Demonstration of Accelerating Machine Learning Inference Queries with Correlative Proxy Models

Zhihui Yang[1], Yicong Huang[2], Zuozhi Wang[2], Feng Gao[1], Yao Lu[3], Chen Li[2], X. Sean Wang[4]
[1]Zhejiang Lab, Hangzhou, China; [2]UC Irvine, CA, USA;
[3]Microsoft Research, WA, USA; [4]Fudan University, Shanghai, China.
[1]{zhyang14,gaof}@zhejianglab.com, [2]{yicongh1,zuozhiw,chenli}@ics.uci.edu,
[3]luyao@microsoft.com, [4]xywangcs@fudan.edu.cn

ABSTRACT

We will demonstrate a prototype query-processing engine, which utilizes correlations among predicates to accelerate machine learning (ML) inference queries on unstructured data. Expensive operators such as feature extractors and classifiers are deployed as user-defined functions (UDFs), which are not penetrable by classic query optimization techniques such as predicate push-down. Recent optimization schemes (e.g., Probabilistic Predicates or PP) build a cheap proxy model for each predicate offline, and inject proxy models in the front of expensive ML UDFs under the independence assumption in queries. Input records that do not satisfy query predicates are filtered early by proxy models to bypass ML UDFs. But enforcing the independence assumption may result in sub-optimal plans. We use *correlative proxy models* to better exploit predicate correlations and accelerate ML queries. We will demonstrate our query optimizer called CORE, which builds proxy models online, allocates parameters to each model, and reorders them. We will also show end-to-end query processing with or without proxy models.

PVLDB Reference Format:
Zhihui Yang, Yicong Huang, Zuozhi Wang, Feng Gao, Yao Lu, Chen Li, X. Sean Wang. Demonstration of Accelerating Machine Learning Inference Queries with Correlative Proxy Models. PVLDB, 15(12): XXX-XXX, 2022.
doi:XX.XX/XXX.XX

1 INTRODUCTION

Consider an example workflow illustrated in Figure 1, where input tweets are processed by one ML UDF `Geotagger` ($\mathcal{F}_1$) followed by a predicate `state = 'CA'` (σ_1), and another ML UDF `Sentiment` ($\mathcal{F}_2$) followed by a predicate `sentiment = positive` (σ_2). It enables downstream visualization and statistics, such as word cloud. ML queries are costly due to the expensive ML UDFs; improving the efficiency for ML inference has been a recent research focus [4, 6]. In our example, classic query optimization techniques such as predicate push-down cannot help much because σ_1 and σ_2 are stuck behind their corresponding ML UDFs regardless of their selectivity.

Recent works [4, 6] propose to rewrite the query and insert a set of light-weight filters in the front of the expensive ML UDFs, thus forming a *proxy model* [8]. Figure 2 demonstrates an example plan with two proxy models $\hat{\sigma}_1$ and $\hat{\sigma}_2$; they quickly discard input records that are unlikely to satisfy the predicates and thus improve the query performance with an acceptable accuracy. In [6], an independence assumption is made to train filters and assemble these filters. But, query predicates are often correlated in many applications. In our example, sentiments may vary in different states – the sentiment in California can be different from that in Texas. The query optimization (QO) in [6] overestimates the reduction when building the filters and thus yields sub-optimal plans.

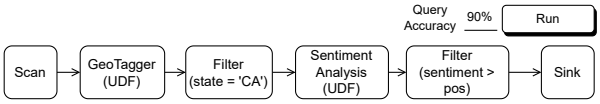

Figure 1: An example workflow for tweet analysis.

In our prior work [10], we proposed an optimizer called "CORE" that better exploits predicate correlations. By relaxing the independence assumption among different predicates, a proxy model is specific not only to a predicate but also its input relation, i.e., prefix σ's and $\hat{\sigma}$'s, as well as parameter choices of prefix $\hat{\sigma}$'s. In Figure 2, $\hat{\sigma}_2$ learns upon filtering the raw input by $\hat{\sigma}_1 \wedge \sigma_1$[1]. Enumerating and building proxy models with different orders and parameter choices offline result in infeasible building and storage costs. CORE builds proxy models *online* to avoid exhaustive offline filter construction.

This demonstration will show an *online* query processing engine on top of CORE, which builds proxy models using a small portion of the input data, and executes the optimized plan on the remaining data. Users will be able to interact with our system in various ways including submitting new queries and comparing performance with or without proxy models. We will also show a correlation score for a new query, and show end-to-end execution of a new query. Performance improvements by up to 63% compared to [6] and by up to 80% compared to running the workflow as it is can be observed in this interaction. In addition, CORE builds proxy models online for a new query and leverages a branch-and-bound search process to reduce the building costs. We will demonstrate CORE including building proxy models, allocating accuracy parameters to proxy models, and reordering with proxy models. Users will be able to interact with the query optimizer by specifying an order or deciding accuracy parameters for proxy models without waiting for the optimizer to finish to accelerate the QO phase.

2 THE DEMONSTRATION SYSTEM

We will use a prototype on top of Texera [9], which is an open-source big data analysis system using Web-based workflows. We

[1]$\mathcal{F}_1$ is a row processor and does not filter as σ_1 and $\hat{\sigma}_1$ do.

Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:XX.XX/XXX.XX

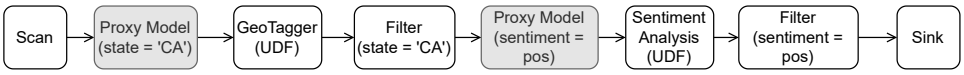

Figure 2: An optimized query plan with proxy models.

implement an operator library by deploying them as UDFs utilizing Texera's feature that supports Java and Python UDFs. Table 1 shows a partial list of operators provided in our system. These operators depict row manipulators; they produce one output row per input row. Using these operators, developers are able to construct a workflow using a Web UI. They can also interact with the system by choosing to execute the workflow as it is, execute it using PPs, or execute it using our CORE.

Module Name	Description
Entity Recognition	Label sequences of words in a text, such as person or company names.
Sentiment Analysis	Predict the sentiment of a text, such as positive or negative.
Pos Tagger	Assign parts of speech to each, such as noun or verb.
Object Detection	Identify objects in an image or video, such as a dog or a cup.
Activity Recognition	Predict the movement of a person, such as applying lipstick.

Table 1: A partial list of ML modules provided in the system.

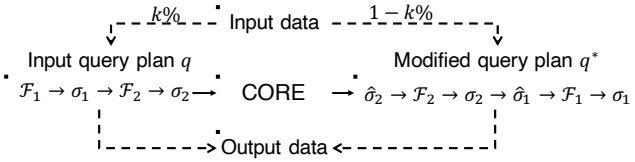

Figure 3: Given a query plan q, CORE generates an optimized plan q^* by applying proxy models. Part of the input data (k%) is used for building proxy models, and the remaining data is processed by q^*.

Figure 3 shows the architecture of the system. The input of CORE is a workflow that includes multiple ML UDFs, such as the workflow in Figure 1. We borrow the AQP-style query interface in [6]. Users are able to specify a global target accuracy $\mathcal{A}$, which sets a trade-off goal between an acceptable accuracy and query-processing speedup. In Figure 1, $\mathcal{A}$ = 90%. CORE optimizes the input workflow by building proxy models online. A proxy model is specific to a predicate $\mathsf{c}\phi\mathsf{v}$, where c is a predicate column, ϕ is a comparison (e.g., > or =), and v is a constant value. A workflow can have one or more predicate clauses in conjunction: $\bigwedge \mathsf{c}\phi\mathsf{v}$. CORE builds a proxy model for each predicate online, considers proxy models' combinations, allocates their accuracy parameters, and injects them into the modified query plan $q*$ (Figure 2). A small portion of the input data (e.g., k%) is used to build proxy models, and the remaining data is processed by the optimized plan q^*. We follow the scope of previous papers such as NoScope [4] and PP [6] to focus on approximate selection queries.

Key technical challenges: The demonstration system has been built to answer the following technical questions to reduce the overhead of building proxy models online.

- *Building proxy models online.* Enumerating and building proxy models offline result in infeasible building and storage costs. To build a $\hat{\sigma}$ online, the demonstration system generates its labeled sample L by pulling initial records from the input, filtering these records by its input relation d, and then labeling L using its predicate σ. Sample L is divided into a training set, a testing set, and a validation set. We re-sample the training data to ensure a label balance. Its classifier M is trained on the training set and the testing set using light-weight classification algorithms, such as a linear SVM and a shallow NN. During training, we leverage a grid-search on the F1-score to decide the best hyper-parameters and a cross-validation to train a classifier using the set of hyper-parameters. After that, we derive its accuracy versus reduction curve R using a validation set.
- *Allocating parameters for proxy models.* The system adopts a hill-climbing search framework to find an optimal accuracy allocation for proxy models. A main challenge is that building proxy models online is time-consuming because (i) there are an exponential number of candidates $\hat{\sigma}$s, and (ii) generating a labeled sample and training a classifier are computationally costly. The system reduces the online allocation overhead by reusing previously materialized samples and trained classifiers to reduce labeling costs and training costs, respectively.
- *Reordering proxy models.* Building all proxy models for different orders can be computationally prohibitive. We adopt a search algorithm based on branch-and-bound to prune candidate plans. Specifically, we compute a lower bound and an upper bound of costs $\sum C$ for a specific order of proxy models. During the search, we tighten the lower and upper bounds of $\sum C$ as we collect more information, such as selectivity and reduction, and prune candidate plans to reduce the optimization overhead. Additionally, we adopt a fine-grained search tree to improve the search process further.

3 DEMONSTRATION SCENARIOS

In this section, we illustrate various features supported in our demonstration system, including computing the correlation score of a workflow, interacting with the QO, executing a workflow using PP or CORE, and the end-to-end processing of a workflow. Details are presented as follows.

3.1 Correlated Workflows

In the demonstration, we will provide three datasets with text, images, and videos, an operator library as illustrated in Table 1, and several pre-constructed workflows over the three datasets with different correlation scores. These workflows retrieve texts, images, and videos that match given filter conditions, which are conjunctions of multiple clauses. Each filter is an equality condition on an ML-generated label column. Users will also be able to construct a workflow using the operator library.

Twitter text dataset. It contains 2M tweets from January 2017 to September 2017 in the United States. Each tweet is a string with

a maximum of 140 characters. We pre-constructed several workflows utilizing various NLP modules such as entity recognition and sentiment analysis to analyze tweets.

COCO image dataset. COCO is a public dataset collected online. It contains 123K images and 80 object classes such as "person," "bicycle," and "dog." Each image is labeled with multiple objects for their class labels and bounding box positions. We pre-constructed several workflows with different correlation scores to retrieve images that contained object classes specified in workflows.

UCF101 video dataset. UCF101 contains 13K videos collected from YouTube. Each video is labeled with one of 101 action categories such as "applying lipstick" and "baby crawling." We pre-constructed several workflows utilizing object detection and activity recognition models to retrieve videos with specific labels.

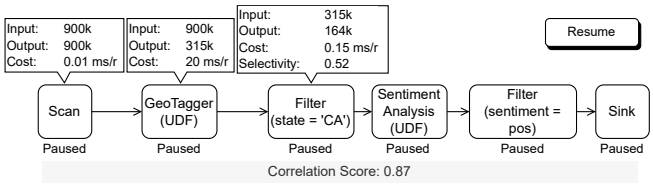

Figure 4: A paused workflow demonstrating its correlation score and the runtime metrics of the first three operators.

Using the interface, we will show the correlation score of a workflow. The correlation score of the workflow in Figure 1 is 0.87 as illustrated in Figure 4. When users start running the workflow, the system displays each operator's runtime metrics, such as the number of processed records, cost, and selectivity. Users will be able to pause the workflow, observe each operator's runtime metrics, and resume the workflow. In Figure 4, the user paused the workflow and observed the first three operators' runtime metrics. The user observes that the cost of the `Geotagger` is 20ms per record, and the selectivity of the `state = 'CA'` filter is 0.52. After that, the user resumes the workflow. Thus, the user will be able to gain valuable insights of the workflow.

3.2 Visualizing Query Optimization with CORE

In this scenario, we will use pre-constructed workflows over the Twitter dataset to demonstrate the query optimizer CORE including building proxy models online, allocating accuracy to each proxy model, and reordering with proxy models. We will show the optimization overhead of the workflow and how users can interact with our query optimizer CORE.

Given a workflow, the query optimizer CORE converts candidate query plans in the space $\mathbb{H}$ to a fine-grained tree. Figure 5 shows the fine-grained tree for the workflow in Figure 1. There are two types of tree nodes. (i) An L-node represents a pair of ML UDF and its filter, and it is to generate labeled samples. (ii) An M-node is to train its corresponding classifiers and derive an accuracy-versus-reduction relationships R. The root of the tree is the `CSV File Scan` operator in the workflow, which reads initial records from the input. Leaf nodes are `View Results` operators, which store results of the workflow. More details about the tree are in [10].

Finding an optimal order of proxy models and allocating their accuracy parameters simultaneously is NP-hard. CORE searches over the tree using a branch-and-bound algorithm to prune candidate plans. Our demo will show each node's runtime metrics during the search process. For an L-node, we will show the number of records in the labeled sample, the number of processed records, cost, and selectivity. For an M-node, we will show the training cost and the accuracy-versus-reduction curve. CORE tightens the lower and upper bounds for different query plans as we collect information, such as selectivity and reduction. The demo will show each candidate's lower and upper bounds of $\sum C$.

In Figure 5, we illustrate the runtime metrics of five nodes, including three L-nodes and two M-nodes. From the runtime metrics of node 1, a user observes that the Geotagger processed 3.85K records, but it only keeps the first 2k records in its labeled sample. The lower and upper bounds of $\sum C$ for the order Geotagger $\rightarrow$ Sentiment become 6 and 829 after collecting the cost and selectivity of the `Geotagger`. From the runtime metrics of M-node 8, the user gains insight of the accuracy-versus-reduction curve. The user also observes that the lower and upper bounds of Geotagger $\rightarrow$ Sentiment become 37 and 352, respectively, after collecting the 0.46 reduction. We will also demonstrate pruned candidate plans. Their corresponding nodes in the fine-grained tree become gray, such as nodes 3, 5, and 6 in Figure 5.

After obtaining more insights about the workflow, a user is able to interact with CORE by specifying an order or deciding an accuracy parameter for a proxy model without waiting for the optimizer to reduce the QO overhead. Specifically, the user can select a query plan from candidates to build proxy models. In Figure 5, the user clicks the `Priority` button and node 4 to select the order Geotagger$\rightarrow$ Sentiment. After observing the accuracy-versus-reduction curve from an M-node, the user can decide an accuracy parameter for a node's proxy model by sliding an accuracy button in the curve based on the user's experience. For node 8 in Figure 5, the button is at a 0.46 reduction; the user can slide the accuracy button to another accuracy.

3.3 End-to-End Query Processing

We will use the pre-constructed workflows over the Twitter dataset to demonstrate the system. A user is able to submit a new workflow and specify a query accuracy $\mathcal{A}$. The demonstration system supports another two modes to run a workflow except using CORE. (i) ORIG runs the original workflow as it is, and (ii) PP builds a light-weight proxy model for each predicate, and injects them early in a plan. PP decides the accuracy parameter for each proxy model using a dynamic programming algorithm with an independence assumption of predicates. Users are able to run the workflow using ORIG, PP, and CORE, respectively.

The demonstration system uses a small portion of the input data to generate an optimal plan, and processes the remaining data using the optimal plan. Our demo will show the percentage of the initial input data used to build proxy models. After the demonstration system finishes running the workflow, it will show the total time of running the workflow including the optimization overhead, the QO cost percentage (i.e., the percentage of the QO time over the total processing time), and the execution cost. Thus, users are able to compare the performance of running the workflow with different modes. Our demo will show that, in general, both PP and CORE

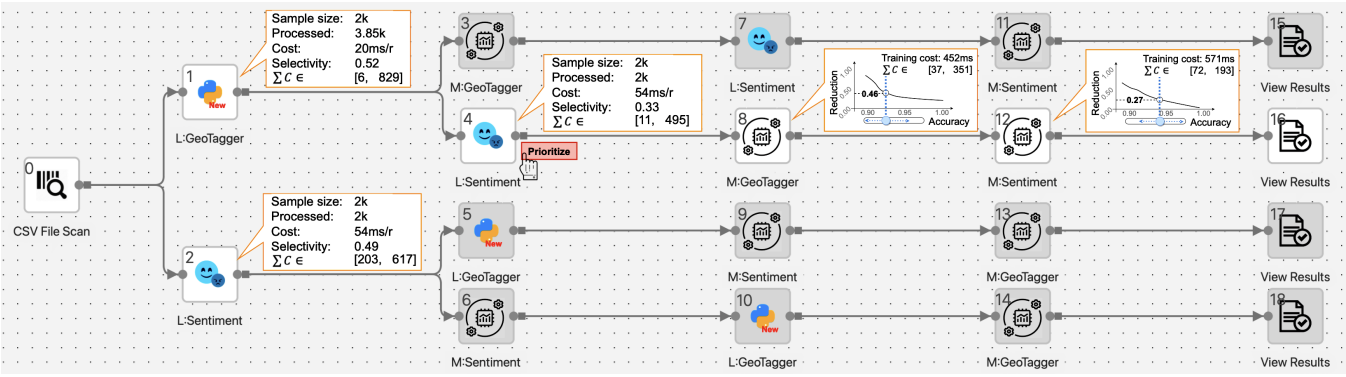

Figure 5: A fine-grained tree generated by CORE demonstrating runtime metrics of five nodes and the selected order Geotagger → Sentiment.

improve the performance of workflows, and CORE achieves more performance improvement on correlated workflows than PP.

4 RELATED WORK

Operator reordering in database optimization. [1] studied how to order correlated predicates in streaming systems. It used a greedy algorithm for selection ordering and collected samples at runtime to estimate selectivity. Our QO gives an optimal solution and uses a branch-and-bound search to quickly prune plans in the space of proxy models. [7] studied various optimization techniques of complex user-defined functions on map-reduce-style big data systems, such as predicate simplification and UDF semantic inference. These techniques were orthogonal to our solution. [2] provided approximate answers to queries by running queries on a small sampling subset of data. Our approach provides approximate answers by exploiting the accuracy of ML inference predicates.

Optimization with Proxy models (a.k.a. cascaded filters). Proxy models have been studied for decades to accelerate ML inference. Jones et al. [8] cascade weak classifiers as proxy models to speed-up face detection in images. Recently proxy models have been applied in big-data systems to accelerate ML inference-based analysis tasks [3–6]. NoScope [4] firstly cascaded a cheap specialized model before expensive DNNs to accelerate selection video queries. Certain classes of video queries including selection without guarantees [3] and selection with statistical guarantees [5] were optimized using proxy models. Probabilistic predicates (PPs) [6] optimized various domain queries by inserting multiple offline-built proxy models before expensive ML UDFs with an assumption of independence between predicates. Different from [3–5], PP and our proposed CORE cascade general proxy models, which are applicable to a variety of domains. CORE follows this line of work and further relaxes the independence assumption of the predicates.

5 CONCLUSIONS

This paper demonstrated a novel query optimizer, CORE, to accelerate ML inference queries. It improved state-of-the-art techniques by relaxing the independence assumption among different predicates. CORE incurs only a small overhead by leveraging a branch-and-bound search algorithm to prune the space of candidate filters and reusing intermediate results.

ACKNOWLEDGMENTS

This work was partially supported by the National Key R&D Program of China (No. 2020AAA0103903), the NSFC (No. 61732004) and the USA NSF award IIS-2107150. Thanks to the Wide-Area Joint Computing team at Zhejiang Lab for their contributions to developing the system.

REFERENCES

[1] Shivnath Babu, Rajeev Motwani, Kamesh Munagala, Itaru Nishizawa, and Jennifer Widom. 2004. Adaptive Ordering of Pipelined Stream Filters. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, June 13-18, 2004*. ACM, Paris, France, 407–418.

[2] Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. 2017. Approximate Query Processing: No Silver Bullet. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, May 14-19, 2017*. ACM, Chicago, IL, USA, 511–519.

[3] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodík, 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 2018, October 8-10, 2018*. USENIX Association, Carlsbad, CA, USA, 269–286.

[4] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Deep CNN-Based Queries over Video Streams at Scale. *PVLDB* 10, 11 (2017), 1586–1597.

[5] Daniel Kang, Edward Gan, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2020. Approximate Selection with Guarantees using Proxies. *Proc. VLDB Endow.* 13, 11 (2020), 1990–2003.

[6] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. 2018. Accelerating Machine Learning Inference with Probabilistic Predicates. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, June 10-15, 2018*. ACM, Houston, TX, USA, 1493–1508.

[7] Astrid Rheinländer, Ulf Leser, and Goetz Graefe. 2017. Optimization of Complex Dataflows with User-Defined Functions. *ACM Comput. Surv.* 50, 3 (2017), 38:1–38:39.

[8] Paul A. Viola and Michael J. Jones. 2001. Rapid Object Detection using a Boosted Cascade of Simple Features. In *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), with CD-ROM, 8-14 December 2001*. IEEE Computer Society, Kauai, HI, USA, 511–518.

[9] Zuozhi Wang, Avinash Kumar, Shengquan Ni, and Chen Li. 2020. Demonstration of interactive runtime debugging of distributed dataflows in Texera. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2953–2956.

[10] Zhihui Yang, Zuozhi Wang, Yicong Huang, Yao Lu, Chen Li, and X. Sean Wang. 2022. Optimizing Machine Learning Inference Queries with Correlative Proxy Models. *CoRR* abs/2201.00309 (2022).