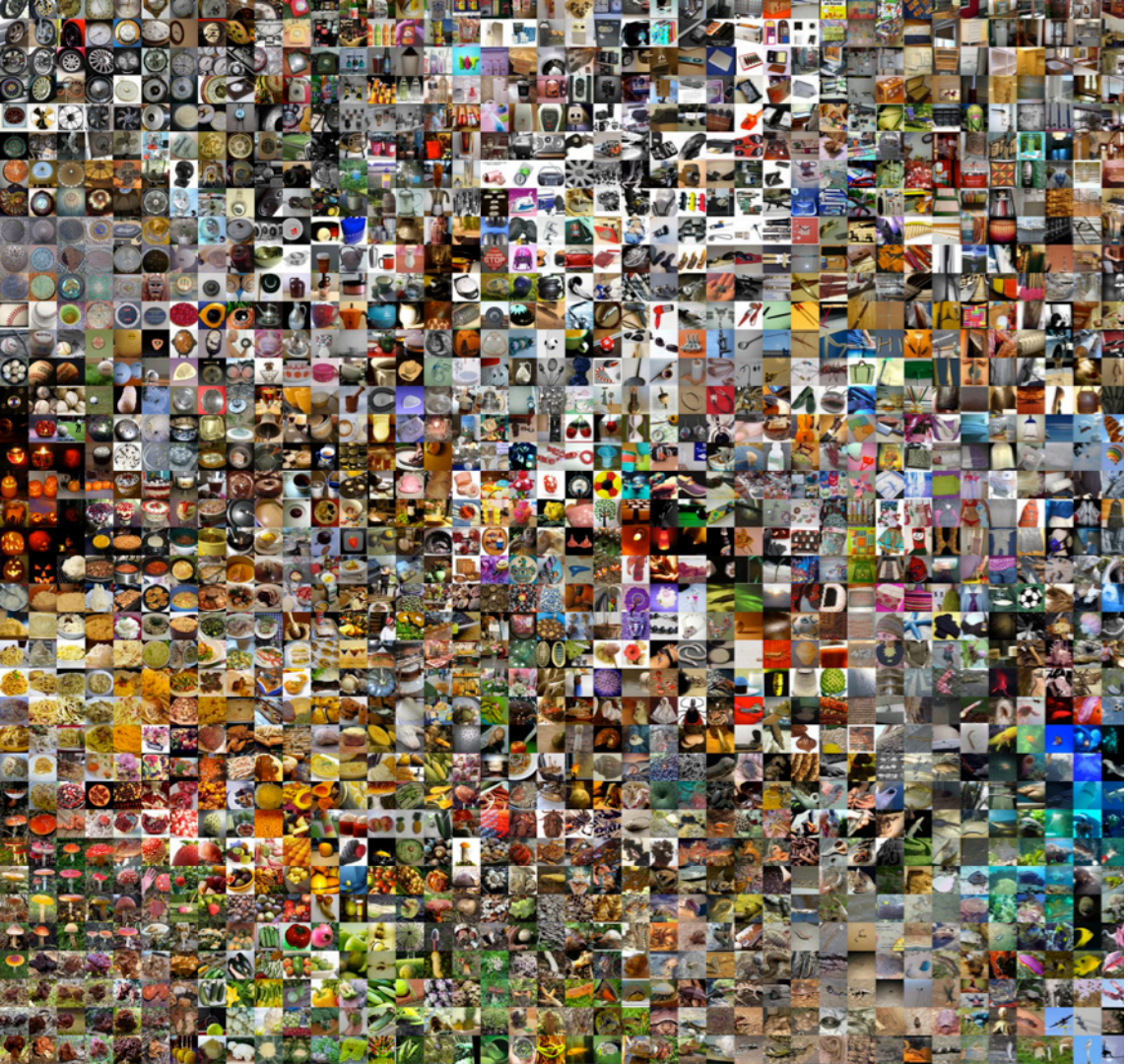
OSDI 2018 PREVIEW:
MACHINE LEARNING

Shivaram Venkataraman
University of Wisconsin, Madison



AlphaGo

MACHINE LEARNING

Classification

Recommendation

WHAT

Dog

Dog

Cat

Cat

Training

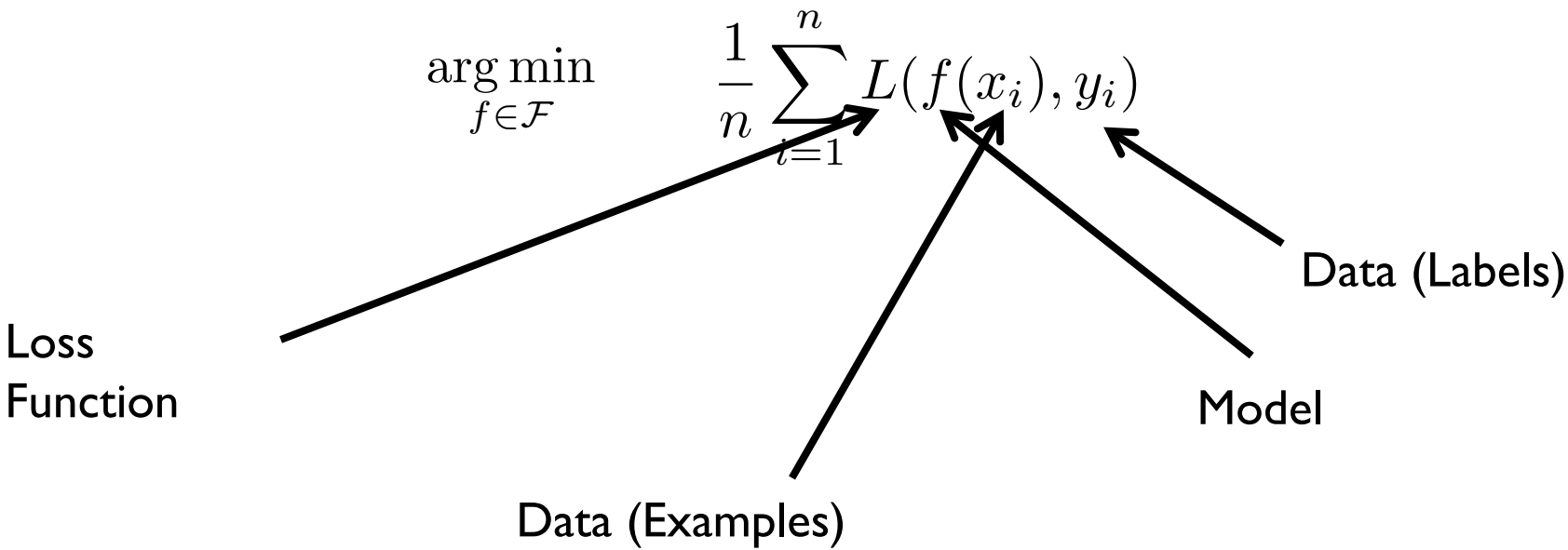
Inference

Model

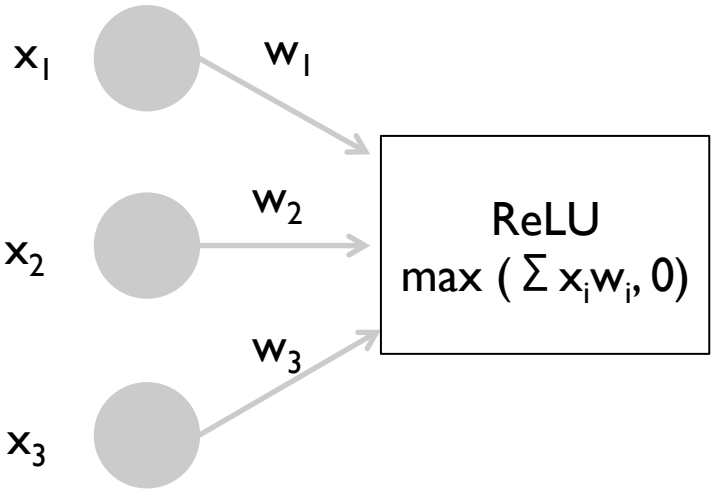
Cat ?

HOW

Optimization Algorithms

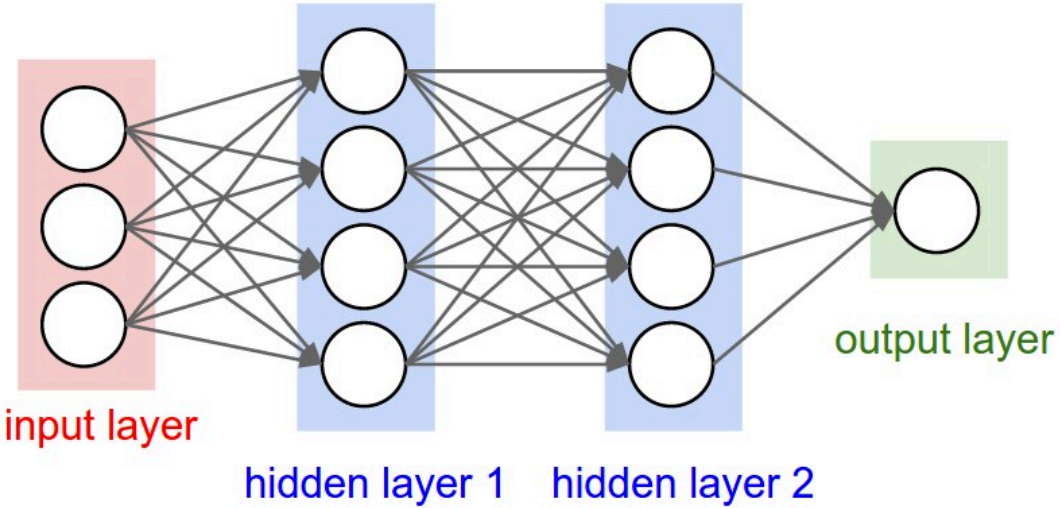


DEEP LEARNING



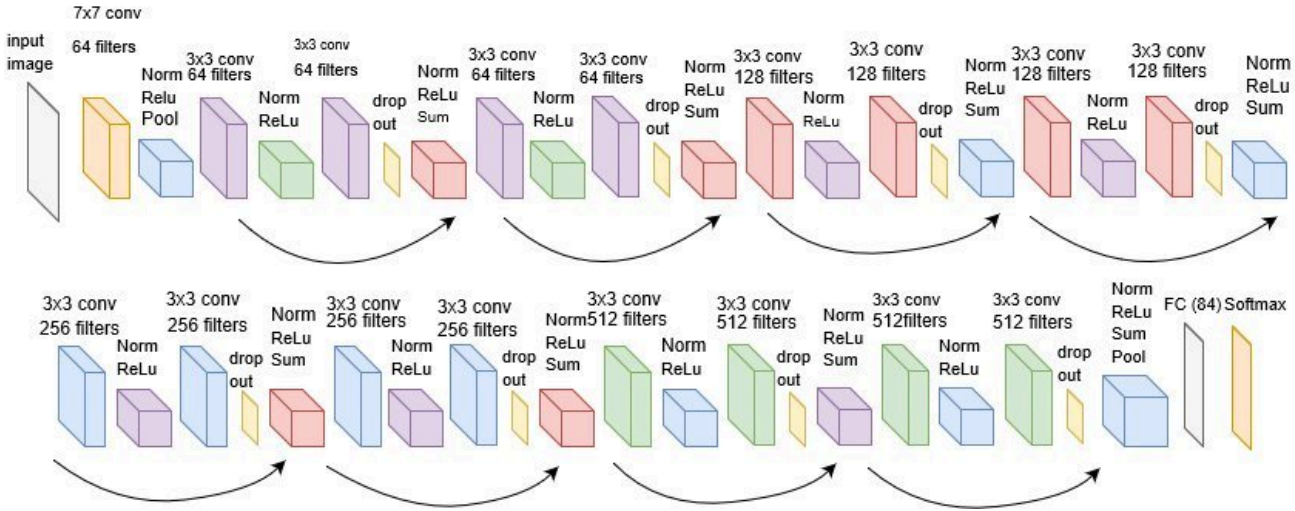
Non-linearity

DEEP LEARNING



Stack them together !

DEEP LEARNING



ResNet18

Convolution
ReLU
MaxPool
Fully Connected
SoftMax

...

MODEL TRAINING

$$w^{(k+1)} = w^{(k)} - \alpha_k \nabla f(w^{(k)})$$

Initialize w

For many iterations:

    Compute Gradient

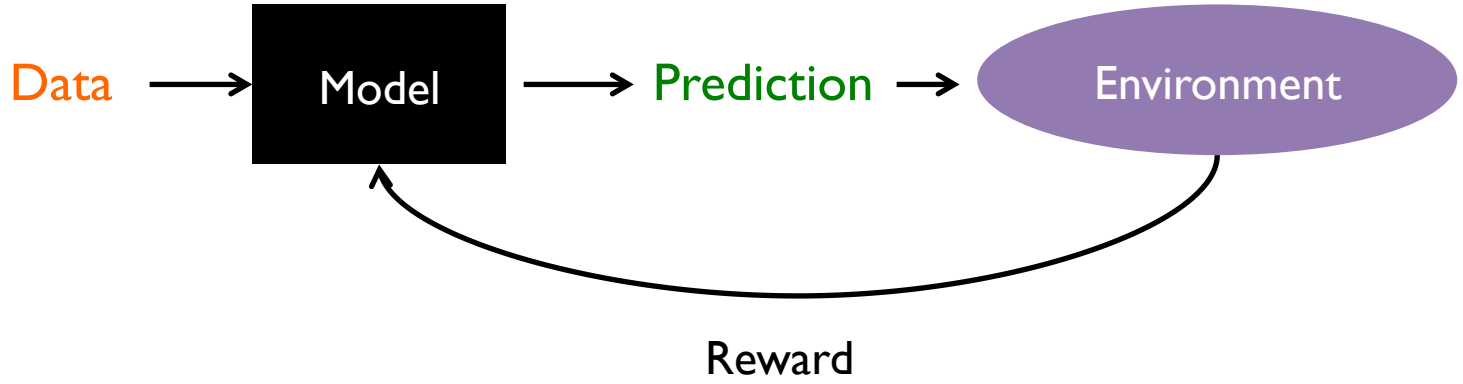
    Update model

End

Stochastic Gradient Descent

Gradient using backprop
Compute Intensive!

MULTIPLE PREDICTIONS



Reinforcement Learning

Robotics
Control Systems
Self Driving Cars

Train with iterative simulations

MACHINE LEARNING TAKEAWAYS

Iterative algorithms

Compute Intensive

Known operators

RECENT ML SYSTEMS

Programming Frameworks

- Naiad [SOSP 2013]
- PowerGraph [OSDI 2012]
- Tensorflow [OSDI 2016]

Scalable Training

- Parameter Server [OSDI 2014]
- Project Adam [OSDI 2014]

Bug Hunting

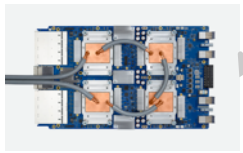
- DeepXplore [SOSP 2017]

MACHINE LEARNING STACK

Resource Manager

GPU

FPGA



TPU

Goals, Challenges

- Heterogeneous hardware
- Low latency (Avg, P99)
- Heterogeneous jobs
- Cluster Utilization

QUESTIONS TO CONSIDER

What is the target machine learning workload ?

- Data or model types
- Training vs. Inference

What is different when running ML workloads ?

- Compared to SQL queries
- Compared to web applications

Ray: A Distributed Framework for Emerging AI Applications

Philipp Moritz*, Robert Nishihara*, Stephanie Wang, Alexey Tumanov, Richard Liaw,
Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, Ion Stoica
University of California, Berkeley

Abstract

The next generation of AI applications will continuously interact with the environment and learn from these interactions. These applications impose new and demanding systems requirements, both in terms of performance and flexibility. In this paper, we consider these requirements and present Ray—a distributed system to address them. Ray implements a unified interface that can express both task-parallel and actor-based computations, supported by a single dynamic execution engine. To meet the performance requirements, Ray employs a distributed scheduler and a distributed and fault-tolerant store to manage the system's control state. In our experiments, we demonstrate scaling beyond 1.8 million tasks per second and better performance than existing specialized systems for several challenging reinforcement learning applications.

1 Introduction

and their use in prediction. These frameworks often leverage specialized hardware (e.g., GPUs and TPUs), with the goal of reducing training time in a batch setting. Examples include TensorFlow [7], MXNet [18], and PyTorch [46].

The promise of AI is, however, far broader than classical supervised learning. Emerging AI applications must increasingly operate in dynamic environments, react to changes in the environment, and take sequences of actions to accomplish long-term goals [8, 43]. They must aim not only to exploit the data gathered, but also to explore the space of possible actions. These broader requirements are naturally framed within the paradigm of *reinforcement learning* (RL). RL deals with learning to operate continuously within an uncertain environment based on delayed and limited feedback [56]. RL-based systems have already yielded remarkable results, such as Google's AlphaGo beating a human world champion [54], and are beginning to find their way into dialogue systems, UAVs [42], and robotic manipulation [25, 60].

Framework for Reinforcement Learning

Key Challenges:
Distributed training
Heterogeneous tasks

Compiler for Deep Learning Models

Key Challenges: Diverse hardware Many Optimizations

TVM: An Automated End-to-End Optimizing Compiler for Deep Learning

Tianqi Chen[1], Thierry Moreau[1], Ziheng Jiang[1,2], Lianmin Zheng[3], Eddie Yan[1]

Meghan Cowan[1], Haichen Shen[1], Leyuan Wang[4,2], Yuwei Hu[5], Luis Ceze[1], Carlos Guestrin[1], Arvind Krishnamurthy[1]
[1]Paul G. Allen School of Computer Science & Engineering, University of Washington

[2] AWS, [3]Shanghai Jiao Tong University, [4]UC Davis, [5]Cornell

Abstract

There is an increasing need to bring machine learning to a wide diversity of hardware devices. Current frameworks rely on vendor-specific operator libraries and optimize for a narrow range of server-class GPUs. Deploying workloads to new platforms – such as mobile phones, embedded devices, and accelerators (e.g., FPGAs, ASICs) – requires significant manual effort. We propose TVM, a compiler that exposes graph-level and operator-level optimizations to provide performance portability to deep learning workloads across diverse hardware back-ends. TVM solves optimization challenges specific to deep learning, such as high-level operator fusion, mapping to arbitrary hardware primitives, and memory latency hiding. It also automates optimization of low-level programs to hardware characteristics by employing a novel, learning-based cost modeling method for rapid exploration of code optimizations. Experimental results show that TVM delivers performance across hardware back-ends that are competitive with state-of-the-art, hand-tuned libraries for low-power CPU, mobile GPU, and server-class GPUs. We also demonstrate TVM's ability to target new accelerator back-ends, such

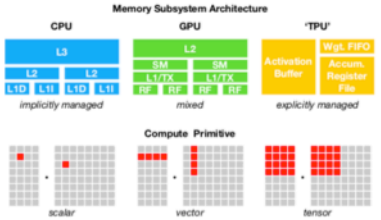


Figure 1: CPU, GPU and TPU-like accelerators require different on-chip memory architectures and compute primitives. This divergence must be addressed when generating optimized code.

terms of memory organization, compute functional units, etc., as shown in Figure 1.

Current DL frameworks, such as TensorFlow, MXNet, Caffe, and PyTorch, rely on a computational graph intermediate representation to implement optimizations, e.g., auto differentiation and dynamic memory management [3, 4, 9]. Graph-level optimizations, however,

Gandiva: Introspective Cluster Scheduling for Deep Learning

Wencong Xiao[†⋆⋆], Romil Bhardwaj[⋆⋆], Ramachandran Ramjee[⋆], Muthian Sivathanu[⋆], Nipun Kwatra[⋆],
Zhenhua Han[◇⋆], Pratyush Patel[⋆], Xuan Peng[‡⋆], Hanyu Zhao[§⋆], Quanlu Zhang[⋆], Fan Yang[⋆], Lidong Zhou[⋆]
[†]*Beihang University,* [⋆]*Microsoft Research,* [◇]*The University of Hong Kong,*
[‡]*Huazhong University of Science and Technology,* [§]*Peking University*

Abstract

We introduce Gandiva, a new cluster scheduling framework that utilizes domain-specific knowledge to improve latency and efficiency of training deep learning models in a GPU cluster.

One key characteristic of deep learning is feedback-driven exploration*, where a user often runs a set of jobs (or a multi-job) to achieve the best result for a specific mission and uses early feedback on accuracy to dynamically prioritize or kill a subset of jobs; simultaneous early feedback on the entire multi-job is critical. A second characteristic is the heterogeneity of deep learning jobs in terms of resource usage, making it hard to achieve best-fit a priori. Gandiva addresses these two challenges by exploiting a third key characteristic of deep learning:* intra-job predictability*, as they perform numerous repetitive iterations called mini-batch iterations. Gandiva exploits intra-job predictability to time-slice GPUs efficiently across multiple jobs, thereby delivering low-latency. This predictability is also used for introspecting job performance and dynamically migrating jobs to better-fit GPUs, thereby improving cluster efficiency.*

An increasingly popular computing trend over the last few years is deep learning [32]; it has already had significant impact; *e.g.*, on widely-used personal products for voice and image recognition, and has significant potential to impact businesses. Hence, it is likely to be a vital and growing workload, especially in cloud data centers.

However, deep learning is compute-intensive and hence heavily reliant on powerful but expensive GPUs; a GPU VM in the cloud costs nearly 10x that of a regular VM. Cloud operators and large companies that manage clusters of tens of thousands of GPUs rely on cluster schedulers to ensure efficient utilization of the GPUs.

Despite the importance of efficient scheduling of deep learning training (DLT) jobs, the common practice today [12, 28] is to use a traditional cluster scheduler, such as Kubernetes [14] or YARN [50], designed for handling big-data jobs such as MapReduce [17]; a DLT job is treated simply as yet another big-data job that is allocated a set of GPUs at job startup and holds exclusive access to its GPUs until completion.

In this paper, we present *Gandiva*, a new scheduling framework that demonstrates that a significant increase

Cluster Scheduler for Deep Learning Jobs

Key Challenges: Prioritize Accuracy Heterogeneous Jobs

PRETZEL: Opening the Black Box of Machine Learning Prediction Serving Systems

Yunseong Lee
Seoul National University

Alberto Scolari
Politecnico di Milano

Byung-Gon Chun
Seoul National University

Marco Domenico Santambrogio
Politecnico di Milano

Markus Weimer
Microsoft

Matteo Interlandi
Microsoft

System for Inference

Key Challenges: Increasing Utilization P99 Latency

Abstract

Machine Learning models are often composed of pipelines of transformations. While this design allows to efficiently execute single model components at training-time, prediction serving has different requirements such as low latency, high throughput and graceful performance degradation under heavy load. Current prediction serving systems consider models as black boxes, whereby prediction-time-specific optimizations are ignored in favor of ease of deployment. In this paper, we present PRETZEL, a prediction serving system introducing a novel white box architecture enabling both end-to-end and multi-model optimizations. Using production-like model pipelines, our experiments show that PRETZEL is able to introduce performance improvements over different dimensions; compared to state-of-the-art approaches PRETZEL is on average able to reduce 99th percentile latency by 5.5× while reducing memory footprint by 25×, and increasing throughput by 4.7×.

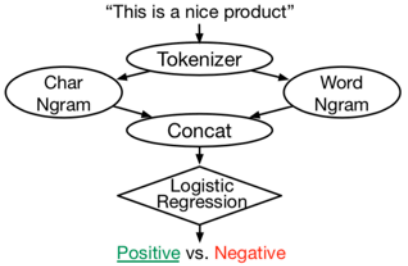


Figure 1: A Sentiment Analysis (SA) pipeline consisting of operators for featurization (ellipses), followed by a ML model (diamond). *Tokenizer* extracts tokens (e.g., words) from the input string. *Char* and *Word Ngrams* featurize input tokens by extracting n-grams. *Concat* generates a unique feature vector which is then scored by a *Logistic Regression* predictor. This is a simplification: the actual DAG contains about 12 operators.

sive iterative algorithms; successively, trained pipelines are used for *inference* to generate predictions through the estimated model parameters. When trained pipelines

CONCLUSION

Machine learning workloads present new systems challenges!

For every paper, consider

- Workload properties
- ML goals / targets
- What is different from SQL/web apps