645.

[49] Benjamin P MacLeod, Fraser GL Parlane, Thomas D Morrissey, Florian Häse, Loïc M Roch, Kevan E Dettelbach, Raphaell Moreira, Lars PE Yunker, Michael B Rooney, Joseph R Deeth, et al. 2020. Self-driving laboratory for accelerated discovery of thin-film materials. *Science Advances* 6, 20 (2020), eaaz8867.

[50] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[51] Tim Maxwell and Liz Bingler. 2023. How major credit card networks protect customers against fraud. https://www.bankrate.com/finance/credit-cards/major-credit-card-networks-protect-against-fraud/.

[52] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. 2018. Do deep generative models know what they don't know? *arXiv preprint arXiv:1810.09136* (2018).

[53] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[54] Thomas Neumann. 2011. Efficiently compiling efficient query plans for modern hardware. *Proceedings of the VLDB Endowment* 4, 9 (2011), 539–550.

[55] Thomas Neumann. 2021. Evolution of a compiling query engine. *Proceedings of the VLDB Endowment* 14, 12 (2021), 3207–3210.

[56] NVIDIA. 2014. *NVIDIA cuDNN.* Technical Report. NVIDIA Corporation, Santa Clara, California, USA. https://developer.nvidia.com/cudnn

[57] Shoumik Palkar, James J Thomas, Anil Shanbhag, Deepak Narayanan, Holger Pirk, Malte Schwarzkopf, Saman Amarasinghe, Matei Zaharia, and Stanford InfoLab. 2017. Weld: A common runtime for high performance data analytics. In *Conference on Innovative Data Systems Research (CIDR)*. 45.

[58] Kwanghyun Park, Karla Saur, Dalitso Banda, Rathijit Sen, Matteo Interlandi, and Konstantinos Karanasos. 2022. End-to-end Optimization of Machine Learning Prediction Queries. In *Proceedings of the 2022 International Conference on Management of Data*. 587–601.

[59] Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277* (2023).

[60] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *Acm Sigplan Notices* 48, 6 (2013), 519–530.

[61] Jun Rao, Chun Zhang, Nimrod Megiddo, and Guy Lohman. 2002. Automating physical database design in a parallel database. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. ACM, 558–569.

[62] Daniel Ratner, Bobby Sumpter, Frank Alexander, Jay Jay Billings, Ryan Coffee, Sarah Cousineau, Peter Denes, Mathieu Doucet, Ian Foster, Alex Hexemer, et al. 2019. *[Office of Basic Energy Sciences (BES)] Roundtable on Producing and Managing Large Scientific Data with Artificial Intelligence and Machine Learning.* Technical Report. DOESC Office of Basic Energy Sciences.

[63] Maximilian Schleich, Amir Shaikhha, and Dan Suciu. 2023. Optimizing Tensor Programs on Flexible Storage. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–27.

[64] Alexander Shapiro. 2003. Monte Carlo sampling methods. *Handbooks in operations research and management science* 10 (2003), 353–425.

[65] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: a GPU cluster engine for accelerating DNN-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 322–337.

[66] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E Gonzalez, et al. 2023. High-throughput generative inference of large language models with a single gpu. *arXiv preprint arXiv:2303.06865* (2023).

[67] Juwei Shi, Jia Zou, Jiaheng Lu, Zhao Cao, Shiqiang Li, and Chen Wang. 2014. MRTuner: a toolkit to enable holistic optimization for mapreduce jobs. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1319–1330.

[68] Sivic and Zisserman. 2003. Video Google: A text retrieval approach to object matching in videos. In *Proceedings ninth IEEE international conference on computer vision*. IEEE, 1470–1477.

[69] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems* 25 (2012).

[70] Yuxin Tang, Zhimin Ding, Dimitrije Jankov, Binhang Yuan, Daniel Bourgeois, and Chris Jermaine. 2023. Auto-Differentiation of Relational Computations for Very Large Scale Machine Learning. *arXiv preprint arXiv:2306.00088* (2023).

[71] 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. *arXiv preprint arXiv:1802.04730* (2018).

[72] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627.

[73] Xiaoying Wang, Weiyuan Wu, Jinze Wu, Yizhou Chen, Nick Zrymiak, Changbo Qu, Lampros Flokas, George Chow, Jiannan Wang, Tianzheng Wang, et al. 2022. ConnectorX: accelerating data loading from databases to dataframes. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2994–3003.

[74] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6

[75] Sheng Xiang, Mingzhi Zhu, Dawei Cheng, Enxia Li, Ruihui Zhao, Yi Ouyang, Ling Chen, and Yefeng Zheng. 2023. Semi-supervised Credit Card Fraud Detection via Attribute-Driven Graph Representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 14557–14565.

[76] Rohan Yadav, Alex Aiken, and Fredrik Kjolstad. 2022. DISTAL: the distributed tensor algebra compiler. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 286–300.

[77] 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

[78] Haitao Yuan, Guoliang Li, Ling Feng, Ji Sun, and Yue Han. 2020. Automatic view generation with deep learning and reinforcement learning. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1501–1512.

[79] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, et al. 2019. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 International Conference on Management of Data*. 415–432.

[80] Wangda Zhang, Junyoung Kim, Kenneth A Ross, Eric Sedlar, and Lukas Stadler. 2021. Adaptive code generation for data-intensive analytics. *Proceedings of the VLDB Endowment* 14, 6 (2021), 929–942.

[81] Bojian Zheng, Ziheng Jiang, Cody Hao Yu, Haichen Shen, Joshua Fromm, Yizhi Liu, Yida Wang, Luis Ceze, Tianqi Chen, and Gennady Pekhimenko. 2022. DietCode: Automatic optimization for dynamic tensor programs. *Proceedings of Machine Learning and Systems* 4 (2022), 848–863.

[82] Lixi Zhou, K Selçuk Candan, and Jia Zou. 2023. DeepMapping: The Case for Learned Data Mapping for Compression and Efficient Query Processing. *arXiv preprint arXiv:2307.05861* (2023).

[83] Lixi Zhou, Jiaqing Chen, Amitabh Das, Hong Min, Lei Yu, Ming Zhao, and Jia Zou. 2022. Serving Deep Learning Models with Deduplication from Relational Databases. *Proc. VLDB Endow.* 15, 10 (2022), 2230–2243. https://www.vldb.org/pvldb/vol15/p2230-zou.pdf

[84] Lixi Zhou, Arindam Jain, Zijie Wang, Amitabh Das, Yingzhen Yang, and Jia Zou. 2022. Benchmark of DNN Model Search at Deployment Time. In *Proceedings of the 34th International Conference on Scientific and Statistical Database Management*. 1–12.

[85] Shuyan Zhou, Uri Alon, Frank F Xu, Zhengbao Jiang, and Graham Neubig. 2022. Docprompting: Generating code by retrieving the docs. In *The Eleventh International Conference on Learning Representations*.

[86] Xuanhe Zhou, Chengliang Chai, Guoliang Li, and Ji Sun. 2020. Database meets artificial intelligence: A survey. *IEEE Transactions on Knowledge and Data Engineering* 34, 3 (2020), 1096–1116.

[87] Erkang Zhu, Fatemeh Nargesian, Ken Q Pu, and Renée J Miller. 2016. LSH ensemble: Internet-scale domain search. *arXiv preprint arXiv:1603.07410* (2016).

[88] Jia Zou. 2021. Using Deep Learning Models to Replace Large Materialized Views in Relational Database. In *11th Conference on Innovative Data Systems Research, CIDR 2021, Virtual Event, January 11-15, 2021, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2021/papers/cidr2021_abstract05.pdf

[89] 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 *Proceedings of the 2018 International Conference on Management of Data*. 1189–1204.

[90] Jia Zou, Amitabh Das, Pratik Barhate, Arun Iyengar, Binhang Yuan, Dimitrije Jankov, and Chris Jermaine. 2021. Lachesis: automatic partitioning for UDF-centric analytics. *Proceedings of the VLDB Endowment* 14, 8 (2021), 1262–1275.

[91] Jia Zou, Arun Iyengar, and Chris Jermaine. 2019. Pangea: monolithic distributed storage for data analytics. *Proceedings of the VLDB Endowment* 12, 6 (2019), 681–694.

[92] Jia Zou, Arun Iyengar, and Chris Jermaine. 2020. Architecture of a distributed storage that combines file system, memory and computation in a single layer. *The VLDB Journal* (2020), 1–25.