Optimizing ML Inference Queries Under Constraints

Ziyu Li[1](✉), Mariette Schönfeld[1], Wenbo Sun[1], Marios Fragkoulis[2], Rihan Hai[1], Alessandro Bozzon[1], and Asterios Katsifodimos[1]

[1] TU Delft, Delft, The Netherlands
{z.li-14,m.a.e.schonfeld,w.sun-2,r.hai, a.bozzon,a.katsifodimos}@tudelft.nl
[2] Delivery Hero SE, Berlin, Germany
marios.fragkoulis@deliveryhero.com

Abstract. The proliferation of pre-trained ML models in public Web-based model zoos facilitates the engineering of ML pipelines to address complex inference queries over datasets and streams of unstructured content. Constructing optimal plan for a query is hard, especially when constraints (e.g. accuracy or execution time) must be taken into consideration, and the complexity of the inference query increases. To address this issue, we propose a method for optimizing ML inference queries that selects the most suitable ML models to use, as well as the order in which those models are executed. We formally define the *constraint-based ML inference query optimization problem*, formulate it as a Mixed Integer Programming (MIP) problem, and develop an optimizer that maximizes accuracy given constraints. This optimizer is capable of navigating a large search space to identify optimal query plans on various model zoos.

Keywords: Machine learning inference query · Constrained-based query optimization · Predicate ordering

1 Introduction

Machine learning (ML) is increasingly used to process unstructured documents (i.e. text, images, videos), or data streams [6,9,21,24]. Take, for instance, the scenario of a self-driving car: when it detects (at certain proximity) that a person is crossing the road, or that another car has turned its emergency lights on, the car has to trigger an emergency action (e.g., breaking hard). This can be modeled as a complex *ML inference query*, and represented as a Boolean expression [5,12]: (`road` $\wedge$ `person`) $\vee$ (`car` $\wedge$ `light`). The literals in the expression are combined using operations such as *and* (conjunction) and *or* (disjunction).

While ML models can be (and often are) tailored to specific inference tasks, there is a growing interest in the reuse and re-purposing of pre-trained ML models [8]. This shift, mostly motivated by computational, economic, and environmental considerations, is evident from the proliferation of public, pre-trained ML

I. Garrigós et al. (Eds.): ICWE 2023, LNCS 13893, pp. 51–66, 2023.
https://doi.org/10.1007/978-3-031-34444-2_4

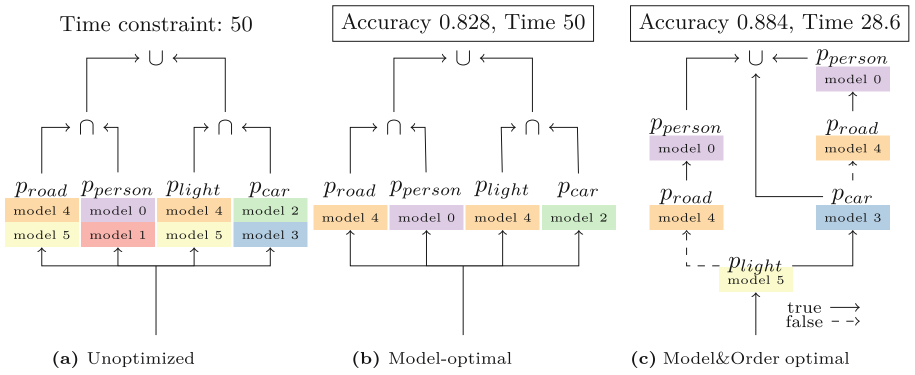


Fig. 1. Alternative ML query plans for the running example query.

model zoos on the Web, such as HuggingFace and PyTorch Hub[1]. These hubs contain thousands of pre-trained models for diverse ML inference needs such as object recognition, sentiment analysis or audio classification. These models are described by metadata detailing their inference capabilities (e.g. identified object classes), and performance (e.g. accuracy and execution time). With the help of the model zoos, ML inference query plans – i.e. complex workflows of ML models as the one shown in Fig. 1 – can be executed by leveraging existing ML models through easily accessible APIs, providing greater flexibility in defining ad-hoc queries.

ML inference queries are often subject to specific performance constraints (e.g. inference execution time, accuracy) [11,18]. In such cases, the selection of a set of models becomes quite complicated: an analyst may manually define a query plan that is excessively expensive and/or inaccurate if they lack considerable systems skills or time. Instead, an *optimizer* could automate the selection of (a set of) ML model(s) from the model zoo, so that the query could be answered under specific execution constraints. That way, data analysts/engineers can focus on the analytical task at hand, while ML researchers and engineers can independently focus on ML model development and enhancement.

Contributions. We propose a method (depicted in Fig. 2) to select the best ML models as well as their execution order, given a complex ML inference query and execution constraints. We formulate the problem of inference query optimization with constraints as a mixed integer program (MIP) and jointly optimize model assignment and predicate ordering (indicated as *model- & order-optimal* optimizer). The model assignment deals with the mapping of models to predicates, while predicate ordering decides the order in which to evaluate them. The contributions of this paper can be summarized as follows:

– We formulate the problem of ML inference query optimization as a (MIP) and propose a MIP-based optimizer that exploits model zoos.

[1] https://huggingface.co/, https://pytorch.org/hub/.

- Our approach is the first that jointly optimizes model assignment and predicate ordering, leveraging the selectivity (i.e., the probability of a predicate to evaluate `true`) of model-based predicates to decide their order of execution.
- We evaluate our `Bypass: Model- & Order-optimal` optimizer against baselines (Sect. 5), showing that our proposed optimizer can generate plans that significantly outperform the baselines in diverse model zoos on different constraint settings.

2 Related Work

ML Inference Query Optimization. The development of specialized models for fast inference of object detection queries has received considerable attention [7,8,17,20]. More recently, related research is targeting the processing efficiency of larger ML inference query [1,3,4,10,18]. NoScope [10] and PP [18] filtered irrelevant frames by training and deploying special lightweight binary classifiers, and Tahoma [1] trained model cascades to process video frames. The cheaper models are trained to achieve very low false negative rates, so that they did not filter out valid tuples/images/frames, since these can be validated by more accurate and expensive models downstream.

The most related work to ours is PP [18]. Our work is complementary to PPs, as it aims at reusing the plethora of existing models available in public and enterprise model zoos without retraining, and at optimally navigating the performance to accuracy trade-off of existing models. PP generates query plans for ML inference queries by first pre-selecting the predicates with a heuristic solution before optimizing the query plan, thus the query plan is suboptimal.

Multiple-objective Query Optimization. We model the ML inference query optimization problem presented in this paper as a multiple-objective query optimization problem with a bounded objective method. Notably, the problem at hand can also be modeled with other methods for multiple-objective optimization [15,19,22,23], which seek to find the set of query plans that dominate all others in terms of the trade-off between two conflicting objectives. However, the problem we tackle in this paper is different from the classic single- and multi-objective query optimization problems in existing literature due to the special treatment that accuracy requires as well as the consideration of predicate ordering in our specific problem setting.

3 Problem Definition

In this section, we define the notions of a *model zoo* and its metadata, and *ML inference query*. We also formalize the *ML inference query optimization problem*. Note that in this work, we consider the case of ML models that perform *classification* tasks.

Table 1. Execution time C of models in a model zoo.

	p_{road}	p_{person}	p_{light}	p_{car}
model 0	∞	25	∞	∞
model 1	∞	35	∞	∞
model 2	∞	∞	∞	20
model 3	∞	∞	∞	40
model 4	5	∞	5	∞
model 5	10	∞	10	∞

Table 2. Example accuracy A of models in a model zoo.

	p_{road}	p_{person}	p_{light}	p_{car}
model 0	0	0.90	0	0
model 1	0	0.95	0	0
model 2	0	0	0	0.91
model 3	0	0	0	0.93
model 4	0.94	0	0.91	0
model 5	0.96	0	0.95	0

3.1 Metadata of a Model Zoo

We formalize the metadata representation of a model zoo [14] as $\mathcal{R}(M, I, P, A, C)$, where M denotes the set of pre-trained ML models; I denotes the set of classes that M can infer; P denotes the corresponding set of a Boolean predicates over the inference classes I; A and C represent the matrices with the dimensions of $|M| \times |P|$, which store the values of model accuracy and execution time, respectively. C is depicted in Table 1 while A is depicted in Table 2. In the following, we explain how we utilize the metadata of a model zoo as prior information in ML inference query optimization in Sect. 4.

3.2 ML Inference Queries

Given a model zoo $\mathcal{R}(M, I, P, A, C)$, we write an ML inference query in the form of $(p_1 \wedge ... \wedge p_i) \vee ... \vee (p_j \wedge ... \wedge p_k)$, where each p_l is a Boolean predicate representing the inference class inferred by the ML model m_l $(1 \leq l \leq k)$. According to the closed-world assumption, we assume that an input ML inference query Q can be answered by a given model zoo $\mathcal{R}$. Note that it is possible that one model is selected for multiple predicates.

CNF and DNF Queries. In above definition, Q is in the *disjunctive normal form (DNF)*, where the clauses $Q_1 \vee \cdots \vee Q_l$ are connected by disjunctions. An ML inference query Q and its subqueries Q_i are Boolean queries. In the rest of the paper, for brevity, we will refer to *ML inference queries in CNF* simply as *CNF queries* (similarly for the DNF ones).

3.3 ML Inference Query Plan

We define a ML inference query plan as the orchestration of ML models supporting the execution of a ML inference query. Note that each predicate can be associated with several models before optimization (Fig. 1(a)). Figure 1(b) presents the query plan with an optimized model assignment, where each predicate is covered by a model. All the models process all the data and results are generated with union. We call this type of query plan a *sequential* plan. Figure 1(c) depicts a plan with optimized model assignment and execution order as a bypass plan [13], where we refer to this type of query plan as *bypass* plan.

3.4 Problem Definition

Given a ML inference query Q, we aim for an optimization target that maximizes the accuracy with constraint on the execution time. The *Accuracy-maximizing Model Assignment (AMA) problem* is defined as follows: given a model zoo $\mathcal{R}$, an ML inference query Q, and an upper bound C_{bound} on execution time, the goal is to assign a model $m \in M$ for each predicate $p \in P$, which maximizes the accuracy a_Q with the constraint of execution time c_Q. Formally, the objective function to optimize is:

$$\begin{aligned} \text{Maximize: } & a_Q = f_{acc}(Q) \\ \text{Subject to: } & c_Q \leqslant C_{bound} \end{aligned}$$

In the above definition, we denote the function to compute a_Q as $f_{acc}(Q)$, which is detailed in Sect. 4.2. The cost of the query plan c_Q is measured by the average inference time on one data instance. C_{bound} represents the given execution time bound that the computation cost of the query should respect.

In a similar way, we can define the *Execution-time-minimizing Model Assignment (EMA) Problem*, where the goal is to assign a model $m \in M$ for each predicate $p \in P$, which minimizes the average execution time on each tuple, i.e., c_Q, with the constraint that the minimum accuracy of the query a_Q stays above a lower bound A_{bound}. We do not detail this version of the problem, for the lack of space, but the interested reader can refer to an extended version of our paper[2].

4 Optimizing ML Inference Queries

Given an ML inference query, the goal is to generate query plan which maximizes the accuracy and satisfies the constraint on execution time. In this section, we outline our optimization and execution workflow for ML inference query in Sect. 4.1. We then present a mixed-integer programming formulation (Sect. 4.2–4.6) for the ML inference query optimization problem as defined previously, including accuracy model, execution time model, objective function, and other relevant components. Due to space limit, we refrain from including the implementation details such as formulation equations and linearization of quadratic terms. Instead, we provide descriptions of the key components and refer the reader to our extended paper version.

4.1 Approach Overview

As depicted in Fig. 2, users can define an ML inference query with ML model-based predicates. To optimize the query, our MIP-based optimizer leverages the metadata of a model zoo containing information about the available models and their performance in terms of accuracy and execution time. The input of our query optimizer also includes the metadata about *selectivity*, i.e., statistics

[2] Extended version: https://www.wis.ewi.tudelft.nl/assets/files/opt-ml-query.pdf.

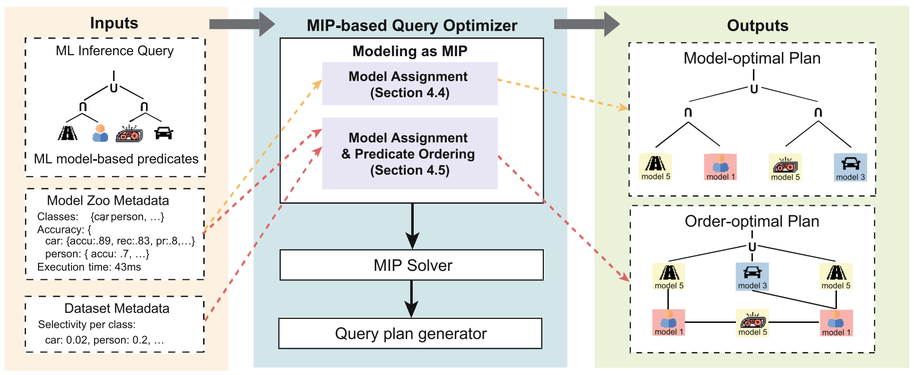


Fig. 2. Approach overview.

regarding the portion of data that a predicate returns as `true`. Both types of metadata are retrieved from a metadata management tool (e.g. [14]). The query optimizer then parses and optimizes the query. Given different input information, the MIP-based optimizer applies different optimization approaches to generate plans that satisfy the constraints.

Modeling as Mixed Integer Programming. The first step in the optimization phase is mathematical modeling, where the optimizer takes in different types of metadata and formulate their relationships. To tackle the *Accuracy-maximizing Model Assignment* (AMA) problem in Sect. 3.4, we resolve *model assignment*, i.e., mapping between ML models and predicates, and *predicate ordering*, i.e., deciding the execution order of predicates.

- *Model assignment.* With the model zoo metadata alone (yellow dashed arrows), the optimization only assign models to predicates adhering to an execution time constraint.
- *Predicate ordering.* To exploit the execution time budget and aim for higher effectiveness, we adopt *bypass* [13] plans and predicate ordering. The bypass plan consists of branches that execute only a defined subset of data, filtered based on prior outcomes. Bypass plans can greatly reduce execution cost by preventing the execution of models on unnecessary data. Together with predicate ordering, we manage to further increase efficiency and take full advantage of the budget by assigning better models for higher effectiveness with the available resource. The optimizer makes use of the selectivity metadata (red dashed arrows). We assume that selectivity is a property of an existing labeled dataset, and is known in advance.

In this work, we jointly optimize model assignment and predicate ordering given time constraints, and have shown significant performance for the objective goal (see Sect. 5 for details).

MIP Solver and Plan Generation. After modeling, we take the constraints and variables, and feed them to a MIP solver. We use *Gurobi* as the optimization

Algorithm 1: BypassPlanGen

```
Input  : query $query$, model-predicate mapping $mapping$,
         execution order of predicates (random or optimized) $order$,
         indication of the current branch $flag$
Output: bypass plan $plan$
1  $plan$ = NULL
2  predicate $p \leftarrow$ the first predicate in the $order$
3  if $p$ is not empty then
4      current node $m = mapping[p]$
5      $suborder \leftarrow$ the remaining order after removing $p$
       // Positive branch
6      $subquery \leftarrow$ subquery of $query$ where $p$ is substituted with true
7      $pos\_branch$ = BypassPlanGen($subquery$, $mapping$, $suborder$, true)
       // Negative branch
8      $subquery \leftarrow$ subquery of $query$ where $p$ is substituted with false
9      $neg\_branch$ = BypassPlanGen($subquery$, $mapping$, $suborder$, false)
       // Generate the plan as a binary tree ([root node, left child, right child])
10     $plan = [m, pos\_branch, neg\_branch]$
11 end
12 return $plan$
```

solver. The outcomes of the solver is optimized model assignment, i.e., mapping between models and predicates, as well as the execution order of the predicates.

Given the model assignment and predicate execution order, the plan generator produces plans in different mechanisms, e.g., sequential plan with `Model-optimal` plan and bypass plan with `Model- & Order-optimal` plan. Sequential plan is a set of ML models executing on all the data. The execution order does not have an impact on the results. Conversely, in bypass plans, models process the data with filtering conditions, allowing the data flow to be divided based on the `true` or `false` results of the predicates. Algorithm 1 presents the pseudo code for generating the bypass plan. The algorithm generates a binary tree as a bypass plan, with the ML models represented as nodes and the predicate filtering conditions indicated by the edges. The root node processes full set of data while the child nodes processes data filtered with different conditions.

4.2 Modeling Accuracy

We now explain the procedure of estimating query accuracy a_Q, i.e., $f_{acc}(Q)$ in the problem definition. The intuition is that the query performance is dependent on the performance of models assigned to the predicates. We assume that predicates are independent to each other (the same assumption made in [18]), i.e., the outcome of one predicate does not impact the performance of others. The accuracy of a conjunctive query, e.g., `road` $\wedge$ `person`, can be estimated by multiplying the accuracy of each model, $a_{road} * a_{person}$. The accuracy of a disjunctive query, e.g., `car` $\vee$ `bus`, can be computed using the inclusion-exclusion principle, as $a_{car} + a_{bus} - a_{car} * a_{bus}$. In the same way, we can calculate the accuracy of more complex Boolean expressions. It is worth noting that the accuracy model is contingent upon the independence assumption, and serves as an indicator of query performance. The actual, real-world query results may be impacted by predicate correlation: when two predicates have high correlation, the performance of one model can influence the output of another. In future work we can leverage the correlated performance of a model (given the output of another) to align the estimation of query accuracy with its actual value.

4.3 Modeling the Execution Time

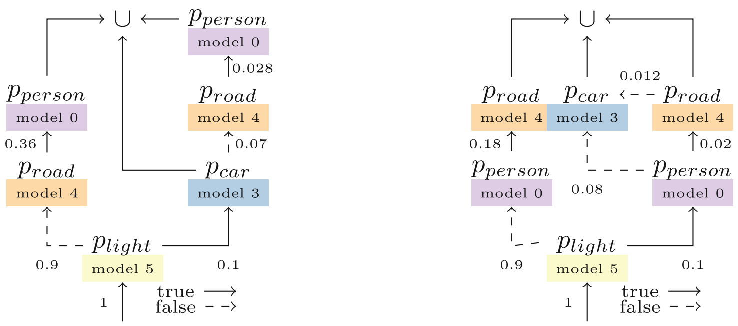
The measurement of execution time is determined by the form of the outcome plan, i.e., *sequential* (a set of models processing all the data) and *bypass* (models processing different subset of data based on the outcomes of the previous executed ones). Execution time is denoted by $f_{time}(Q)$.

Sequential Plan. In this case, the optimization does not take into account selectivity. The execution plan is a set of selected models executing on complete data. When computing the execution time, we only need to consider whether a model is selected, and we sum the cost of all the selected models. The models' execution time should be measured only once: the model can be executed once on the input and can output predictions for multiple classes.

Bypass Plan. In this case, not every model needs to process all the data: models in the subsequent steps only have to process a subset of the origin data filtered on the outputs of the previously executed models. The plan's execution time for this mechanism is measured with the sum of all the selected model cost proportioned to the data it need to process. For example in Fig. 1c, p_{car} processes images from two different data flows: images with `light` but without `person` (`light` $\wedge$ $\neg$`person`), and images with `light` and `person` but without `road` (`light` $\wedge$ `person` $\wedge$ $\neg$`road`). The execution cost of answering p_{car} is the execution cost of running the model proportioned to the amount of data it need to process, which is determined by the input data flows. The key challenge is to determine the portion of data processed by each predicate, which we will tackle in Sect. 4.5.

4.4 Modeling Model Assignment

Model assignment is the mapping between models and predicates. It determines the models used to answer predicates. To perform model assignment, we set a few constraints: *i*) we need to allocate exactly one model to each predicate; *ii*) only models with non-zero accuracy on a predicate can be assigned. Note that a model can be assigned to answer multiple predicates. Figure 1(b) presents the plan that only takes into account of model assignment that maximizes the accuracy given the time constraint.



(a) Model & Order-optimal bypass plan (**cost: 28.55)**

(b) Bypass plan with random predicate order (**cost: 38.4**)

Fig. 3. Bypass plans with different predicate execution order (numbers near by the arrows indicating the selectivity of the predicates in that path).

4.5 Modeling Predicate Ordering

Predicate ordering has a significant impact when we generate a bypass plan. If the plan is a sequential execution of models without filtering any data, the results are the union of the predictions of all the models and the execution order will not make an effect on the results. On the other hand, a predicate in a bypass plan can filter insignificant data, which results in different execution cost when adopting a different execution order of the predicates.

The emphasis of predicate ordering is to measure the selectivity of predicates, clauses and subqueries, given a certain order, i.e., the portion of data that retained by the previous answered predicates. For example, consider a query $p_{road} \wedge p_{person}$. If the execution order is $p_{road} \rightarrow p_{person}$, the portion of data processed by p_{road} is 100%, while p_{person} the portion of data where p_{road} returns `true`. If p_{road} returns `false`, the whole query returns `false`, which ends the evaluation. The portion of data processed by p_{person} is thus the selectivity of p_{road}. When the execution order changes and p_{road} is answered before p_{person}, the amount of data being processed in general is different from the previous plan. Thus, when considering bypass plan, predicate ordering matters, and selectivity of predicates are taken into account.

Take the previous query as example. In Fig. 3, we present two bypass plans based on different predicate execution order. Though the model assignment is the same, the execution cost of these plans are different. Figure 3(b) shows the plan when we jointly optimize model assignment and predicate ordering. The predicate execution order follows $p_{light} \rightarrow p_{car} \rightarrow p_{road} \rightarrow p_{person}$, which achieves lower cost than random predicate order in Fig. 3(a).

4.6 Objective Function and Constraints

Our proposed `Model- & Order-optimal` approach has transformed the objective functions into the following forms. Given an execution time constraint (solving the AMS problem):

Maximize: $f_{acc}(Q)$
Subject to: Exactly one model is assigned to a predicate;
Only models with non-zero accuracy can be assigned to a predicate;
Execution time of the query plan $f_{time}(Q)$ is calculated depending on the type of output plan and execution order of the predicates;
$f_{time}(Q) \leqslant C_{bound}$

5 Experimental Evaluation

In this section, we empirically evaluate our method on both real and synthetic datasets, covering different modalities, i.e., texts and images. We first evaluate the efficacy of the optimizer with other competing methods on different datasets, and observe significant performance of our advanced optimizer. We then evaluate the optimizers' optimization time on a synthetic setting with different query sizes, which verifies the complexity of the problem.

5.1 Experimental Settings

Datasets and Evaluation Metrics. We used public datasets covering object detection in images with COCO [16] as well as sentiment analysis in text with TweetEval [2]. *COCO* contains 123K images and 80 distinct classes of objects, lending themselves to complex queries with multiple predicates. *TweetEval* is a corpus of tweets collected from Twitter. We focus on 18 inference classes, belonging to different categories, such as text sentiments, entity types, etc. We finetune some NLP models to fit Tweeteval to perform the tasks. We use F1-score to measure the quality of the models, and milliseconds per instance for execution time. Each dataset is divided into a validation set (60%) and a test set (40%). We use the validation set to measure selectivity on each dataset, as well as execution time. The query execution time shown in the following is obtained by executing the queries on the test set.

Model Zoos. We collected all of our pre-trained from HuggingFace (NLP tasks) and PytorchHub (object detection). To navigate the space of different model zoos that may be encountered in the real world, we manually curated different types of model zoos – each with different characteristics in terms of included models, the inference classes they support, as well as accuracy and performance characteristics. Those are summarized in Table 3:

– *Real-World:* Model Zoo ❶ contains 48 real-world models that can tackle NLP tasks. Each model in this model zoo, covers all inference classes of the NLP tasks. Model Zoo ❷ includes 33 models that can be used in object detection tasks in images; each model in this model zoo covers all object classes in COCO.
– *Synthetic: Model Zoo* ❸ , *Model Zoo* ❹ are derived from Model Zoo ❷. Each of the 33 models has 5 variants; to that end, we have introduced a 0–30% accuracy penalty to all models uniformly, while we have also added an execution time

Table 3. Summary of model zoos.

Repo. Name	Modality	Class Coverage	Performance Variation	Number of Models
Model Zoo ❶	Text	All	None	48
Model Zoo ❷	Image	All	None	33
Model Zoo ❸	Image	1	Accuracy, Cost	165
Model Zoo ❹	Image	13 (avg)	Accuracy, Cost	165

penalty of 0–50%. By applying these variations we obtain 165 models in total. These three model zoos differ in terms of the inference classes that the models can answer (see Table 3).

Optimization Methods. We compare four strategies for optimizing ML inference query given a certain constraint. Note that there are two ways to execute the query plans: in *sequential*, i.e., not applying bypass and executing the plans in sequence; and in *bypass*, i.e., executing the plan using the bypass mechanism, given a predicate execution order.

Baseline 1 - `Sequential: Greedy`. This optimizer applies greedy heuristic and loops over predicates and selects the model with the highest rank greedily, i.e., $\frac{accuracy}{cost}$ (similar to predicate ordering based on rank). The optimizer stops when every predicate is assigned to a model and the constraint is met.

Baseline 2 - `Sequential: Model-optimal`. The model selection optimizer relies on MIP to optimize the model assignment under constraints, as compared to the greedy optimizer that approximates model assignment.

Baseline 3 - `Bypass: Model-optimal.` This baseline extends Baseline 2. Given the model assignment optimized with `model-optimal` approach, this baseline generates bypass plan.

Proposed method - `Bypass: Model-` & `Order-optimal`. This approach jointly optimizes for both model assignment and predicate ordering and create a bypass plan. It takes into account of the *selectivity* of predicates in a dataset and creates bypass plans.

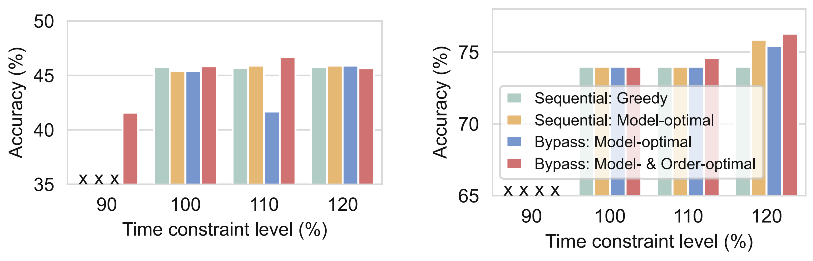
Queries. Since there are no benchmark queries that we could use from other works for our datasets, we adopted a similar approach as [18] to curate queries. We generate queries for two scenarios: comparing optimizer quality (Sect. 5.2, 5.3) and measuring optimization time (Sect. 5.4).

Optimizer Performance. We manually curated 10 queries (exemplified in Table 4) for image analysis (classes adopted from COCO), and 6 queries for text processing (tasks including name entity recognition, topic classification and sentiment analysis), in CNF and DNF forms. The queries range from 2 to 6 predicates with varying constraints on execution cost.

Query Optimization Time. We generate a set of queries in different complexity levels (the number of predicates ranging from 2 to 64), in total, 60 queries in CNF and DNF. The classes are adopted from COCO. For each predicate, we sample the classes with a uniform distribution.

Table 4. Examples of different ML inference queries (accuracy measured by F1-score, and cost measured by average inference time per instance).

Modality	Example Query	Constraint
text	e.g., ner=`person` ∧ sentiment=`negative` ∧ (topic=`news` ∨ topic=`sport`)	e.g., accuracy > 80%
image	e.g., `person` ∧ (`car` ∨ `bike`) ∧ `emergency_light`	e.g., cost < 100 ms



(a) Model Zoo ❶

(b) Model Zoo ❷

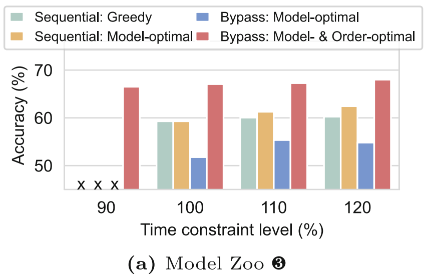
Fig. 4. Average speedups of query execution time compared to the *Greedy* approach on the query workload with different accuracy constraints.

Exec. Time Constraints. We create a number of experiment settings by enumerating different execution time bounds to verify optimizers' performance on different levels of constraints. We regard *Baseline 1* as the reference and record the minimum time constraint on which it can generate a query plan. The time constraints are set to be proportional to the minimum time constraint with scales of {80%, 90%, 100%, 110%, 120%} (we have observed that the performance converges from 120% onwards).

Hardware. We perform our experiments on a Ubuntu server with a single GPU (Nvidia A40, 8 GB RAM).

5.2 Using Uniform Model Zoos

We analyse the behavior of our optimizer using the model zoos Model Zoo ❶ and Model Zoo ❷. In this experiment we consider the constraint of 100% to be the execution time that allowed the `Sequential:Greedy` optimizer to find a solution to all the queries. We constrain the execution time to gradually increase from 90% - 120% to observe how the optimizers behave with different constraints. We present those results in bar plots (e.g., Fig. 4). The first observation is that when we put a low constraint on the execution time, our solution, `Bypass: Model- & Order-optimal`, succeeds to find proper solutions. Since the models used in both model zoos ❶ and ❷ have very similar accuracy, we do not observe large differences. It is worth noting that generating a bypass plan for the `Model-optimal`



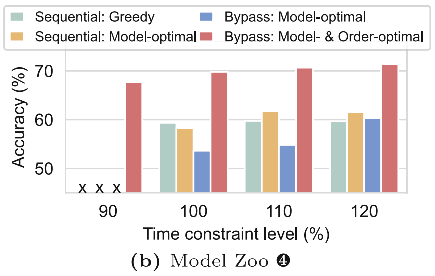


Fig. 5. The average accuracy on the query workload with different time (objective) constraint levels.

query plan can lead to a reduction in accuracy. This is because the random ordering of predicates can sometimes result in poor performance when a low-performing model is executed early in the process. Applying bypass plan can increase efficiency when executing the plan, however, with early filtering, this approach may wrongly filter data in an early stage, leading to decrease in accuracy.

5.3 Using Model Zoos with Diverse Model Distributions

We study the effect of diverse accuracy and execution time distributions, and class coverage in model zoos. More specifically, we run experiments using Model Zoo ❸ where each model answers exactly one inference class and Model Zoo ❹ average of 13 inference classes per model. We want to see if in such constrained environment the order optimizer can bring benefits.

Figure 5 shows the accuracy of all queries, for different values of execution time constraint. We observe that `Bypass: Model- & Order-optimal` consistently obtains higher query accuracy than the baselines. As in earlier experiment, bypass plans do not gain benefits when execution time is constrained. While `Bypass: Model- & Order-optimal` jointly optimizes for both model selection and predicate ordering can make use of predicate ordering and perform early filtering, making better use of execution time budget.

Results show that using bypass plans can lead to higher efficiency, while not necessarily increasing accuracy. The `Bypass: Model- & Order-optimal` optimizer outperforms the other baselines and can achieve higher query accuracy, especially given very diverse model zoos with different execution time and accuracy tradeoffs.

5.4 Query Optimization Time

We now evaluate the scalability of different approaches. We are interested in finding the limit of the `Bypass: Model- & Order-optimal` optimizer, with respect to the number of predicates that can be included in a query. Note that for brevity we exclude Baseline 3 (`Bypass: Model-optimal`): converting a given plan to its

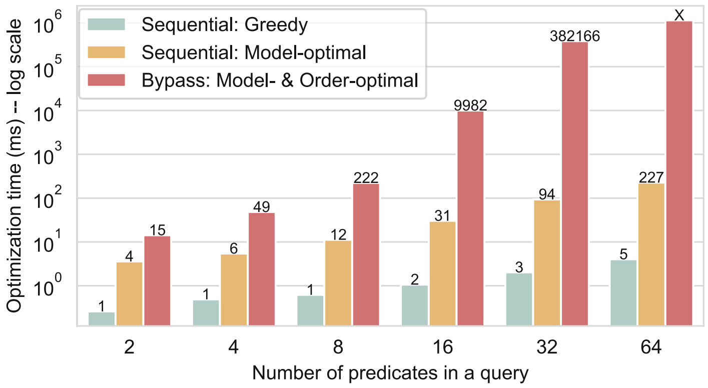


Fig. 6. Optimization time on queries with varying number of predicates.

bypass version requires a very small fraction of the optimization time. Thus, `Bypass: Model-optimal` in this case does not differ from `Bypass: Model- & Order-optimal`. We evaluate the efficiency of our optimizers in generating a query plan by varying the number of predicates in a query as shown in Fig. 6. The experiments were performed on Model Zoo ❹.

All the optimizers show exponential increase in execution time with the increase of predicate number in a query (Fig. 6 is plotted in log scale), except the `Sequential:Greedy` approach. The exponential increase also hints that the problem we are tackling has a very high complexity (Sect. 3). We observe that the advanced optimizers require much longer time to generate a plan as the number of predicates increases. In fact, when accuracy is constrained, the optimization time for 64 predicates did not finish (X). Future work can focus on applying approximation schemes to increase efficiency.

6 Conclusions and Future Work

In this paper we address the problem of ML inference query optimization, which aims for high accuracy given constraints on execution time. We formulate the problem as an MIP to perform optimal model selection and predicate ordering. Our optimizer that considers both model selection and predicate ordering achieves high performance, especially when the constraints are tight. In future work, we will consider additional objectives, such as model power consumption and memory footprint. Further research can focus on *i*) exploring multi-objective optimization problems, *ii*) applying approximation schemes in the MIP formulation of the problem and *iii*) lifting the assumptions made in this paper, considering especially the correlation of inference predicates and concept drift.

References

1. Anderson, M.R., et al.: Physical representation-based predicate optimization for a visual analytics database. In: 2019 IEEE 35th ICDE, pp. 1466–1477. IEEE (2019)
2. Barbieri, F., et al.: TweetEval: unified benchmark and comparative evaluation for tweet classification. arXiv preprint arXiv:2010.12421 (2020)
3. Cai, Z., et al.: Learning complexity-aware cascades for pedestrian detection. IEEE PAMI **42**(9), 2195–2211 (2019)
4. Cao, J., et al.: Thia: accelerating video analytics using early inference and fine-grained query planning. arXiv preprint arXiv:2102.08481 (2021)
5. Chang, J.Y., Lee, S.: An optimization of disjunctive queries: union-pushdown. In: Proceedings of COMPSAC, pp. 356–361. IEEE (1997)
6. Chowdhary, K., et al.: Natural language processing. In: Chowdhary, K.R. (ed.) Fundamentals of Artificial Intelligence, pp. 603–649. Springer, New Delhi (2020). https://doi.org/10.1007/978-81-322-3972-7_19
7. Howard, A.G., et al.: MobileNets: efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
8. Huang, J., et al.: Speed/accuracy trade-offs for modern convolutional object detectors. In: Proceedings of the IEEE CVPR, pp. 7310–7311 (2017)
9. Jiang, J., et al.: Chameleon: scalable adaptation of video analytics. In: Proceedings of SIGCOMM, pp. 253–266 (2018)
10. Kang, D., et al.: NoScope: optimizing neural network queries over video at scale. arXiv preprint arXiv:1703.02529 (2017)
11. Karanasos, K., et al.: Extending relational query processing with ml inference. CIDR (2020)
12. Kastrati, F., Moerkotte, G.: Generating optimal plans for Boolean expressions. In: IEEE ICDE, pp. 1013–1024. IEEE (2018)
13. Kemper, A., et al.: Optimizing disjunctive queries with expensive predicates. ACM SIGMOD Rec. **23**(2), 336–347 (1994)
14. Li, Z., Hai, R., Bozzon, A., Katsifodimos, A.: Metadata representations for Queryable ML model zoos. arXiv preprint arXiv:2207.09315 (2022)
15. Li, Z., et al.: Optimizing machine learning inference queries for multiple objectives. In: 39th ICDE Workshop on DBML. IEEE (2023)
16. Lin, T.-Y., et al.: Microsoft COCO: common objects in context. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8693, pp. 740–755. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10602-1_48
17. Liu, W., et al.: SSD: single shot multibox detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9905, pp. 21–37. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46448-0_2
18. Lu, Y., et al.: Accelerating machine learning inference with probabilistic predicates. In: Proceedings of the SIGMOD, pp. 1493–1508 (2018)
19. Papadimitriou, C.H., et al.: Multiobjective query optimization. In: Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2001, pp. 52–59. Association for Computing Machinery, New York (2001)
20. Redmon, J., et al.: YOLO9000: better, faster, stronger. In: Proceedings of the IEEE CVPR, pp. 7263–7271 (2017)
21. Shen, H., et al.: Fast video classification via adaptive cascading of deep models. In: Proceedings of the IEEE CVPR, pp. 3646–3654 (2017)

22. Trummer, I., Koch, C.: Approximation schemes for many-objective query optimization. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, pp. 1299–1310 (2014)
23. Trummer, I., Koch, C.: Multi-objective parametric query optimization. SIGMOD Rec. **45**(1), 24–31 (2016)
24. Zhang, H., et al.: Live video analytics at scale with approximation and delay-tolerance. In: 14th USENIX (NSDI), pp. 377–392 (2017)