Willump: Statistically-Aware Optimizations for Fast Machine Learning Inference

Peter Kraft, Daniel Kang, Deepak Narayanan, Shoumik Palkar, Peter Bailis, Matei Zaharia

1 Overview

The increasing prominence of ML in data centers today has led to interest in systems for ML prediction serving, which perform ML inference and serve predictions to user applications [4, 17]. These systems, called model serving systems, typically approach ML inference as an extension of conventional data serving workloads, missing critical opportunities to exploit the statistical nature of ML inference. Most of these systems, such as Clipper [4], Amazon Sagemaker [1], and Microsoft AzureML [2], treat ML inference as a black box and implement generic systems optimizations such as caching and adaptive batching. Some, such as Pretzel [10], also apply traditional compiler optimizations.

These optimizations are useful in ML inference, just as they are for web applications or database queries. However, systems for optimizing ML inference can leverage two key properties of ML not found in general data serving:

- **ML models can often be approximated efficiently on many inputs:** For example, the computer vision community has long used "model cascades" where a low-cost model classifies "easy" inputs and a higher-cost model classifies inputs where the first is uncertain, resulting in much faster inference with negligible change in accuracy [16, 18]. By contrast, existing multi-purpose model serving systems handle all data inputs the same way.
- **ML models are often used in higher-level application contexts**, such as top-K queries. However, existing model serving systems are unaware of these query modalities. As we show, tailoring inference to the query (for Willump, top-K queries) can improve performance.

To leverage these opportunities for optimization, we present Willump, a statistically-aware end-to-end optimizer for ML inference. Willump targets a common class of ML inference applications: those whose performance bottleneck is feature computation. In these applications, a pipeline of transformations converts raw input data into numerical features used by an ML model to make predictions. These applications are common, especially when performing ML inference over tabular data. For example, Pretzel's study of ML inference at Microsoft found that feature computation accounted for as much as 99.7% of the runtime of some production Microsoft ML inference applications [10]. Willump improves ML inference performance through two novel optimizations:

1) Automatic End-to-end Cascades: ML inference pipelines often compute many features for use in a model, but because ML applications are amenable to approximation, it is often possible to classify data inputs using only a subset of these features. For example, in a pipeline that detects toxic online comments, we may need to compute expensive TF-IDF vectorizations to classify some comments, but we can classify others simply by checking for curse words.

However, selectively computing features is challenging because features vary by orders of magnitude in computational cost and importance to the model and are often computationally dependent on one another. Therefore, one cannot pick an arbitrary set of features (e.g,. the least computationally intensive) and expect to efficiently classify data inputs with them.

To address these challenges, Willump uses a dataflow analysis algorithm and cost model to identify important but inexpensive features. With these features, Willump trains an approximate model that can identify and classify "easy" data inputs. For example, an approximate model for toxic comment classification might classify comments with curse words as toxic and cascade other comments to a more powerful model. Willump automatically tunes its own parameters to select the features that minimize expected query execution time while meeting a target accuracy level. The concept of model cascades has a long history in the ML literature [5, 7, 16], but to the best of our knowledge, Willump is the first system to automatically generate feature-aware and model-agnostic cascades from input programs. Willump's cascades deliver speedups of up to 5× on real-world ML inference pipelines without a statistically significant effect on accuracy.

2) Top-K Query Approximation: Willump automatically optimizes an important class of higher-level application queries: top-K queries. Top-K queries request a ranking of the K top-scoring elements of an input dataset [6]. They are fundamentally asymmetric: predictions for high-scoring data inputs must be more precise than predictions for low-scoring data inputs. Willump leverages this asymmetry by automatically constructing a computationally simple approximate pipeline to filter out low-scoring inputs.

Some ML recommender systems approximate top-K queries is a similar way using fast retrieval models [3]. However, they develop these models manually. Because top-K optimization is not automatic, existing ML model serving systems such as Clipper or Pretzel do not optimize top-K queries, instead naively scoring all elements of the input dataset. Unlike prior work, Willump automatically constructs its approximate models and automatically tunes their parameters to maximize performance while meeting a target accuracy level. Willump's

automatic top-K query approximation improves performance on real-world serving workloads by up to 10×, with negligible impact on pipeline accuracy.

Willump complements end-to-end cascades and top-K query approximation with powerful compiler optimizations. Willump compiles a subset of Python to machine code through the Weld system [12, 13], in the process applying optimizations such as loop fusion and vectorization. Compilation improves query throughput by up to 4×.

Willump users write ML inference pipelines as functions from raw inputs to model predictions in a dialect of Python. Specifically, these functions must register model training, prediction, and scoring functions, must be written as a series of function calls, and must represent data using NumPy arrays, SciPy sparse matrices, or Pandas dataframes. If an ML inference pipeline conforms to these rules, Willump can automatically parse it into a graph of transformations and optimize it.

2 Evaluation

We evaluate Willump on several pipelines curated from entries to major data science competitions hosted by Kaggle, CIKM, and WSDM, listed in Table 1. We first evaluate Willump on offline batch queries, showing results in Figure 1. First, we apply Willump's compiler optimizations. These improve the performance of all compilable benchmarks by up to 4.3×.

Then, we apply end-to-end cascades to all classification benchmarks. For all benchmarks, we set an accuracy target of 0.1% less than the accuracy of the original model, but we did not observe a statistically significant change in accuracy for any benchmark. End-to-end cascades improve benchmark performance by up to 5×.

Interestingly, cascades are least effective on `music`, which queries pre-computed features from an in-memory database. This is very fast when compiled, so feature computation accounts for a small portion of overall runtime and potential gains from cascades are limited. In `music-remote`, we moved the features to a remote database, so querying them was more costly and cascades became more effective, providing a 2.4× speedup.

We then evaluate Willump on top-K queries, showing results in Figure 2. We use $K = 20$, query over the entire validation set, and set a minimum precision of 0.95.

We first apply Willump's end-to-end compiler optimizations to compilable benchmarks; these perform the same as before. We then apply Willump's top-K query approximation optimization. This produces performance improvements ranging from 1.3-10× with precision always above the minimum. Smaller speedups occur in benchmarks with relatively expensive models, such as `music`, as well as in benchmarks where differences between scores of high-scoring candidates were small (less than a hundredth of a percent), like `product`.

Benchmark	Feature-Computing Operators	Prediction Type	Model
Toxic [15]	String processing, N-grams, TF-IDF	Classification	Linear
Music [14]	In-memory database lookup	Classification	GBDT
Music-Remote [14]	Remote (Redis) data lookup	Classification	GBDT
Product [11]	String processing, N-grams, TF-IDF	Classification	Linear
Instant [8]	Model Stacking	Classification	Ensemble
Purchase [9]	Automatically Generated Features	Classification	GBDT

Table 1. Properties of Willump's benchmark workloads.

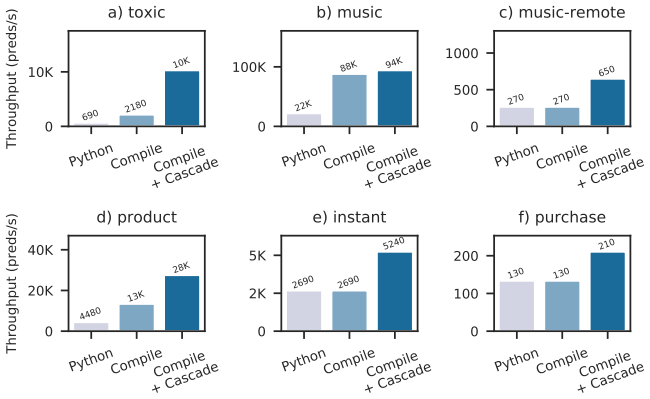

Figure 1. Willump performance on offline batch queries.

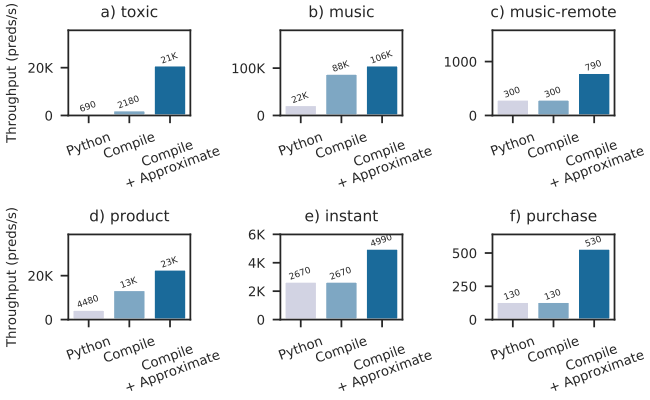

Figure 2. Willump performance on top-K queries.

Benchmarks with less expensive models and more differentiation between high-scoring candidates, like `music-remote` and and `toxic`, have larger speedups.

3 Contributions

In summary, we make the following contributions:

- We introduce Willump, a statistically-aware end-to-end optimizer for ML inference pipelines.
- We automatically cascade feature computation, improving ML inference performance by up to 5× without statistically significant accuracy loss.
- We automatically approximate top-K queries, improving performance by up to 10× with minimal accuracy loss.

4 Acknowledgments

This research was supported in part by affiliate members and other supporters of the Stanford DAWN project—Ant Financial, Facebook, Google, Infosys, Intel, Microsoft, NEC, SAP, Teradata, and VMware—as well as Toyota Research Institute, Keysight Technologies, Amazon Web Services, Cisco, and the NSF under CAREER grant CNS-1651570. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] 2019. Amazon SageMaker on AWS. https://aws.amazon.com/sagemaker/

[2] 2019. Azure Machine Learning Service. https://azure.microsoft.com/en-us/services/machine-learning-service/

[3] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 7–10.

[4] 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.

[5] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. 2018. Focus: Querying large video datasets with low latency and low cost. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 269–286.

[6] Ihab F Ilyas, George Beskales, and Mohamed A Soliman. 2008. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)* 40, 4 (2008), 11.

[7] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. Noscope: optimizing neural network queries over video at scale. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1586–1597.

[8] Prashant Kikani. 2019. IG PCA + NuSVC + KNN + LR stack. https://www.kaggle.com/prashantkikani/ig-pca-nusvc-knn-lr-stack

[9] Will Koehrsen. 2019. A Machine Learning Framework with an Application to Predicting Customer Churn. https://github.com/Featuretools/predict-customer-churn

[10] 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.

[11] Tam T Nguyen, Hossein Fani, Ebrahim Bagheri, and Gilberto Titericz. 2017. Bagging Model for Product Title Quality with Noise. (2017).

[12] Shoumik Palkar, James Thomas, Deepak Narayanan, Pratiksha Thaker, Rahul Palamuttam, Parimajan Negi, Anil Shanbhag, Malte Schwarzkopf, Holger Pirk, Saman Amarasinghe, et al. 2018. Evaluating end-to-end optimization for data analytics applications in weld. *Proceedings of the VLDB Endowment* 11, 9 (2018), 1002–1015.

[13] 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)*.

[14] rn5l. 2018. rn5l/wsdm-cup-2018-music. https://github.com/rn5l/wsdm-cup-2018-music

[15] Bojan Tunguz. 2018. Logistic regression with words and char n-grams. https://www.kaggle.com/tunguz/logistic-regression-with-words-and-char-n-grams

[16] Paul Viola and Michael Jones. 2001. Rapid object detection using a boosted cascade of simple features. In *null*. IEEE, 511.

[17] Wei Wang, Jinyang Gao, Meihui Zhang, Sheng Wang, Gang Chen, Teck Khim Ng, Beng Chin Ooi, Jie Shao, and Moaz Reyad. 2018. Rafiki: machine learning as an analytics service system. *Proceedings of the VLDB Endowment* 12, 2 (2018), 128–140.

[18] Xin Wang, Yujia Luo, Daniel Crankshaw, Alexey Tumanov, Fisher Yu, and Joseph E Gonzalez. 2017. Idk cascades: Fast deep learning by learning not to overthink. *arXiv preprint arXiv:1706.00885* (2017).