Journal of Data-centric Machine Learning Research (2023) Submitted 9/23; Revised 10/23; Published 11/23

Deploying Data Selection Techniques on Dynamic Datasets

Maximilian Böther MBOETHER@INF.ETHZ.CH
ETH Zurich
Switzerland

Ana Klimovic AKLIMOVIC@ETHZ.CH
ETH Zurich
Switzerland

Reviewed on OpenReview: *https://openreview.net/forum?id=CytNIWMFKN*

Editor:

1 Introduction and Motivation

The data fueling today's production machine learning (ML) models, which typically comes from a myriad of sensors or real-time user click streams, is continuously evolving. Real-world training datasets are inherently *dynamic*, as samples get added or removed over time. To maintain high accuracy and adjust to data distribution shifts, models deployed in the wild need to incorporate new data during training [Shankar et al. (2022); Huyen (2022)]. While models are typically (re)trained at simple time-based or data volume-based intervals, selecting which data to train on is an active area of ML research. To reduce to cost of ML training, researchers are exploring how to optimally reduce the number of samples using various strategies, such as continual learning [Prabhu et al. (2020)], importance sampling [Katharopoulos and Fleuret (2018)], or submodularity [Ramalingam et al. (2021)].

Deploying data selection methods in large-scale and dynamic environments is challenging, since data ingestion is a common bottleneck in ML training [Murray et al. (2021); Zhao et al. (2022)]. The selection algorithm itself has a runtime overhead. However, even assuming no cost for the selection algorithm, just fetching the data to ingest at sample-level granularity can significantly slow down data loading and bottleneck training, compared to sequentially reading input data. We are not aware of any training platform that supports sample-level data selection decisions, even for training on static datasets. We present Modyn, an open-source, extensible, modular, and easy-to-use platform for model training that addresses this gap. Modyn orchestrates continuous running pipelines over their lifecycle. We design Modyn to enable sample-level data selection and support basic triggering policies, while alleviating users from having to optimize the system infrastructure. Modyn separates the concerns of pipeline execution into several components. It uses several layers of partitioning and parallelization across components to avoid data stalls.[1]

2 System Design and Evaluation

Figure 1 shows Modyn's system architecture. Modyn takes data from a data generating source (e.g., a sensor or click stream) as input. Modyn outputs a stream of trained ML

1. A full preprint on Modyn is available at `https://arxiv.org/abs/2312.06254`.

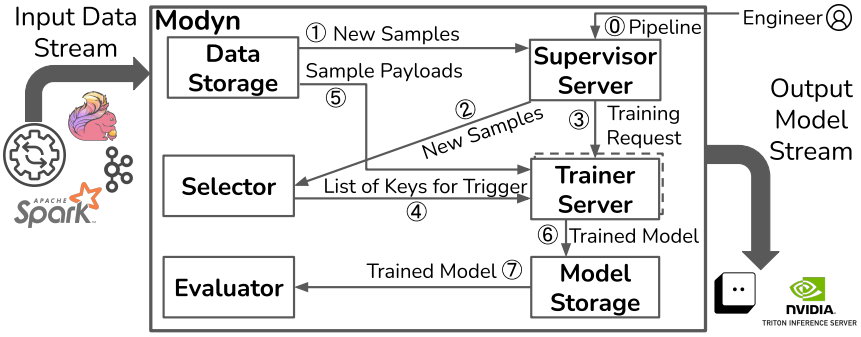

Figure 1: Modyn's system architecture.

models that can be deployed for serving. To balance performance and ease-of-use, Modyn components are either written in C++, purely in Python, or Python with C++ extensions. Having a clean, extendable Python interface allows ML researchers to implement their techniques without worrying about systems aspects. The codebase, totaling over 30,000 lines of Python and 5,000 lines of C++, is publicly accessible[2] and actively being developed.

Overview of data flow. ⓪ The user submits a training pipeline definition via Modyn's CLI to the supervisor server, which implements the triggering policy and orchestrates the training pipelines. Modyn stores data samples streaming in from external sources in its storage component, which assigns a unique key to each sample. ① The data storage component informs the supervisor server about new samples by their key. The supervisor checks whether any data point in the incoming batch causes a trigger and ② forwards potential triggers and the sample keys to the selector, which implements the data selection policy. ③ Upon trigger, the supervisor contacts the trainer server to start a training process. ④ The trainer requests the trigger training set (keys and weights to train on) from the selector. ⑤ Then, it loads the actual data from the storage and, depending on the retraining policy, also the previous model from the model storage. The trainer then runs a training according to the configuration. ⑥ The trained model, which is the output of Modyn, is then stored in the model storage component. ⑦ Afterwards, in an experimental setup, the evaluator component evaluates the newly trained model.

Evaluation. We compare Modyn to local training in order to measure the overhead of a well-performing configuration. We train a DLRM model since this is memory-bound and stresses Modyn's infrastructure the most. In order to compare, we use Modyn's training loop and replace its network data loader with a custom local dataset reading data directly from 90 binary files on local NVMe containing 30 M samples. Note that this not only removes the communication and gRPC overhead, but also removes the sample-level data selection. Modyn loads each sample individually by key, but the baseline loads entire files and emits all samples in them. Modyn reaches 92 %, 87.2 %, 82.6 %, and 87 % of local performance for 1, 4, 8, and 16 workers, respectively.

Research outlook. Modyn enables the data-centric ML community to easily explore retraining and data selection policies at scale, as it optimizes the infrastructure under the hood for high-throughput sample-level data selection. It offers a platform to validate their novel selection policies under continuous training scenarios experiencing distribution shift.

2. Available at `https://github.com/eth-easl/modyn`

We hope to foster cross-community collaboration and development in an open-source process.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Chip Huyen. Real-time machine learning: challenges and solutions. `https://huyenchip.com/2022/01/02/real-time-machine-learning-challenges-and-solutions.html`, 2022.

Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018. URL `http://proceedings.mlr.press/v80/katharopoulos18a.html`.

Derek G. Murray, Jiří Šimša, Ana Klimovic, and Ihor Indyk. tf.data: a machine learning data processing framework. *Proceedings of the VLDB Endowment*, 14(12), 2021. doi: 10.14778/3476311.3476374.

Ameya Prabhu, Philip H. S. Torr, and Puneet K. Dokania. GDumb: A simple approach that questions our progress in continual learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020. doi: 10.1007/978-3-030-58536-5_31.

Srikumar Ramalingam, Daniel Glasner, Kaushal Patel, Raviteja Vemulapalli, Sadeep Jayasumana, and Sanjiv Kumar. Less is more: Selecting informative and diverse subsets with balancing constraints. 2021. doi: 10.48550/arXiv.2104.12835.

Shreya Shankar, Rolando Garcia, Joseph M. Hellerstein, and Aditya G. Parameswaran. Operationalizing machine learning: An interview study, 2022.

Mark Zhao, Niket Agarwal, Aarti Basant, Buğra Gedik, Satadru Pan, Mustafa Ozdal, Rakesh Komuravelli, Jerry Pan, Tianshu Bao, Haowei Lu, Sundaram Narayanan, Jack Langman, Kevin Wilfong, Harsha Rastogi, Carole-Jean Wu, Christos Kozyrakis, and Parik Pol. Understanding data storage and ingestion for large-scale deep recommendation model training. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, 2022. doi: 10.1145/3470496.3533044.