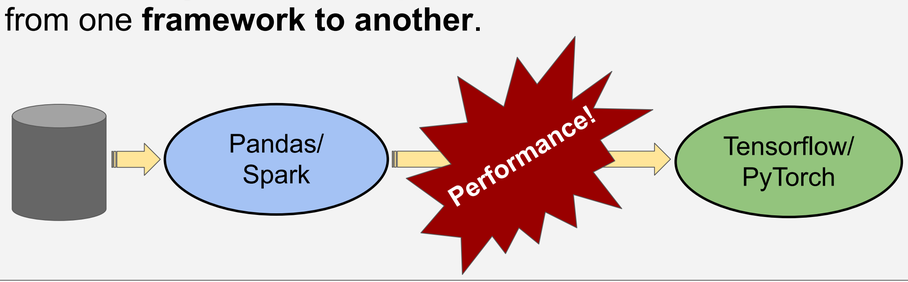
End-to-end Optimization for High-performance Machine Learning Pipelines

Supun Abeysinghe, Fei Wang, Tiark Rompf

Motivation

- Deploying cutting edge machine learning (ML) models at scale requires ingesting data from various sources.
- Today, this data preprocessing is typically implemented using separate tools like **Pandas** or **Spark** that then feed into **PyTorch** or **TensorFlow**.
- While these ML systems go to great lengths to optimize the performance of the ML kernels, large amounts of **performance are lost** when **data needs to move** from one **framework to another**.



In this work...

- We present a common runtime system for an **end-to-end high-performance ML pipeline**
- Integrates two systems that are built around the central theme of **runtime compilation** and **native code generation**.
  - **Flare** - an accelerator for Spark SQL, used for data manipulation
  - **Lantern** - an accelerator for TensorFlow and PyTorch, used for building ML models
- Leverages the power of **Lightweight Modular Staging (LMS)** for runtime code generation.
- The resultant system generates a **globally optimized end-to-end compiled data path** that would perform the entire process from data preprocessing to training and inference
- Eliminates the interaction overhead between systems without the need for sacrificing program expressivity.

Flare

- Flare is a **query accelerator** built for Apache Spark
- Achieves **order of magnitude speedups** on DataFrame and SQL workloads
- Flare compiles optimized query plans generated by **Catalyst** (in-built query optimizer for Apache Spark) to **native code**
- Leverages **Lightweight Modular Staging (LMS)** to generate native code

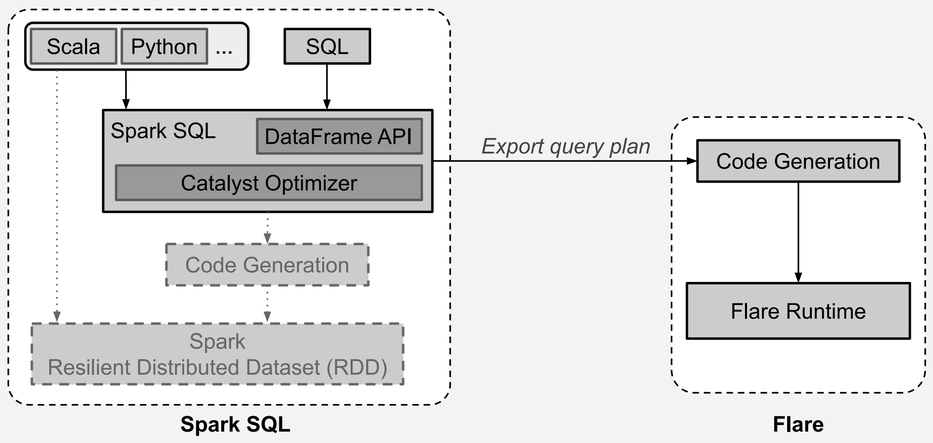
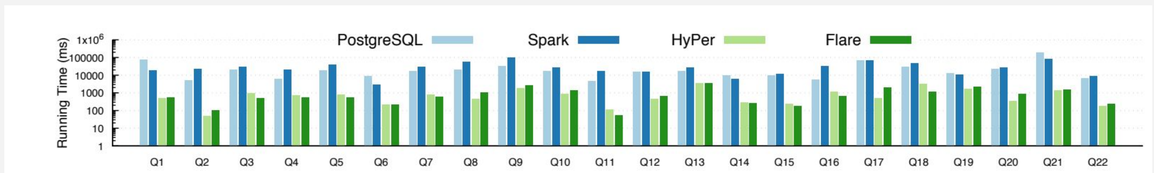


Figure 1 : Flare system overview



SF10	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20	Q21	Q22
Postgres	72599	5064	20790	6032	17904	9071	16408	20003	31728	16409	4729	15755	16152	9369	9607	5662	66254	30104	12852	22795	184736	6900
Spark SQL	17692	21062	28039	20693	40008	2826	28262	55346	98216	27585	16193	15431	25632	5913	11538	30973	66089	47542	11062	27448	78896	8687
HyPer	498	47	969	725	821	207	804	467	1782	846	111	460	3307	283	227	1112	485	3061	1666	345	1349	177
Flare	550	98	481	544	527	212	596	1016	2688	1429	53	643	3499	256	175	679	1924	1150	2236	836	1533	241

Figure 2 : Performance comparison of Postgres, HyPer, Spark SQL, Flare in TPC-H (SF10) (from: Essertel et al OSDI'18)

Lantern

- Lantern is a highly expressive machine learning framework written in **Scala**
- Attains the **performance of "define-then-run"** machine learning frameworks like Tensorflow while preserving **expressiveness of "define-by-run"** frameworks such as PyTorch
- Backpropagation is implemented using functions with callbacks; where forward pass is executed as a sequence of function calls and the backward pass with the corresponding function returns (uses CPS in particular)
- Leverages **Lightweight Modular Staging (LMS)** to generate low-level (C/CUDA) code

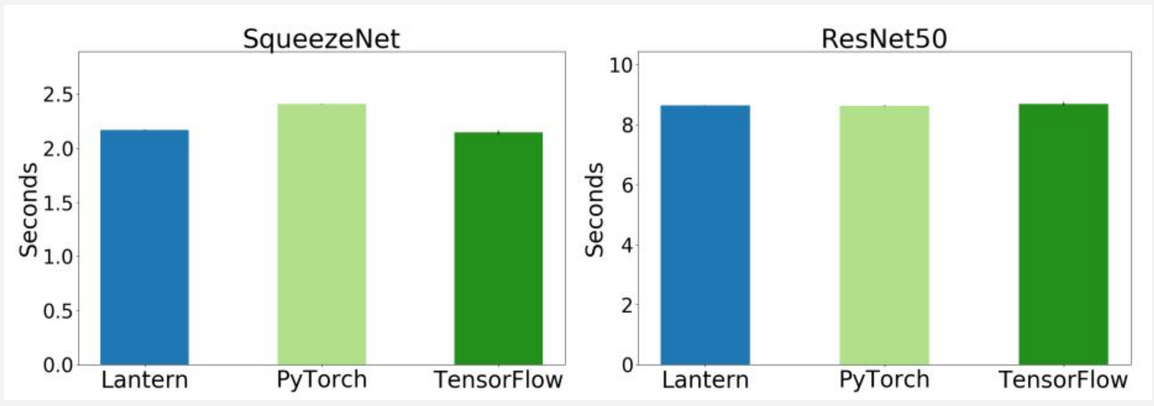


Figure 3 : Running time of SqueezeNet and ResNet50 for different frameworks (from: Fei et al ICFP '19)

Flare + Lantern

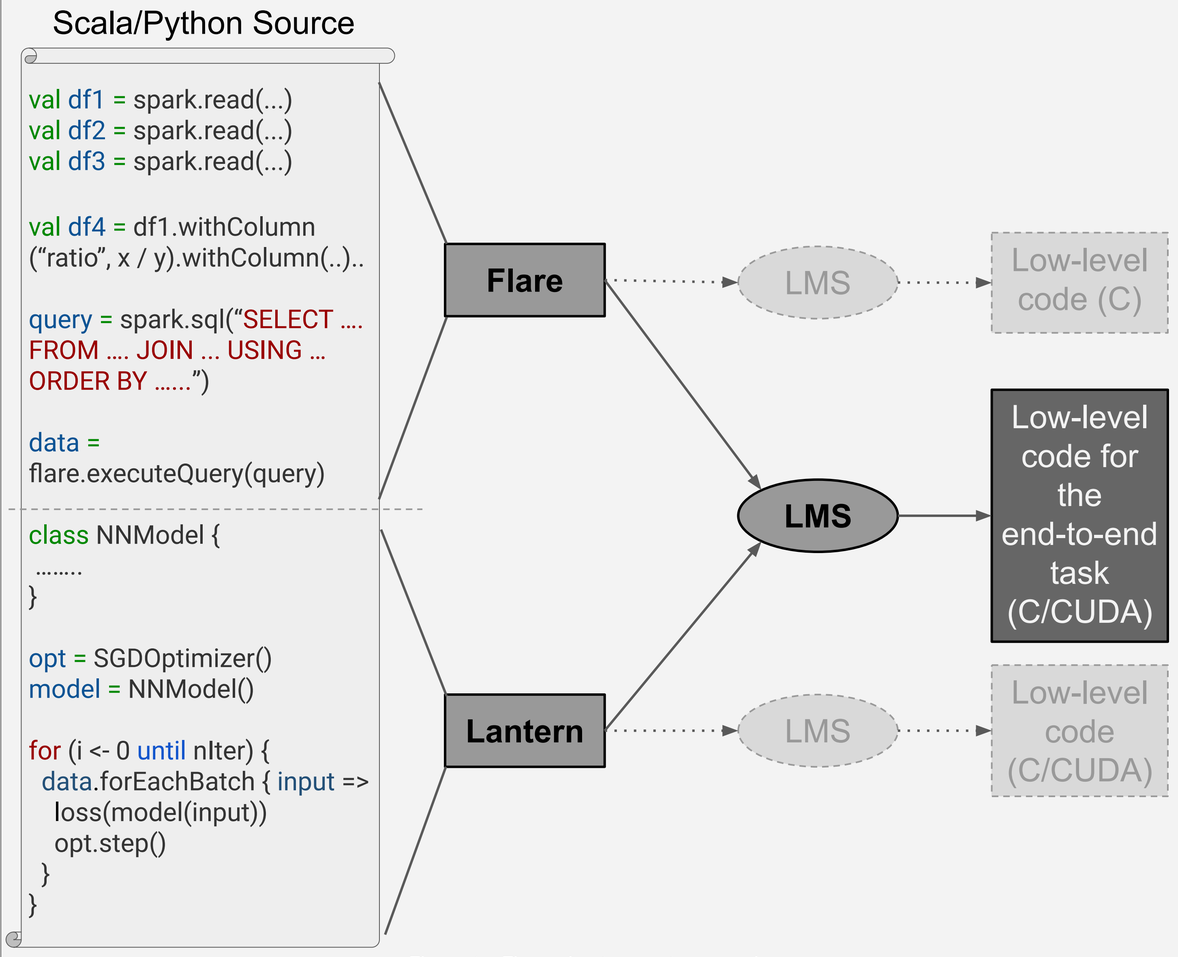


Figure 4 : Flare + Lantern system overview

Experiments

- Prior experiments have shown promising results on using **Flare** in conjunction with **Tensorflow**
- For a simple case where Tensorflow classifier is used as a Spark UDF, Flare produced over **1,000,000x speed up**! for some cases

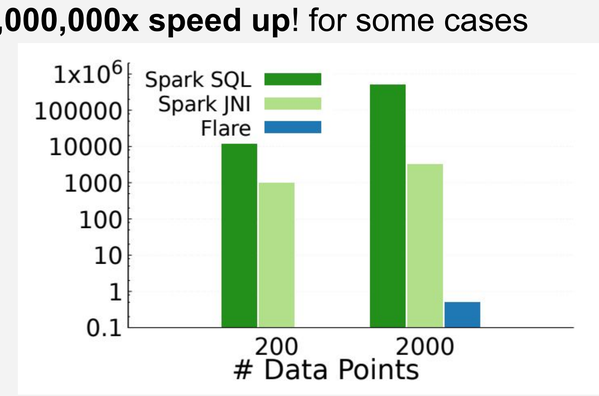


Figure 5 : Running time (ms) of a query containing a TF classifier as an UDF (from: Essertel et al OSDI '18)

- Our preliminary experiments show significant potential gains (**18x speed up**) for the proposed system

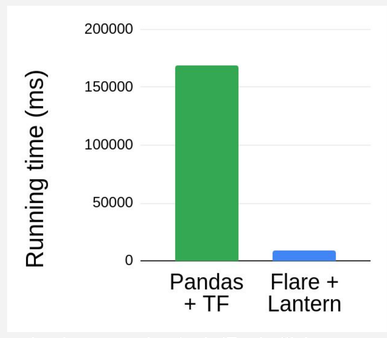


Figure 6 : Total Running time (ms) for a basic regression task (Fuel efficiency prediction) taken from TF documentation