A Scalable Serving System for a Deep Neural Network

Giulio Marcon
giulio.marcon@aalto.fi

Tutor: Thanh-Phuong Pham

Abstract

Serving a Deep Neural Network (DNN) efficiently on a cluster of Graphical Processing Unit (GPU) is an important problem.
When we think about ML, we usually only think about the great models that we can now create. But when we want to take that amazing model and make it available to the world we need to think about all the things that a production solution requires, including scalability, consistency, modularity, and testability, as well as safety and security.

KEYWORDS: Machine Learning, Tensorflow, Serving, BentoML

1 Introduction

Consider a Reinforcement Learning agent that has to learn how to play the game of Pong just from the pixels that form a frame of the video-game, and has to analyze thousands of these images to successfully beat the opponent. The core computations of this workload are Deep Neural Network (DNN), which are networks of dense linear algebra computations where several layers of nodes are used to build up progressively more abstract

representations of the data. Graphical Processing Units (GPU) are specialized hardware accelerators for DNNs that have been used to solve the complex computations, and in the recent years Tensor Processing Units (TPUs) have emerged to further increase the computational power available.
A fundamental problem is therefore to distribute the large incoming workload onto the machines whilst improving the workflow of the data scientist that has to develop models that will be deployed into production.
This paper does not offer novel ML serving algorithms, but instead seeks to increase the community's awareness on the importance of designing an infrastructure that is able tackle the three key challenges of prediction serving: latency, throughput, and accuracy.

2 What is serving?

Serving is the process of applying machine learning models after they have been trained.
All applications of machine learning depend mainly on two stages: training and inference.
Consider the example of a machine learning inference server that executes the model algorithm given data gathered from an IoT sensor, for instance a camera, and returns the inference output, as shown in Figure 1.
Training is the process of building a model from data, which is often computationally expensive and requires multiple passes over potentially large datasets. In our example, it involves the use of a deep-learning framework and training dataset. In this case, IoT data provides a source of training data that data scientists and engineers can use to train the models for a variety of use cases.
Inference is the process of using the model to make a prediction given an input, is typically part of user-facing applications, and must run in real-time, often on orders of magnitude more queries than during training. In our example, IoT data can be used as the input to a trained machine learning model, enabling predictions that can guide decision logic on the device, at the edge gateway or elsewhere in this IoT system. It is important to notice that the input can is usually very diverse and significant in its size, which is one of the main problems that needs to be tackled when designing this type of infrastructure.

If we analyze what a typical workflow of a data scientist looks like, we will notice that the process of deploying models requires additional effort and attention, which results in them being distracted from their problem at hand. Apart from that, having many data scientists build and maintain their own serving solutions means that there may be a lot of duplicated effort.

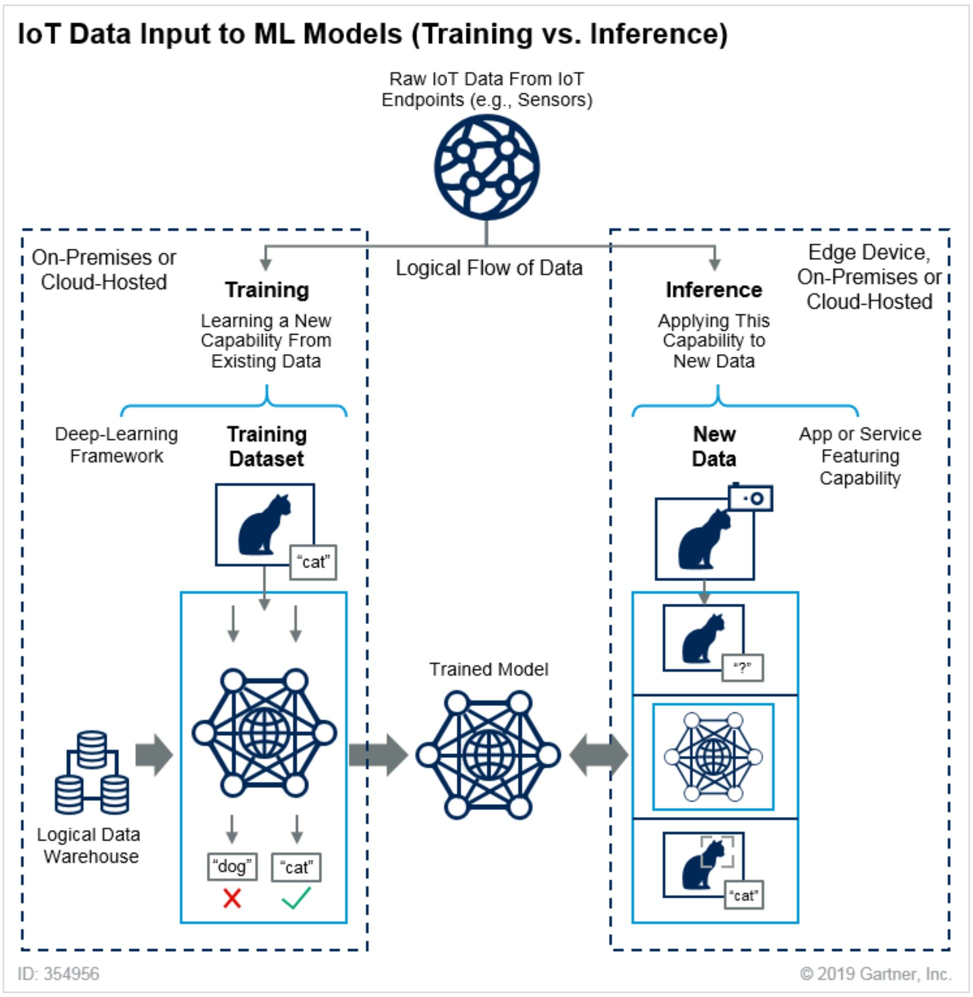
IoT Data Input to ML Models (Training vs. Inference)
Raw IoT Data From IoT Endpoints (e.g., Sensors)
On-Premises or Cloud-Hosted
Training
Learning a New Capability From Existing Data
Logical Flow of Data
Inference
Applying This Capability to New Data
Edge Device, On-Premises or Cloud-Hosted
Deep-Learning Framework
Training Dataset
New Data
App or Service Featuring Capability
"cat"
Trained Model
"?"
Logical Data Warehouse
"dog"
"cat"
"cat"
ID: 354956
© 2019 Gartner, Inc.

Figure 1. Inference is where capabilities learned during deep learning training are put to work. Source: [12]

3 Related work solutions.

Most of the solutions that try to tackle this problem are done using custom code developed by individual teams for specific use cases, leading to duplicated effort and fragile systems with high technical debt.
Over the last few years, some companies have tried to develop an infrastructure that was able to successfully produce and deploy machine

learning models, such as Clipper [14], a prediction serving system with a layered architecture that abstracts away the complexity associated with serving predictions, a set of novel techniques to reduce and bound latency while maximizing throughput, and a model selection layer that enables online model selection and composition to provide robust and accurate predictions for interactive application.
Another team developed Nexus [19], a scalable and efficient system that operates directly on models and GPUs, instead of serving the entire application in an opaque CPU-based container with models embedded in it, enabling several optimizations in batching and allowing more efficient resource allocation.
InferLine [16] was developed as a system which efficiently provisions prediction pipelines subject to end-to-end latency constraints by combining a low-frequency Planner that finds cost-optimal configurations, with a high-frequency Tuner that rapidly re-scales pipelines to meet latency Service Level Objectives (SLOs) in response to changes in the query workload.
Other solutions include: TorchServe [11], designed specifically for PyTorch models, NVIDIA TensorRT [4], an SDK build to work the the company's own GPUs, and Microsoft Custom Decision Service [13] [5], which provides a cloud-based service for optimizing decisions using multi-armed bandit algorithms and reinforcement learning.

Unfortunately, most of them are now deprecated, and Google's TensorFlow Serving [17] and BentoML [1] seem to be the two most advanced ones currently available to the public.
In this seminar paper, I will compare the platforms mentioned above to test their performance while keeping in mind the advantages and disadvantages of each, and in the end, try to propose a solution that can improve the existing solutions.

4 Tensorflow Serving

Tensorflow Serving [10] is an open-source ML model serving project by Google. It aims to be a lexible, high-performance serving system, designed for production environments, that facilitates the deployment of new algorithms and experiments, while keeping the same server architecture and APIs. Tensorflow Serving provides out-of-the-box integration with Tensorflow [8] models, but can be easily extended to serve other types of models

and data.
Some of the advantages of using this technology include:

- High performance. It has proven performance handling tens of millions of inferences per second at Google [9].

- High availability. It has a model versioning system to make sure there is always a healthy version being served while loading a new version into its memory

- Actively maintained by the developer community and backed by Google

5 BentoML

BentoML is an open-source platform for high-performance ML framework for serving, managing, and deploying machine learning models.
The main advantages of the project are:

- Ability to create basic API endpoints for serving trained models

- High-Performant online API for serving with adaptive micro-batching support.

- Provides support for all major machine learning training frameworks.

- Flexible deployment orchestration that follow DevOps best practices, such as Docker, Kubernetes, Kubeflow, Knative, AWS Lambda, SageMaker, Azure ML, and GCP.

6 Comparing the two platforms

The two frameworks have been compared using the Fashion MNIST dataset [2], a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples, often used to compare machine learning models, and used in both Tensorflow and BentoML's

documentation.
Both frameworks, both written in Python, have some important differences:

- BentoML has multi-framework support, and is fully compatible with Tensorflow, PyTorch, Scikit-Learn, XGBoost, FastAI. Tensorflow-serving, on the other hand, only supports Tensorflow framework as of now, even though it can be adapted for other frameworks with some workarounds [6].

- Tensorflow loads the model from a tf.SavedModel format, which means that all the graphs and computations must be compiled into the SavedModel. BentoML keeps the Python run time in serving time, making it possible to do pre-processing and post-processing in serving endpoints.

- Tensorflow-serving can serve different versions of the same model, which can be useful when comparing changes during testing.

Some of the code used in the comparison can be found in this GitHub repository: [3].

7 Possible improvement

MLOps is an emerging field that aims to solve the technical debt that currently exists in machine learning systems [15].
They consist in a combination of philosophies and practices designed to enable data science and IT teams to rapidly develop, deploy, maintain, and scale out Machine Learning models. By following these guidelines, new versions of the models can be deployed into production constantly and reliably.
Important improvements could be done in the testing phase of the cycle shown id Figure 2. The validation of the trained model is still a tedious task that takes a lot of effort and is prone to error [18].
A specific testing support and methodology for detecting ML-specific errors needs to be established and followed in order to successfully tackle this problem.
Some of the important type of testing includes:

- Stress tests to ensure that the infrastructure can handle high volumes of data.

- Model staleness test, to check whether the trained model includes up-to-date data.

- Checking that the calculated metrics are satisfactory and improve the previous version of the same model, for example by measuring loss metrics and bias.

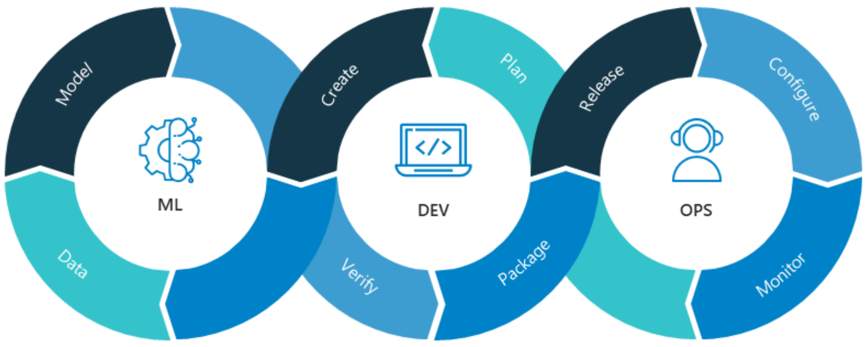

Figure 2. Proof of Concept MLOps cycle. Source: [7]

8 Conclusion

Concepts such as continuous delivery and integration, immutable infrastructure, and serverless computing, have been the focus of DevOps engineers in the last few years, and they can be adapted for the development, test, and serving of DNNs and ML models.
The so-called MLOps practises are still being designed and refined to this day, due to how innovative and current they are, but improvements can already be noticed, even if the practices are still in the early stages of development.

References

[1] Bentoml - the easiest way to build machine learning apis.
https://github.com/bentoml/BentoML. Accessed: 29-10-2020.

[2] Fashion mnist - an mnist-like dataset of 70,000 28x28 labeled fashion images.
https://www.kaggle.com/zalando-research/fashionmnist. Accessed: 29-11-2020.

[3] Github - giuliomarcon/a-scalable-serving-system-for-a-deep-neural-network.

https://github.com/giuliomarcon/A-Scalable-Serving-System-for-a-Deep-Neural-Network. Accessed: 29-11-2020.

[4] Nvidia tensorrt - programmable inference accelerator.
https://developer.nvidia.com/tensorrt. Accessed: 29-11-2020.

[5] Project custom decision.
https://www.microsoft.com/en-us/research/project/custom-decision/. Accessed: 29-11-2020.

[6] Running pytorch models in production.
https://medium.com/styria-data-science-tech-blog/running-pytorch-models-in-production-fa09bebca622. Accessed: 29-11-2020.

[7] Running pytorch models in production.
https://nealanalytics.com/expertise/mlops/. Accessed: 29-11-2020.

[8] Tensorflow.
https://www.tensorflow.org. Accessed: 29-10-2020.

[9] Tensorflow extended.
https://www.tensorflow.org/tfx. Accessed: 29-10-2020.

[10] Tensorflow extended - serving models.
https://www.tensorflow.org/tfx/guide/serving. Accessed: 29-10-2020.

[11] Torchserve - a flexible and easy to use tool for serving pytorch models.
https://pytorch.org/serve/. Accessed: 29-11-2020.

[12] Training versus inference.
https://blogs.gartner.com/paul-debeasi/2019/02/14/training-versus-inference/. Accessed: 29-11-2020.

[13] Markus Cozowicz Luong Hoang John Langford Stephen Lee Jiaji Li Dan Melamed Gal Oshri Oswaldo Ribas Siddhartha Sen Alex Slivkins Alekh Agarwal, Sarah Bird. Making Contextual Decisions with Low Technical Debt. Technical report.

[14] G. Zhou M. J. Franklin J. E. Gonzalez I. Stoica D. Crankshaw, X. Wang. Clipper: A Low-Latency Online Prediction Serving System. Technical report.

[15] Daniel Golovin Eugene Davydov Todd Phillips Dietmar Ebner Vinay Chaudhary Michael Young Jean-Francois Crespo Dan Denniso D. Sculley, Gary Holt. Hidden Technical Debt in Machine Learning Systems. Technical report.

[16] Simon Mo Corey Zumar Joseph E. Gonzalez Ion Stoica Alexey Tumanov Daniel Crankshaw, Gur-Eyal Sela. InferLine: ML Prediction Pipeline Provisioning and Management for Tight Latency Objectives. Technical report.

[17] 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 Martin Zinkevich Denis Baylor, Eric Breck. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. Technical report, Google Inc., 2017.

[18] Eric Nielsen Michael Salib D. Sculley Eric Breck, Shanqing Cai. The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction. Technical report, Google Inc.

[19] Y. Jin L. Zhao B. Kong M. Philipose A. Krishnamurthy R. Sundaram H. Shen, L. Chen. Nexus: A GPU Cluster Engine for Accelerating Neural Networks Based Video Analysis. Technical report.