Optimizing ML Inference Queries under Constraints

Ziyu Li Mariëtte Schönfeld Wenbo Sun Marios Fragkoulis† Rihan Hai
Alessandro Bozzon Asterios Katsifodimos
Delft University of Technology, †Delivery Hero SE
{z.li-14,w.sun-2,a.katsifodimos,r.hai}@tudelft.nl,marios.fragkoulis@deliveryhero.com

ABSTRACT

Machine Learning (ML) inference queries enable the execution of complex predicates over databases and streams of unstructured content. The increasing availability of ML model zoos allows for the reuse of pre-trained models in ad-hoc ML inference queries. But the problem of model assignment to the predicates is hard, especially when constraints (e.g. accuracy or execution time) have to be satisfied, and the complexity of the query increases. We propose a method for optimizing ML inference queries that aims at picking the best models to use in a given query, according to constraints on accuracy or execution time. We define the *constraint-based ML inference query optimization problem*, and formulate it as a Mixed Integer Programming (MIP) problem. We prove that the ML inference query optimization problem under constraints is NP-hard, and propose an optimizer aiming at high efficiency and effectiveness. Our optimizer can navigate a massive search space and find (close to) optimal query plans on various model zoos, outperforming baselines and complementing existing work on probabilistic predicates. Our approach can achieve up to 7x speedup on query execution time and 10% improvement in terms of accuracy performance.

1 INTRODUCTION

Machine learning (ML) is increasingly used across application domains such as video analytics [24, 58], autonomous driving [56], content moderation [16], traffic monitoring [41] and crowd detection [27]. ML models are commonly used to enable *ML inference queries* over unstructured documents (i.e. text, images, videos) that are stored in databases, or produced by data streams. An ML inference query evaluates the presence of (a combination of) known entities to filter the documents, or to trigger a specific action. Take, for instance, the scenario of a self-driving car: when the camera feed 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). Figure 1 exemplifies the inference query, as expressed in SQL.

Model Zoos. While ML models can be (and often are) trained for specific inference queries, there is a growing interest in the reuse and re-purposing of pre-trained ML models [22]. This shift, mostly motivated by computational, economic, and environmental reasons, is evident from the proliferation of public, pre-trained ML model zoos such as HuggingFace, Tensorflow Hub, and PyTorch Hub[1]. These model zoos contain thousands of pre-trained models for diverse ML inference needs (e.g. recognition of classes/objects/concepts). The models are described by metadata about their inference capabilities (e.g. identified object classes), and performance (e.g. accuracy and execution time). Thanks to model zoos, complex ML inference queries could be executed through the re-use of existing ML models, allowing for great flexibility in the definition of ad-hoc queries. In the example of Figure 1, ML models (e.g. `model1`) are chosen by a practitioner by hand. As the size of model zoos and the complexity of queries increase, a *query optimizer* could automate the assignment of a specific (set of) ML model(s) from the model zoo. 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.

```
SELECT  frame_id,
        mlpred(img, 'road', model5 ) AS road,
        mlpred(img, 'person', model0 ) AS person,
        mlpred(img, 'emergency light', model5 ) AS light,
        mlpred(img, 'car', model3 ) AS car
FROM    video_stream  -- tuple contains img, and frame_id
WHERE   (road AND person) OR (light AND car)
```

Figure 1: This query evaluates `mlpred` as a user-defined function (UDF). `mlpred` receives an image/frame (`img`) as input, an object class to identify (e.g. `person`), and an ML model (e.g. `model5`). The UDF, as a predicate, evaluates the presence of the class in the given image. For simplicity, we omit spatial predicates and other complex spatial relationships among detected objects.

Optimization. ML inference queries are often subject to specific performance constraints [28, 39]. If an ML inference query is evaluated over a live video, as in our example, model execution time is a fundamental objective as each frame should be processed within 1/30th of a second. Other applications [13, 18, 25, 26, 33, 55] impose multiple constraints (e.g. execution time *and* accuracy). A *query optimizer*'s job is to automatically assign ML models that can answer to constrained ML inference queries where various predicates may have conflicting time or accuracy objectives.

Classic query optimization [23, 30] and predicate ordering [19, 31] approaches do not apply in this setting. In ML inference queries, the mapping of a model to a predicate needs to happen dynamically, accounting for the capabilities of models in the model zoo, as well as the models' accuracy and performance tradeoffs. Differently to traditional relational predicates, ML models produce *probabilistic* outcomes. It requires a particular formulation to calculate the accuracy of ML models over a query plan that includes arbitrary combinations of conjunctive and disjunctive predicates.

Original Contributions. We describe an approach for the optimization of ML inference queries under constraints (Figure 2). The approach selects the best ML models for an ad-hoc query, given a cost (e.g., evaluate each data item within 100ms) or accuracy constraint (e.g., evaluate a query with 90% accuracy). We address the problem using a Mixed Integer Programming (MIP) approach, with

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

the optimization goal of either maximizing inference accuracy, or minimizing total execution time.

We model both ML model assignment and predicate ordering in a MIP formulation (indicated as *order-optimal* optimizer). The model assignment deals with the mapping of models to predicates, and predicate ordering decides the order in which to apply them. The contributions of this paper can be summarized as follows:

- We show that the problem of ML inference query optimization under constraints is NP-hard (Section 3).
- We formulate the problem at hand as a (MIP) and propose a MIP-based optimizer that takes into account model assignment (Section 5) and predicate ordering (Section 6). To scale the MIP formulation further, we show how to linearize product formulations.
- Our approach is the first to consider selectivity (the probability of an inference predicate to evaluate `true`) of model-based predicates to arrange their order of execution.
- We evaluate our order-optimal optimizer against two baselines and the work using probabilistic predicates (PP) [39] (Section 7), showing that our order-optimal optimizer can generate plans that significantly outperform the baselines in diverse model repositories on different constraint settings.

2 RELATED WORK

Multimedia Query Systems. Multimedia databases date back to the 90s starting with Fagin's Garlic system. They retrieved media from a database system according to a weight (i.e., accuracy in our case) [14], and CVQL [34], which dealt with queries over raw videos. Our system shares ideas and motivation with CVQL [34] and our modeling uses notions from Garlic [14]. However, our work does not consider fuzziness at query time, as Garlic did. Instead, our query optimizer considers accuracy during query planning, but considers the results of the classifier as correct at query time. Finally, modern systems like Velox [12], Macrobase [4], VideoStorm [59], SVQ [57] and others [21] can benefit from our optimizer.

ML Inference Query Optimization. The development of specialized models for fast inference of object detection queries has received considerable attention [6, 20, 22, 38, 45]. More recently, related research is targeting the processing efficiency of larger ML pipelines [2, 7, 8, 26, 39, 52]. NoScope [26] and PP [39] filtered irrelevant frames by training and deploying special lightweight binary classifiers, and Tahoma [2] trained models and constructed 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 [39]. The main idea behind PPs is to train specialized models to achieve a high data reduction rate in order to avoid the application of more expensive models up in the query tree. Our work is complementary, 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 greedy solution before optimizing the query plan, thus the query plan is suboptimal. We provide a general solution for the model selection and predicate ordering problem and it can be used to perform better plans than PP. We proved that with an experiment in Section 7.5.

Predicate optimization. Traditionally, relational database management systems have optimized disjunctive queries by converting Boolean questions into either CNF or DNF [23]. Disjuncts within Boolean factors or conjunctive predicates can be ordered optimally (locally) by ranking with the ratio between cost and selectivity [17, 32]. Works have proposed to apply bypass techniques to reduce cost by avoiding expensive predicates altogether [29]. The scope of this work extends the traditional problem by optimizing various objectives beyond cost, i.e., accuracy. The traditional selection predicates do not yield uncertainty values and thus do not cause accuracy issues.

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. In particular, our work can consider execution time as the objective and accuracy as the constraint, and vice versa. Notably, the problem at hand can also be modeled with other methods for multiple-objective optimization [42, 53, 54], which seeks 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.

Probabilistic Databases. Our work connects to probabilistic databases because the inference for each input is uncertain: it relates to the effectiveness of the selected model that identifies a predicate. Before a model is executed, the inference outcome is unknown. Therefore, we model the "metadata" of the ML models and generate query plans based on the metadata. Instead, probabilistic databases model the uncertainty probability of the tuples and generate query results according to an uncertainty bound [51].

3 PROBLEM DEFINITION

In this section, we define the notions of a *model zoo* and its metadata, and *ML inference query*. Also, we formalize the *ML inference query optimization problem* and discuss its complexity. In this work, we only consider ML classification models.

3.1 Metadata of a Model Zoo

A *model zoo* is a repository of pre-trained ML models [1, 47, 50, 60], also known as *model hubs*. Building model zoos has become a popular mechanism for sharing and developing ML models. In a recent work [36], we have designed a conceptual metadata model for model zoos, and developed an open-source tool[2] that retrieves, stores and searches such model metadata. We retrieved performance metadata such as inference *accuracy* (specific to a model's class), as shown in Table **??**, and *inference execution time*, as shown in Table 3.2. The tool also gathers dataset-related metadata such as the *selectivity* of a predicate per inference class, that is an estimation

[2] https://modelsearch.io

of the fraction of data instances (e.g. images) for a certain object class in the ML inference query in a given dataset. The construction and maintenance (update) of the model zoo and its metadata is orthogonal to this work, which we take as given inputs.

We formalize the metadata representation of a model zoo [36] $\mathcal{R}$ 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 predicate 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. Continuing with the running example in Figure 1, we have $M = \{model\ 0, model\ 1, \ldots\}$, $I = \{road, person, \ldots\}$, $P = \{p_{road}, p_{person}, \ldots\}$. A is depicted in **??** while C is depicted in subsection 3.2.

3.2 ML Inference Queries

Our formalization of *ML inference queries* is inspired by Unions of Conjunctive Queries (UCQs) [10]. UCQs are a fundamental class of database query language, which corresponds to the subset of relational algebra containing selection, projection, join and union (SPJU). Drawing upon UCQs, our goal is to express an inference need on a given dataset such as images, which is satisfied through the execution of one or more ML models on the input data items. Given a model zoo $\mathcal{R}(M, I, P, A, C)$, we write an ML inference query in the form of $(p_1(m_1) \wedge \ldots \wedge p_i(m_i)) \vee \ldots \vee (p_j(m_j) \wedge \ldots \wedge p_k(m_k))$, 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 (CWA), we assume that an input ML inference query Q can be answered by a given model zoo $\mathcal{R}$. Note that $m_1, \ldots, m_k$ are not necessarily distinct, since it is possible that one model is selected for multiple predicates. Consider the below query of running example. In Fig 3d, *model* 5 is selected for both p_{light} and p_{road}.

	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 1: Execution time C 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

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

$$q: \ (p_{\texttt{road}}(m_4) \wedge p_{\texttt{person}}(m_1)) \ \vee (p_{\texttt{light}}(m_4) \wedge p_{\texttt{car}}(m_3))$$

CNF and DNF queries. An ML inference query Q and its subqueries Q_i are Boolean queries. In above definition, Q is in the *disjunctive normal form (DNF)*, where the subformulas $Q_1 \vee \cdots \vee Q_l$ are connected by disjunctions. A sentence in DNF can be equivalently transformed to its *conjunctive normal form (CNF)*, where the subformulas are connected by conjunctions. For example, the equivalent CNF of q is:

$$(p_{\texttt{road}}(m_4) \vee p_{\texttt{light}}(m_4)) \ \wedge (p_{\texttt{road}}(m_4) \vee p_{\texttt{car}}(m_3)) \wedge$$
$$(p_{\texttt{person}}(m_1) \wedge p_{\texttt{light}}(m_4)) \ \wedge (p_{\texttt{person}}(m_1) \vee p_{\texttt{car}}(m_3))$$

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 Optimization of ML Inference Queries

Given an ML inference query Q, we aim for two optimization targets. The first target is the *execution time*: the goal is to select the models that minimize the execution time of the query. However, since accuracy and execution time may conflict, the query with the lowest execution time may also suffer from low accuracy. There are multiple ways to deal with conflicting objectives, such as multi-objective optimization [43]. *In this work, we deal with this conflict by establishing bounds: an upper bound on execution time, when optimizing for accuracy; and a lower bound for accuracy, when optimizing for execution time.* In the following, we formalize the definitions of these two problem variants.

Definition 1 (Accuracy-maximizing Model Assignment (AMA) Problem). *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. The form of the objective function 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 will be elaborated in Section 5.1. 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.

Use case. The problem in Definition 1 specifies the bounding of the execution time. It is a typical requirement in use cases where execution speed is of importance, as in the query of Figure 1.

Definition 2 (Execution time-minimizing Model Assignment (EMA) Problem). *Given a model zoo $\mathcal{R}(M, I, P, A, C)$, an ML inference query Q, and a lower bound on accuracy A_{bound}, 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}. The form of the objective function is:*

$$\begin{aligned} &\text{Minimize:} && c_Q = f_{time}(Q) \\ &\text{Subject to:} && a_Q \geqslant A_{bound} \end{aligned}$$

In the above definition, we denote the function to compute c_Q as $f_{time}(Q)$, which will be elaborated in Section 5.2.

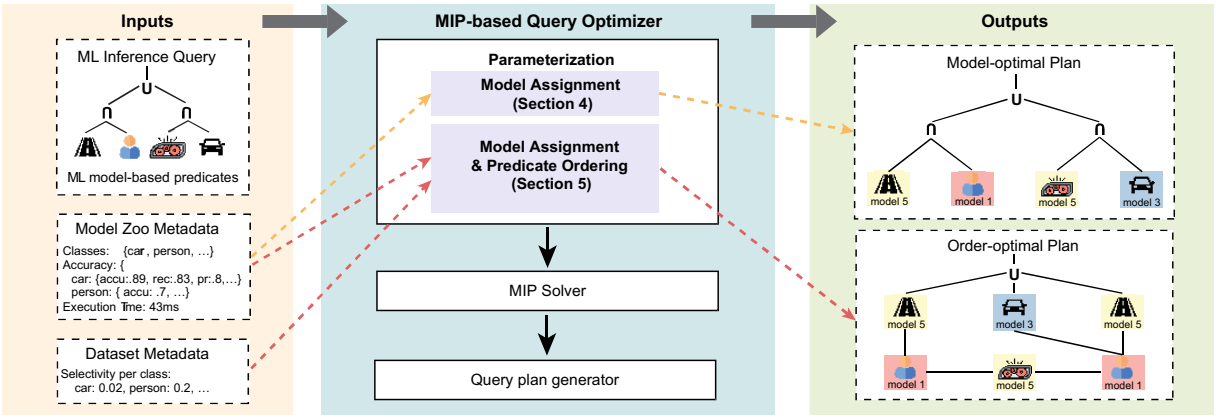


Figure 2: Approach overview.

Use case. Lower bounds on accuracy are expected in many ML inference queries. When the query is not used on a critical functionality (such as taking the correct turn rather than breaking for a pedestrian), a certain degree of accuracy could be sacrificed for a faster execution time, as long as the actual accuracy meets the minimum design requirement.

3.4 Complexity Analysis

In the following, we will show that solving the AMA problem (Definition 1) is NP-hard. For simplicity, we will consider the special case of our problem where *i*) the queries are conjunctive, i.e., $Q: p_0 \wedge \cdots \wedge p_r$ and *ii*) a given model can only answer a single predicate.

The Multiple Choice Knapsack Problem (MCKP). Our problem of assigning an ML model from a model zoo to every predicate of a query, is very similar to the Multiple-Choice Knapsack Problem (MCKP) [48], a generalization of the classic knapsack problem. The MCKP receives a set of items in disjoint classes, and its goal is to select exactly one item out of each class of items, maximizing the total value of items, while not exceeding the given weight capacity of the knapsack.

Theorem 1. The Accuracy-maximizing Model Assignment Problem is NP-hard.

Proof Sketch. The MCKP problem can be mapped to the AMA problem as follows. The items in MCKP are mapped to the models that need to be selected in the AMA problem. Moreover, those ML models will be divided into different classes, one class for each predicate that a model can cover (e.g., all models that can detect a "car" , would belong to the same class), as for items divided into different classes. For the needs of this reduction, we consider the special case that a model covers a single predicate, instead of multiple ones. The logarithm of accuracy of the selected model can be mapped as the value of the item. The total value of the items can be calculated as the sum of logarithms of accuracy. The weight of each item (model) can be directly mapped to the execution time of the selected model. Thus, the execution time bound of our original problem can be mapped to the total weight of the items.

A solution to the AMA problem selects exactly one model from each class (like MCKP does with items from each class), respecting the execution time constraint (weight in MCKP) and maximizing accuracy. Similar mechanism can be applied to map MCKP to EMA problem[3] by mapping the item profit with model execution time and the item weight with logarithm of the accuracy.

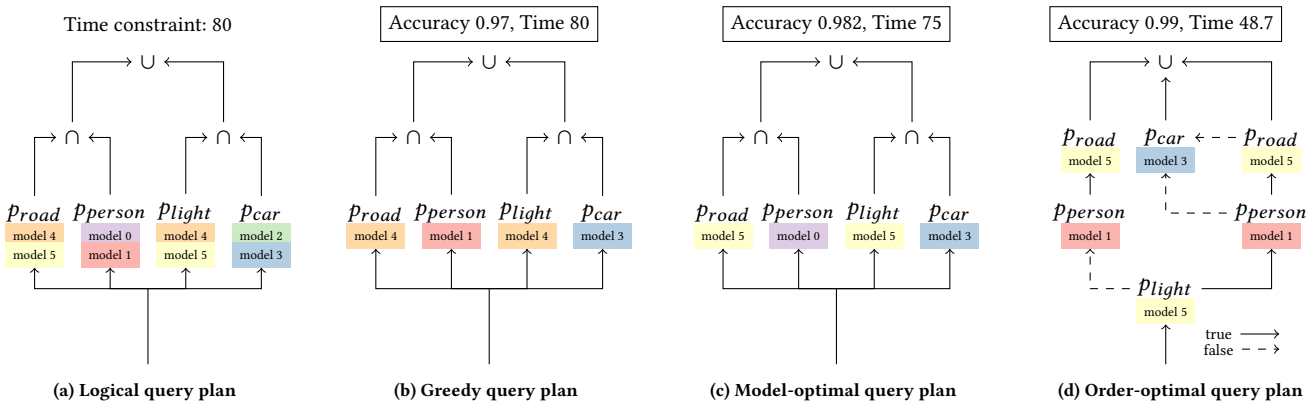
4 APPROACH OVERVIEW

As depicted in Figure 2, users can define an ML inference query with ML model-based predicates. To optimize the query our MIP-based optimizer receives 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 selectivity metadata, i.e., statistics regarding a predicate selectivity. Both types of metadata are retrieved from a metadata management tool [36]. The query optimizer then parses and optimizes the query. With the model zoo metadata alone (yellow dashed arrow), the resulting query plan is *i*) a *model-optimal* query plan that minimizes query execution time, adhering to an accuracy constraint, or vice versa. In addition, by utilizing the selectivity metadata, our query optimizer produces *ii*) a *model- & order-optimal query plan* (red dashed arrows). Such a query plan is more desirable: in addition to assigning the selected models for each predicate, it optimizes the order in which ML-based predicates are executed. The plan follows *bypass* processing [31], in which each branch only executes a specific set of the data with significant results, resulting in less query execution time.

4.1 Challenges

Tackling the accuracy/execution time trade-off. In an ML inference query, a predicate can be inferred by multiple models (Figure 3(a)). A naive planning approach would assign the models greedily as in Figure 3(b): for each predicate, the model with the highest accuracy is selected, as long as it meets the constraint, which leads to exponential complexity. To tackle this challenge, we propose an MIP-based formulation (section 5) by parameterizing the factors relevant to AMA/EMA problems, e.g., variables for model assignment. Figure 3(c) shows our approach, which produces the best possible assignment of models by examining all predicates before allocating them. Thus, it provides better overall accuracy compared to the greedy optimizer.

[3]Proving the same for the EMA problem of Definition 2 is very similar, and we omit it for the sake of space.



(a) Logical query plan **(b) Greedy query plan** **(c) Model-optimal query plan** **(d) Order-optimal query plan**

Figure 3: Alternative query plans for the running example query.

Predicate ordering. Not all data instances are equally likely to have a positive outcome for the query. By considering predicates *selectivity* it is possible to arrange the execution order of models, and generate *order-optimal* plans using *bypass* processing [31]. Without loss of generality, we assume that selectivity is a property of an existing labeled dataset, and is known in advance. Selectivity is further applied by the query optimizer to generate query plans for unknown/unlabeled data. As shown in Figure 3(d), compared to the aforementioned alternatives, this query plan is more effective in terms of minimizing the number of data instances (e.g., images) that have to be processed for answering the query. We explain the formulation of selectivity-related variables in Section 6.

4.2 Assumptions

Our approach is a natural extension of relational databases: we consider multimedia data such as images or text, instead of relational tuples. In this work, an image can be seen as a tuple, and the inference classes of that image can be seen as boolean columns. This paper makes three assumptions.

(1) First, similar to relational DBMS query optimization works, this paper makes the attribute-value independence assumption [46]. This assumption translates to our setting as follows: the data *distributions of inference classes within a dataset are independent* of each other. For instance, we consider the probabilities of a road or a person being found in an same image to be independent. This is a common assumption made by query optimization works [11, 44]. We make that assumption because it greatly simplifies our calculations of model accuracy (section 5), and plan to lift it in future work.
(2) Second, we assume that the *classification decisions that a model makes are independent*. This assumption allows us to simplify the calculation of selectivities for predicate-order optimization (section 6).
(3) Finally, we assume that the characteristics of datasets on which the models have been trained/tested (accuracy on a given class, class distributions, etc.) are the same as the unseen dataset that we are querying. This means that if the optimizer chooses a given order of predicate execution, then that order will indeed be effective when the query is evaluated (section 7).

5 MODEL ASSIGNMENT AS A MIP

In this section, we present a mixed-integer programming formulation for the ML inference query optimization problems as defined in Definition 1 and Definition 2.

Model-assignment variables. To perform model assignment we need to allocate exactly one model to each predicate. Given a model zoo $\mathcal{R}(M, I, P, A, C)$ we define the *decision variables*, denoted as $X_{m,p}$, where $m \in M$ and $p \in P$. We represent the set of all possible decision variables of $\mathcal{R}$ as X. The decision variable $X_{m,p}$ is a binary variable that indicates whether a model is selected:

$$X_{m,p} = \begin{cases} 1, & if\ model\ m\ is\ assigned\ to\ predicate\ p \\ 0, & otherwise \end{cases}$$

Based on decision variables $X_{m,p}$, we now define the constraints.

Choosing exactly one model per predicate. The constraints guarantee that exactly one model is selected and assigned for each predicate in the query. Since $X_{m,p}$ is set to 1 when a model m is assigned for predicate p, among all the decision variables for the same predicate p, only one decision variable has the value of 1. That is, the sum of decision variable $X_{m,p}$ for different models but for the same predicate is 1. We express this constraint as:

$$\sum_{m \in M} X_{m,p} = 1 \tag{1}$$

This equation alone is not sufficient since it is possible for the optimizer to assign the cheapest model to every predicate and may result in 0% accuracy. This issue is somewhat mitigated by setting an upper bound on $X_{m,p}$ using A_{bound}:

$$X_{m,p} \leqslant \lceil A_{m,p} \rceil \tag{2}$$

which ensures that only models with non-zero accuracy on a predicate can be assigned. By setting this upper bound, the size of the search space also becomes smaller as the optimizer discards these non-valid solutions.

5.1 Query Accuracy Calculation

In what follows, we explain the procedure of calculating query accuracy a_Q, i.e., $f_{acc}(Q)$ in Definition 1.

For example, given the query q in Section 3.2, the accuracy is computed as follows:

$$f_{acc}(q) = (a_{road} \cdot a_{person}) + (a_{light} \cdot a_{car}) - (a_{road} \cdot a_{person}) \cdot (a_{light} \cdot a_{car})$$

In this work, we assume that the predicates are independent and we did not consider the effect of correlation between predicates (See 4.2 for explanations). Similar assumption has been made in [39]. If two predicates are independent, we can regard the accuracy as the probability of getting true predictions. Thus we can compute the accuracy model following probability theory.

With decision variable $X_{m,p}$ and accuracy value $A_{m,p}$, we now turn to calculate the accuracy of a query, i.e., $f_{acc}(Q)$. Recall that an ML inference query can come in as DNF or CNF. The first step of calculating a_Q for a DNF query is to calculate the accuracy of the individual conjunctive subexpressions by using the following formula:

$$f_{acc}(Q) = \prod_{p \in P} \left(\sum_{m \in M} A_{m,p} X_{m,p} \right) \tag{3}$$

The disjunction of the accuracy values of the conjunctive subexpressions is computed with the following formula:

$$\begin{aligned} f_{acc}(Q) = & \sum_{p \in P} \sum_{m \in M} A_{m,p} X_{m,p} - \prod_{i \in \{p0,p1\}} \sum_{m \in M} A_{m,i} X_{m,i} \\ & + \prod_{j \in \{p0,p1,p2\}} \sum_{m \in M} A_{m,j} X_{m,j} - \ldots \\ & + (-1)^{|P|-1} \prod_{p \in P} \left(\sum_{m \in M} A_{m,p} X_{m,p} \right) \end{aligned} \tag{4}$$

The calculation of $f_{acc}(Q)$ for a CNF query is conducted similarly: first we calculate all the individual disjunctive subexpressions with Eq 4, and then calculate the final conjunction of those disjunctions with the formula of Eq 3. To summarize, for a CNF query or DNF query, we will parse different operators and compute the accuracy according to the query.

5.2 Modeling the Execution Time

In this subsection, we will introduce how to represent the problem if we only consider model assignment, and how to compute f_{time} in Definition 2. Suppose that a user sets a constraint on execution time. There are situations where a given model will be assigned to multiple predicates. In this case, however, the model's execution time should be measured only once: the model can be executed once on the input and can output predictions for multiple classes. Therefore we define a binary variable B_m to indicate the assignment of the model m, where $m \in M$. We use B to denote the set of variables B_m for different models in M. If the model m is selected, possibly more than once, the corresponding variable B_m is set to 1, otherwise it is set to 0.

We first enforce that B_m has to be non-zero if at least one $X_{m,p}$ is non-zero, by setting $X_{m,p}$ as a lower boundary for B_m:

Table 3: Variables in formalization.

Symbol	Domain	Semantic
$X_{m,p}$	{0,1}	If model m is assigned to predicate p
B_m	{0,1}	If model m is selected
$O_{p,j}$	{0,1}	If predicate p is answered in the jth execution step
$G_{g,j}$	{0,1}	If the predicates within the same group (conjunction / disjunction) have all been answered
$H_{g,j}$	$\mathbb{R}$	The percentage of data being computed at step j when predicates in group g have all been answered
$W_{g,j}$	$\mathbb{R}$	The percentage of data being computed at step j considering the predicates in the same group have been answered
$Q_{g,p,j}$	$\mathbb{R}$	The product of $H_{g,j-1}$ and $O_{p,j-1}$
S_j^J	$\mathbb{R}$	The percentage of data being selected in step j
$Y_{m,p,j}$	$\mathbb{R}$	The product of S_j^J, $X_{m,p}$, and $O_{p,j}$
$R_{m,j}$	$\mathbb{R}$	The execution time of running model m at step j

$$X_{m,p} \leqslant B_m \tag{5}$$

In this case $X_{m,p}$ "pushes" B_m upwards. To reduce the size of the search space we introduce an upper bound as well, making sure that B_m can only become non-zero if there is at least one $X_{m,p}$ that is non-zero:

$$B_m \leqslant \sum_{p \in P} X_{m,p} \tag{6}$$

Finally, the execution time for the query plan is :

$$f_{time}(Q) = \sum_{m \in M} C_m B_m \tag{7}$$

Objective functions. To conclude, with Eq(4) and Eq(7), we have transformed the two problem variants in Section 3.3 to a matter of MIP by defining two objective functions as below.
Given an execution time constraint (solving problem described in Definition 1):

Maximize: $f_{acc}(Q)$
Subject to: $Eq(1), Eq(2), Eq(5), Eq(6), f_{time}(Q) \leqslant C_{bound}$

Given an accuracy constraint (solving the problem in Definition 2):

Minimize: $f_{time}(Q)$
Subject to: $Eq(1), Eq(2), Eq(5), Eq(6), f_{acc}(Q) \geqslant A_{bound}$

6 PREDICATE ORDERING AS MIP

Algorithm 1 outlines the main steps of our proposed order-optimal query optimizer. To distinguish the known and unknown variables in an objective function, we use K to present the list of input variables. It includes the given ML inference query Q and model repository R (defined in Section 3), objective type T (execution time or accuracy) and bound β (C_{bound} or A_{bound}). We write the objective functions introduced in Section 5 as $f(K, X)$, where K is known and we try to decide the value of X. Our main contribution lies in line 2 in Algorithm 1. We design variables (e.g., O, G) that represent predicate ordering and predicate selectivity. They allow us to transform $f(K, X)$ to new objective functions $f'(K, X, O, G)$ with embedded information of predicate order, cost, and accuracy. We elaborate the variable definitions, their computation rules in

Table 4: Constraints in formalization.

Eq index	Constraints
Eq(1)	$\forall p : \sum_{m \in M} X_{m,p} = 1$ Semantics: Only select one model for each predicate
Eq(2)	$\forall p, m$: $X_{m,p} \leqslant \lceil A_{m,p} \rceil$ Only assign a model to a predicate it can successfully inference on
Eq(5,6)	$\forall m, p : X_{m,p} \leqslant B_m; B_m \leqslant \sum_{i \in P} X_{m,i}$ Identify whether model m is selected
Eq(8,9)	$\forall j : \sum_{p \in P} O_{p,j} = 1; \forall p : \sum_{j \leqslant \lvert P \rvert - 1} O_{p,j} = 1$ At each step, only one predicate is answered
Eq(13)	$\forall g \; \forall j \; \forall p \in P_g : G_{g,j} \leqslant \sum_{0 \leqslant k \leqslant j-1} O_{p,k}$ Variables are applicable if all the predicates are answered within the same group
Eq(14)	$\forall g \forall j : W_{g,j} = 1 - G_{g,j} S_g$ Determines the selectivity produced by group g in step j
Eq(15,16)	$H_{g,j} = H_{g,j-1} \cdot (1 - \sum_{p \in P_g} O_{p,j-1}(1 - S_p^P))$ $H'_{g,j} = \max(1 - \sum_{p \in P_g} O_{p,j}, G_{g,j}, H_{g,j})$ Determines the selectivity produced by the predicate in the same group
Eq(17)	$\forall j : S_j = \prod_{g \leqslant \lvert group \rvert} W_{g,j} H_{g,j}$ Selectivity at step j
Eq(18)	$\forall m \forall j : Z_{m,j} = \sum_{p \in P} Y_{m,p,j} C_{m,p}$; $\forall m \forall p \forall j Y_{m,p,j} \leqslant M \cdot X_{m,p}; Y_{m,p,j} \leqslant M \cdot O_{p,j}$; $Y_{m,p,j} \geqslant S_j^J - (2 - X_{m,p} - O_{p,j})M; 0 \leqslant Y_{m,p,j} \leqslant S_j^J$; Determines the cost of executing model m at step j

Section 6.1, and 6.2, and the transformed objective functions in 6.3. Finally, in Section 6.4 we explain how we apply an MIP solver to obtain the values of unknown variables X, O, G, and use them to generate the optimized query plan that performs bypass processing (i.e., Algorithm 2).

6.1 Predicate-order Variables

To order the predicates we consider *steps* of an ML inference query. We assume sequential model execution and use a step to represent the execution of one predicate in the query.

To allocate exactly one predicate at one step we introduce the binary variables $O_{p,j} \in O$, where $p \in P$. j represents the step and its value is the index of the order with the range of $[0, |P| - 1]$. The variable $O_{p,j}$ indicates whether predicate p is evaluated during the step j.

$$O_{p,j} = \begin{cases} 1, & \text{if predicate } p \text{ is answered at step } j \\ 0, & \text{otherwise} \end{cases}$$

Algorithm 1: Order-optimal Optimizer

Input : ML inference query Q, model repository R, objective type T, bound β
Output : Query plan $optPlan$ for query Q

```
K ← [Q, R, T, β] // input variables
f'(X, O, G, K) ← OrderOpt(f(X, K)) // transform obj func
X, O, G ← MILP_Solver(f'(X, O, G, K))
optPlan ← QueryPlanGen(X, O, G)
return optPlan
```

Table 5: $O_{p,j}$ with different predicates and steps

p \ j	**0**	**1**	**2**	**3**
$p_{\texttt{road}}$	0	0	1	0
$p_{\texttt{person}}$	0	1	0	0
$p_{\texttt{light}}$	1	0	0	0
$p_{\texttt{car}}$	0	0	0	1

Continuing with the running example, Table 5 shows an example of a possible order of the four predicates. The order is $p_{\texttt{light}} \rightarrow p_{\texttt{person}} \rightarrow p_{\texttt{road}} \rightarrow p_{\texttt{car}}$.

Answering exactly one predicate at each step. Similar to Eq(1), we design the following constraint to restrict the number of predicates executed at each step j:

$$\sum_{p \in P} O_{p,j} = 1, \tag{8}$$

A similar constraint is set on the execution of the predicates, i.e., each predicate p must be executed once:

$$\sum_{j=0}^{|P|-1} O_{p,j} = 1 \tag{9}$$

6.2 Considering Selectivity and Order

Before establishing a cheap order of execution we need to measure the cost of the plan. The cost of a query plan depends highly on the order of predicate evaluation if we consider selectivity. The lower the selectivity of a model, the more data tuples/items can be filtered out, which reduces the computation time. However, the amount of saved computation can be easily offset with high model execution time. For instance, a very expensive predicate/model that is very selective may not save costs if it is run for all input tuples/images of a dataset. This is a cost-based decision that we model in the following.

6.2.1 Predicate Ordering on Two Simple Types of Queries. Before introducing predicate ordering on an ML inference query, we first consider two simpler cases: conjunction-only queries and disjunction-only queries.

Conjunction-only queries. Consider an ML inference query with only conjunctions of predicates, i.e., in the form of $Q : p_1 \wedge \cdots \wedge p_r$. Predicate ordering for such queries is straightforward: the selectivity of the query would be the product of the selectivity of all the predicates in the query. We define the *selectivity of the predicates* as S_p^P [4], where $p \in Q$. The selectivity of the conjunctive query is $\prod_{p \in Q} S_p^P$. Taking into account the selectivity and cost of a predicate, as well as their execution order for a query Q, the cost C_Q can be calculated as follows (simplified version):

$$C_Q = C_0 + C_1 S_0^P + C_2 S_0^P S_1^P + \ldots + C_{|r|-1} \prod_{i \in [0, |r|-2]} S_i^P \tag{10}$$

[4] Due to the need to distinguish between selectivity of predicates S^P (dataset-defined constants) versus groups S^G (query-dependent constants) versus timesteps S^J (MIP variables), the superscript denotes which type of selectivity is meant, and the subscript the set indexation.

Disjunction-only queries. Next, we consider an ML inference query with only disjunctions of predicates, i.e., in the form of $Q: p_1 \vee \cdots \vee p_r$. For such a query Q, the selectivity of the query is the multiplication of $1 - S_p^P$ for each predicate $p \in Q$. The cost of this query is (simplified version):

$$\begin{aligned} C_Q =& C_0 + C_1(1 - S_0^P) + C_2(1 - S_0^P)(1 - S_1^P) \\ &+ ... + C_{r-1} \prod_{i \in [0, r-2]} (1 - S_i^P) \end{aligned} \tag{11}$$

6.2.2 Predicate Ordering on CNF or DNF queries. Now we explain predicate ordering on an ML inference query in CNF or DNF. In all the following examples we will continue with the running example query q.

Example 6.1. *In this query q, $p_{\texttt{road}}$ and $p_{\texttt{person}}$ are the literals in the first conjunction, while $p_{\texttt{light}}$ and $p_{\texttt{car}}$ are in the second conjunction. We refer to the predicates in the same conjunction subformula of DNF queries (or in the same disjunction subformula in CNF queries) as a group. q is of DNF, and it has two groups $(p_{\texttt{road}}, p_{\texttt{person}})$ and $(p_{\texttt{light}}, p_{\texttt{car}})$.*

In the following, we aim to model predicate execution order based on groups and optimize it with MIP. To this end, we define three kinds of variables for presenting groups (G), selectivity among groups (W), and selectivity within groups (H).

Representing groups. G are binary variables representing whether all predicates in the same group have been fully evaluated at a given step j. If yes, the value of $G_{g,j} \in G$ is 1, otherwise it will be 0. g ($\in \{0, 1, ..., \#groups\}$) refers to the index of different conjunction groups, and j refers to the step number. We add constraints of the form $G_{g,j} \leqslant \sum_{0 \leqslant k \leqslant j-1} O_{p,k}$, where $p \in P_g$. To make sure that the value of $G_{g,j}$ is set to 1 if all predicates in the same group g have been evaluated, we define the following constraints:

$$G_{g,j} \geqslant 1 - |P_g| + \sum_{p \in P_g} \sum_{0 \leqslant k \leqslant j-1} O_{p,k}. \tag{12}$$

$$G_{g,j} \leqslant \sum_{0 \leqslant k \leqslant j-1} O_{p,k} \tag{13}$$

Representing selectivity among groups. We introduce the variable $W_{g,j}$ to represent the percentage of data being processed at every step when one group of predicates has all been answered. W is the set of all possible variables $W_{g,j}$, and it models the effect of the predicates from different groups. Thus, we can see that the selectivity of the predicate can affect other predicates in other groups. We use G to compute, since G indicates whether the predicates in the same group have been answered. Moreover, there is a selectivity for each group of conjunctions, $S_0^G = S^P_{\texttt{road}} S^P_{\texttt{person}}$ for group 0 and $S_1^G = S^P_{\texttt{light}} S^P_{\texttt{car}}$ for group 1. For CNF queries the selectivity for each group is the probability that each disjunction returns true. For DNF queries however, the selectivity for each group is the probability that each conjunction returns false. We model the reduction rate of the current step as follows:

$$W_{g_i,j} = 1 - G_{g_i,j} S_g^G \tag{14}$$

Since the variables G are binary, when $G_{g,j}$ equals 1, $W_{g,j}$ equals $1 - S_g^G$, indicating that the proportion of data being selected for further processing is $1 - S_g^G$. When $G_{g,j}$ equals 0, $W_{g,j}$ equals 1, which means that all the data should be processed. To continue with the previous example, if $G_{0,3}$ equals 1, then $W_{0,3}$ equals $1 - S_0^P$, where S_0 is the selectivity of $(p_{\texttt{road}} \wedge p_{\texttt{person}})$, i.e., $S_0^G = S^P_{\texttt{road}} S^P_{\texttt{person}}$ as mentioned above.

Representing selectivity within groups. H are continuous variables representing the selectivity within the group at each step. For each $H_{g,j} \in H$, g is the group index and j the step number. H models the effect of predicates in the same group.

We compute H as follows:

$$H_{g,j} = H_{g,j-1} \cdot (1 - \sum_{p \in P_g} O_{p,j-1}(1 - S_p^P)) \tag{15}$$

As mentioned before unnecessary product variables should be avoided in MIP. Products of binary and continuous variables can oftentimes be linearized [3] without any error.

When all the predicates within a group is executed, then the effect within the same group is demolished. Thus we introduce the following equation to remove the effect from within group. We introduce a new variable H' to represent the new relationship.

$$H'_{g,j} = \max(1 - \sum_{p \in P_g} O_{p,j}, G_{g,j}, H_{g,j}) \tag{16}$$

The percentage of the data being processed at each step is affected by the answered predicates within the same group and across groups (W and H'). The percentage is represented as the selectivity in each step, $S_j^J \in S$, and can be computed as:

$$S_j^J = \prod_{g=0}^{|group|} W_{g,j} H'_{g,j} \tag{17}$$

So far, we have obtained the measured image processing rate at each step. At each step we have S_j^J to indicate the current proportion of images to process. With this variable we can further measure the cost model of the plan.

Calculating the execution cost (time) of a query. We combine the model assignment variables $X_{m,p} \in X$ to compute the cost model. Considering the selectivity and model performance, we define the variables $Z_{m,j}$ to represent the execution cost of a model m for each step j. The set of all possible variables $Z_{m,j}$ is R. The cost of a query plan can be computed as follows:

$$Z_{m,j} = S_j^J \sum_{p \in P} X_{m,p} O_{p,j} C_{m,p} \tag{18}$$

We apply a similar mechanism to linearize the equation by adding additional variables. Transformations can be found in Table 4. The execution time of each model should be computed only once, even though it can answer multiple predicates. The cost model is:

$$\sum_{m \in M} \max_{0 \leqslant j \leqslant |P|-1} Z_{m,j} \tag{19}$$

For example, In Figure 3d, *model* 6 is assigned to answer both $p_{\texttt{light}}$ and $p_{\texttt{road}}$. If $p_{\texttt{light}}$ is answered prior to $p_{\texttt{road}}$, we need to only consider the execution time of the model when it is firstly executed for $p_{\texttt{light}}$.

6.3 Objective Functions

Finally, our proposed order-optimal approach has transformed the objective functions in Section 5.2 to the following forms. Given an execution time constraint (solving the problem of Definition 1):

$$\text{Maximize:} \quad f_{acc}(Q)$$
$$\text{Subject to:} \quad Eq(8), Eq(9), \sum_{m \in M} \max_{0 \leqslant j \leqslant |P|-1} Z_{m,j} \leqslant C_{bound}$$

Given an execution time constraint (solving the problem of Definition 2):

$$\text{Minimize:} \quad \sum_{m \in M} \max_{0 \leqslant j \leqslant |P|-1} Z_{m,j}$$
$$\text{Subject to:} \quad Eq(8), Eq(9), f_{acc}(Q) \geqslant A_{bound}$$

6.4 Query Plan Generation

With the above transformed objective functions ready, we obtain the values of all defined variables, such as O, G. We use Gurobi 9.0 to solve the optimization problem. The solver generates the MIP solutions, and we obtain the values of all the defined variables, then with Algorithm 2 we generate the bypass plans.

6.5 Inference Query Optimization in Practice

Our work can be applied in applications that require executing composition of models given constraints. There are two main trends that incorporate ML models in a query in the database field, namely in-database-ML and dataflow.

In-database Inference. Multiple works in the last years (e.g., [15, 28]) focus on bringing ML inference queries closer to the data by natively supporting ML models within a database system, such that practitioners can perform ML model training and inference on database data through extended SQL queries. ML models can be used within user defined functions (UDFs) – a common practice in the database community [40, 61]. These works operate in a similar fashion: ML models are executed as UDFs on incoming tuples during query execution. In the context of our work, the query plans on the right of Figure 2, would be translated to classic database UDF-based filters and would be executed by any DBMS execution engine. For this approach to work in our case, the original DBMS optimizer needs to be extended in order to support ML query inference. To this end, *i*) the UDFs have to be annotated with e.g., the classes that they can recognize, and *ii*) the information about the models that can be executed alongside the model zoo metadata have to be passed on to the optimizer.

Dataflow pipeline. At the smae time, we see a proliferation of dataflow systems that are being used for ML inference (a.k.a. prediction) queries. Prime examples of such works are Cloudflow [49] and Pretzel [35], while in practice, a lot of ML inference queries take place in stream processing engines [9]. In this approach the inference queries are defined as dataflows where the data is read from the sources, and then they are evaluated by dataflow operators. Our optimized query plans can be compiled to dataflows as follows: a `map` receives an image, infers its labels and outputs the labels alongside the original image, which are then forwarded to a `filter` that outputs that image if the predicate is satisfied.

Algorithm 2: QueryPlanGen

```
Input  : X: matrix indicating selected models assigned to predicates
         O: matrix indicating the execution order of predicates
Output : Query plan optPlan
T ← model assignment for predicates identified from X
S ← execution order of predicates identified from O
for each predicate p running at each step s in S do
    model m ← mapping from T given p
    if All the predicates in group g have all been executed then
        condition ← Boolean value of g that keeps the Boolean
          expression unsolved
    end
    plan ← {s, p, m, condition}
    add plan to optPlan
    for other predicate q in the same group of predicate do
        if p has been executed then
            condition ← Boolean value of p that keeps the Boolean
              expression unsolved
            plan ← {s, p, m, condition}
            add plan to optPlan
        end
    end
end
return optPlan
```

7 EXPERIMENTAL EVALUATION

7.1 Setup

7.1.1 Input Datasets. We used public datasets covering object detection in images with COCO [37] which is the most well-known real-world annotated dataset of images, as well as sentiment analysis in text with TweetEval [5]. The data in TweetEval is real-world data taken from twitter. *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. It was generated for emotion recognition, hate speech detection, sentiment analysis, etc. In our experiments, we use the ground truth for the sentiment analysis task. There are 18 inference classes, belonging to different categories, such as text sentiments, entity types, etc. The ground truth for sentiment analysis was taken from the original dataset, while for the rest, we inferred the ground truth with the prediction of the best-performing models for the task in our model zoo.

7.1.2 Model Zoos. We collected all of our pre-trained models from public model zoos: HuggingFace, and PytorchHub. To navigate the space of different model zoos that may be encountered in the public space, we opted for curating four 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 6 and presented as follows:

– **Real-world Model Zoo ❶** . This model zoo contains 48 real-world models that can tackle NLP tasks. Each model in this model zoo, covers all inference classes of the NLP tasks.

– **Real-world Model Zoo ❷** . This model zoo 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 Zoos, derived from real-world: Model Zoo ❸ , Model Zoo ❹ , Model Zoo ❺** . These model zoos have been

Table 6: 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
Model Zoo ❺	Image	All	Accuracy, Cost	165

Table 7: Ex generate ample ML inference queries (accuracy measured by F1-score, and cost measured by average inference time per instance).

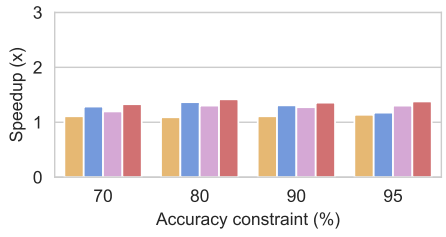
Modality	Example Query	Constraint
text	e.g., `ner=person` ∧ `sentiment=negative` ∧ (`topic=news` ∨ `topic=sport`)	e.g., accuracy > 80%
image	e.g., `person` ∧ (`car` ∨ `bike`) ∧ `emergency_light`	e.g., cost < 100 ms

derived from Model Zoo ❷ . Each of the 33 models model 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 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 6).

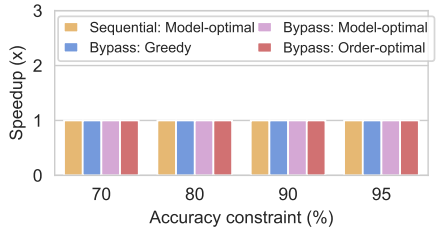
7.1.3 Optimization Strategies. We compare five strategies for optimizing ML inference query given a certain constraint. Note that there are two ways to execute the query plans: in *sequential*, i.e., not applying bypass and executing the plans in sequence; and in *bypass*, i.e., executing the plan in bypass mechanism given certain 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 (Section 5) relies on MIP to optimize the model assignment under constraints, as compared to the greedy optimizer that approximates model assignment.



(a) Model Zoo ❶



(b) Model Zoo ❷

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

Baseline 3 - Bypass: Greedy. This baseline extends Sequential: Greedy by converting the plan into bypass plan given random predicate execution order.

Baseline 4 - Bypass: Model-optimal. This baseline extends Sequential: Model-optimal by converting the plan into bypass plan given random predicate execution order.

Bypass: Order-optimal. This approach jointly optimizes for both model assignment and predicate ordering by considering the selectivity of predicates in a dataset and create a bypass plan.

7.1.4 Evaluation metrics. We use F1-score to measure accuracy, and milliseconds per instance for execution time. We divide each dataset 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 (assumption (3) applied here, see 4.2). The query execution time shown in the following is obtained by executing the queries on the test set.

7.1.5 Queries. Since there are no benchmark queries that we could use from other works for our datasets, we adopted a similar approach as [39] to curate queries. We generate queries for two scenarios: comparing query quality and measuring optimization time.

Query performance. We manually curated 10 queries (exemplified in Table 7) 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 either accuracy or 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, where the predicates share the same probability of being selected.

7.1.6 Accuracy & Exec. Time Constraints. We create a number of experiment settings by enumerating different execution time and accuracy bounds to verify optimizers' performance on different levels of constraints. The accuracy bounds are {70%, 80%, 90%, 95%}, representing relaxed targets, moderate target and restricted target. 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%, 130%, 140%, 150%}. We observe that, when constraint scales are lower than 100% (minimum time constraint, our approach (*Bypass: Order-optimal*) can still generate plans while other baselines cannot.

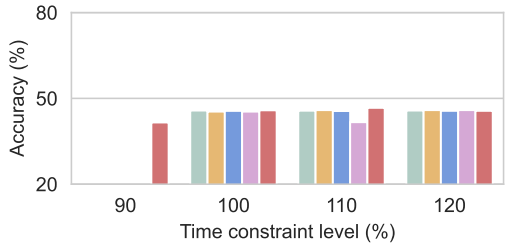
7.1.7 Hardware. We perform our experiments on a Ubuntu server with a single GPU (Nvidia A40, 4GB RAM) and 24-core CPU.

7.2 Performance on Uniform Model Zoos

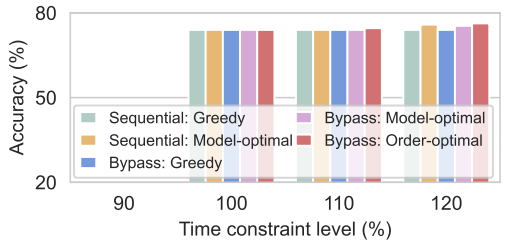
In this section, we observe the behavior of our optimizer using the model zoos Model Zoo ❶ and Model Zoo ❷ . We first constrain the accuracy (70% - 95%) and then constrain the execution time (90% - 120%). We execute all the queries in the query workload and report average speedups compared to the query plans generated by the Sequential:Greedy optimizer. We present those speedups in bar plots (e.g., Figure 4).

Constraining Accuracy. As seen in Figure 4, the first observation is that using bypass plan can increase efficiency. We find out that the differences in model performance are small, model zoos containing very similar models in terms of architecture (e.g., transformer-based for NLP models). In Model Zoo ❷ , we observe that most of the time, a single model was enough to answer the complete query. In those cases we do not observe any speedups or very limited speedups across the two model zoos.

Constraining Execution Time. 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. The first observation (Figure 5) is that when we put a low constraint on the execution time, our solution, Bypass:Order-optimal, succeeds to find proper solutions. Since the models used in both model zoos ❶ and ❷ have very similar performance in terms of accuracy, we do not observe large differences in accuracy.



(a) Model Zoo ❶



(b) Model Zoo ❷

Figure 5: Accuracy of queries, given execution time constraints.

7.3 Performance on Model Zoos with Diverse Model Distributions

In this section, we want to observe the effect of diverse performance 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 ❺ answering all inference classes, and we want to see if in such constrained environment the order optimizer can bring benefits. Finally, Model Zoo ❹ (average of 13 inference classes per model) stands in the middle of the two, offering possibilities for optimization but not as many as in Model Zoo ❺

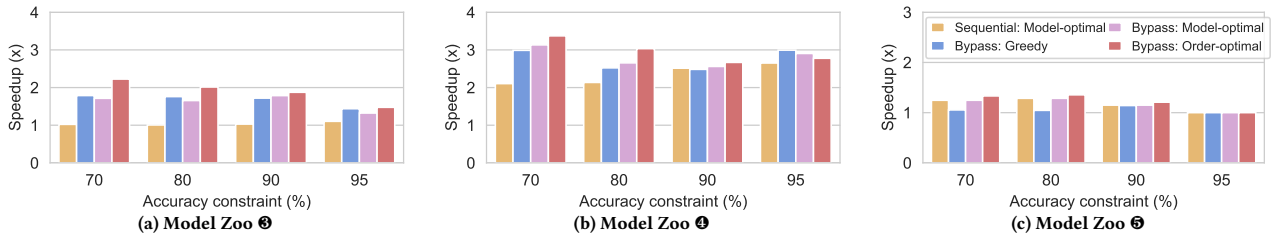
Constraining Accuracy. Figure 6 shows the average speedups (more efficient compared to *Sequential: Greedy)* of all queries under different accuracy constraints. We observe speedups when applying bypass plan to execute the models, compared to Greedy and Model-optimal. In most cases, *Bypass: Order-optimal* outperforms the baselines across all model zoos. Specifically, *Bypass: Order-optimal* achieved up to 7x speedup in some cases, compared to *Sequential: Greedy*. Compared to the previous experiments, we notice that when the Bypass:Order-optimal optimizer is presented with more opportunities, namely more models of different accuracy and execution time tradeoffs, it can navigate the search space efficiently and optimize queries, resulting in great speedups. While in Model Zoo ❺ most of the time one model is feasible to answer the query, leading to limited speedups.

Constraining Execution Time. Figure 7 shows the accuracy of all queries, for different values of execution time constraint. We observe that Bypass:Order-optimal consistently obtains higher query accuracy than the baselines. Even though bypass is applied given a model selection plan in Bypass:Greedy and Bypass:Model-optimal, bypass plans do not gain benefits when execution time is constrained. By applying bypass plan, these two approaches may filter some significant images in the early stage leading to a decrease in performance. While Bypass: Order-optimal jointly optimizes for both model selection and predicate ordering and can make use of predicate ordering and perform early filtering, making better use of execution time budget.

Summary. Using bypass plans can lead to higher efficiency, while not necessarily increasing accuracy. The Bypass: Order-optimal optimizer can speedup certain queries even by 7x compared to Sequential:Greedy, and it can find optimal solutions, especially given very diverse model zoos with different execution time and accuracy tradeoffs.

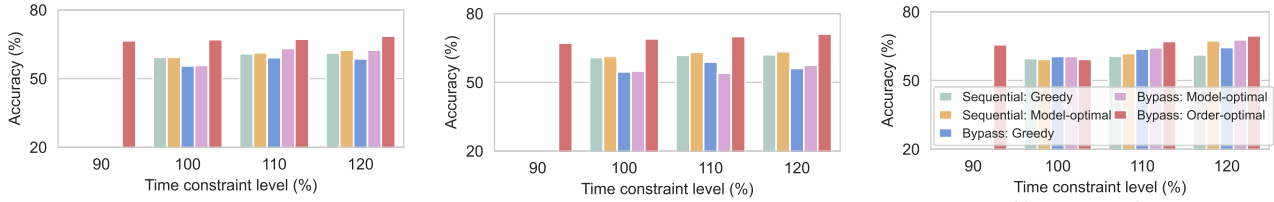
7.4 Query Optimization Time

In this section, we evaluate the scalability of different approaches (we exclude Baseline 3 and Baseline 4 in this case, since converting output to bypass plan can be executed in polynomial time.): increasing the number of predicates is reminiscent of increasing the number of items in a knapsack (see complexity analysis section 3). Hence, we are interested in finding the limit of the Bypass:Order-optimal optimizer, with respect to the number of predicates that can be included in a query before it becomes too slow.



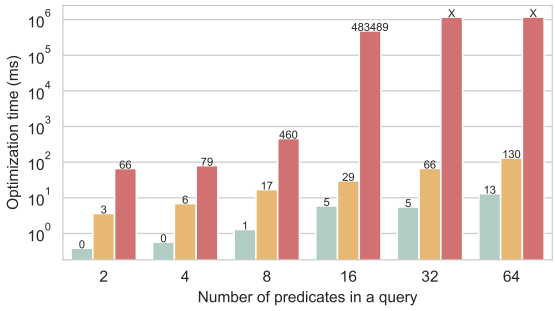
(a) Model Zoo ❸ (b) Model Zoo ❹ (c) Model Zoo ❺

Figure 6: The average speedups of query execution time compared to the *Greedy* approach on the query workload with different accuracy (objective) bounds.

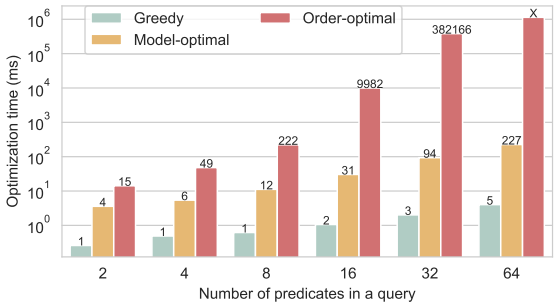


(a) Model Zoo ❸ (b) Model Zoo ❹ (c) Model Zoo ❺

Figure 7: The average accuracy performance on the query workload with different time (objective) constraint levels.



(a) Constrained on accuracy



(b) Constrained on execution time

Figure 8: Optimization time on queries with varying number of predicates.

Query Plan Generation Time. We evaluate the efficiency of our optimizers in generating a query plan by varying the number of predicates in a query as shown in Figure 8. The experiments were performed on Model Zoo ❹ (a medium size model zoo).

We observe that all the optimizers show very abrupt growth with the increase of predicate number in a query, except the Sequential:Greedy approach, which verifies that the problem we are tackling has a very high complexity (section 3). We observe that the advanced optimizers require much longer time to generate a plan as the number of predicates increase. We also observe that constraining on the execution time has yielded faster optimization time than constraining on the accuracy, especially when the number of predicates in a query is high. In fact, when accuracy is constrained, the optimization time for 32 and 64 predicates did not finish (X). The discrepancies between optimizing with accuracy constraints versus time constraints in Figure 8 are likely due to the number of product variables that have to be calculated for accuracy. Though taking longer time to generate plans, for queries having up to 16 predicates, the time to answer a query is dominated by the total inference time. Thus, for the majority of queries, the order-optimal optimizer continues to offer considerable benefit in reducing execution time (see Figure 6), and this benefit will grow with the size of the data.

7.5 Further Optimizing Probabilistic Predicates

In this experiment, we integrate the proposed *order-optimal* algorithm in the probabilistic predicates (PP) [39] framework. We chose order-optimal as it is the one that has shown to perform best. In short, the goal of PP is to filter insignificant images at an early stage by executing a set of efficient (but less accurate) models first. Our approach can be integrated in PP by replacing PP's early stage plan. The threshold applied in PP regards recall: the proportion of true positive labels. In the rest of the experiment, we refer to the recall as accuracy. We generate plans subject to different accuracy bounds and expect shorter execution time that translates to larger throughput measured in frames per second (fps). The aim of PP [39] is to increase execution efficiency. Thus, in this experiment, we only compare the execution time given recall constraints. Our order-optimal optimizer can be deployed to decide on both the model selection when multiple probabilistic predicates are available, and also to set the order of predicate execution. To show this, we crafted 5 example queries, two of which are the ones used in the original paper [39].

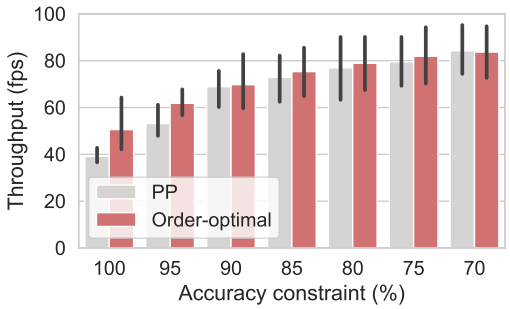


Figure 9: Comparison with PP on throughput (frames per second) when accuracy (measured with recall) was under constrained.

The results are shown in Figure 9. In general, our order-optimal optimizer improves throughput against PP in all but one queries. Notably, the tighter the accuracy constraint, the better is the improvement that our optimizer achieves. The improvement is owed to our optimizer's capacity to optimally allocate the execution order of predicates. PP orders predicates by rank, i.e., $\frac{accuracy}{cost}$, within conjunctions or disjunctions, while our order-optimal optimizer specifies the order based on global cost instead.

8 CONCLUSIONS & FUTURE WORK

In this paper we addressed the problem of ML inference query optimization, which regards the optimal selection of ML models for answering an inference query under constraints. We formulated 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*) the application of approximation schemes in the MIP formulation of the problem and *iii*) the lifting the assumptions made in this paper, considering especially the correlation of inference classes and concept drift.

REFERENCES

[1] Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. 2019. FLAIR: An easy-to-use framework for state-of-the-art NLP. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics (demonstrations)*. 54–59.

[2] Michael R Anderson, Michael Cafarella, German Ros, and Thomas F Wenisch. 2019. Physical representation-based predicate optimization for a visual analytics database. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1466–1477.

[3] Mohammad Asghari, Amir M Fathollahi-Fard, SMJ Mirzapour Al-e hashem, and Maxim A Dulebenets. 2022. Transformation and Linearization Techniques in Optimization: A State-of-the-Art Survey. *Mathematics* 10, 2 (2022), 283.

[4] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. 2017. Macrobase: Prioritizing attention in fast data. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 541–556.

[5] Francesco Barbieri, Jose Camacho-Collados, Leonardo Neves, and Luis Espinosa-Anke. 2020. Tweeteval: Unified benchmark and comparative evaluation for tweet classification. *arXiv preprint arXiv:2010.12421* (2020).

[6] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934* (2020).

[7] Zhaowei Cai, Mohammad Saberian, and Nuno Vasconcelos. 2019. Learning complexity-aware cascades for pedestrian detection. *IEEE transactions on pattern analysis and machine intelligence* 42, 9 (2019), 2195–2211.

[8] Jiashen Cao, Ramyad Hadidi, Joy Arulraj, and Hyesoon Kim. 2021. THIA: Accelerating Video Analytics using Early Inference and Fine-Grained Query Planning. *arXiv preprint arXiv:2102.08481* (2021).

[9] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015).

[10] Ashok K Chandra and Philip M Merlin. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*. 77–90.

[11] S. Christodoulakis. 1984. Implications of Certain Assumptions in Database Performance Evauation. *ACM Trans. Database Syst.* 9, 2 (jun 1984), 163–186. https://doi.org/10.1145/329.318578

[12] Daniel Crankshaw, Peter Bailis, Joseph E. Gonzalez, Haoyuan Li, Zhao Zhang, Michael J. Franklin, Ali Ghodsi, and Michael I. Jordan. 2015. The Missing Piece in Complex Analytics: Low Latency, Scalable Model Management and Serving with Velox. In *Seventh Biennial Conference on Innovative Data Systems Research, CIDR 2015, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2015/Papers/CIDR15_Paper19u.pdf

[13] 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.

[14] Ronald Fagin. 1998. Fuzzy Queries in Multimedia Database Systems. In *ACM PODS*.

[15] Apurva Gandhi, Yuki Asada, Victor Fu, Advitya Gemawat, Lihao Zhang, Rathijit Sen, Carlo Curino, Jesús Camacho-Rodríguez, and Matteo Interlandi. 2023. The Tensor Data Platform: Towards an AI-centric Database System. *CIDR* (2023).

[16] Tarleton Gillespie. 2020. Content moderation, AI, and the question of scale. *Big Data & Society* 7, 2 (2020), 2053951720943234.

[17] Michael Z Hanani. 1977. An optimal evaluation of boolean expressions in an online query system. *Commun. ACM* 20, 5 (1977), 344–347.

[18] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, et al. 2018. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 620–629.

[19] Joseph M Hellerstein and Michael Stonebraker. 1993. Predicate migration: Optimizing queries with expensive predicates. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*. 267–276.

[20] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[21] 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)*. USENIX Association, Carlsbad, CA, 269–286. https://www.usenix.org/conference/osdi18/presentation/hsieh

[22] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. 2017. Speed/accuracy trade-offs for modern convolutional object detectors. In

Proceedings of the IEEE conference on computer vision and pattern recognition. 7310–7311.

[23] Matthias Jarke and Jurgen Koch. 1984. Query optimization in database systems. *ACM Computing surveys (CsUR)* 16, 2 (1984), 111–152.

[24] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 253–266.

[25] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. Challenges and Opportunities in DNN-Based Video Analytics: A Demonstration of the BlazeIt Video Query Engine.. In *CIDR*.

[26] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. Noscope: optimizing neural network queries over video at scale. *arXiv preprint arXiv:1703.02529* (2017).

[27] Kai Kang and Xiaogang Wang. 2014. Fully convolutional neural networks for crowd segmentation. *arXiv preprint arXiv:1411.4464* (2014).

[28] Konstantinos Karanasos, Matteo Interlandi, Doris Xin, Fotis Psallidas, Rathijit Sen, Kwanghyun Park, Ivan Popivanov, Supun Nakandal, Subru Krishnan, Markus Weimer, et al. 2020. Extending relational query processing with ML inference. *CIDR* (2020).

[29] Fisnik Kastrati and Guido Moerkotte. 2017. Optimization of disjunctive predicates for main memory column stores. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 731–744.

[30] Fisnik Kastrati and Guido Moerkotte. 2018. Generating optimal plans for boolean expressions. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1013–1024.

[31] A. Kemper, G. Moerkotte, K. Peithner, and M. Steinbrunn. 1994. Optimizing Disjunctive Queries with Expensive Predicates. In *ACM SIMGOD* (Minneapolis, Minnesota, USA) *(SIGMOD '94)*. 336–347.

[32] Alfons Kemper, Guido Moerkotte, Klaus Peithner, and Michael Steinbrunn. 1994. Optimizing disjunctive queries with expensive predicates. *ACM SIGMOD Record* 23, 2 (1994), 336–347.

[33] Peter Kraft, Daniel Kang, Deepak Narayanan, Shoumik Palkar, Peter Bailis, and Matei Zaharia. 2020. Willump: A statistically-aware end-to-end optimizer for machine learning inference. *Proceedings of Machine Learning and Systems* 2 (2020), 147–159.

[34] Tony C. T. Kuo and Arbee L. P. Chen. 1996. A Content-Based Query Language for Video Databases. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems, ICMCS 1996, Hiroshima, Japan, June 17-23, 1996*. IEEE Computer Society, 209–214. https://doi.org/10.1109/MMCS.1996.534976

[35] 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.

[36] Ziyu Li, Rihan Hai, Alessandro Bozzon, and Asterios Katsifodimos. 2022. Metadata Representations for Queryable ML Model Zoos. *Proceedings of the Workshop on Benchmarking Data for Data-Centric AI (DataPerf)*.

[37] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*. Springer, 740–755.

[38] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*. Springer, 21–37.

[39] 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*. 1493–1508.

[40] Nantia Makrynioti and Vasilis Vassalos. 2019. Declarative data analytics: a survey. *IEEE Transactions on Knowledge and Data Engineering* 33, 6 (2019), 2392–2411.

[41] Dinithi Nallaperuma, Rashmika Nawaratne, Tharindu Bandaragoda, Achini Adikari, Su Nguyen, Thimal Kempitiya, Daswin De Silva, Damminda Alahakoon, and Dakshan Pothuhera. 2019. Online incremental machine learning platform for big data-driven smart traffic management. *IEEE Transactions on Intelligent Transportation Systems* 20, 12 (2019), 4679–4690.

[42] Christos H. Papadimitriou and Mihalis Yannakakis. 2001. Multiobjective Query Optimization. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Santa Barbara, California, USA) *(PODS '01)*. Association for Computing Machinery, New York, NY, USA, 52–59. https://doi.org/10.1145/375551.375560

[43] Christos H. Papadimitriou and Mihalis Yannakakis. 2001. Multiobjective Query Optimization. In *ACM PODS*. 52–59.

[44] Viswanath Poosala and Yannis E. Ioannidis. 1997. Selectivity Estimation Without the Attribute Value Independence Assumption. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB '97)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 486–495.

[45] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7263–7271.

[46] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. 1979. Access Path Selection in a Relational Database Management System. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data* (Boston, Massachusetts) *(SIGMOD '79)*. Association for Computing Machinery, New York, NY, USA, 23–34. https://doi.org/10.1145/582095.582099

[47] Yang Shu, Zhi Kou, Zhangjie Cao, Jianmin Wang, and Mingsheng Long. 2021. Zoo-Tuning: Adaptive Transfer from A Zoo of Models. In *International Conference on Machine Learning*. PMLR, 9626–9637.

[48] Prabhakant Sinha and Andris A Zoltners. 1979. The multiple-choice knapsack problem. *Operations Research* 27, 3 (1979), 503–515.

[49] Vikram Sreekanti, Harikaran Subbaraj, Chenggang Wu, Joseph E Gonzalez, and Joseph M Hellerstein. 2020. Optimizing prediction serving on low-latency serverless dataflow. *arXiv preprint arXiv:2007.05832* (2020).

[50] Felipe Petroski Such, Vashisht Madhavan, Rosanne Liu, Rui Wang, Pablo Samuel Castro, Yulun Li, Jiale Zhi, Ludwig Schubert, Marc G Bellemare, Jeff Clune, et al. 2019. An Atari Model Zoo for Analyzing, Visualizing, and Comparing Deep Reinforcement Learning Agents. In *IJCAI*.

[51] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. Probabilistic databases. *Synthesis lectures on data management* 3, 2 (2011), 1–180.

[52] Yi Sun, Xiaogang Wang, and Xiaoou Tang. 2013. Deep convolutional network cascade for facial point detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3476–3483.

[53] Immanuel Trummer and Christoph Koch. 2014. Approximation schemes for many-objective query optimization. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1299–1310.

[54] Immanuel Trummer and Christoph Koch. 2016. Multi-Objective Parametric Query Optimization. *SIGMOD Record* 45, 1 (2016), 24–31.

[55] Wei Wang, Sheng Wang, Jinyang Gao, Meihui Zhang, Gang Chen, Teck Khim Ng, and Beng Chin Ooi. 2018. Rafiki: Machine learning as an analytics service system. *arXiv preprint arXiv:1804.06087* (2018).

[56] Jianping Wu, Zhaobin Liu, Jinxiang Li, Caidong Gu, Maoxin Si, and Fangyong Tan. 2009. An algorithm for automatic vehicle speed detection using video camera. In *2009 4th International Conference on Computer Science & Education*. IEEE, 193–196.

[57] Ioannis Xarchakos and Nick Koudas. 2019. Svq: Streaming video queries. In *Proceedings of the 2019 International Conference on Management of Data*. 2013–2016.

[58] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live video analytics at scale with approximation and delay-tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 377–392.

[59] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 377–392. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang

[60] Wen-Yang Zhou, Guo-Wei Yang, and Shi-Min Hu. 2021. Jittor-GAN: A fast-training generative adversarial network model zoo based on Jittor. *Computational Visual Media* 7, 1 (2021), 153–157.

[61] Xuanhe Zhou, Chengliang Chai, Guoliang Li, and Ji Sun. 2020. Database meets artificial intelligence: A survey. *IEEE Transactions on Knowledge and Data Engineering* (2020).