On the utility of metadata to optimize machine learning workflows

Li, Z.

DOI
10.4233/uuid:0037aafa-5960-435b-9aea-52e266c9857c

Publication date
2024

Document Version
Final published version

Citation (APA)
Li, Z. (2024). *On the utility of metadata to optimize machine learning workflows*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:0037aafa-5960-435b-9aea-52e266c9857c

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

ON THE UTILITY OF METADATA TO OPTIMIZE MACHINE LEARNING WORKFLOWS

ON THE UTILITY OF METADATA TO OPTIMIZE MACHINE LEARNING WORKFLOWS

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology,
by the authority of the Rector Magnificus, Prof.dr.ir. T.H.J.J. van der Hagen,
chair of the Board of Doctorates,
to be defended publicly on Wednesday, 11th of September 2024 at 15:00 o'clock.

by

Ziyu LI

Master of Science in Computer Science,
Delft University of Technology, the Netherlands
and Bachelor of Engineering in Network Engineering,
South China University of Technology, China
born in Guangdong, China

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	voorzitter
Prof. dr. ir. G.J.P.M. Houben,	Delft University of Technology, promotor
Prof. dr. ir. A. Bozzon,	Delft University of Technology, promotor
dr. A. Katsifodimos,	Delft University of Technology, co-promotor
independent members:	
Prof. dr. G. Smaragdakis	Delft University of Technology, the Netherlands
Prof. dr. C. Pautasso	University of Lugano (USI), Switzerland
Prof. dr. Y. Velegrakis	Utrecht University, the Netherlands
Prof. dr. D. Abbink	Delft University of Technology, the Netherlands
Prof. dr. M. Specht	Delft University of Technology, the Netherlands, reserve member

This research was partially supported by the HyperEdge Sensing project of Cognizant.

SIKS Dissertation Series No. 2024. The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Published and distributed by: Ziyu Li

Keywords: Metadata, Machine Learning, Query Optimization, Model Selection

Printed by: ProefschriftMaken

Front & Back: Beautiful cover art that captures the entire content of this thesis in a single illustration.

ISBN 978-94-6510-112-5

CONTENTS

1

INTRODUCTION

The field of machine learning (ML) is witnessing a fast-paced improvement in terms of both available models, and their application to real-world problems. This continuous improvement creates the need for standardized metadata format and tools to describe and access the expanding repositories of ML models and associated artifacts - for instance, datasets, model architectures, training configurations, and evaluations. We explore how repositories of ML models can be described through structured metadata, and we show how the availability of rich metadata can be used to enhance ML workflows, with specific focuses on model inference and fine-tuning.

1.1. BACKGROUND AND MOTIVATION

The past two decades witnessed an explosion in the machine learning field, primarily due to the ever-growing data size and massive computing capabilities. ML in general, and deep learning (DL) specifically, has shown its excellent performance in multiple areas, including but not limited to healthcare, mobility, life sciences, energy systems, and more. Thanks to extensive computation power and open-source libraries (e.g., ML frameworks such as TensorFlow[1] and Pytorch[2]), ML approaches have never been more accessible and efficient to apply.

The success of ML techniques led to a proliferation of models and related artifacts (e.g., datasets). The absence of standardized representations and tools to access these artifacts presents a significant impediment to practitioners. ML/DL scripts are often managed through traditional software repositories like GitHub[3] or Kaggle[4]. While suitable for software engineering workflows, these systems hinder the ability of practitioners to effectively reuse (through selection or composition of models) and compare (via retrieval, evaluation, or benchmarking) ML models and associated artifacts. The current land-

[1] https://www.tensorflow.org/
[2] https://pytorch.org/
[3] https://github.com/
[4] https://kaggle.com/

scape lacks cohesive practices and support tools, highlighting a critical gap that needs to be addressed.

Model zoos have recently emerged as solutions to address the challenges above. A model zoo serves as a platform for sharing diverse information about a model and its associated artifacts, including the task and the training dataset used. One notable example is PaperWithCode[5], which presents and compares results documented in research papers. While this helps rank models based on metrics, it still necessitates external exploration, either through code or the papers themselves, to access critical information like hyperparameters and model architecture. Other model zoos may go further by maintaining the implementation scripts of (pre-)trained models and storing their weight files. Prominent public model zoos in this category include HuggingFace[6], Tensorflow Hub[7], and PyTorch Hub[8]. These platforms enable users to search for models based on properties such as the intended task (e.g., image classification), modality (e.g., text, image), model name, or development framework (e.g., PyTorch). These model zoos allow practitioners to store ML models in dedicated spaces where a minimum amount of metadata is stored, and serve as collaborative hubs for sharing ML models, allowing users to explore and employ these models with accessible APIs.

1.2. LACK OF METADATA MANAGEMENT IN CURRENT MODEL ZOOS AND ML SYSTEMS

Developers of ML frameworks indeed acknowledge the pivotal role of metadata in supporting the work of practitioners, enhancing the understanding of existing models' characteristics, and, ultimately, increasing transparency and accountability. Initiatives such as model cards [131] or data sheets [24, 59, 126] are examples of efforts to enable developers to describe their models and datasets comprehensively.

Although model zoos offer valuable repositories for locating and reusing specific models, they often fall short of providing comprehensive information encompassing all the artifacts associated with the ML lifecycle.

In the following, we will describe a general ML lifecycle [11, 31, 146, 161, 180], and discuss how current model zoos and ML systems fall short in managing metadata.

It is possible to identify four main stages: i) data preparation and data management; ii) model learning; iii) model evaluation and model verification; and iv) model deployment. Figure 1.1 depicts the four stages, and highlights which artifacts (and their properties) require consideration (and representation through metadata) in the context of managing model repositories.

1. Data preparation and data management. The first stage of the ML lifecycle relates to the acquisition and transformation of data used by the ML models. This stage can be split into the following steps.

i) Gather data samples through observations or measurements.

[5] https://paperswithcode.com/
[6] https://huggingface.co
[7] https://www.tensorflow.org/hub
[8] https://pytorch.org/hub/

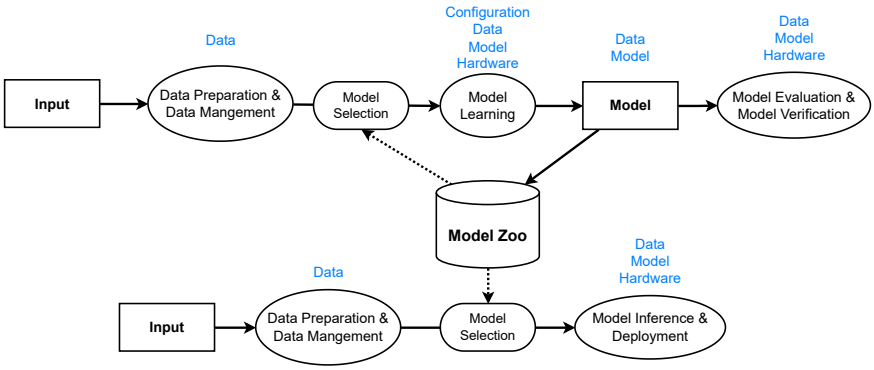


Figure 1.1: ML lifecycle with four main stages. The blue texts indicate the types of metadata associated with each stage.

ii) Data analysis: need for additional data, augmentation, and preprocessing.

iii) Data cleaning: replace or remove incomplete data from the dataset.

iv) Preprocessing: convert raw data such that ML models can use it.

v) Feature engineering: extract relevant features from raw data such that it can be used for model building.

vi) Splitting data: separate data into training, validation, and test sets.

The metadata shall be able to capture information about data preparation and processing (e.g., data source, data curation method, data statistics), as such data-related operations influence the performance of a ML model [15].

2. Model learning. The *model learning* stage concerns the design and training of the ML model. This stage can be divided into the following steps.

i) Model selection depends on the type of data (structured or unstructured).

ii) Select the loss function to measure training error.

iii) Select and tune hyperparameters to control overfitting, underfitting, and other characteristics.

iv) Model training or fine-tuning on datasets to minimize error.

v) Repeat steps 3 and 4 until good precision numbers and low training error.

Capturing metadata about the configurations, such as model architecture, hyperparameters, etc., is important to understand and interpret the performance of a model. In addition, at this stage, the metadata regarding the dataset and execution environment, e.g., hardware, also plays a role in impacting the model training. Thus, the trained model is associated with the dataset, hyperparameter, as well as the training environment.

3. Model evaluation and model verification. After the model training, the model needs to be verified on unseen (validation or testing) data. The main goal of this stage is to ensure that the model, after training, performs as expected on new inputs. Usually, this is done by assessing the performance of the trained model against a test dataset that was generated in the data management stage. The relevant metadata regarding the dataset and model shall be captured. In addition, the relevant configurations, such as hardware settings, should also be recorded.

4. Model inference and deployment. The outcome of this stage is a properly functioning, fully-fledged, and deployed ML system. At this stage, all the information from previous stages shall be revealed, such as dataset information, model training details, and, most importantly, the performance under different environment specifications. The environment specifications include hardware specifications and any specific software configurations or constraints. Fine-grained metadata helps ensure compatibility and optimal performance of the deployed model. Practitioners can thus choose the appropriate models for their needs and requirements.

1.2.1. LIMITATIONS OF CURRENT PUBLIC MODEL ZOOS AND ML SYSTEMS

Recent efforts within the ML community have been centered on democratizing ML practices, emphasizing both the utilization and sharing of ML artifacts. Platforms, commonly referred to as model zoos, exemplified by Tensorflow Hub, PyTorch Hub, and HuggingFace, have taken significant strides in exposing metadata associated with ML artifacts. It is noteworthy that some of these platforms have been developed during the duration of this thesis. These platforms are grounded in the principles of open-source sharing, making model parameters, scripts, and APIs readily accessible to users. With this thesis, we advocate for an extension in their functionalities, to capture and make full use of rich metadata throughout the ML lifecycle.

Notably, HuggingFace has introduced an abstraction layer with the *Transformers* library, simplifying the consumption and inference of these models [83]. This thesis proceeds to delineate the capabilities and limitations inherent in presently available model zoos. As a prominent public model zoo, HuggingFace currently hosts model cards for an extensive collection of over 803k models. The number of model cards is substantial, but the quality is less satisfactory. Only a few model cards are thoroughly validated by officials with details describing the models; others are customized content generated by users. There are also missing model cards for some models, making it challenging for users to understand and effectively utilize the models. Even in cases where model card content is adequate, critical information, such as training details, dataset information, and performance metrics, is often buried within lengthy textual descriptions.

Similar issues are evident in other model repositories, including ONNX[9], TensorFlow Hub, and PyTorch Hub. While these platforms offer convenient APIs for accessing models, along with associated scripts and weights, their primary focus remains on facilitating efficient model access. Comprehensive metadata coverage for related artifacts is typically lacking. Description content is frequently sourced from Markdown files hosted in external GitHub repositories. In some instances, essential performance metrics are con-

[9] https://github.com/onnx/models

spicuously absent, as observed in TensorFlow Hub and PyTorch Hub.

Other ML experimentation platforms like MLflow [196] and Kubeflow [21] often focus on tracking metrics and code versions but may not emphasize the systematic capture of metadata related to data preprocessing, feature engineering, or execution environments. This can hinder the reproducibility of complex ML workflows. A similar case applies to AutoML tools, where they automate the training process while may not adequately capture the data preprocessing and feature selection decisions. These limitations hinder the comprehensive understanding and effective utilization of models and related artifacts.

1.3. METADATA USAGE: DATABASE SYSTEMS VS. MODEL ZOOS

In the database (DB) community, on the other hand, metadata management assumes paramount importance and constitutes a well-established practice. It is common for database systems to maintain a *catalog* [108], also known as a data dictionary or data repository, used to systematically organize, store, and maintain metadata relevant to the management of the systems and its operations. The metadata within the DB catalog is not merely an abstract concept but an invaluable resource. It plays multifaceted roles by catering to the informational needs of various stakeholders, including end-users, application developers, and the Database Management System (DBMS) itself. These roles encompass a wide array of functionalities integral to the DBMS ecosystem.

Despite the availability of rich metadata, the extent to which these valuable resources are leveraged for various ML tasks and applications remains limited. ML frameworks and model repositories often do not seamlessly integrate metadata management into their workflows, and fail to exploit the metadata, which hinders the potential of these metadata exploited within the current ML ecosystems. This lack of integration can make metadata an afterthought rather than a fundamental part of the ML lifecycle. This underutilization signifies a missed opportunity to enhance ML models' transparency, interpretability, and utility and their associated artifacts within the broader ML landscape. Thus, there is a compelling need to bridge this gap and unlock the full potential of metadata to advance the field of ML.

In the following sections, we introduce examples of the application of metadata in the field of databases to provide intuitions as to how model zoo metadata could prove beneficial in ML practice. A comparison is presented in Table 1.1. We look at three aspects, namely: *catalog and profiling, provenance, query optimization and execution.*

Table 1.1: Comparison of metadata applications in database and ML research

Database practices	**ML practices**
Data catalog Data Profiling	Model card/Data sheet
Schema evolution Data lineage and Provenance	Model provenance
Query optimization/execution	Query optimization/execution (Model inference/training/deployment)

1.3.1. DATABASE PRACTICES USING METADATA

Catalog and Profiling. The first category delves into the applications of metadata in terms of building a catalog and profiling the data. These elements are essential in harnessing the full potential of an organization's data assets and ensuring their efficient and effective use.

- **Data Catalogs**. Metadata is used to create data catalogs that provide a comprehensive inventory of available data assets within an organization. Data catalogs offer a structured view of the database schema, table definitions, column attributes, and data types, and usually include the meaning and attributes of the contained variables as well as information about the creation, format, and usage of the data [26]. They help users discover relevant datasets, understand their characteristics, and promote data sharing and reuse [185]. With robust metadata as the core of the data catalog, many other features and functions are supported, e.g., dataset searching and dataset evaluation.

- **Data Profiling and Quality Assessment** Metadata can be used to profile data quality by recording statistics on data completeness, accuracy, and consistency. This information assists data quality assessments and data cleansing processes and helps identify data anomalies [93].

Provenance. In the dynamic world of data management, understanding the origins and transformations of data is essential for maintaining data integrity and quality. This necessity brings to the forefront the concept of provenance, a critical aspect of data management underpinned by the effective use of metadata. We explore the role of metadata in facilitating provenance, focusing on schema evolution and data lineage, and how these aspects are integral to modern database management practices.

- **Schema evolution** In evolving database schemas, metadata can help manage schema changes smoothly. Database administrators can track and document alterations over time by recording metadata about schema versions and changes, aiding in database version control and migration [2].

- **Data Lineage and Provenance**. Metadata can be used to track the lineage and provenance of data, ensuring that users can trace data back to its source. This is essential for data quality assurance, auditing, and understanding data origins [27].

Query optimization and execution. Metadata plays a crucial role in query processing and execution in database research, serving as a foundational element that enhances efficiency and accuracy. Its application spans various aspects of DBMS, from optimizing query execution plans to ensuring efficient data retrieval and processing.

- **Query processing**. Metadata is a valuable aid during query processing, guiding the DBMS in several ways. Metadata can be leveraged to acquire and share data, support data integration [57], reduce query processing overhead [145], and boost interactivity [88].

- **Query optimization**. Query optimization is a fundamental research topic in database research [13, 34, 82, 105, 121]. Metadata is instrumental in optimizing the execution plans of queries within a database management system. By providing essential insights into the data distribution, indexing structures, and access patterns, metadata enables the query optimizer to make informed decisions about query execution strategies. For instance, knowing the cardinality of tables, the distribution of values in columns, and the presence of relevant indexes allows the optimizer to select the most efficient join algorithms and access methods, thereby enhancing query performance.

- **Performance tuning** Database administrators (DBAs) and system operators utilize metadata to monitor and finetune the DBMS's performance [155]. Metadata can reveal valuable insights into resource utilization, query execution times, and bottlenecks. Armed with this information, administrators can proactively address performance issues, allocate resources optimally, and ensure the smooth operation of the DBMS.

1.3.2. PRACTICES AND INSIGHTS FOR ML RESEARCH USING METADATA

The significance of metadata can extend to ML practices, albeit with different focuses. Leveraging the practices of using metadata in database research provides valuable insights for enhancing the ML workflow. The principles of metadata management in databases can be adapted and applied to various stages of the ML lifecycle, from model profiling to model deployment.

Below, we focus on a few practices that exploit metadata for ML, i.e., model profiling, model inference, and model fine-tuning. These focal points align with the thematic areas addressed in this thesis.

- **ML lifecycle profiling**. Similar to data catalogs in databases, metadata in ML can be used to create comprehensive catalogs of models and datasets. These catalogs can provide detailed information about model architectures, training configurations, dataset characteristics, and performance metrics. This aids in discovering and selecting appropriate models and datasets for specific ML tasks.

- **Optimizing model selection for complex inference tasks.** Addressing complex inference tasks often necessitates the utilization of multiple ML models to fulfill intricate queries [86, 120, 192]. Identifying the most suitable models for specific tasks can pose a formidable challenge in scenarios where a substantial collection of models is available within a model zoo. To navigate such complexities, practitioners require a deep understanding of the capabilities of available models. Detailed metadata can empower practitioners to make informed decisions regarding model selection, ensuring that the chosen models align with the desired outcomes.

- **Facilitating model selection with a model zoo for fine-tuning.** Model fine-tuning in ML is a process that retrain a pre-trained ML model to make it perform better for a specific task or dataset. This is particularly common in deep learning, where

large pre-trained models on vast datasets are adapted to work on smaller, task-specific datasets. The process of fine-tuning ML models from a model zoo can be resource-intensive and time-consuming. Without adequate information about the models' characteristics and performance, practitioners may be compelled to finetune numerous models, consuming extensive computational resources. The availability of comprehensive metadata can alleviate this burden by enabling practitioners to identify the most promising candidates for fine-tuning. This targeted approach, utilizing metadata, not only enhances efficiency but also increases the likelihood of achieving superior fine-tuning results while conserving computational resources [110].

In essence, the DB community's meticulous approach to metadata usage underscores its significance and the tangible benefits it brings to the efficient operation of DBMSs. By drawing parallels with this established practice, the ML domain can gain valuable insights into the potential advantages of adopting a similarly robust metadata management framework to enhance ML workflows' transparency, reproducibility, efficiency, and effectiveness.

The utilization of metadata within ML artifacts holds significant promise for addressing prevailing challenges and enhancing the transparency of the ML workflow. By providing detailed information on the properties and dependencies of ML artifacts throughout the entire ML lifecycle, we can open up a wealth of opportunities for improvement, besides the ones mentioned earlier. These opportunities span a spectrum of applications, offering benefits such as increased workflow efficiency, more effective model selection for complex inference tasks, and overall productivity enhancements for ML practitioners.

1.4. RESEARCH QUESTIONS

In this thesis, we address the following research question:

How to represent and utilize metadata within ML model zoos with the aim of enhancing the processes of model training, inference, and fine-tuning?

It is apparent that this single doctoral thesis alone cannot address this broad research question. Yet, it highlights the focus on this thesis, i.e., how to *use metadata* to support ML workflows. Before we exploit the usage of the metadata, we first require structured representations to capture the comprehensive metadata. Thus, we start by describing work on ML metadata representation. With the representation, we further utilize the metadata to enhance various ML workflows, i.e., model inference and fine-tuning by selecting the 'right' models. We aim to improve the efficiency and effectiveness of these workflows.

We formulate the following four high-level questions to address the main research question:

Q1: How to represent metadata in a model zoo in a structured and queryable fashion?

By answering this question, we identify the metadata of different artifacts captured throughout the ML lifecycle. We design a metadata model that represent metadata of different artifacts (e.g., model, dataset, configuration). We implement the metadata model

with a actively-used database, and present a tool that can extract, store and query the metadata. The structured and queryable metadata representations enables the management and access to the diverse metadata, allowing practitioners to explore and monitor the metadata, and providing opportunities to be applied in downstream tasks for different purposes.

With the metadata being captured and stored, we continue to identify the metadata that can help speed up the model training and inference process. Arun's work [36] was identified to be effective in improving the speed of model training either in a factorized way or materialized way depending on the attributes of the data. In this research question, we identify the necessary metadata that can be applied to make decision on whether to factorize or to materialize. To train a model in a factorized or materialized way may affect the training and inference time, which is the efficiency of the ML workflows. The answer to this question shows the fine-grained metadata can assist ML tasks by improving the efficiency.

This work is presented in Chapter 2, and it is based on the following publications:

- Li, Z., Kant, H., Hai, R., Katsifodimos, A., Brambilla, M., & Bozzon, A. (2023). Metadata Representations for Queryable Repositories of Machine Learning Models. *IEEE Access.*
- Li, Z., Kant, H., Hai, R., Katsifodimos, A., & Bozzon, A. (2023, June). Macaroni: Crawling and Enriching Metadata from Public Model Zoos. *In International Conference on Web Engineering* (pp. 376-380). Cham: Springer Nature Switzerland.
- Li, Z., Sun, W., Zhan, D., Kang, Yan., Chen, L., Bozzon, A., & Hai, R. (2024, February). Amalur: the Convergence of Data Integration and Machine Learning. *IEEE Transactions on knowledge and data engineering 39, no. 1.*

Q2: How can the metadata be leveraged to enhance the efficiency and effectiveness of ML inference queries?

We move on to the scenarios of serving ML models. In this stage, practitioners would use ML models to perform certain task, e.g., complex analytic tasks with several models involved. In this context, we only focus on ML inference. We referred to this type of queries with ML models as operators as ML inference queries. Given different constraints, e.g., accuracy requirement or latency constraint, it would be time-consuming and expensive to use human labor to select models to answer the queries, or it might yield poor performance if the model selection only based on a single objective, e.g., accuracy, ignoring the effect on other objectives.

This work focuses on optimizing ML inference queries in the context of large model zoos, where selecting the right ML models and determining their execution order is challenging. To address this, we propose a method that formulates the problem as a Mixed Integer Programming (MIP) problem and develops an optimizer. This optimizer maximizes accuracy while considering constraints, offering a solution to the constraint-based ML inference query optimization problem, even in complex scenarios.

This work is presented in Chapter 3, and it is based on the following publications:

- Li, Z., Schönfeld, M., Sun, W., Fragkoulis, M., Hai, R., Bozzon, A., & Katsifodimos, A. (2023, June). Optimizing ML Inference Queries Under Constraints. *In International Conference on Web Engineering* (pp. 51-66). Cham: Springer Nature Switzerland.

- Li, Z., Schönfeld, M., Hai, R., Bozzon, A., & Katsifodimos, A. (2023, April). Optimizing machine learning inference queries for multiple objectives. In 2023 *IEEE 39th International Conference on Data Engineering Workshops (ICDEW)* (pp. 74-78). *IEEE.*

Q3: How can metadata, along with other representations be employed to predict the performance (i.e., accuracy) of models without undergoing the fine-tuning process?

In the previous scenarios, we assume that the data distribution remains the same, such that the trained models can be used for inference directly. In this research question, we investigate in the situation when there is a shift in data domain. The pre-trained models are no longer available. We aim to select a good (effective) set of models which would perform well after fine-tuning. In such a way, we reduce the computation cost and time (efficiency) by fine-tuning only the good-performing models. Otherwise, we may need to fine-tune a large set of models, which is costly and also time-consuming. The answer to this question shows that we can utilize the metadata (e.g., of models, and of datasets) to select models for fine-tuning with the purpose of improving the effectiveness and the efficiency.

This work is presented in Chapter 4, and it is based on the following publications:

- Li, Z., Van der Wilk, H., Zhan, D., Khosla, M., Katsifodimos, A., Bozzon, A., & Hai, R., (2024, May). Model Selection with Model Zoo via Graph Learning, *In 2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE.

1.5. THESIS OUTLINE

This thesis is organized in 5 chapters, as depicted in Figure 1.2. In Chapter 2, we propose a metamodel that represents the metadata within the ML context. Continue in Chapter 3, we explores the exploitation of metadata for various ML applications. In specific, Chapter 4 utilizes metadata for ML inference query optimization, where constraints are specified. We aim to select models from a model zoo for fine-tuning. We investigate how to utilize the metadata to understand and predict the model performance.

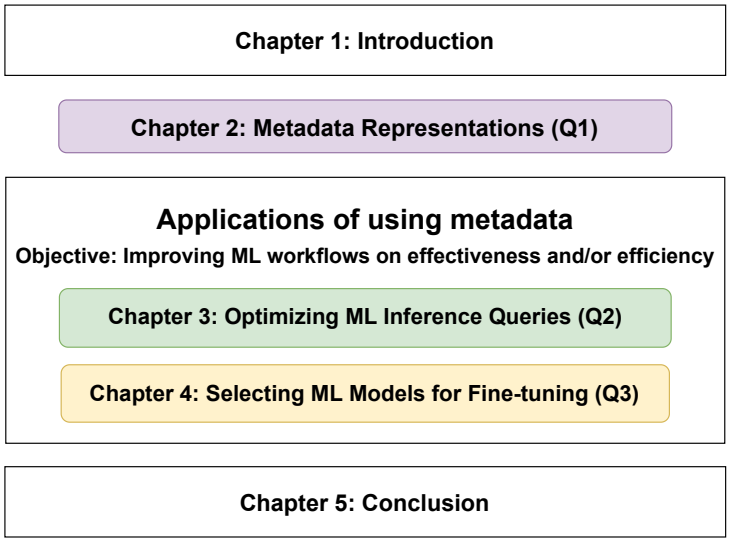


Figure 1.2: Structure of the thesis

2

METADATA REPRESENTATIONS

Machine learning (ML) practitioners and organizations are building model repositories of pre-trained models termed *model zoos*. These model zoos contain metadata describing properties of the ML models and datasets that are useful for reporting, auditing, reproducibility, and interpretability purposes. Despite the growing adoption of description formats like datasheets and model cards, metadata present in current model zoos is very limited. Moreover, existing formats have limited expressiveness, thus ultimately limiting the potential use of model repositories beyond the simple storage of pre-trained models. In this chapter, we introduce a unified metadata representation format for model zoos. We show how such rich metadata enables a broad set of use cases, which include the search, reuse, comparison and composition of ML models, and the acceleration of ML model training, and inference. We also describe the design, and showcase the implementation of an advanced model zoo system built on top of our metadata representation.

This chapter is based on the following publications:

- Li, Z., Kant, H., Hai, R., Katsifodimos, A., Brambilla, M., & Bozzon, A. (2023). Metadata Representations for Queryable Repositories of Machine Learning Models. *IEEE Access*.
- Li, Z., Kant, H., Hai, R., Katsifodimos, A., & Bozzon, A. (2023, June). Macaroni: Crawling and Enriching Metadata from Public Model Zoos. *In International Conference on Web Engineering* (pp. 376-380). Cham: Springer Nature Switzerland.
- Li, Z., Sun, W., Zhan, D., Kang, Yan., Chen, L., Bozzon, A., & Hai, R. (2024, February). Amalur: the Convergence of Data Integration and Machine Learning. *IEEE Transactions on knowledge and data engineering 39, no. 1*.

2.1. INTRODUCTION

Machine learning (ML) is increasingly used across application domains such as video analytics [156, 192], autonomous driving [76], content moderation [62], traffic monitoring [133] and crowd detection [157]. While ML models can be (and often are) trained for specific purposes, there is a growing interest in reusing and re-purposing of pre-trained ML models [78]. This shift, motivated mainly 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). Thanks to model zoos, complex predictive and analytics tasks can benefit from reusing existing ML models.

The potential of model zoos is currently hindered by the lack of *structured, comprehensive, and queryable* metadata representations. Current repositories include a wide range of information, e.g., using model cards [131]. However, such information is mostly for human consumption, and the level of detail remains coarse-grained, thus preventing advanced repository automation and management functionalities. Table 2.1 presents the information provided by different public model zoos. The categories of the information cover different aspects of ML artifacts (e.g., model, dataset, performance). We observe that current model zoos only provide limited information; for instance, PyTorch Hub provides only the ReadMe files from the source (e.g., a GitHub repository). Insufficient information forces practitioners to search for additional metadata in external repositories and descriptive documents or repeatedly go through the ML lifecycle. These processes impede the reuse of models and hamper their evaluation and assessment.

Table 2.1: The existing public model zoos encompass various categories of metadata. The presence of metadata in each category can be denoted by symbols: ✓ indicates that the metadata is available, ~ indicates that the metadata is available only in certain cases or provides limited information. Additionally, *queryability* is used to describe the type of queries that can be supported by the model zoo.

	Dataset Metadata				Evaluation Metadata		
Model Zoos	**Source**	**Statistics**	**Configuration Metadata**	**Prediction Metadata**	**Metrics**	**Hardware specifications**	**Queryability**
HuggingFace [83]	✓	~	✓		~		Faceted search
PyTorch Hub [147]			~		~		Faceted search
TensorFlow Hub [173]	✓		~		~	~	Faceted search
Papers with Code [144]	✓				✓		Faceted search
OpenVino [142]	~		~		✓		
DawnBench [43]	✓		~		✓	✓	Faceted search
Macaroni's Metamodel	✓	✓	✓	✓	✓	✓	Structured queries

The software market provides several tools and platforms designed to help manage the ML lifecycle and experiment tracking with the support of metadata (e.g., MLflow [35], Weights & Biases [184], ClearML [41], comet [44]). However, the metadata is not standardized and requires custom formats from its users through API calls. While this approach may be convenient for users who wish to manage their private repositories, it falls short of facilitating broader knowledge sharing, integration, and reuse. This limitation

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

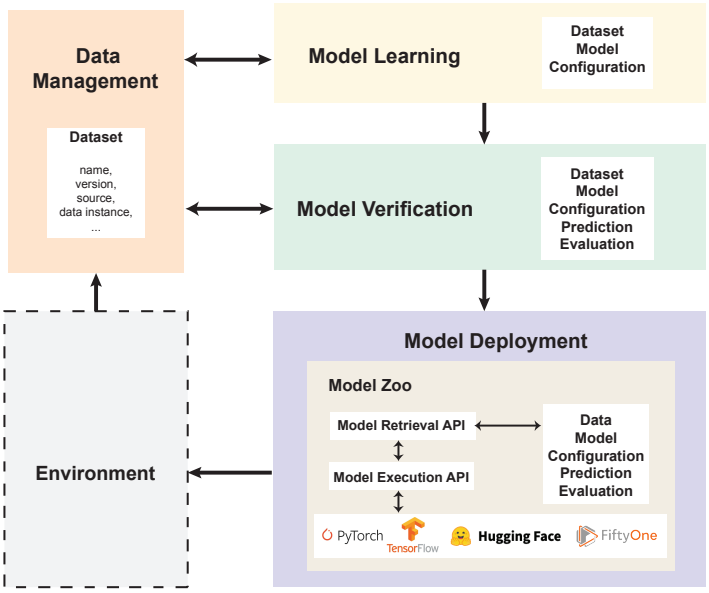


Figure 2.1: ML lifecycle along with metadata of different artifacts. We highlight the model zoo in model deployment phase with detailed components.

ultimately hampers the full potential of leveraging metadata to enhance and optimize the ML lifecycle. Unlike these works focusing on managing the ML lifecycle, our work is specifically dedicated to addressing the metadata needs required to support model repositories and enable advanced functionalities, such as model retrieval and composition.

While data profiling techniques can determine metadata for datasets [1], the growing scope of ML applications (including their undesired effects on individuals and society [195]) requires metadata able to describe all the ML artifacts included in model zoos. The metadata should include information such as a model's inference capabilities (e.g., identified object classes), architecture, inference time, datasets (for training/validation/test), configurations (e.g., hyper-parameters values), and evaluation performance (refer to Table 2.3 for a complete list of metadata).

Such metadata can be helpful in different phases across the entire ML lifecycle [160, 161], as shown in Figure 2.1. A few examples include: i) *Data cleaning and preprocessing*: The metadata of the training dataset can assist practitioners in identifying erroneous instances such that they can exclude the problematic data points during training [81]. ii) *Model selection*: Metadata accelerates the learning process by setting warm-starting of hyper-parameter searches instead of random search [191]. iii) *Model evaluation*: Metadata can facilitate keeping track of the performance of different objectives [78], as well as the predictions of each instance for further analysis (e.g., model explainability). iv) *Model explainability and reproducibility*: Metadata management can be used in the

context of Trustworthy and Responsible AI[2]. Metadata can facilitate model explanation with model predictions and annotations [95]. v) *Model serving*: Metadata about model inference can perform model comparison and help practitioners decide which to use in production [66].

In addition, the availability of model metadata can also facilitate machine learning operations, commonly referred to as MLOps [101], and opens up new opportunities for advanced use cases such as i) retrieving models from large repositories with complex filtering conditions; ii) continuous integration of models in production (e.g., using transfer learning [171, 205]); iii) (semi-)automatic model composition; and iv) advanced model management system. We will further discuss the use cases in Section 2.2.3.

In this chapter, we advocate for expressive metadata representation for model zoos. Beyond the current state-of-the-art (and practices) [5, 159], we propose a metadata format that can capture information about relevant artifacts (e.g., models, datasets, data instances, training configurations, evaluation) and their relationships. We also describe the design and current implementation of an advanced ML models management platform called *Macaroni* [111][3] that can be used to query and make use of such metadata. As seen in Table 2.1, our proposed metadata representations cover various categories of metadata and can support the querying of it, as in the example in Table 2.2.

The contributions of this chapter are the following:

- We introduce a structured and queryable metadata model designed to provide comprehensive representations within model zoos, as detailed in Section 2.3.
- To validate our metadata model, we present *Macaroni*, a reference tool offering retrieval and analytical functionalities, operating across several model zoos. These functionalities are explained in Section 2.4.1.
- Our work demonstrates how the metadata model facilitates automatic model reuse and composition. This is achieved through Boolean expressions over inference predicates and performance constraints, as outlined in Section 2.6.

2.2. ML MODEL ZOOS

The purpose of a model zoo is to store and provide access to different artifacts – and their descriptions – created throughout the lifecycle of ML. In this section, we first briefly describe the ML lifecycle, highlighting relevant artifacts. We then describe the organization and functionalities of model zoos.

2.2.1. MACHINE LEARNING ARTIFACTS

In the proposed metadata model, we tackle relationships among the artifacts in the entire ML lifecycle, including data collection, model training, model inference, serving, and reporting. We include the following artifacts that we believe should form the pillars of a rich model zoo: i) *Model*, ii) *Dataset*, iii) *Configuration*, iv) *Prediction* - semantic capabilities, v) and *Performance*.

[2] https://partnershiponai.org/paper/responsible-publication-recommendations/
[3] Prototype available at https://sites.google.com/view/macaroni-model-zoo/home

ML Model. Throughout the years, we have witnessed the advances of ML, and new models are developed with performance even surpassing human-level capabilities in many real-world tasks. Compared to the traditional ML models, such as regression models and decision trees, deep neural networks are far more complicated due to their complex architecture and their large number of parameters (some model sizes going beyond billions or trillions of parameters). To differentiate one model from others, simply knowing its name is insufficient. Additional information needs to be provided, not only for model reporting and reproducibility purposes but also for explainable AI.

Dataset. In recent years, there has been a notable emergence of the Data-centric discourse within the ML community, as elucidated in the study by Miranda et al. [129]. This emphasizes the crucial role of datasets in the training and testing of ML models. Datasets are crucial in ML for training models, evaluating performance, addressing biases, guiding feature selection, supporting transfer learning, assessing generalization and robustness, and promoting reproducibility and transparency. They serve as the foundation for developing effective and trustworthy ML models. Especially, understanding the data source and the collection methods becomes essential when diagnosing or debugging a model; yet, unfortunately, it is a frequently overlooked aspect in dataset reporting [24].

Configuration. A set of important configurations for model training are hyperparameters. By carefully selecting and tuning the hyperparameters, practitioners aim to optimize the model's ability to learn patterns from the data, reduce overfitting, and achieve better generalization to unseen samples. Besides hyperparameters, the configuration settings for ML model training or inference also include hardware configurations, such as CPUs, GPUs, or specialized accelerators. The hardware configurations not only affect the performance on latency measurement [154] but also on accuracy [170].

Prediction. Training a ML model is to fulfill a specific task for a real-world problem, e.g., image classification or named-entity recognition. Usually, the predictions are associated with the labels of the tasks. Researchers nowadays are investigating the semantic meaning of the predictions as concepts [16] to debug or explain the capability of a model. We highlight that the prediction outputs of a model with associated semantics play a significant role in explainable AI [16].

Evaluation. The most straightforward way to observe the capability of a model is to evaluate it on a particular task. The majority of current deep learning performance benchmarks only measure the aggregated performance of the model, such as overall accuracy or processing time for a single minibatch of data. ML model performance is much more complicated in practice. Recent works, such as DAWNBench [42], propose measuring ML models' performance from diverse perspectives, including training time, inference latency, accuracy, etc. Besides these measurements, our proposed metadata system for model zoos also presents the per-class performance and supports a broader range of models.

2.2.2. EXISTING MODEL ZOOS

Recently, communities have been focusing on democratizing ML, for both using and sharing. Platforms, often referred to as model zoos, such as Tensorflow Hub, PyTorch Hub, and HuggingFace, are now exposing metadata related to ML artifacts. These plat-

forms/hubs are building on the principles of using open-source model parameters, scripts, and APIs. HuggingFace also offers an abstraction with the *Transformers* library, which makes it easy to consume and infer these models [83]. In the following, we describe the capabilities and limitations of currently available model zoos.

Metadata. Existing model zoos offer users model cards [131] that describe the models in different levels of detail. Model cards contain metadata regarding different artifacts: For example, in TensorFlow Hub, they include metadata such as model publisher, architecture, data that train the model, inputs and outputs. Pytorch Hub includes model description, usage, and results. Besides the mentioned metadata, HuggingFace also includes discussions on the bias and limitations of the model. The metadata in these model zoos provides practitioners with different aspects of the artifacts.

Functionalities. Given the metadata, model zoos provide access to retrieving the models by means of filtering by name, task, or trained data. Some of the model zoos also support data retrieval given provided metadata of the related dataset. On top of all functionalities, model inference and sharing is the most important feature of these model zoos. They provide corresponding APIs for model execution. Practitioners can consume these public ML models for downstream tasks, e.g., building applications on top of the models, and finetuning the models on the specified dataset. For example, the APIs of HuggingFace are built on top of transformers [186], and users can easily infer models in NLP domains.

Limitations. Model zoos differ in the type and level of detail for the metadata associated with the different ML artifacts. Therefore, practitioners will need to manually retrieve and integrate information from various platforms if the same artifact (e.g., a dataset) is hosted in multiple zoos. Metadata quality and consistency are also an issue: often, models are described with little or no metadata; for example, it is very common for users of the HuggingFace platform to upload only the trained models without additional descriptions. Finally, metadata are often not offered in a computer-readable format: most of the descriptions of the artifacts are presented in plain texts, created only for human consumption. Obviously, this greatly hinders the retrieval capabilities of the model search engines and the ability of practitioners to compare different models.

2.2.3. REQUIREMENTS FOR FUTURE MODEL ZOOS

With the existing public model zoos, we observe some limitations that hinder the reuse, sharing and management of ML models. We foresee that the future model zoo should contain one of the following attributes: i) rich metadata representations that enable querying the repository; ii) rich analytic capabilities to facilitate advanced performance evaluation and comparison; iii) offering different functionalities, including serving the models through APIs or endpoints, and integrating the downstream systems that (e.g.,) accelerate model inference.

Rich metadata representations. The model zoo should contain information regarding different artifacts, e.g., data, ML model details, model performance, etc. This information allows practitioners to be aware of how the model is defined, on what dataset it is trained, and the corresponding evaluation performance. With rich metadata representation, practitioners may not only have access to comprehensive information but also

Table 2.2: Example queries

Property	ID	Query
Dataset information	1	Retrieve text classification models trained on dataset crowdsourced by at least a group of 50 people
	2	Find a dataset collected from COCO and OpenImage with all the images containing "dog"
Model performance	3	Retrieve models trained on ImageNet with an accuracy higher than 90%
	4	Which model performs the best on COCO for person detection?
Interpretability	5	Retrieve a person detection model with no gender bias
	6	Retrieve text generation models that do not generate hate speech
Hardware-related	7	Retrieve image classification models that are suitable to deploy on edge devices

search/query on top of it, which enables data/model search, data/model discovery, and comparison. Table 2.2 lists some example queries that could be valuable for ML developers and users. These queries require more fine-grained model metadata that current model repositories, such as HuggingFace, do not support. This highlights the need for detailed metadata of trained ML models and datasets in a structured and queryable representation.

Operations that facilitate advanced performance evaluation. A model zoo is not a mere information presentation platform, but it shall also serve as a tool for practitioners to facilitate advanced performance evaluation. When a new/updated dataset is provided (provided by the practitioners or added by the system), the model zoo shall allow automated evaluation/finetuning or provides operations for the practitioners to evaluate/finetune models on top of it. For example, a practitioner would like to test the robustness of a model by evaluating the model performance on a perturbed dataset. To facilitate this, the model zoo shall first allow operations on a dataset for perturbations (e.g., adding noise, adversarial attacks) or users uploading their own data. In addition, the model zoo shall also provide APIs to perform the evaluation of the dedicated models on the perturbed dataset.

Other functionalities. The purpose of having a model zoo includes sharing and serving a model. The model zoo shall also provide easily-access APIs or endpoints to serve/deploy a model. With models of various characteristics (e.g., tasks to answer, evaluation performance), the model zoo makes it feasible to solve complex analytic tasks by constructing workflows through the composition of models. Optimizations can focus on how to define the workflow under requirement constraints (e.g., accuracy or latency) [113], or how to assign workload on heterogeneous hardware (e.g., edge or server) [187].

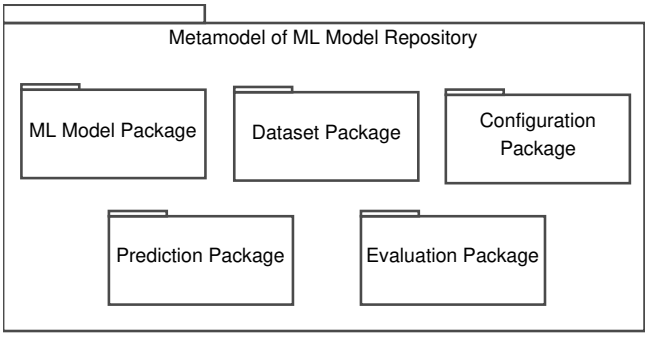


Figure 2.2: The packages in the metamodel

2.3. PROPOSED METAMODEL

Based on the analysis of the current capabilities and limitations of current model zoos, we now describe the metadata format (i.e., the *metamodel*) that can be used to represent different ML artifacts and their relations. With our proposed structured representation along with comprehensive metadata, users can retrieve metadata in fine-grained details.

Figure 2.2 depicts the main sub-models that compose our *metamodel* in a bird's eye view and Figure 2.3 presents the details.The *metamodel* comprises five packages:

i) the `ML Model` package, which defines the ML models, their architecture, input, and output formats;

ii) the `Dataset` package, which contains the information on the datasets;

iii) the `Configuration` package, which summarizes the configuration settings when training and using a model for inference;

iv) the `Prediction` package, which describes the inference output of the model, possibly enriched with description from a knowledge graph;

v) the `Evaluation` package, which presents the different evaluation metrics, including the ones related to output accuracy and to time performance.

Table 2.3 shows the summary of the metadata included in our metamodel, and the associated artifacts.

2.3.1. ML MODEL PACKAGE

The taxonomy for the `ML Model` package, depicted in Figure 2.4, encompasses classes with a white background, delineating various components such as ML Model basic information and the algorithm. The metadata associated with the `ML Model` encompasses essential details such as name, version, tasks, input and output specifications (I/O), as well as a URL linking to the script files. The `algorithm` specify the ML algorithms in different categories: traditional ML algorithms, such as SVM, decision tree, and Deep Neural Network (DNN). DNN can be further divided into different types of networks, e.g.,

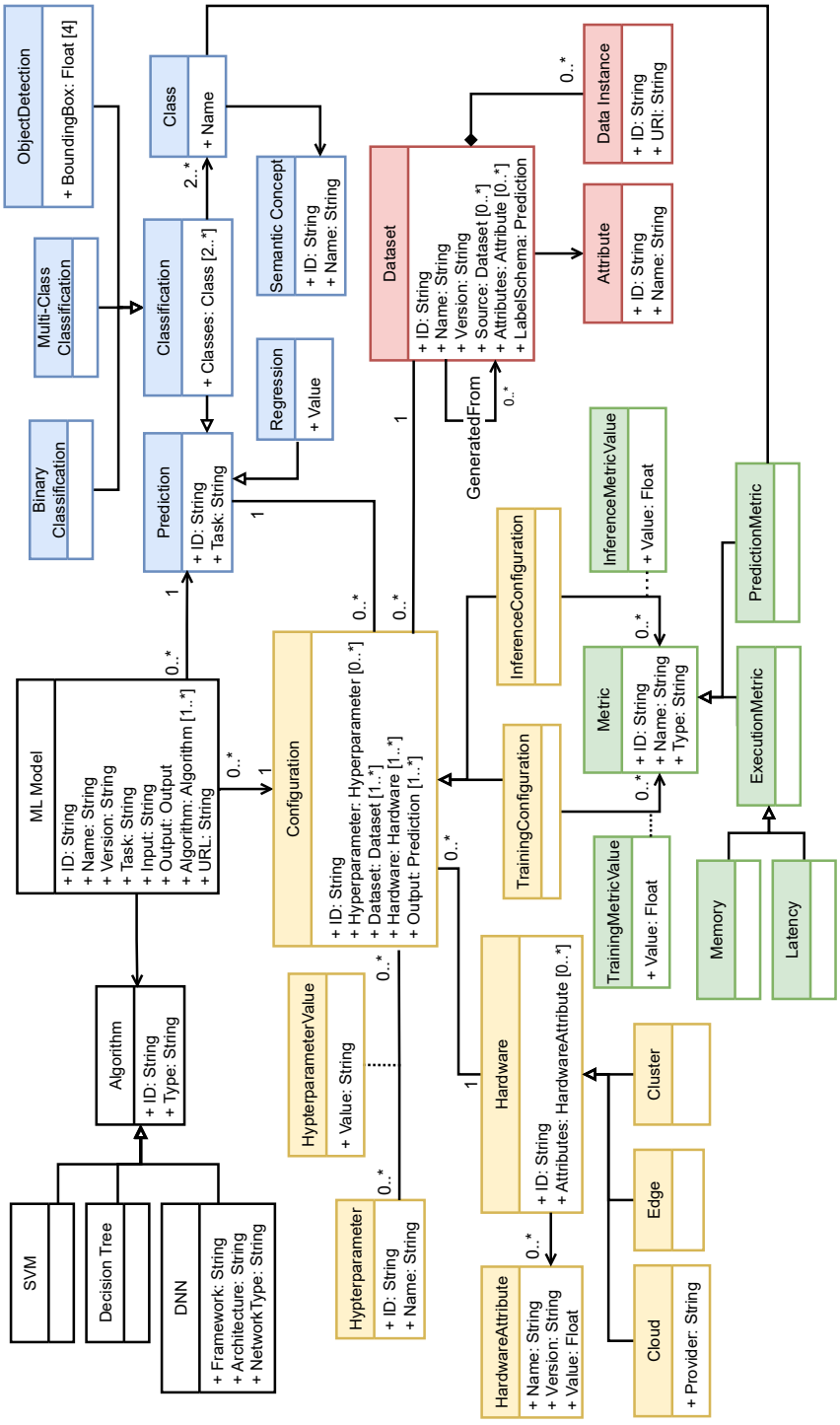


Figure 2.3: Modeling the metadata throughout ML lifecycle

Table 2.3: Metadata summary

Artifacts	Metadata type
ML Model	ID, name, version, task, architecture, input & output format/size, source URL, algorithm (e.g., SVM, decision tree, DNN
Dataset	ID, name, version, source, label schema, data instance ID, data instance attribute, data instance URI
Configuration	ID, configuration type (training/inference/testing), hyperparameter name and value, hardware attribute version
Prediction	ID, task, classes (classification task), values (regression task), bounding box (object detection task), semantic concepts
Evaluation	evaluation type, evaluation metric name, evaluation value

CNN and RNN. One of the trends of advancement in DNNs is characterized by continuous innovation and the development of increasingly sophisticated architectures, e.g., transformers [186], GPT [150], BERT [50]. Recording the framework and architecture of these advanced models is thus fundamental. The availability of such metadata is of importance in facilitating model management, model understanding, and interoperability within the ML ecosystem, and it promotes trust and confidence in the models.

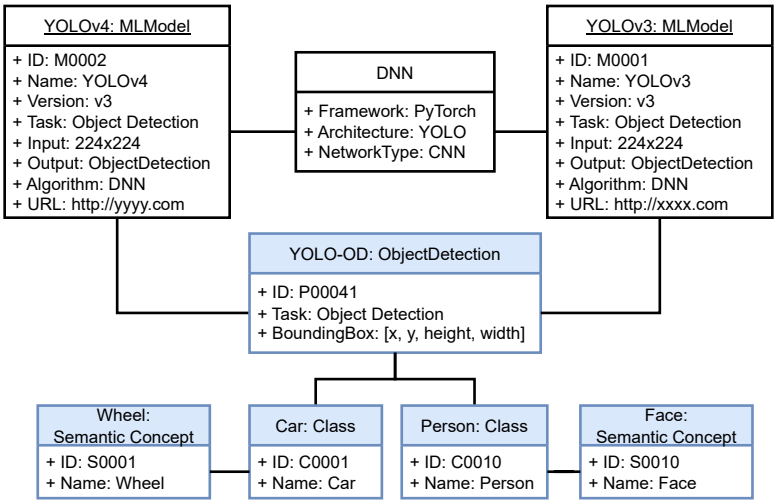


Figure 2.4: *ML Model* and *Prediction* model extract for a running example

2.3.2. DATASET PACKAGE

The behavior of a ML model heavily relies on the data that has been used for training it. Thus the *metamodel* includes a `Dataset` package, representing both `Datasets` and their *DataObject*s. With the `Dataset` element, we present the metadata of the datasets that is significant for data management and reporting. Examples of the metadata are

i) *ID*, to uniquely refer to a dataset; ii) *Name*, name of the dataset; iii) *Version*, version of the dataset, e.g., COCO has multiple versions constructed in different years; iv) *Source*, reference to other dataset(s) which (partially) construct the current dataset; v) *Attributes*, the attributes contained in the dataset, specifically for structured datasets, specifying different columns.

A dataset consists of multiple data objects. With `DataObject`, we denote the i) *ID*, to uniquely refer to a piece of content; and ii) *URI*, a string that unambiguously identifies the location of the content.

Figure 2.5 shows the datasets applied in the example. COCO is a popular image dataset that is usually used to train object detection and image segmentation models. In this case, we split COCO into a training set and a testing set. Both datasets have the same source, which is the complete COCO dataset. The COCO training set and test set contain a different subset from the complete dataset.

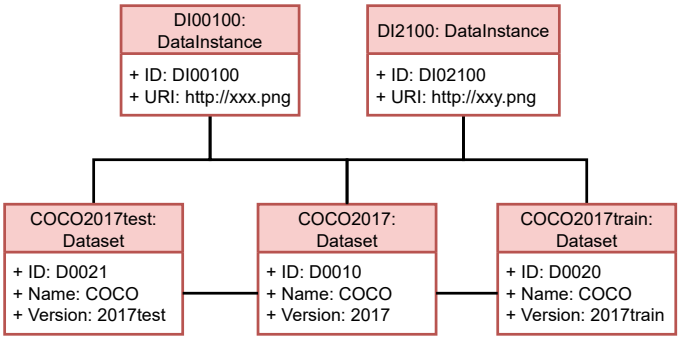


Figure 2.5: *Dataset* model extract for a running example

2.3.3. CONFIGURATION PACKAGE

The `Configuration` package encompasses essential concepts that establish connections to various packages. These packages define the associated model, dataset, hardware specifications, predictions, etc. The `Configuration` model is related to multiple entities, `ML Model`, `Hardware`, `Predication`, and `Dataset`. It associates the model with dataset and hardware, indicating where the model is trained on with which dataset. A different associated training dataset will result in a different model, with different learned parameters/weights. Within the `Configuration` package, `Hyperparameter` model, `Hardware` model, and `Configuration` are three main components. These components collectively define the essential settings for both training and inference processes. They play a crucial role in determining the behavior and performance of the model throughout its lifecycle.

The `Hardware` model contains the concepts that denote the hardware that a model is trained on or executed on. The `Hardware` model comprises metadata that describes the hardware setting, associated with different `HardwareAttributes`. We list three types of hardware, i.e., *Cloud* resources, *Edge* devices, and *Cluster*s. The *Cloud* resources associated with the public cloud services, such as AWS, Google Cloud, and Azure. An example

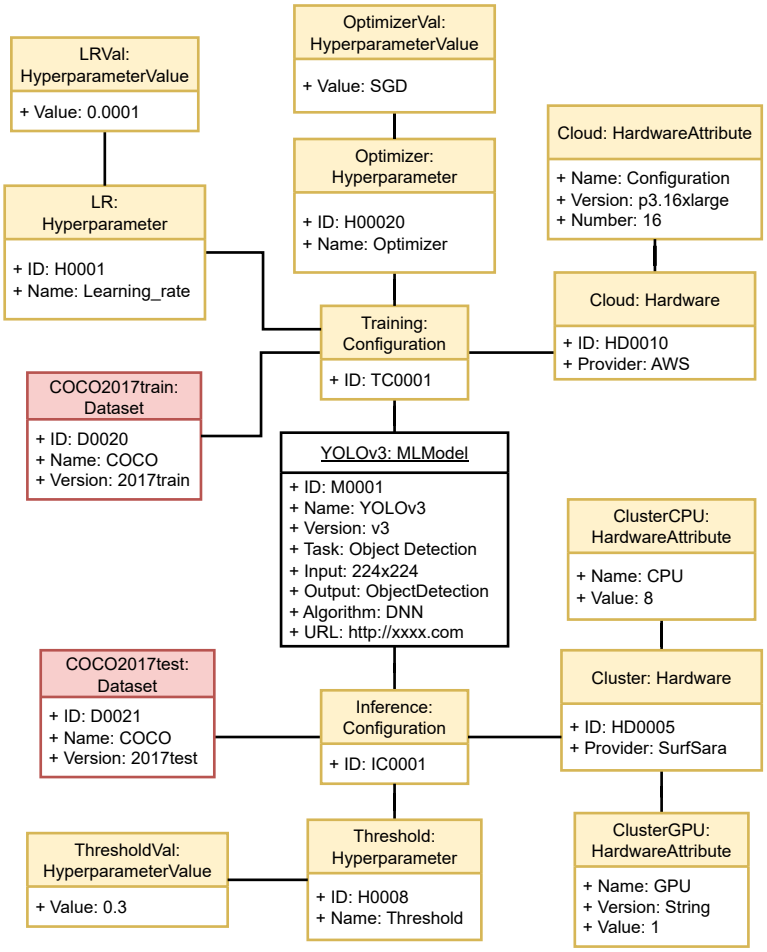


Figure 2.6: *ML Model* and *Configuration* model extract for a running example

of the hardware attributes for the *Cloud*, is the cloud configuration, e.g., 16 p3.16xlarge supported by AWS. For *Edge*, the hardware attributes include the type of the device (e.g., mobile phone, camera), storage capacity, memory, CPU, or GPU settings. Similar to *Edge*, a *Cluster* comprises metadata such as storage capacity, types, and the number of GPUs and CPUs.

The `Configuration` model is categorized into `TrainingConfiguration` and `InferenceConfiguration`. They have various sets of hyperparameters associated with different `HyperparameterValue`. For example, the former model specifies the hyperparameters related to optimization, e.g., type of optimizer, and learning rate, while these are not necessary for inference.

2.3.4. PREDICTION PACKAGE

The `Prediction` package contains the concepts that denote the model prediction/output, associated with the semantic concepts of the outputs (e.g., wheel to a car). The model elements of `Prediction` are presented with blue background.

The `Prediction` class allows the description of different types of tasks, i.e., regression and classification. Each `Prediction` is defined by *ID* and *Type* of the prediction. The prediction of a regression model is associated with a value. While the prediction of classification model can be further divided into multiple subclasses, e.g., binary classification, multiclass classification, and object detection. Classification model prediction is associated with a different number of classes. Specifically, `ObjectDetection` is associated with additional output, *BoundingBox*, which is defined by an array indicating the coordinate of the detected object in a picture. The classification prediction can be expanded and implemented for different kinds of models, e.g., image segmentation, with an additional attribute of an array.

Optionally a `Class` can be linked with a *Semantic Concept* from a knowledge base, thus allowing complex reasoning. Consider the example that an ML practitioner is building an ML model for image classification of cars, and she tries to conduct a model diagnosis. She may have several questions: what makes the model identify a car as a car? What are the semantic concepts that the model is capable of identifying? Are the wheels that make it believe that it is a car? To support such use-cases, the *metamodel* allows storing information about inference performance on specific data instances, which can be used to describe the behavior of a model, i.e., in what circumstances a model can perform well and why. Such information and awareness of the model prediction may significantly improve ML model interpretability in various applications such as health care, law enforcement, and finance.

Figure 2.4 presents an extract of the `ML Model` and `Prediction` Model as an example. *YOLOv3* and *YOLOv4* are two example models. Both models tackle object detection tasks, and the algorithm is *DNN*. Their output follows *ObjectDetection* types of prediction, with different classes (e.g., car and person in this case) and associated bounding boxes that locate the detected object. The object classes are related to different semantic concepts, for example, the wheel as part of the car, and the face as part of a person.

2.3.5. EVALUATION PACKAGE

The performance evaluation of an ML model is critical during both the training and deployment phases. The practitioners need to deploy a suitable ML model for the task given a specified environment, e.g., `Hardware` parameters. A mismatch of the deployment will lead to latency issues or reliability concerns, which results in user dissatisfaction. Hence, we present the `Metric` model to denote the model performance associated with the `Configuration` model. Thus the model performance is related to the model, hyperparameter settings, hardware, and applied dataset. Specifically, we associate `Metric` model with `TrainingConfiguration` and `InferenceConfiguration`, since a model may train on a dataset while inference on another dataset. A `Configuration` is related to zero or more `Metric`, each associated with a unique `MetricValue`.

The `Metric` is categorized into multiple types of metrics, e.g., `ExecutionMetric` such as `MemoryFootprint` and `Executiontime`; and *Prediction Metric* denoting the cor-

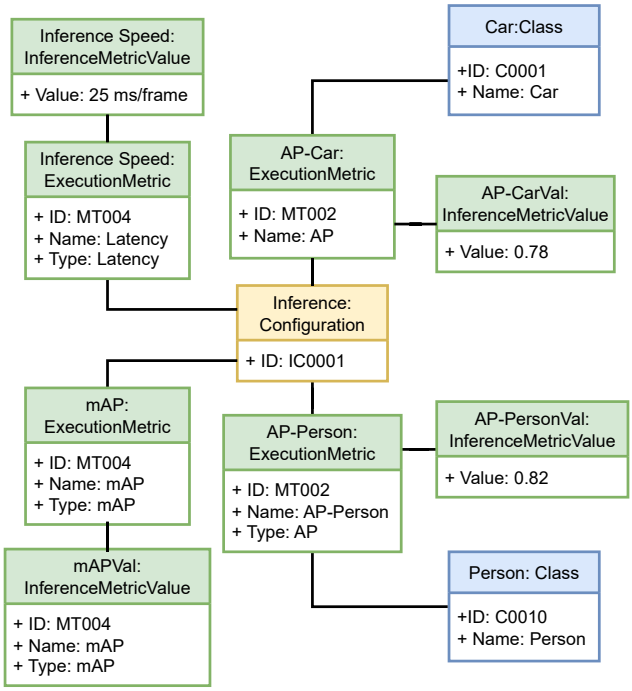


Figure 2.7: *Evaluation* model extract for a running example

rectness of the performance.

`Evaluation Metric` contains metric-related metadata. Different models will need different evaluation metrics, for instance, a regression model would have a *Mean Square Error - MSE*, while a multi-class classification model would have an *accuracy* value for each class.

Figure 2.7 presents the example of `Evaluation` package. The model inference configuration with ID *IC0001* is illustrated in Figure 2.6, associated with execution hardware, ML model, and dataset. To present the performance of a model under specified configuration settings, we include various types of metrics. In general, two kinds of metrics are introduced, i) correctness performance, verifying the prediction performance of the model, ii) and execution performance, including the runtime and memory footprint of the model. In the example, the latency and memory of the model are recorded, with a latency of 25ms and 90MB for the model size. In terms of correctness performance, we apply the common metrics to evaluate the object detection model. The metrics included in the example are average precision (AP) and mean average precision (mAP). In particular, we provide the AP metric for each class, allowing practitioners to assess the model's performance at a granular class level. Compared to the existing benchmarks or platforms that only report aggregate results, the performance metric that we include is more fine-grained.

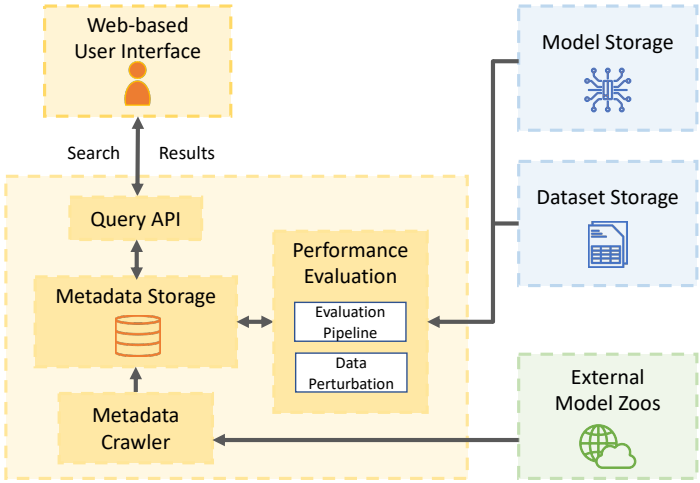


Figure 2.8: The system architecture of Macaroni

2.4. MACARONI: A REFERENCE IMPLEMENTATION OF AN ADVANCED MODEL ZOO

This section describes the architecture and functionality of *Macaroni* [111], a tool designed and implemented to demonstrate how our metamodel can be used in ML model zoos. The implementation[4] is based on the metadata model described in the previous section. The architecture of Macaroni is shown in Figure 2.8. The tool is currently designed to retrieve metadata from different sources, integrate and enrich them, and allow for complex retrieval queries. The available APIs also allow for model execution and model composition.

2.4.1. SYSTEM DESIGN

The system aims to query and enrich the metadata for model zoos. We present the system architecture in Figure 2.8. The system includes a web-based interface as front-end and back-end with storage and computation. We collect metadata in two ways: crawling ML-related metadata from external model zoos; and enriching metadata, e.g., obtaining model performance by evaluating the model on raw or perturbed dataset. The metadata obtained is later stored in metadata storage supported by a metadata model. Macaroni allows users to i) interactively search/discover models, ii) compare multiple models on various objectives (e.g., accuracy, runtime, size), iii) and, specially, measure the robustness of models on perturbed dataset. These functionalities are novel compared to public model zoos and other metadata management tools, which can be adopted seamlessly to support research studies such as explainable AI and AutoML.

The system includes a web-based interface as a front-end to present the metadata and a back-end with storage and computation. The system comprises three main components: i) user interface; ii) acquiring and storing metadata; iii) storing and serving

[4] https://sites.google.com/view/macaroni-model-zoo/home

models and datasets. The second component is the core of the system. The system is built not only to present and query metadata but also to support ML model serving.

User interface. Users can explore and query the metadata of the model zoo. We provide query API for users to filter and retrieve information. The ingestion API allows users to ingest information regarding different artifacts. The metadata being retrieved is presented in an interactive visualization.

Metadata crawler. The tool can automatically extract information from external model zoos, including HuggingFace and FiftyOne[5]. Metadata can be extracted from their APIs or information on the web pages and is recorded along with the source in the metadata storage. In particular, the model name, hyperparameters and task are obtained from the model cards from both model zoos through their APIs, and stored alongside the origin of the model. Information not available through their API (e.g., datasets and other tags) is parsed from the model's original readme files shown as the model cards. Other metadata could also be retrieved depending on the content provided by the external model zoos. Future work can investigate on extracting knowledge as metadata from the textual descriptions. Since more models will be added/updated , the crawling and extraction of the metadata shall be updated from time to time.

Model Evaluation Pipeline. The execution of models from different model zoos can be varied, differing by framework, algorithm, tasks, training dataset(s), and input format. To obtain the model performance and compare the evaluation results, we provide a unified evaluation pipeline, which facilitates evaluating models from different external model zoos on various datasets conveniently. Our pipeline is extensible, i.e., add support for new types of models or data after the initial pipeline deployment. We achieve extensibility in two ways. i) We apply a modular design, in which each evaluation module defines how to evaluate for a subset of models. ii) From each evaluation module, one or more evaluations are conducted, based on configuration of the module, such as which metrics to calculate or datasets to use.

2.4.2. METADATA RETRIEVAL AND STORAGE

Recent works developed tools/systems to record metadata for the purpose of monitoring the experiments, especially during model training. Works such as Cerebro [104], MLflow [196], and ModelDB [180] provide APIs/logging libraries for users to track and log interested metadata in different levels of details. While others, such as ModelKB [61], automate the extraction process by identifying metadata from the source code of different deep learning frameworks.

In this work, we obtain the metadata in multiple ways and process and store it in the back-end. We collect metadata in the following three ways. i) A user can add the metadata of a model or a dataset by filling in specified fields (with Ingestion API), e.g., the content presented in a model card. Then such information is processed by the *Ingestion API* and converted into structured representation according to the above-mentioned metadata model, and stored in the *Metadata Storage*. ii) The tool can also automatically extract information from external model zoos, e.g., HuggingFace and PyTorch Hub. We can extract metadata from their API or information from the web pages and

[5] https://docs.voxel51.com/

record the source of the external metadata in our metadata field. iii) To gather the information regarding the model performance, we apply a third way to obtain the metadata. We obtain the performance by evaluating the model on a dataset offline with specified hardware settings. This process utilizes cloud resources and is performed offline.

The metadata is stored in the structure described by the data model (Section 2.3). We implemented the data model and stored the metadata in MongoDB.

Model/Data Storage and Serving. To support performance evaluation, we store related models and data scripts/files. The models and data are further applied with an automated evaluation pipeline (in the following subsection 2.4.3). The models are perceived in docker images, and they also support model serving. Users can apply the model to their data and obtain prediction results.

2.4.3. PERFORMANCE EVALUATION AND AUTOMATED PIPELINE

The tool supports the integration of models described and hosted in external model zoos. By the time we were writing the paper, we had imported more than 171k models hosted on the HuggingFace and FiftyOne model zoos, among which 986 of them were evaluated on 14 different datasets in texts or images. We develop an automated pipeline to execute/evaluate models from various platforms adapted from external APIs.

The performance metadata is gathered in three steps: i) The model is evaluated using the API from the model zoo the model was extracted from. ii) The output from the inference is then processed and transformed to a standardized format, and finally iii) The performance of the model is evaluated based on the predictions, and the values will be stored in the proposed structure.

It is important to note that external APIs may have their own methods for performance calculation that deviate from the norm, or may not have some capability at all (performance of a class). To ensure the comparability of model performance, we implement unified evaluation methods.

2.4.4. RETRIEVAL AND EXPLORATION OF MODEL REPOSITORY

Throughout the ML lifecycle, ML practitioners will require different metadata for tracking the ML model status, editing, comparing, or reporting. An ML practitioner often needs to query models in large repositories with complex filtering conditions, e.g., data instance, performance, and inherit mechanism. In Table 2.2, we list a few example queries revealing different properties of the metadata. For example, Queries 1 and 2 require the metadata regarding the dataset, i.e., its attribute and source. This type of metadata helps practitioners understand the dataset, which allows them to gain insights into the characteristics and properties of the data and thus determine the relevant features in the feature selection and engineering stage. Queries such as Queries 3 to 6 require other metadata properties, such as the model performance and its interpretability. These properties are crucial for model discovery and comparison, and in addition, assist in decision-making processes and solving complex analytic problems. Query 7, on the other hand, requires more complex information regarding the inference performance with specified hardware settings. For example, an edge device may have constraints such as limited computation power and storage. To answer this query, practitioners will need to obtain the model performance of different objectives, e.g., inference speed and mem-

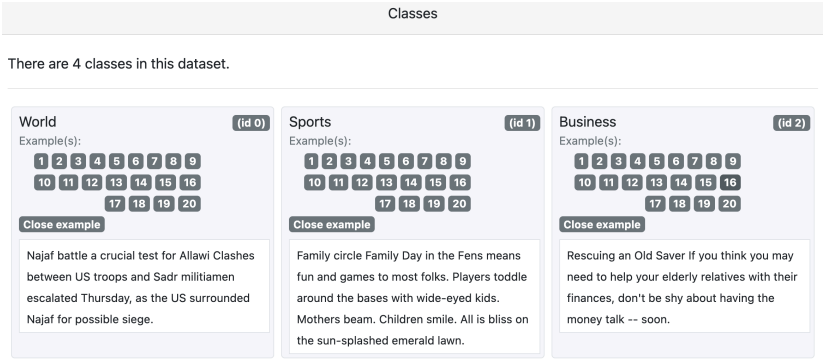


Figure 2.9: Macaroni interfaces: Dataset page with data instance examples

ory footprint.

The interface of the tool aims to let users find a model that is relevant to them, shown in Figure 2.9, Figure 2.10, Figure 2.11, and Figure 2.12. To do this, users can filter all available models by relevant properties, e.g., the name, task, or training dataset of the model. Once a model is selected, the user is presented with an overview of all available metadata, alongside the average evaluation results of the model (such as inference time and accuracy, if available) and a brief description of the associated dataset (if available).

If evaluation results are available, the user can opt to view more detailed information. We present the model performance with the following visualization types.

- *Table.* The tables present the (aggregated/disaggregated) performance results of a model with numbers, such that users can identify the best score of model performance on each task or on average.

- *Bar chart.* Bar charts and tables are interchangeable when presenting the performance of a model. For a clearer presentation, we only apply bar charts when comparing models. Users can also select the interested evaluation metrics and tasks for presentation.

- *Confusion matrix.* A confusion matrix is useful for model explanation, as users can observe when the model performs poorly. We also record the predictions of the model on data instances. Users can further explore the performance of the model by investigating the wrong predictions given the data examples.

- *Scatter plot.* As earlier stated, accuracy shall not be the only evaluation criterion of model performance, especially given complex requirements in the production environment. Specially, we also support comparing models on multiple objectives, e.g., accuracy against inference speed. For instance, one may observe that no single model can dominate in all objectives, e.g., having the highest accuracy score while being the most efficient to run.

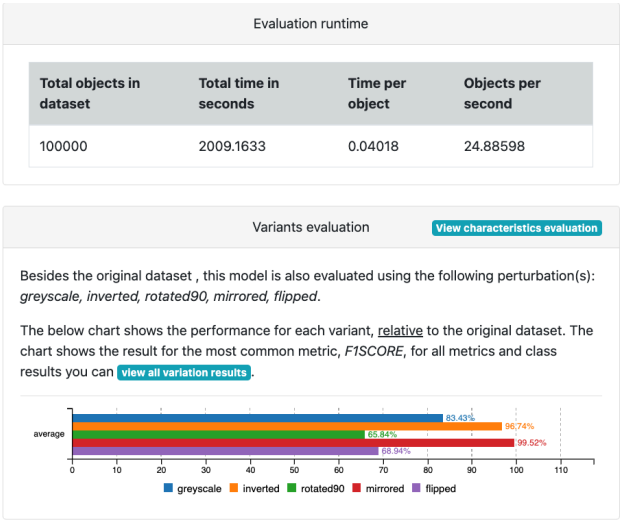


Figure 2.10: Macaroni interfaces: Model page with aggregated performance

Figure 2.11: Macaroni interfaces: Model page with fine-grained performance

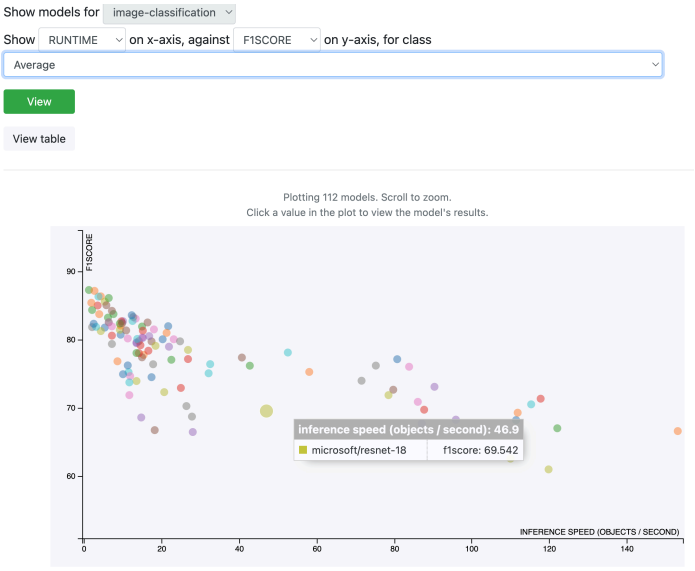


Figure 2.12: Macaroni interfaces: Model comparison page

2.4.5. USAGE AND FUNCTIONALITY

The querying of the metadata is performed in an online manner, while obtaining and updating the metadata can be processed offline. For instance, the metadata can be extended by evaluating the model and crawling the external model zoos regularly, e.g., once a week. Users can also trigger to evaluate the model and push the results to the metadata storage.

Model discovery. The tool supports different evaluation metrics, accuracy, inference speed, memory footprint, etc. Specifically, the accuracy of a model is not only presented in aggregated results of a task but also at a class level. Besides ingesting, extracting, and storing the metadata, our proposed tool allows a user to (i) retrieve the models that help them identify a model, (ii) compare multiple models, (iii) or explore the properties of models/data by composing queries on the metadata.

Data Perturbation to Measure Model Capabilities. In addition to model performance on dedicated dataset, we also allow practitioners to investigate the model robustness on various types of data changes. Identifying the model performance on different data shifts is fundamental in understanding the model capabilities. We define a few types of perturbations on input dataset, e.g., converting to greyscale, flipping or mirroring the images, and observe the difference in performance. In such a way, we manage to identify how the model is generalized to data with different properties/changes. Future work can incorporate different perturbation methods, e.g., adversarial attacks, adding noise, on various modalities. We view the establishment of such a way to observe the model capabilities as the starting step, and further techniques and methods from explainable AI can enrich the description for model capabilities.

2.5. APPLICATION 1: AMALUR, AN DI-AWARE ML SYSTEM

In this section, we first explain the challenges of two common ML scenarios, i.e., feature augmentation and federated learning, and propose a novel system *Amalur* to tackle these challenges. Amalur is a unified ML system designed to enhance ML pipelines' efficiency, effectiveness, and privacy in data integration contexts. It facilitates model training and inference over data silos by leveraging DI metadata.

2.5.1. AMALUR OVERVIEW

We are currently developing *Amalur*, a *machine learning* system that is based on our work on data lakes [67] and model zoos [111]. With DI metadata, Amalur solves the challenges of efficient training and inference of ML models over data silos and reducing the manual work of integrating the data. Figure 2.13 provides a high-level overview of Amalur with key components relevant to this paper.

User inputs. Amalur empowers users, including domain experts like physicians or data scientists, to run predictive or ML model training tasks on data silos. Through the metadata provided by the catalog, users can choose the desired features and relevant data silos. They can also initiate model training using either custom models or Amalur's built-in ML models with metadata from the catalog. Furthermore, specific constraints, such as data privacy regulations like the GDPR [182], can be specified by individual users or particular silos.

Hybrid metadata catalog. One of the fundamental components of Amalur is the metadata catalog. It stores the metadata of data, ML models, and hardware settings. Data-related metadata includes the basic metadata and DI metadata. Collected from the silos, the basic metadata describes each data source, e.g., source table schema, data types, and silo location. The DI metadata includes column relationships from schema matching and row matching from entity resolution. Model-related metadata includes information describing models and the evaluation performance (e.g., model accuracy). We have addressed the representations of basic metadata of source tables [148] and ML models [111]. Amalur utilizes a mixture of heterogeneous metadata, including DI metadata, in Sec. 2.5.3.

Estimator and planner. The cost estimator and task planner play a critical role in the system. The cost estimator leverages the metadata from the catalog (e.g., data characteristics, hardware specifics) and constraints to determine the approach to execute the input task over silos: materialization and factorization. The initial cost estimator utilizing basic hardware information and the computational complexity of the target model. The planner identifies the appropriate physical operators and generates an execution plan for task orchestration.

Task orchestrator and dispatcher. The execution plan from the planner is translated into specific programs tailored to the training approach, i.e., factorization, materialization, or FL, and the execution environment, such as TensorFlow, PyTorch, Spark, or ONNX. For materialization, *Dataloader* pulls data from the silos for processing. Model training or inference will be performed in the centralized server. Alternatively, if factorization is preferred, programs are sent to remote silos as the metadata dictates, i.e., silo location, ensuring they reach the appropriate data location. The main computations are

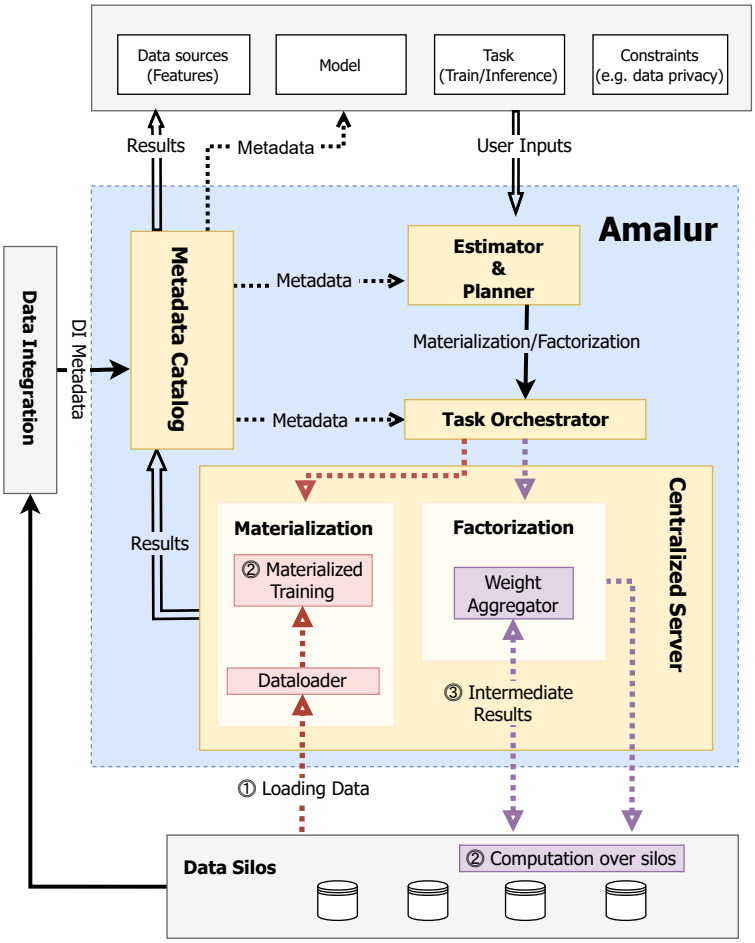


Figure 2.13: An overview of Amalur.

performed over each silo.

Aggregator. For factorized learning and federated learning, a crucial component is the aggregator. Some computations are pushed to the silos while a central server aggregates the results. The computations are performed locally, and the parameters are learned globally. The role of the aggregator is to collect the result of local computations and then distribute the loss to the silos and aggregate the gradient of the parameters.

2.5.2. AMALUR WORKFLOWS FOR ML TRAINING AND INFERENCE

With the core components in Amalur being introduced, we will explain the main workflow among the components in Figure 2.14. Given the user inputs in Amalur, different workflows are performed: either performing inference or training, either factorizing or materializing the data, etc.

Amalur allows users to determine the data sources and models. If there is available DI

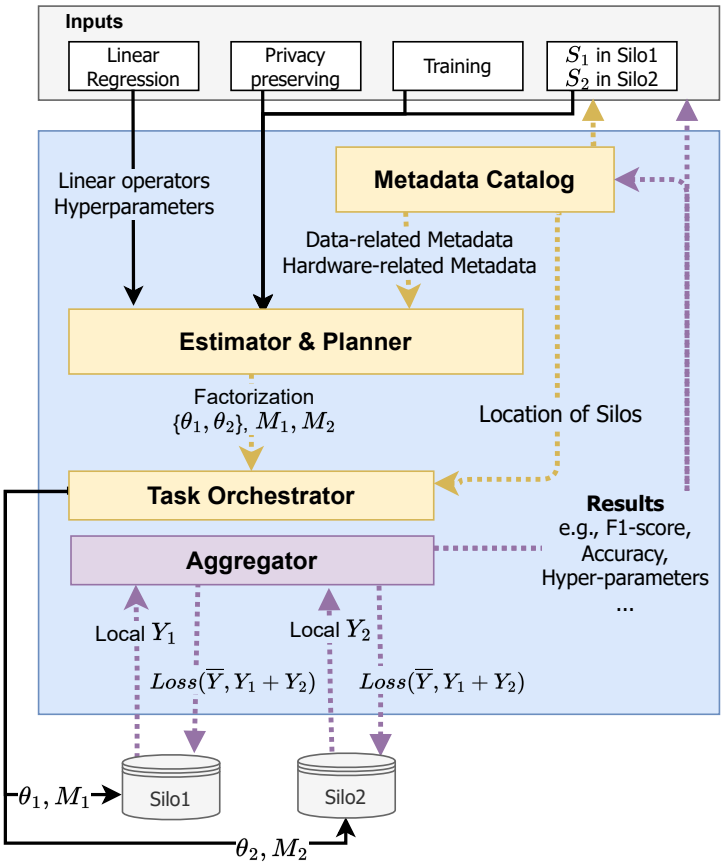


Figure 2.14: Example workflows of factorized learning in Amalur

metadata, Amalur will provide data sources that can be connected. To increase the effectiveness of ML training, a user can select feature columns from the available schemata. Model-related metadata is also retrieved from the metadata catalog and provided to users, which enables them to decide what algorithm and hyper-parameters to use. Users may use their customized model and hyper-parameter sets. In the input phase, the user selects data sources (e.g., name of the table, name of the schema), chooses the model, and determines the task (classification or regression) and constraints (e.g., privacy).

With the inputs, model training or inference will be performed. In the end, all results, which include predictive outcomes and trained models, are gathered in a centralized cluster. Concurrently, the system logs the training or inference method (materialization/factorization), the hyper-parameters, and performance (e.g., F1-score, runtime) in the metadata catalog, making them accessible for future reference and used by other users. Below, we will introduce the training and inference in more detail.

Model training. After Amalur receives the inputs from a user, the cost estimator will determine the computation strategy for training, i.e., to materialize or factorize, with

metadata from the inputs and metadata catalog. For materialization, Amalur will integrate the source datasets and generate the target table in the centralized server, and training will be performed on the server. For factorization, the model is decomposed and pushed down to silos. When privacy constraints are present, Amalur executes privacy-aware model training processes over the silos [158], i.e., federated learning.

Figure 2.14 depicts a workflow for ML model training in a factorized manner. The planner will split the model into the parameters θ_1, θ_2 along with the DI metadata M_1, M_2, which are pushed to Silo1 and Silo2 for computations respectively. Subsequently, the central server will collect the computations and aggregate the result, i.e., Y_1 from Silo1, computes the loss calculated from $Loss(\bar{Y}, Y_1 + Y_2)$ which is sent back to the silos for gradient updates. Once the loss meets a predefined criterion, a central orchestrator records performance metrics in the metadata catalog. In addition to illustrated workflow, due to privacy considerations in FL, the partial parameters are stored locally within the silos.

Model inference. A user can select a specific model and perform model inference if models are available for the prepared dataset. Like model training, the cost estimator determines whether the computation is performed in a factorized or materialized manner. Model inference in a materialized manner is similar to model training. Inference in a factorized manner is slightly different, with only the local predictive results being sent to the centralized aggregator to generate the predictive results, while nothing is returned from the server.

2.5.3. METADATA IN AMALUR

Metadata is crucial for DI systems [19, 51, 96]. At the core of this vision, we reveal the importance of metadata, particularly DI metadata, for ML training and inference. In the following, we divide the relevant metadata into three categories, i.e., data-related, ML-related, and hardware-related metadata. As shown in Table 2.4, in each category, we showcase the representative types of metadata and their roles in improving the effectiveness, efficiency, and privacy of ML model training and inference. The metadata is stored and managed by the metadata catalog described in Sec. 2.5.1.

Data-related metadata Metadata in databases and data lakes refers to the information that describes the structure, and characteristics of databases or data lakes and their objects [72, 164, 166]. The metadata includes, e.g., information regarding schemata, statistical and descriptive data about relations and attributes, integrity constraints, and silo location.

Data integration metadata. By DI metadata, we refer to the information that describes the relevance and overlap between data sources, e.g., schema-level correspondences between source tables and the target table (schema mapping), and row matching between source tables (entity resolution).[6] To address the research question in Sec. 4.1 , we employ DI metadata in a two-fold manner.

1. Efficiency. By representing schema mapping and row matching as matrices, Amalur facilitates a unified execution of data transformation operations and linear algebra op-

[6] How to obtain DI metadata is not the focus of this work, as schema matching and mapping, and entity resolution are intensively studied topics with open-source tools [97, 98] and commercial products. We assume that the DI metadata is part of Amalur's input. We are interested in how to utilize DI metadata for machine learning.

Table 2.4: Example metadata types used in Amalur

Category	Function	Metadata Type
Data-related Metadata	Dataset	Source location
		Schema (attribute, data type)
		Cardinality
	Data Integration	Schema mapping
		Row matching
Model-related Metadata	Model	Name
		Algorithm
		Hyper-parameters
	Trained Results	Training method (factorized/materialized)
		Parameters
		Performance (e.g., accuracy)
Hardware-related Metadata	Hardware and Environment	#GPU cores
		#CPU cores, cache sizes
		Memory bandwidth
		Memory latency

erations, which improves the efficiency of training tasks.
2. Effectiveness. DI metadata brings new opportunities for improving the effectiveness of federated learning frameworks, e.g., through discovering the redundancy among source datasets.

Challenges. Another type of DI metadata is the similarity among source datasets. In recent studies on data lakes, it is a crucial step to first capture the similarity between source datasets, i.e., joinable or unionable dataset discovery [20, 70, 92], before data integration. The similarity between datasets is also valuable for improving the effectiveness of ML training, e.g., resolving the inconsistency across datasets and recommending models to train if the model was trained on a similar dataset [193]. In recent data integration works [20, 30, 80], the embeddings are applied to capture the similarities between features or tuples in source tables. Taking one step further, representations of the entire table [109, 127] allow for many more applications, e.g., transfer learning and multimodal machine learning. Many research questions remain open, regarding more types of DI metadata, their representations, and their roles in improving ML tasks.

ML-related metadata. Amalur not only supports efficient ML training but also manages the trained models, which makes it necessary to design a metadata catalog that includes heterogeneous metadata for ML, i.e., ML-related metadata. ML-related metadata captures the metadata from various ML lifecycle stages [111], such as model definition and model training. The metadata describing the model includes architecture, frame-

work, configurations (e.g., hyper-parameters), input/output (e.g., prediction classes), etc. These types of metadata are utilized in different components in Amalur. For example, the cost estimator requires information regarding the complexity of the model (e.g., algorithm, parameters) to predict workload and model training requires hyper-parameters. The metadata catalog also keeps track of the connections between the model and its training datasets. Thus, given a generated dataset, if a model was previously trained on this dataset, the model will be recommended to the user for inference or re-training.

Besides the information describing a model, we also record the performance of ML models (e.g., model accuracy, runtime, memory footprint) under different execution environments. This information helps recommend models to users in the model preparation stage. Recommending a good model to train based on different strategies [153] can improve the effectiveness and efficiency of ML training.

Metadata regarding hardware. Hardware and environment settings are important to measure the performance of databases [177], the interested information including, e.g., number of CPU cores, memory. This information is also essential for measuring ML performance during training and inference [43]. For ML, the hardware devices may also include GPU or TPU. In general, the hardware-related metadata is regarding the execution environment in the central cluster and distributed silos.

In this paper, the hardware-related metadata also supports an essential component for improving the efficiency of ML training. This type of metadata, along with the data-related and ML-related metadata, supports the cost estimator to decide whether to train a model on materialized or factorized data. The result of the cost estimation results in a more efficient plan for model training or inference.

2.6. APPLICATION 2: MODEL COMPOSITION UNDER CONSTRAINTS

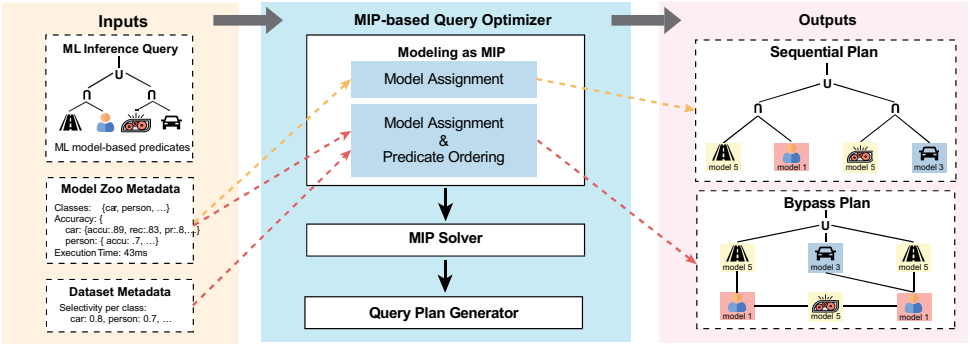


Figure 2.15: ML inference query optimization

Now we introduce a more advanced yet common use case. With metadata being captured and well-represented, ML practitioners can make good use of the models trained offline and apply them to answer complex, ad-hoc inference queries. We will further illustrate this application in more details in Chapter 3.

Recent research has focused on ML applications for different modalities. For example, systems have been built to serve ML models for specific tasks [86]. Others aim to accelerate ML inference on domain tasks, such as NoScope [86], and PP [120]. These applications have shown a trend of applying model composition (usually with multiple models in cascades or in sequence) for complex tasks. The key idea of these works is to filter out insignificant data as early as possible, which is extremely useful when processing large-scale data, e.g., video, or streaming data, e.g., tweets.

The trend of applying and optimizing the usage of ML models in complex tasks has provided insights on how to manage ML models to better serve the tasks. A solution is to identify the capability of the models as fine-grained as possible. One of our previous works [113] has proposed to optimize for ML inference query by utilizing the metadata of ML models, especially the performance of models in multiple objectives.

As shown in Figure 2.15, the ad-hoc query can consist of multiple ML inference tasks with different dependencies and relations. Moreover, the query can be composed of specified constraints/requirements (e.g., latency and accuracy restrictions). Practitioners can select a composition of models from the model zoo to answer the ad-hoc queries. Optimization can be applied to further increase the efficiency of answering the query by carefully scheduling the tasks in a different order and applying early filtering, as the Bypass Plan in Figure 2.15. With the same example in Figure 2.15, an ML practitioner would like to design an application for video analysis that can capture a pedestrian crossing the road or the rear light of the car in front getting red. Since the data volume is significant and latency is also an essential factor to consider, the practitioner should select image classification or object detection models with fast inference speed. And the inference speed is greatly affected by the hardware being applied. If the application is deployed on a mobile phone, then the memory footprint is also a fundamental objective to be concerned with. To identify which set of models could best address the query and constraints, they may require information regarding the model performance with different objectives (e.g., accuracy, inference speed, memory footprint). The metadata of the dataset can also provide information to detect concept drift, for instance.

2.7. RELATED WORK

Recent studies focus on different aspects of management during the ML lifecycle, from model versioning, and model reporting to model evaluation. Each is important for practitioners to manage and understand the models. We observe a gap among the profound works [161], a comprehensive and queryable metadata representation. With the metadata representation, we can thus better manage the ML models and data, including the interactions between them.

2.7.1. ML MODEL MANAGEMENT SYSTEMS AND TOOLS

Due to the complexity of the ML models and ML lifecycle, managing the ML models in different phases are challenging. And yet, multiple systems have been developed to tackle the challenge of managing the ML models during training in experiments.

Modeldb [180] is one of the first systems that allow tracking, storing, and exploring ML models. Modeldb keeps track of the ML pipelines defined by the users and allows

them to visualize and explore the models and pipelines. Other systems, such as ModelHub [125], ModelKB [61], and Runway [178] also allow managing ML experiments and their associated models. These systems allow model storage, versioning, and querying, with metadata being extracted from scripts or manually logged. None of them, unfortunately, made it explicit what metadata should be included/tracked during the experiments or when serving the models. Enterprises platforms, such as MLFlow [35], Amazon SageMaker, Google TFX [132], comet [44], Airbnb Bighead [25] and etc., provide rich APIs and tools to support ML experiment management. Users can customize what metadata to log by calling APIs. The metadata can be visualized or used to specify new experiments. In this work, we do not cover the aspect of managing ML model training experiments. Related works, e.g., [35, 132, 184], can be served as support and complement to our scope. We report models with rich and comprehensive metadata covering different artifacts and their relationships. We strive to support practitioners with the necessary information to know about a model and its necessary components.

2.7.2. ML INFERENCE/SERVING SYSTEMS

Instead of managing the end-to-end process of the ML lifecycle, multiple ML systems aim at a particular phase in the lifecycle, e.g., ML inference or ML serving.

Accelerating ML inference. Systems, such as Clipper [45], Willump [99], and GATI [14], optimize and accelerate ML inference when serving. The goal of these systems is to serve and infer ML models for downstream tasks. Clipper is a general-purpose low-latency prediction serving system that sits between end-user applications and a wide range of machine learning frameworks. It introduces a modular architecture to simplify model deployment across frameworks and applications. Clipper reduces prediction latency and improves prediction throughput, accuracy, and robustness without modifying the underlying ML frameworks.

ML benchmark in specific domains. Researchers have been building ML benchmarks for different domains, for example, PMLB [141], PennAI [140](biomedical and health), Moleculenet [188] (molecule), facies classification [7], DLHub [33] (science), Kipoi [12] (genomics).

2.7.3. AI-CENTRIC DATA MANAGEMENT SYSTEMS

Systems have been developed and built to manage data for AI. DescribeML [63] and Amalur [68] propose dataset models to describe ML datasets in detail and preserve relevant metadata. The preservation of dataset information greatly facilitates, for example, the search for suitable datasets for ML projects. For ML dataset management and versioning, research work such as Mldp [5], Chimera [56] , and DataLab [202] are complements to the above-mentioned model management systems that served as support for managing data versions.

Another type of data platform is to move the DBMS engine from a relational to a tensor abstraction, which unseemly integrates databases with external ML tools. TDP [58] provides access to multi-model data and leverages PyTorch to run queries over data on a wide range of hardware devices. TDP integrates the flexibility of PyTorch's programming model with the declarative power of SQL.

2.7.4. MODEL CARDS AND DATA SHEETS

Recent research also focuses on the reporting of models and datasets, covering aspects not only limited to basic informative components but also including ethical, inclusive, and fair considerations. Model cards [131], for example, proposed to include information regarding model intended use cases, potential pitfalls, and other contexts that can improve model understanding. A similar idea also lies in data cards/sheets. Examples include [24, 59, 126]. Though model cards and data cards contain rich information, the Q&A format is nonetheless unfriendly to machines to process and thus cannot be easily managed and retrieved.

2.7.5. MODEL PERFORMANCE BENCHMARKING

A growing body of published work also focuses on the benchmarking of ML model performance, such as MLperf [123, 151], fathom [4], and DAWNBench [43]. These platforms covered a set of metadata, including metrics, and training and inference configurations with specified hardware/software settings. Their focus is the report of the model performance at different ML lifecycle stages (training or inference). They paid little attention to the dataset the model used, whose path is provided as an argument filled by the user. The model process pipeline is also not covered besides the model scripts.

2.8. CONCLUSION AND OUTLOOK

In this chapter, we advocate for the need for a structured, queryable, and comprehensive metadata representation for model zoos. We propose a metadata model for such metadata representation to tackle different use cases. We also develop a tool that helps practitioners to manage and query the metadata.

We urge practitioners to prioritize the collection and organization of metadata, utilizing it for future applications. In order to enhance the applications of metadata, we present several aspects that can be explored in future research endeavors.

Integration of ML model metadata to current platforms. Existing work has developed tools to record (log/extract automatically) metadata during the ML lifecycle. Recent works only identify the public pre-trained model. Future work can integrate the systems seamlessly such that practitioners can have access to self-trained models as well as public pre-trained models.

NLP-based Extraction of metadata from text. With the support of large language models, future research can develop tools to extract useful information from the textual descriptions in the model and data cards by applying natural language processing techniques and mapping it to the predefined metadata representation.

Personalized AI and Finetuning. Many applications, such as behavior detection and virtual assistants, are user-specific and take into account user beh aviors and preferences. Companies adapt the models to their own datasets and context by re-training and finetuning the models. Instead of randomly searching the hyper-parameter values to train a model from scratch, practitioners can utilize the metadata to accelerate the learning process by setting a warm-start for hyper-parameter search [160]. These AI applications can utilize the capabilities of the pre-trained models that already have a good performance on a certain task. We support these use cases by providing rich and comprehen-

sive metadata, either the configuration settings or performance evaluation.

Provenance, Lineage, and Versioning. Researchers also propose to utilize metadata for other purposes, such as workflow data provenance [124, 203] and ML pipeline lineage [64, 159, 179]. ModelHub [125] and Modeldb [180], on the other hand, track metadata about models throughout the lifecycle and provide version management. However, they focus on the abstractions of the model, while they lack information on the model performance under different hardware settings (e.g., inference speed, accuracy, memory footprint), and they pay little attention to the datasets that the models consume.

3

Optimizing ML Inference Queries with metadata

The previous chapter introduced a metamodel to encapsulate metadata associated with diverse artifacts. With this foundation, this chapter delves into the practical utilization of metadata for model selection in ML inference.

Nowadays, 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 an 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 through the strategic use of metadata, encompassing elements like model performance, accuracy, and operator selectivity The optimizer selects the most suitable ML models to use and 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 problem, and develop an optimizer that maximizes accuracy given constraints. This optimizer can navigate a large search space to identify optimal query plans on various model zoos.

This chapter is based on the following publications:

- Li, Z., Schönfeld, M., Sun, W., Fragkoulis, M., Hai, R., Bozzon, A., & Katsifodimos, A. (2023, June). Optimizing ML Inference Queries Under Constraints. *In International Conference on Web Engineering* (pp. 51-66). Cham: Springer Nature Switzerland.

- Li, Z., Schönfeld, M., Hai, R., Bozzon, A., & Katsifodimos, A. (2023, April). Optimizing machine learning inference queries for multiple objectives. In 2023 *IEEE 39th International Conference on Data Engineering Workshops (ICDEW)* (pp. 74-78). *IEEE*.

3.1. INTRODUCTION

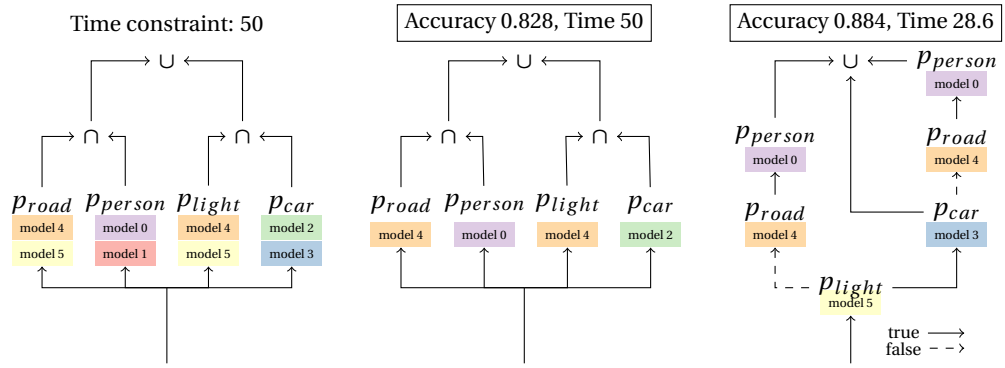
Machine learning (ML) is increasingly used to process unstructured documents (i.e. text, images, videos), or data streams [39, 84, 163, 199]. Take, for instance, the scenario of a self-driving car: when it detects (at certain proximity) that a person is crossing the road, or that another car has turned 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 [32, 89]: (`road` ∧ `person`) ∨ (`car` ∧ `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 [78]. This shift, mostly motivated by computational, economic, and environmental considerations, is evident from the proliferation of public, pre-trained ML model zoos on the Web, such as HuggingFace and PyTorch Hub. 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 3.1 – 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)[87, 120]. 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 Figure 4.5) to select the best ML models as well as their execution order, given a complex ML inference query and execution constraints. The contributions of this chapter can be summarized as follows:

- We formulate the problem of ML inference query optimization as a mixed integer progarm (MIP) and propose a MIP-based optimizer that exploits model zoos.
- 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 (Section 3.8), showing that our proposed optimizer can generate plans that significantly outperform the baselines in diverse model zoos on different constraint settings.



(a) Unoptimized (b) Model-optimal (c) Model&Order optimal

Figure 3.1: Alternative ML query plans for the running example query.

- We extend the model assignment problem by considering three objectives, i.e., accuracy, speed and memory consumption. We formulate the problem as multi-objective mixed integer program (MOMIP). We contribute an analysis of applying different multiple-objective optimization methods.

3.2. RELATED WORK

ML Inference Query Optimization. The development of specialized models for fast inference of object detection queries has received considerable attention [77, 78, 117, 152]. More recently, related research is targeting the processing efficiency of larger ML inference query [9, 28, 29, 86, 120]. NoScope [86] and PP [120] filtered irrelevant frames by training and deploying special lightweight binary classifiers, and Tahoma [9] 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 [120]. 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 [113, 143, 176, 177], 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.

Table 3.1: 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 3.2: 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

3.3. 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.

3.3.1. METADATA OF A MODEL ZOO

We formalize the metadata representation of a model zoo [112] as $\mathscr{R}\{M, I, P, A, C\}$, 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 3.1 while A is depicted in Table 3.2. In the following, we explain how we utilize the metadata of a model zoo as prior information in ML inference query optimization in Section 3.4.

3.3.2. ML INFERENCE QUERIES

Given a model zoo $\mathscr{R}\{M, I, P, A, C\}$, 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 $\mathscr{R}$. 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).

3.3.3. 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 (Figure 3.1(a)). Figure 3.1(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 3.1(c) depicts a plan with optimized model assignment and execution order as a bypass plan [91], where we refer to this type of query plan as *bypass* plan.

3.3.4. 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 [143]. *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 3.3.1 (Accuracy-maximizing Model Assignment (AMA) Problem)**.** Given a model zoo $\mathscr{R}$, 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:

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

In the above definition, we denote the function to compute a_Q as $f_{accu}(Q)$, which will be elaborated in Section 3.5.2. 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.

Use case. The problem in Definition 3.3.1 specifies the bounding of the execution time. It is a typical requirement in use cases where execution speed is of importance.

Definition 3.3.2 (Execution time-minimizing Model Assignment (EMA) Problem)**.** Given a model zoo $\mathscr{R}(M, I, P, A, C)$, 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 3.5.3.

3.4. 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 Section 3.4.1. We then present a mixed-integer programming formulation (Section 3.5.2-3.6.3) for the ML inference query optimization problem as defined previously, including accuracy model, execution time model, objective function, and other relevant components.

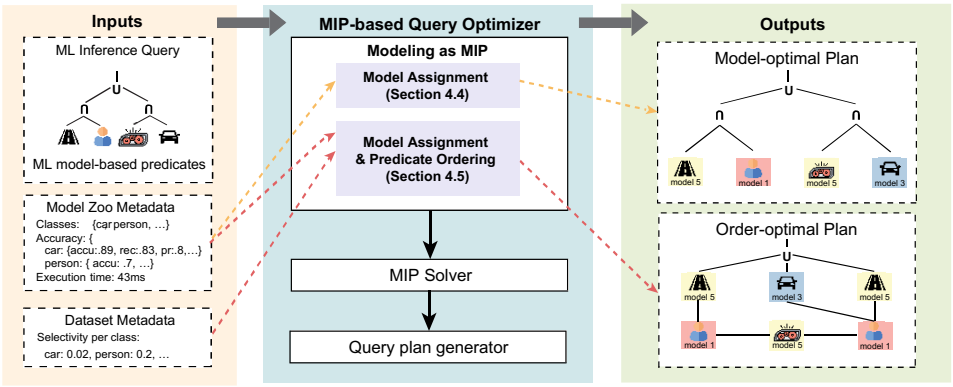


Figure 3.2: Approach overview.

3.4.1. APPROACH OVERVIEW

As depicted in Figure 3.2, 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 regarding the portion of data that a predicate returns as `true`. Both types of metadata are retrieved from a metadata management tool (e.g. [112]). 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 Section 3.3.4, 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* [90] 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.

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 $plan$

$plan$ = NULL predicate $p \leftarrow$ the first predicate in the $order$ **if** p *is not empty* **then**
  current node $m = mapping[p]$
  $suborder \leftarrow$ the remaining order after removing p
  `// Positive branch`
  $subquery \leftarrow$ subquery of $query$ where p is substituted with `true`
  pos_branch = BypassPlanGen($subquery$, $mapping$, $suborder$, `true`)
  `// Negative branch`
  $subquery \leftarrow$ subquery of $query$ where p is substituted with `false`
  neg_branch = BypassPlanGen($subquery$, $mapping$, $suborder$, `false`)
  `// Generate the plan as a binary tree ([root node, left child, right child])`
  $plan = [m, pos\_branch, neg\_branch]$
end
return $plan$

In this work, we jointly optimize model assignment and predicate ordering given time constraints, and have shown significant performance for the objective goal (see Section 3.8 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 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.

3.5. MODEL ASSIGNMENT AS MIXED INTEGER PROGRAMMING

In this section, we present a mixed-integer programming formulation for the ML inference query optimization problems as defined in Definition 3.3.1 and Definition 3.3.2.

3.5.1. MODEL ASSIGNMENT CONSTRAINTS

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 con-

straints: *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 3.1(b) presents the plan that only takes into account of model assignment that maximizes the accuracy given the time constraint.

Model-assignment variables. To perform model assignment we need to allocate exactly one model to each predicate. Given a model zoo $\mathscr{R}(M, I, P, A, C)$ 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 $\mathscr{R}$ as X. The decision variable $X_{m,p}$ is a binary variable that indicates whether a model is selected:

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

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:

$$\sum_{m \in M} X_{m,p} = 1 \tag{3.1}$$

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}:

$$X_{m,p} \leqslant \lceil A_{m,p} \rceil \tag{3.2}$$

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.

3.5.2. MODELING ACCURACY

Query Accuracy Calculation . In what follows, we explain the procedure of calculating query accuracy a_Q, i.e., $f_{accu}(Q)$ in Definition 3.3.1.

For example, given the query q in Section 3.3.2, the accuracy is computed as follows:

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

In this work, we assume that the predicates are independent and we did not consider the effect of correlation between predicates. Similar assumption has been made in [120]. 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., $f_{accu}(Q)$. 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:

$$f_{accu}(Q) = \prod_{p \in P} (\sum_{m \in M} A_{m,p} X_{m,p}) \tag{3.3}$$

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

$$\begin{aligned} f_{accu}(Q) = & \sum_{p \in P} \sum_{m \in M} A_{m,p} X_{m,p} - \prod_{i \in \{p0,p1\}} \sum_{m \in M} A_{m,i} X_{m,i} \\ & + \prod_{j \in \{p0,p1,p2\}} \sum_{m \in M} A_{m,j} X_{m,j} - \ldots + (-1)^{|P|-1} \prod_{p \in P} (\sum_{m \in M} A_{m,p} X_{m,p}) \end{aligned} \tag{3.4}$$

The calculation of $f_{accu}(Q)$ for a CNF query is conducted similarly: first we calculate all the individual disjunctive subexpressions with Eq(3.4), and then calculate the final conjunction of those disjunctions with the formula of Eq(3.3). To summarize, for a CNF query or DNF query, we will parse different operators and compute the accuracy according to the query.

We now explain the procedure of estimating query accuracy a_Q, i.e., $f_{accu}(Q)$ in the problem definition. The intuition is that the query performance is dependent on the performance of models assigned to the predicates. In this paper, we follow the assumption below.

Assumption 1. The predicates are independent to each other.

The same assumption is also made in [120], i.e., the outcome of one predicate does not impact the performance of others. The accuracy of a conjunctive query, e.g., `road ∧ person`, can be estimated by multiplying the accuracy of each model, $a_{road} * a_{person}$. The accuracy of a disjunctive query, e.g., `car ∨ 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.

3.5.3. 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 $f_{time}(Q)$.

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 3.3.2. 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

Table 3.3: 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

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.

$$B_m = \begin{cases} 1 & \text{if } \sum_{p \in P} X_{m,p} \geqslant 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

B_m is constrained as follows:

$$\begin{aligned} B_m &\geq X_{m,p} \\ B_m &\leq \sum_{p \in P} X_{m,p}, \forall m \in M \end{aligned} \tag{3.6}$$

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. The execution time for the query plan is :

$$f_{time}(Q) = \sum_{m \in M} C_m B_m \tag{3.7}$$

Objective functions. To conclude, with Eq(3.4) and Eq(3.7) we have transformed the two problem variants in Section 3.3.4, to a matter of MIP by defining two objective functions as below.
Given an execution time constraint (solving problem described in Definition 3.3.1):

$$\begin{aligned} &\text{Maximize:} && f_{accu}(Q) \\ &\text{Subject to:} && Eq(3.1), Eq(3.6), \\ & && f_{time}(Q) \leqslant C_{bound} \end{aligned}$$

Given an accuracy constraint (solving problem described in Definition 3.3.2):

$$\begin{aligned} &\text{Minimize:} && f_{time}(Q) \\ &\text{Subject to:} && Eq(3.1), Eq(3.6), \\ & && f_{accu}(Q) \geqslant A_{bound} \end{aligned}$$

3.6. PREDICATE ORDERING AS MIP

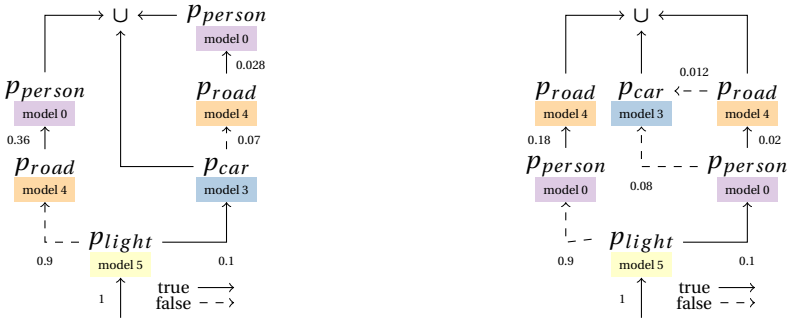
Predicate ordering has a significant impact when we generate a bypass plan. If the plan is a sequential execution of models without filtering any data, the results are the union of the predictions of all the models and the execution order will not make an effect on the results. On the other hand, a predicate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

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 Figure 3.1(c), 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 Section 3.6.1.

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 Figure 3.3, 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 3.3(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 Figure 3.3(a).

Algorithm 2 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 3.3), objective type T (execution time or accuracy) and



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

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

Figure 3.3: Bypass plans with different predicate execution order (numbers near by the arrows indicating the selectivity of the predicates in that path)

bound β (C_{bound} or A_{bound}). We write the objective functions introduced in Section 3.5 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 2. 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 Section 3.6.1, and Section 3.6.2, and the transformed objective functions in Section 3.6.

3.6.1. 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.

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

Continuing with the running example, Table 3.4 shows an example of a possible order of the four predicates. The order is $p_{\texttt{light}} \rightarrow p_{\texttt{person}} \rightarrow p_{\texttt{road}} \rightarrow p_{\texttt{car}}$.

Algorithm 2: Order-optimal Optimizer

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

1 $K \leftarrow [Q,R,T,\beta]$ `// input variables`
2 $f'(X,O,G,K) \leftarrow OrderOpt(f(X,K))$`// transform obj func`
3 $X,O,G \leftarrow MILP\_Solver(f'(X,O,G,K))$
4 $optPlan \leftarrow QueryPlanGen(X,O,G)$
5 **return** $optPlan$

Table 3.4: $O_{p,j}$ 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

Answering exactly one predicate at each step. Similar to Eq(3.1), we design the following constraint to restrict the number of predicates executed at each step j:

$$\sum_{p \in P} O_{p,j} = 1, \tag{3.8}$$

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

$$\sum_{j=0}^{|P|-1} O_{p,j} = 1 \tag{3.9}$$

3.6.2. 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.

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 [1], 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):

$$C_Q = C_0 + C_1 S_0^P + C_2 S_0^P S_1^P + \ldots + C_{|r|-1} \prod_{i \in [0,|r|-2]} S_i^P \tag{3.10}$$

[1] 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):

$$\begin{aligned} C_Q = & C_0 + C_1(1-S_0^P) + C_2(1-S_0^P)(1-S_1^P) \\ & + \ldots + C_{r-1} \prod_{i \in [0,r-2]} (1-S_i^P) \end{aligned} \tag{3.11}$$

3

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 3.6.1. 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:

$$G_{g,j} \geqslant 1 - |P_g| + \sum_{p \in P_g} \sum_{0 \leqslant k \leqslant j-1} O_{p,k}. \tag{3.12}$$

$$G_{g,j} \leqslant \sum_{0 \leqslant k \leqslant j-1} O_{p,k} \tag{3.13}$$

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, $S_0^G = S_{\texttt{road}}^P S_{\texttt{person}}^P$ for group 0 and $S_1^G = S_{\texttt{light}}^P S_{\texttt{car}}^P$ 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:

$$W_{g_i,j} = 1 - G_{g_i,j} S_g^G \tag{3.14}$$

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., $S_0^G = S_{\texttt{road}}^P S_{\texttt{person}}^P$ 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:

$$H_{g,j} = H_{g,j-1} \cdot (1 - \sum_{p \in P_g} O_{p,j-1}(1 - S_p^P)) \tag{3.15}$$

As mentioned before unnecessary product variables should be avoided in MIP. Products of binary and continuous variables can oftentimes be linearized [10] without any error. We introduce a new variable $Q_{g,p,j}$ that takes the value of $H_{g,j-1} \cdot O_{p,j-1}$:

$$Q_{g,p,j} \leqslant M \cdot O_{p,j-1}, \qquad 1 \leqslant j \leqslant |P| \tag{3.16}$$

$$Q_{g,p,j} \geqslant H_{g,j-1} - (1 - O_{p,j-1})M, \qquad 1 \leqslant j \leqslant |P| \tag{3.17}$$

$$0 \leqslant Q_{g,p,j} \leqslant H_{g,j-1}, \qquad 1 \leqslant j \leqslant |P| \tag{3.18}$$

with M an upper bound on $Q_{g,p,j}$ that holds in every case, but in our case $M = 1$ suffices as $H_{g,j}$ is a number between 0 and 1.

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:

$$S_j^J = \prod_{g=0}^{|group|} W_{g,j} H_{g,j} \tag{3.19}$$

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 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 $R_{m,j}$ to represent the execution cost of a model m for each step j. The set of of all possible variables $R_{m,j}$ is R. The cost of a query plan can be computed as follows:

$$R_{m,j} = S_j^J \sum_{p \in P} X_{m,p} O_{p,j} C_{m,p} \tag{3.20}$$

Note that S^J is an MIP variable that depends on decision variables $X_{m,p}$ and $O_{p,j}$. Linearizing products containing S^J is therefore desirable. Eq.(3.20) can be linearized by

introducing another variable $Y_{m,p,j}$ that takes the value of $S_j^J X_{m,p} O_{p,j}$:

$$Y_{m,p,j} \leqslant M \cdot X_{m,p} \tag{3.21}$$
$$Y_{m,p,j} \leqslant M \cdot O_{p,j} \tag{3.22}$$
$$Y_{m,p,j} \geqslant S_j^J - (2 - X_{m,p} - O_{p,j})M \tag{3.23}$$
$$0 \leqslant Y_{m,p,j} \leqslant S_j^J \tag{3.24}$$

and thus the products can be replaced in Eq(3.20):

$$R_{m,j} = \sum_{p \in P} Y_{m,p,j} C_{m,p} \tag{3.25}$$

The execution time of each model should be computed only once, even though it can answer multiple predicates. Thus, the cost model is:

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

For example, In Figure 3.1(d) $model$ 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 $p_{\texttt{light}}$.

3.6.3. 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:	$f_{accu}(Q)$
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;
	$f_{time}(Q) \leqslant C_{bound}$

Objective functions. To conclude, with Eq(3.4) and Eq(3.7), we have transformed the two problem variants in Section 3.3.4 to a matter of MIP by defining two objective functions as below. With the above transformed objective functions ready, we obtain the values of all defined variables, such as X, 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 1 we generate the bypass plans.
Given an execution time constraint (solving problem described in Definition 3.3.1):

Maximize:	$f_{accu}(Q)$
Subject to:	$Eq(3.1), Eq(3.2), f_{time}(Q) \leqslant C_{bound}$

Table 3.5: Constraints for formalizing model assignment for query plans

Eq index	Constraint
Eq(3.1)	$\forall p : \sum_{m \in M} X_{m,p} = 1$ Semantics: Only select one model for each predicate
Eq(3.2)	$\forall p, m$: $X_{m,p} \leqslant \lceil A_{m,p} \rceil$ Semantics: Only assign a model to a predicate it can successfully inference on
Eq(3.6)	$\forall m, p : X_{m,p} \leqslant B_m$; $B_m \leqslant \sum_{i \in P} X_{m,i}$ Semantics: Identify whether model m is selected
Eq(3.8,3.9)	$\forall j : \sum_{p \in P} O_{p,j} = 1$; $\forall p : \sum_{j \leqslant \lvert P \rvert - 1} O_{p,j} = 1$ Semantics: At each step, only one predicate is answered
Eq(3.13)	$\forall g \ \forall j \ \forall p \in P_g : G_{g,j} \leqslant \sum_{0 \leqslant k \leqslant j-1} O_{p,k}$ Semantics: Variables are applicable if all the predicates are answered within the same group
Eq(3.14)	$\forall g \forall j : W_{g,j} = 1 - G_{g,j} S_g$ Semantics: Determines the selectivity produced by group g in step j
Eq(3.15, 3.16, 3.17, 3.18)	$\forall g \forall j \geqslant 1 : H_{g,j} = H_{g,j-1} - \sum_{p \in P_g} Q_{g,p,j-1}(1 - S_p^P))$; $Q_{g,p,j} \leqslant M \cdot O_{p,j}$; $Q_{g,p,j} \geqslant H_{g,j-1} - (1 - O_{p,j-1})M$; $0 \leqslant Q_{g,p,j} \leqslant H_{g,j-1}$; $\forall g : H_{g,0} = 1$ Semantics: Determines the selectivity produced by the predicate in the same group
Eq(3.19)	$\forall j : S_j = \prod_{g \leqslant \lvert group \rvert} W_{g,j} H_{g,j}$ Semantics: Selectivity at step j
Eq(3.21, 3.22, 3.23, 3.24,3.25)	$\forall m \forall j : R_{m,j} = \sum_{p \in P} Y_{m,p,j} C_{m,p}$; $\forall m \forall p \forall j \, Y_{m,p,j} \leqslant M \cdot X_{m,p}$; $Y_{m,p,j} \leqslant M \cdot O_{p,j}$; $Y_{m,p,j} \geqslant S_j^J - (2 - X_{m,p} - O_{p,j})M$; $0 \leqslant Y_{m,p,j} \leqslant S_j^J$; Semantics: Determines the cost of executing model m at step j

Given an accuracy constraint (solving the problem in Definition 3.3.2):

$$\begin{aligned} &\text{Minimize:} && f_{time}(Q) \\ &\text{Subject to:} && Eq(3.1), Eq(3.2), f_{accu}(Q) \geqslant A_{bound} \end{aligned}$$

3.7. MULTI-OBJECTIVE MIXED INTEGER PROGRAMMING

We extend the problem further by considering multiple objectives, including accuracy, speed and memory consumption. We formulate the model selection problem as *Multi-Objective Mixed Integer Program* (MOMIP). In this scenario, we only consider *sequential plan*, thus predicate ordering is not included.

Example 1. Consider the query from the Section 4.1 with the following simplified model zoo example:

Model name	C	S	A_{car}	A_{human}
model1	10	3000	80%	0%
model2	30	8000	95%	0%
model3	20	5500	0%	85%
model4	40	9000	0%	95%

Table 3.6: A model zoo example

Example 2. Following the above definition, we reformulate the query from the introduction as follows:

$$q := model2_{car} \wedge model4_{human} \tag{3.26}$$

In short, this means that *model2* will be used to evaluate the predicate `car` and *model4* will be used to evaluate the predicate `human`.

Example 3. Continuing with Example 2, this model selection achieves 90.25% accuracy, execution cost of 70ms, and 17000B memory footprint, according to the model zoo in table 3.6, calculated with the objective functions in section 3.7.

OBJECTIVE FUNCTIONS

In this work we consider three objectives, accuracy $f_{accu}(Q)$, execution cost $f_{cost}(Q)$, and memory footprint $f_{mem}(Q)$. Below we will introduce the fomular to compute different objectives. Other constraints on, e.g., X and B, is the same as above.

Calculating accuracy. Consider an example DNF query, (`car` $\wedge$ `outdoor`) $\vee$ (`chair` $\wedge$ `indoor`). The clauses in the query is connected by disjunction. We first estimate the accuracy of the clauses, referred as c_1, c_2. We assume the predicates are independent to each other. Similar assumption has been made in [120]. The accuracy of a conjunctive query, e.g., `car` $\wedge$ `outdoor`, can be estimated by multiplying the accuracy of each models, $a_{car} * a_{outdoor}$. The accuracy of a disjunctive query, e.g., `car` $\vee$ `bus`, can be computed using inclusion–exclusion principle, as $a_{car} + a_{bus} - a_{car} * a_{bus}$. Following the same manner, we can calculate the accuracy, f_{accu}, of more complex Boolean expressions.

In the canonical form of MOO problems all objectives either have to be minimized or maximized however. Therefore we introduce "accuracy loss":

$$f_{accu\_loss}(Q) = 1 - f_{accu}(Q) \tag{3.27}$$

Minimizing $f_{accu\_loss(Q)}$ is equivalent to maximizing $f_{accu}(Q)$.

Calculating execution cost. The second objective, execution cost $f_{cost}(Q)$, is obtained by summing the execution cost of all used models:

$$f_{cost}(Q) = \sum_{m \in M} C_m B_m \tag{3.28}$$

Calculating memory. The third objective, memory footprint f_{mem} is obtained similarly. We assume all the models are loaded in advance before executing them.

$$f_{mem}(Q) = \sum_{m \in M} S_m B_m \tag{3.29}$$

We derive the following MOMIP:

$$\begin{array}{ll} \min_{y \in Y} & \left[f_{accu\_loss}(Q), f_{cost}(Q), f_{mem}(Q)\right] \\ \text{such that} & Eq.(3.1), Eq.(3.2), Eq.(3.6) \text{ hold.} \end{array} \tag{3.30}$$

Where Y denotes the set of valid query plans.

FUNCTION NORMALIZATION

Many MOO methods will prioritize optimizing the objectives that naturally take larger values (such as memory that ranges in the thousands, versus accuracy that ranges from 0 to 1). To avoid this 'preferential treatment' we use a normalization method [122]:

$$f_i^{norm}(y) = \frac{f_i(y) - f_i^{\min}}{f_i^{\max} - f_i^{\min}} \tag{3.31}$$

Which normalizes the objective functions f_i between the minimum and maximum obtainable value.

3.7.1. MULTI-OBJECTIVE MIXED INTEGER OPTIMIZATION METHODS

Before we introduce the MOO methods, some important definitions need to be introduced [122]:

Definition 3.7.1. A point $\mathbf{y} \in \mathbf{Y}$ is **Pareto optimal** iff there does not exist a point $\mathbf{y}^* \in \mathbf{Y}$ such that $f_i(\mathbf{y}^*) \leq f_i(\mathbf{y})\ \forall\ 0 \leq i \leq k$ and $f_j(\mathbf{y}^*) < f_j(\mathbf{y})$ for some j. y is **weakly Pareto optimal** iff there does not exist a solution $\mathbf{y}^* \in Y$ such that $\mathbf{f}(\mathbf{y}^*) < \mathbf{f}(\mathbf{y})$.

Example 4. The query plan in Example 3 is Pareto optimal: with the models in Example 1 we can only achieve lower cost or memory footprint by decreasing accuracy.

Preference methods. We name the relative importance of objectives *preferences*. Preferences can be indicated in may ways (e.g. weights or hierarchies), which we name.

MOO methods. We consider three methods commonly used in MOMIPs [8] and a naive greedy method of our own contribution.

THE WEIGHTED SUM METHOD

The weighted sum method minimizes the weighted sum of all objectives. It guarantees a Pareto optimal solution. The weighted sum model selection MOMIP is formulated as follows:

$$\begin{array}{ll} \min_{y \in Y} & w_{acc\_loss} f_{accu\_loss}^{norm} + w_{cost} f_{cost}^{norm} + w_{mem} f_{mem}^{norm} \\ \text{such that} & Eq.(3.1), Eq.(3.2), Eq.(3.6) \text{ hold.} \end{array} \tag{3.32}$$

THE WEIGHTED MIN-MAX METHOD

In the weighted min-max method, also known as the weighted Tchebycheff method, an ancillary variable λ is introduced as an upper bound to every (weighted) objective, which is then minimized. It generates weakly Pareto optimal solution. The weighted min-max MOMIP model selection is formulated as follows:

$$\begin{array}{lccc} \min_{y \in Y} & \lambda & & \\ \text{such that} & Eq.(3.1), Eq.(3.2), Eq.(3.6), & & \\ & w_i f_i^{norm} & \leq & \lambda, \\ & i \in \{acc\_loss, cost, mem\} & \text{hold.} & \end{array} \tag{3.33}$$

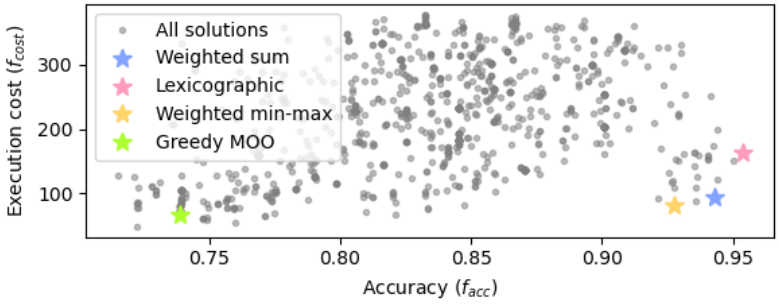


Figure 3.4: MOO solutions in an example search space

THE LEXICOGRAPHIC METHOD

In the lexicographic method a hierarchy of objectives is used to convey their importance. First the program is solved for the most importance objective. The objective value is then added as an upper bound to that objective, and next the second most important objective is used as the objective function. This means that the MOMIP needs to be solved several times, and the method is inefficient for finding compromise solutions. The lexicographic method guarantees Pareto optimal solutions.

GREEDY-MOO (BASELINE)

Our greedy baseline pairs predicates to models using a basic weighted sum utility function:

$$\begin{aligned} U(m,p) &= w_{acc\_loss}\frac{1-A_{m,p}}{1-\min\{A_{n,p}|n\in M\}} \\ &+ w_{cost}\frac{C_m}{\max C} + w_{mem}\frac{D_m}{\max D} \end{aligned} \tag{3.34}$$

which is then used to calculate the following model selection:

$$X_{m,p} = \begin{cases} 1 & \text{if } U(m,p) = \min\{U(n,p)|n \in M, A_{n,p} > 0\} \\ 0 & \text{elsewhere} \end{cases} \tag{3.35}$$

Greedy-MOO does not guarantee Pareto optimal solutions, which can be shown using simple counterexamples.

Example 5. We visualize a sample search space with solutions found by the different MOO methods in Figure 3.4. Note how the solutions vary in their objective values, even when they use similar preference profiles.

3.8. EXPERIMENTAL EVALUATION

In this section, we empirically evaluate our optimizer 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.

Table 3.7: 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

3.8.1. EXPERIMENTAL SETTINGS

DATASETS AND EVALUATION METRICS.

We used public datasets covering object detection in images with COCO [115] as well as sentiment analysis in text with TweetEval [18]. *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 3.7:

– *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 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 3.7).

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

Table 3.8: 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

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` (In the plots, it is short for `Bypass: 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 [120] to curate queries. We generate queries for two scenarios: comparing optimizer quality (Section 3.8.2, 3.8.3) and measuring optimization time (Section 3.8.4).

Optimizer performance. We manually curated 10 queries (exemplified in Table 3.8) 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.

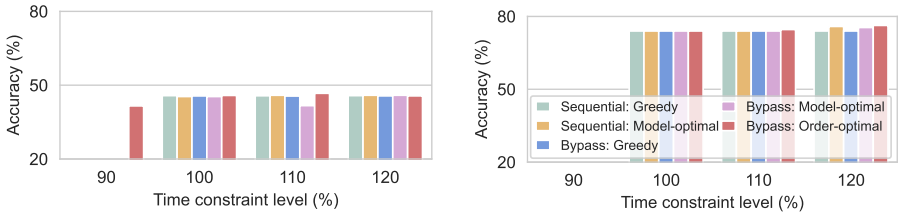
Query optimization time. We generate a set of queries in different complexity levels (the number of predicates ranging from 2 to 64), in total, 60 queries in CNF and DNF. The classes are adopted from COCO. For each predicate, we sample the classes with a uniform distribution.

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, 8GB RAM).

3.8.2. USING UNIFORM MODEL ZOOS

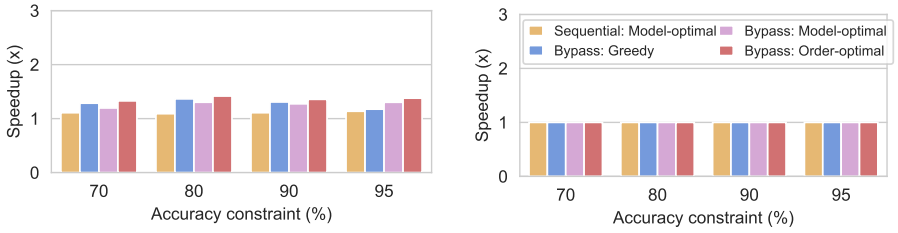
We analyse the behavior of our optimizer using the model zoos Model Zoo ❶ and Model Zoo ❷ . In this experiment we consider the constraint of 100% to be the execution time that allowed the `Sequential:Greedy` optimizer to find a solution to all the queries. We constrain the execution time to gradually increase from 90% - 120% to observe how the optimizers behave with different constraints. We present those results in bar plots (Fig-



(a) Model Zoo ❶ (Text)

(b) Model Zoo ❷ (Image)

Figure 3.5: Average accuracy performance on the query workload with different execution time constraints.



(a) Model Zoo ❶ (Text)

(b) Model Zoo ❷ (Image)

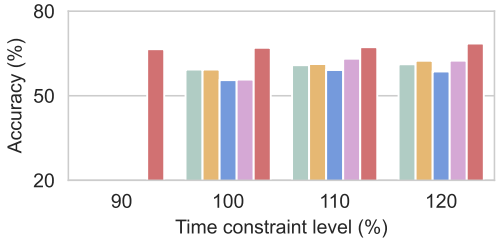
Figure 3.6: Average speedups of query execution time compared to the *Greedy* approach on the query workload with different accuracy constraints.

ure 3.5, Figure 3.6). 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, Model Zoo ❶ and Model Zoo ❷ have very similar accuracy, we do not observe large differences. It is worth noting that generating a bypass plan for the `Model-optimal` 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.

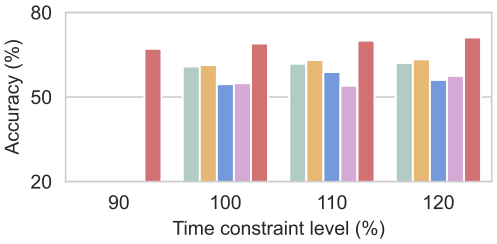
3.8.3. USING MODEL ZOOS WITH DIVERSE MODEL DISTRIBUTIONS

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.

Constraining Cost. Figure 3.7 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, by-



(a) Model Zoo ❸



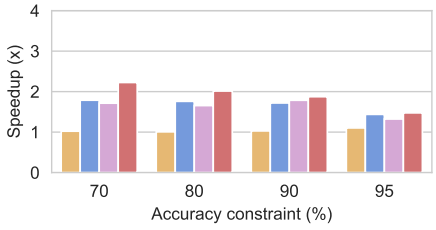
(b) Model Zoo ❹

Figure 3.7: The average accuracy on the query workload with different time (objective) constraint levels.

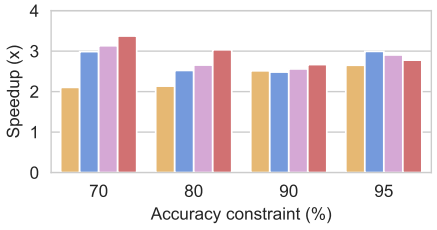
pass plans do not gain benefits when execution time is constrained. While `Bypass: Model-&Order-optimal` jointly optimizes for both model selection and predicate ordering can make use of predicate ordering and perform early filtering, making better use of execution time budget.

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.

Constraining Accuracy. Figure 3.8 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



(a) Model Zoo ❸



(b) Model Zoo ❹

Figure 3.8: The average speedups of query execution time compared to the *Greedy* approach on the query workload with different accuracy (objective) bounds.

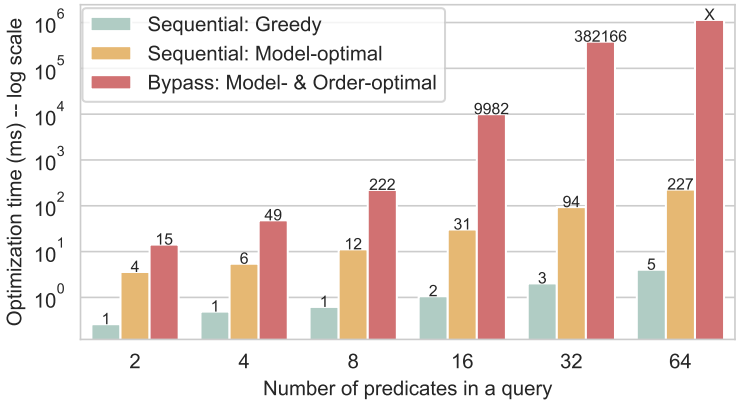


Figure 3.9: Optimization time on queries with varying number of predicates.

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.

3.8.4. 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 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 Figure 3.9. 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 (Figure 3.9 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 (Section 3.3). 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.

3.8.5. RESULTS OF MOMIP

Setup. We adopt the same setting as previous. In this subsection, we only tackle the *Real-world* setting, with Model Zoo ❶ containing 48 real-world models that can tackle

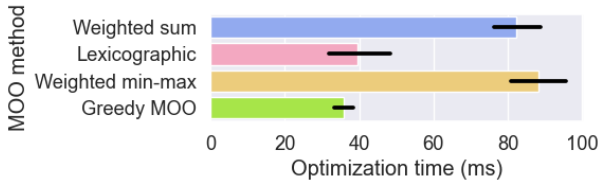
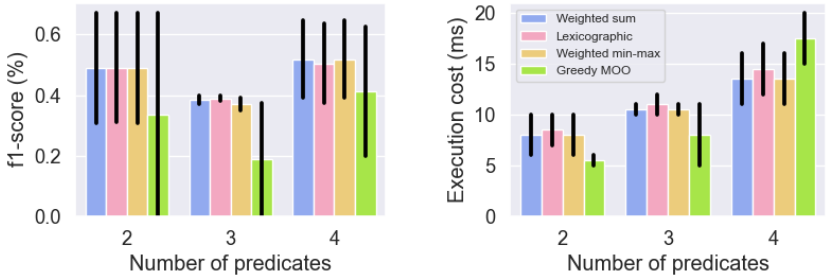


Figure 3.10: Optimization time over different MOO methods



(a) f1-score

(b) Execution cost

Figure 3.11: Query plan performance for the NLP scenario

NLP tasks and Model Zoo ❷ including 33 models that can be used in object detection tasks in images

Queries. To run our experiments we formulated 10 queries for Model Zoo ❷ and 6 for Model Zoo ❶ , half in CNF and half in DNF. The queries vary from 2 to 6 predicates, with 1 to 2 predicates per group. Queries are similar to (`person` ∧ (`car` ∨ `bike`) ∧ `emergency_light`).

OPTIMIZATION

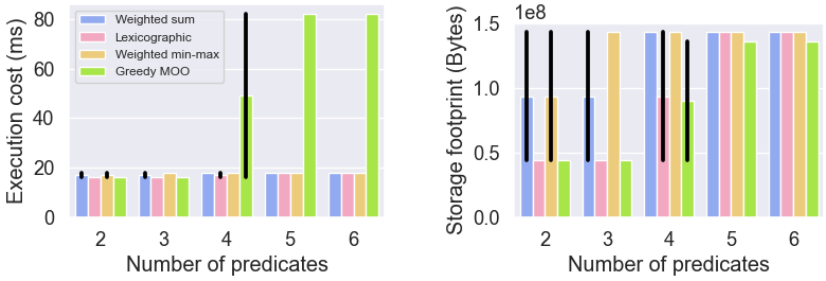
We generate query plans for every query over two preference profiles (see Section 3.8.5), execute them over the test set, and record the resulting f1-score, and execution cost. A preference profile consists of a hierarchy of the three objectives, where the most important objective gets a weight of $\frac{1}{2}$, the second $\frac{1}{3}$, and the last $\frac{1}{6}$. We also compare the methods on the time it takes to generate a query plan, for which we use a larger number of randomly generated queries.

OPTIMIZATION TIME

For optimization time, visualized in Figure. 3.10 with aggregated results from different queries, we see that the `greedy-MOO` and the lexicographic method perform significantly better than the weighted sum and weighted min-max method, that perform comparatively.

QUERY EXECUTION

To compare query execution performance, we spotlight two use case scenarios and compare query plans calculated with our 4 MOO methods on their two most important ob-



(a) Execution cost

(b) Memory footprint

Figure 3.12: Query plan performance for the OD scenario

jectives.

- NLP: accuracy > execution cost > memory;
- OD: execution cost > memory > accuracy.

We see that for both scenarios in Figure 3.11 and 3.12 that the greedy method performs poorly. Even for the most important objective (accuracy in the NLP scenario, execution cost in the OD scenario) it has bad scores. The lexicographic method manages to score well for its most important objective, but underperforms for the others. The weighted sum and weighted min-max method perform very comparably, finding highly similar query plans in most cases and balancing objectives adequately. Due to its slightly lower computation time and guarantee for Pareto-optimal query plans, the weighted sum method would be the better choice for our optimizer.

3.9. CONCLUSIONS AND FUTURE WORK

In this chapter 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. The optimizer utilizes the metadata of both models and datasets, which indicates the significance of metadata in this application. In addition, we investigated the multi-objective ML inference query optimization problem and formulated as MOMIP that optimizes for accuracy, execution cost, and memory footprint. We tested several commonly used MOO methods and compared them on their theoretical suitability and tested their performance in different experimental settings. We note that the weighted-sum based optimizer can process user preferences and balance objectives accordingly, outperforming naive methods. Future work can investigate effect of the assumptions.

Further research can focus on i) applying approximation schemes in the MIP formulation of the problem and ii) lifting the assumptions made in this work, considering especially the correlation of inference predicates and concept drift.

4

Model Selection with Model Zoo for fine-tuning

In this chapter, we continue to explore another application of using metadata for enhancing ML task, i.e., model selection for fine-tuning. Given a large number of pre-trained models, it presents a computational challenge when fine-tuning them for a new dataset, often proving costly and even infeasible. Selecting the right pre-trained models is crucial, yet complicated by the diversity of models from various model families (like ResNet, Vit, Swin) and the hidden relationships between models and datasets. Existing methods, which utilize basic information from models and datasets to compute scores indicating model performance on target datasets, overlook the intrinsic relationships, limiting their effectiveness in model selection. In this study, we introduce *TransferGraph*, a novel framework that reformulates model selection as a graph learning problem. TransferGraph constructs a graph using extensive metadata extracted from models and datasets, while capturing their intrinsic relationships. Through comprehensive experiments across 12 real datasets, we demonstrate TransferGraph's effectiveness in capturing essential model-dataset relationships, yielding up to a 21.8% improvement in correlation between predicted performance and the actual fine-tuning results compared to the state-of-the-art methods.

This chapter is based on the following publication:

- Li, Z., Van der Wilk, H., Khosla, M., Katsifodimos, A., Bozzon, A., & Hai, R., (2024, May). Model selection with model zoo for fine-tuning, *In 2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE.

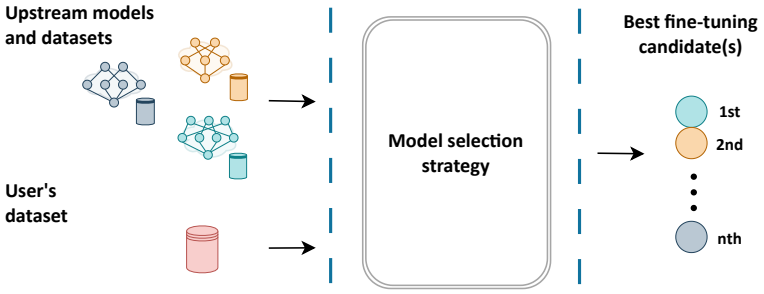


Figure 4.1: Illustration of the model selection problem setting.

4

4.1. INTRODUCTION

Deep learning has been widely used in handling image data, such as image classification, and object detection. The paradigm of *first pre-training, then fine-tuning* has become the de facto of applying deep learning in practice. Pre-training is the phase of training a neural network on a large, diverse dataset, typically drawn from a general domain, e.g., ImageNet [47]. Subsequently, the fine-tuning step refines the model for a specific task, often a smaller target dataset. This two-step process leverages the general knowledge acquired during pre-training, facilitating effective adaptation to a narrower and more specialized context. The general representations learned during pre-training, speed up model convergence during fine-tuning and help reduce the risk of over-fitting.

Today, many pre-trained models are available in public online platforms, e.g., HuggingFace [1], TensorFlow Hub[2], and PyTorch Hub[3]. Such repositories of pre-trained models are referred to as *model zoos*. Model zoos have been widely adopted in recent years, as they offer convenient access to a collection of pre-trained models, including cutting-edge deep learning architectures. This lowers the expertise barrier, enabling non-expert individuals to apply complex deep learning models in their applications. Moreover, utilizing a model zoo for fine-tuning facilitates the adaptation across a wide range of target datasets, which have varying quantities of training data [49]. In addition, by fine-tuning pre-trained models from the model zoo, machine learning practitioners can bypass the need for training from scratch—a resource-intensive process, resulting in significant savings in both development time and computational resources. However, it is a non-trivial task to pick the *right* pre-trained models as the starting point of fine-tuning, which has a substantial impact on the effectiveness of the fine-tuning results [49]. A straightforward solution is to fine-tune all pre-trained models relevant, which is computationally expensive, and sometimes infeasible in practice. For instance, there are 7,411 models for image classification tasks in the HuggingFace repository and 900 variations on TensorFlow Hub. It took 1178 hours of GPU time to fine-tune all the 185 models in our model zoo on a single dataset.

The practical choice is to identify pre-trained models that exhibit promising perfor-

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

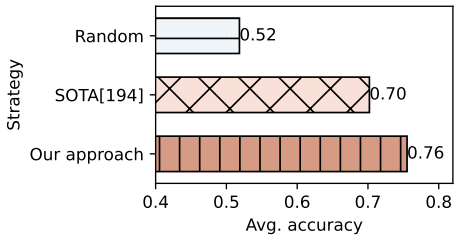


Figure 4.2: Average fine-tuned accuracy of the top 5 selected models compared between random selection strategy and our proposed solution learning from a graph along with metadata (example dataset: stanford-cars).

mance even without fine-tuning, i.e., *model selection*. As in Figure 4.1, given a target dataset and several pre-trained models over existing datasets, model selection aims to rank and select optimal candidates from the model zoo to perform fine-tuning. Different strategies may yield disparate rankings of the candidates.

A naive approach is to randomly select models for fine-tuning. When pre-trained models all have similar fine-tuning accuracy, this random selection strategy may suffice. Yet, in the more general case where the performance of models varies, the random strategy is ineffective. In Figure 4.2, we report the results of the average accuracy of the top five models selected through diverse strategies (full results in Section 4.7). `Random` denotes a random selection strategy, which only achieved an unsatisfying accuracy value of 0.52.

Existing studies [23, 79, 110, 138, 175, 194] mainly focus on extracting information about the pre-trained models and datasets, and mapping model features to the target dataset labels. The effectiveness of feature is expected to decline as the source dataset (training dataset of the pre-trained model) and target dataset become less similar [193]. However, such information does not provide insight into whether a model would exhibit similar performance when the source and target datasets are similar. That is, existing methods consider only basic characteristics of datasets and models, while neglecting their inherent relationships.

Therefore, in this chapter, we are interested in exploring such intrinsic relationships, and investigating whether they help the model selection problem, and improve the fine-tuning accuracy given the target dataset. We borrow the inspiration from data management systems for data repositories, such as data lakes [71, 134, 174]. For managing a collection of datasets, a common approach is to structure these datasets as graphs [54, 135, 201]. This involves representing tables as nodes and their relationships as edges. For instance, an edge can indicate two tables are semantically similar [54]. For model selection problem, rich relationships exist not only between models and datasets but also among datasets themselves.

Borrowing this concept, we reformulate the challenge of model selection for image datasets as a graph link prediction problem. We propose *TransferGraph*, which explores how the relationships among *dataset-dataset* and *dataset-model* can facilitate more effective model selection, offering a structured and intuitive method to navigate and un-

derstand these complex relationships. To represent and analyze these intricate relationships, we represent these relationships using graph structures. We prove that with graph features learned from the graph structure and along with only metadata information (e.g., model architecture, data size), *TransferGraph* is able to identify suitable pre-trained models for the target dataset. As shown in Figure 4.2, TransferGraph outperforms the state-of-the-art method [110] with a notable improvement in fine-tuning accuracy.

Contributions. We summarize our contributions as follows.

- We reformulate the model selection problem as a graph learning problem.
- Different from the existing works only taking into account the dataset labels or solely information about models or datasets, we exploit the metadata of both models and datasets and further learn the intrinsic relationships between the artifacts by learning a graph.
- We propose a framework that tackles the model selection problem via graph learning. The framework consists of end-to-end processes, from feature collection, graph learning, to model performance prediction.
- Extensive experiments are conducted to evaluate the validity of our graph-based model selection strategy. We show that with a simple graph learning method, we can predict model performance with a high correlation to the actual fine-tuning accuracy.

4.2. BACKGROUND AND PROBLEM DEFINITIONS

It is a challenging task to select the *right* models for fine-tuning, especially given the abundant pre-trained models in the model zoo. In this section, we explain existing model selection strategies and identify their limitations.

4.2.1. MODEL SELECTION STRATEGIES

Previous model selection strategies consider features of models and datasets from different perspectives. They mainly differ in the information of datasets and pre-trained models they use. Some works like SHiFt [153] have developed systems which combine these approaches, while also taking user inputs such as budget constraints into account. We categorizes these model selection strategies based on the features they employ to rank the models for model selection.

Task similarity model selection. Early model selection methods use the similarity of the source and target tasks to measure the transferability of a model. When the target task is similar to source task, a model with good performance on the source task is likely to have good fine-tuning performance [183]. Methods in this group include EMD [46] and NCE [175], Task2Vec [3]. To obtain the similarity of the source and target tasks, EMD [46] and NCE [175] compare source and target task features and labels. Task2Vec [3] embeds tasks as vectors using a single probe model and computes their pairwise distances as transferability scores.

Feature-based model selection. More recent approaches leverage the target task specific features, which are extracted by executing a forward pass of the target task on each

pre-trained model (model inference on the target dataset). Methods in this group include LEEP [138], LogME [194], PARC [23] and TransRate [79]. These approaches circumvent the need for fine-tuning. However, as the number of pre-trained models in a model zoo grows, it becomes inefficient to perform a forward pass over all pre-trained models, even infeasible. Moreover, methods in this group overlook basic features of both the target dataset (like the number of samples and labels) and the pre-trained model (such as input size and architecture), which are crucial for fine-tuning efficiency. For instance, a mismatch in input size or the number of classes, leads to significant deterioration in fine-tuning performance [110].

Learning-based model selection. The third group of approaches [23, 110] trains a simple linear model, e.g., a linear regression model, to predict model performance and recommend pre-trained models given a new target task. The used features extracted from models or *metadata* of the target dataset and pre-trained models. The state-of-the-art approach, Amazon LR [110], employs only basic metadata of the target dataset and pre-trained models. It achieved competitive results when learning a linear regression model. The authors suggest that incorporating additional features could potentially enhance this method even further.

4.2.2. LIMITATIONS AND CHALLENGES

We summarize the limitations of existing model selection strategies, and outline the challenges to tackle these limitations.

OVERLOOKING THE HETEROGENEITY OF MODEL ZOO

A model zoo may encompass a variety of heterogeneous models and datasets. Models within this context can exhibit differences in pre-trained domain, architecture, and hyper-parameter settings. At the same time, datasets vary in terms of the tasks they address and the distribution of their data. Predicting the performance and capability of models is challenging, given that the models are trained differently, and the inductive biases of models are different. Featured-based model selection strategy usually use the model as feature extractors or assumes the fine-tuning process does not change the backbone weights much [23, 48]. However, such an assumption does not hold in practice.

Prior studies [48, 79] have often restricted model architectures to certain categories, e.g., ResNet, MobileNet or DenseNet. In LogMe [194], only models pre-training on the same source dataset (e.g., ImageNet) are included. However, the optimal architecture or Pareto-optimal models are usually task-dependent, relying on the inductive bias of the model and the dataset properties [110]. Fine-tuning with a model zoo helps transfer to a diverse set of target tasks with different downstream datasets. Due to the diversity of the model characteristics, e.g., architecture family, pre-trained domain, and hyper-parameter settings, it is even more challenging to identify suitable candidate models for the downstream task.

INSUFFICIENT FEATURES COVERAGE

Feature-based model selection strategies [23, 48] often rely on the model features and the target dataset labels. They assume that models can generalize better if the features extracted by the model are similar and labels are similar. However, such approaches

struggle to accurately predict top-performing models for target datasets significantly different from source datasets used for pre-training [110]. Conversely, incorporating simple prior knowledge, such as dataset characteristics, is proven to help predict the model performance on downstream datasets [110]. Learning from the basic metadata of models and datasets are beneficial yet limited due to its coarse-grained nature. It often overlooks the intricate relationships between models and datasets. Therefore, a central challenge lies in identifying and utilizing such *intrinsic* relationships for more effective model selection.

4.3. MODEL SELECTION AS A GRAPH LEARNING PROBLEM

In this section, we first explain the problem setting of model selection. To tackle the challenges in Sec. 4.2.2, we propose transforming the model selection problem to a graph learning problem.

4.3.1. PROBLEM DEFINITION

Consider a set of models, denoted as $M = \{m_1, ...m_N\}$, and a collection of datasets, represented as $\chi = \{d_1, ...d_K\}$. We denote the actual fine-tuning accuracy as $T_{i,j}$, with respect to the model m_i and the target dataset d_j, where $m_i \in M$, and $d_j \in \chi$. Given a pre-trained model m_i, we are interested in predicting a score S_{ij} which approximates its fine-tuning accuracy on the target dataset d_j.

Example 4.3.1. Consider two models m_1 and m_2, and two target datasets d_1 and d_2. The predicted scores of the models on each dataset can be presented by a matrix $S = \begin{pmatrix} 0.6 & 0.8 \\ 0.7 & 0.3 \end{pmatrix}$. For dataset d_1, the predicted score S_{11} of m_1 on d_1 is 0.6, and S_{21} (m_2 on d_1) is 0.8. It indicates that m_2 is predicted to have better fine-tuning performance than m_1. Whereas it is a different case on the dataset d_2, with S_{12}=0.7 higher than S_{22}=0.3.

The predicted score should be a good approximation of the actual fine-tuning results and exhibit a strong correlation with the target dataset. Such an alignment would enable the predicted score to be a reliable indicator of fine-tuning performance on the target dataset, allowing for the effective ranking of pre-trained models.

To measure the effectiveness of the model selection score, we use *Pearson's correlation coefficient*, following the common practice [110]. We use $\tau \in [-1, 1]$ to represent the Pearson's correlation. Given N paired data $\{(m_1, d_t), (m_2, d_t), ...(m_N, d_t)\}$, τ is defined as:

$$\tau = \frac{\sum_{i=1}^{N}(T_i - \bar{T})(S_i - \bar{S})}{\sqrt{\sum_{i=1}^{N}(T_i - \bar{T})^2 \sum_{i=1}^{N}(S_i - \bar{S})^2}} \tag{4.1}$$

The goal is to maximize the correlation between the predicted scores and the model performance. An absolute value of 1 implies that a S perfectly aligns with the trend of T with all data points lying on a line.

4.3.2. CONVERT MODEL SELECTION TO GRAPH LEARNING

We formulate the model selection problem as a graph learning process, which maximizes the Pearson correlation τ, between the predicted score S_{ij} and the actual fine-tuning accuracy T_{ij}.

Notation	Definition
$S_{i,j}$	Predicted transferability score of m_i on d_j
$T_{i,j}$	Fine-tuning performance of m_i on d_j
χ, d_j	Set of datasets collection and dataset j
M, m_i	Model collection and model i
ϕ	Dataset similarity
G	Graph
E	Edge of Graph
V	Vertex / Node
L	Edge labels
τ	Pearson's Correlation
$f_G()$	Function over the graph
$W^{(k)}$	Weights of the graph
$Q^{(k)}$	Operation that allows aggregation afterwards
$\bar{X}$	Mean of the variable X
$\hat{X}$	Prediction for the variable X
$F()$	Learned function to predict the model performance

Table 4.1: Notation definitions

Definition 4.3.1 (Graph)**.** We denote a graph as $G = (V, E)$ where V is the set of vertices and $E \subseteq V \times V$ denotes the set of edges that connect the vertices in V.

In our setting, a vertex either represents a dataset or a model. Given a set of datasets $\chi = \{d_1, d_2, ..., d_K\}$, and a set of pre-trained deep learning models $M = \{m_1, m_2, ..., m_N\}$, we build our vertex set as $V = \chi \cup M$. Here $K = |\chi|$ is the number of datasets, and $N = |M|$ is the number of models.

In our graph, we construct three types of edges, depending on the nodes connected by the edges. The first edge type connects two dataset nodes based on our computed similarity measure. The second edge type connects a model and a dataset and are the existing model selection scores, e.g., LogME [194], PARC [23]. The third edge type also connects a model and a dataset, and comes from the training history of each model on each dataset, such as the pre-trained performance and fine-tuning performance. So, in the graph instead of having the binary adjacency matrix, the respective scores will be used as the weights of the adjacency matrix. Instead of having a fully connected graph, a pruning threshold will be used to decide the existence of the edges.

Example 4.3.2. If we have two datasets d_1, d_2 and two models m_1, m_2 then we can form the graph with edge sets $E_G = \{(d_1, d_2), (d_1, m_2), (d_2, m_2), (d_2, m_1)\}$. Each of these edges will have a value, as per a weighted adjacency matrix, the value for (d_1, d_2) will be the similarity score $\phi_{1,2}$ between the datasets. The value between edges $\{(d_1, m_1), (d_1, m_2), (d_2, m_2), (d_2, m_1)\}$ will be the training performance of the model on the corresponding dataset. These values can be taken from Example 4.3.1 and drawing the graph using those values.

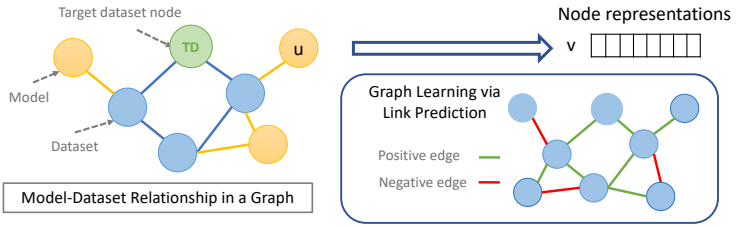


Figure 4.3: Link prediction in the context of model selection

In this work, we are interested in exploiting the intrinsic relationships between models and datasets for model selection problem in a model zoo. Borrowing the concept from data lake management, we represent the model and dataset relationships in a graph and learn the graph structure by performing a link prediction task.

Link prediction. We extend Definition 4.3.1 to $G = (V, E, L)$, by adding the set of labels or representations of each edge, denoted as L. The goal of link prediction is to learn a predictive model that assigns a score to pairs of vertices (u, v) indicating the likelihood of the existence of an edge between them.

In our problem setting, we aim to identify the models that have high performance on the datasets. We can specify the positive edges with models receiving high performance in the training history. Thus edges of models performing well on a dataset have the label of 1 and 0 if the model has poor performance on a dataset, which forms the labels in L.

We formulate the model selection problem through a learned function over the graph, represented by the following formula.

$$\hat{T}_{i,j} = F(f_G(m_i), f_G(d_j))), \tag{4.2}$$

In Equation 4.2, we use the graph learners f_G to learn the set of labels L for the link prediction task on our constructed graph. $f_G(m_i)$ obtains the vertex embeddings of m_i, and $f_G(d_j)$ obtaining the vertex embeddings of d_j. F denotes the prediction model that maps from the model and data representations to the fine-tuning results. The prediction model is trained on the training history.

We reformulate the problem of model selection with a model zoo as a graph link prediction problem. In what follows, we will introduce the information needed to tackle the problem in our proposed graph-learning-based strategy.

4.4. DATA COLLECTION: METADATA AND FEATURES

Extensive research [23, 138, 194] has been conducted to investigate the relationship between the model features and the target dataset labels. Yet, the metadata of models and datasets are often neglected. Though simple and coarse-grained, such metadata are of great value to specify the characteristics of the models and datasets in some sense, and prove to be useful for predicting the fine-tuning performance [110]. Below, we will introduce the main metadata and features considered.

4.4.1. METADATA AS NEW FEATURES

In the following subsections, we present the considered metadata of both models and datasets. All the metadata information is easy to obtain and without any computation requirement.

METADATA OF DATASETS

The metadata of datasets can be indicators of the fine-tuning difficulty. The properties of a dataset can affect a model's performance. For example, a dataset with many classes is more difficult to learn than a dataset with binary classes. We do not exclude the information from the pre-trained model, as in most feature-based model selection strategies. We consider the metadata of both source and target datasets for model selection.

Number of data samples. A small dataset contains less information and is likely easier to learn. In contrast, a large dataset with more diverse features may require a more complex model to learn to obtain good performance.

Number of label classes. A multi-label classification problem is more challenging than binary classification and may require more data samples to learn.

METADATA OF MODELS

The metadata of models reveals their learning capability from a certain perspective. A model with more parameters may capture more generalized features. Models with different architectures may have varying inductive biases for different datasets.

Input shape. More information can be captured with a larger input shape. A higher-resolution image contains more information.

Architecture. The architecture of a model plays an important role in determining how well a model can capture complex patterns in a dataset. A more complex architecture, e.g., ResNet [74], Inception [52], might be more suitable to learn more complex and larger inputs than e.g., LeNet [107].

Pre-trained dataset. The source data quality significantly impacts the learned features and knowledge that a model can capture. A model trained on a large dataset with diverse data may have more generalized ability than one trained on a small and biased dataset.

Model performance. The performance identifies the capability of a model. For example, when two models are trained on the same dataset, the model with higher accuracy indicates that it has better knowledge of the dataset and may be adaptable to new datasets.

Number of parameters. A bigger model with more parameters can capture more generalized features from a large dataset. Compared to an SVM model, a more complex model, e.g., ResNet, can perform better in image classification on ImageNet.

Memory consumption. The memory consumption of a model is correlated to the number of parameters. It is another indicator of the complexity of a model.

This work does not include all the metadata mentioned in Amazon LR [110]. Some metadata included in Amazon LR needs further computation to obtain, e.g., dataset difficulty. The metadata mentioned above are more accessible to obtain. In addition, we include some other features, e.g., models' pre-trained performance compared to Amazon LR. We find that even with the simple metadata, the model selection strategies can make good predictions on the model performance.

4.4.2. DATASET FEATURES

Together with the metadata, we capture the dataset features in the feature collection stage. Similar to feature-based model selection strategies, which acquire features by executing a forward pass over all candidate models on the target dataset, we can capture dataset representations through a comparable approach. By utilizing a reference ML model, referred to as a *probe network*, for inference on datasets as the initial step. We acknowledge that the probe network exhibits varied performance on different datasets, resulting in distinct embeddings within a vector space. We expect these embeddings to unveil the distinctive characteristics of the datasets, and the distance between embeddings captures the semantic similarities between the datasets.

DATASET REPRESENTATIONS

Prior studies, including Task2Vec [3] and Taskonomy [197], focus on learning dataset representations within the realm of transfer learning. We adopt two kinds of methods to extract the embeddings for dataset representations; *Domain Similarity* and *Task2Vec*.

Domain Similarity embeddings. We adopt a similar mechanism to extract features of an image from large pre-trained model as in Domain Similarity [46]. We aggregate all the image representations of the dataset inferred by a probe network as the dataset features. The probe network is usually a large network, such as VGG [165], ResNet [74]. These networks are pre-trained on ImageNet [47] and are considered to be able to capture good generic features from the images and thus serve as reference models to retrieve features. The embedding of a dataset d_k is defined as:

$$\tilde{E}_k = \sum_{j=1}^{|d_k|} g(x_j), \quad x_j \in d_k, \tag{4.3}$$

$g(\cdot)$ represents the features obtained by extracting the feature layers of the reference model. We adopt *ResNet34* pretrained on ImageNet as the reference model.

Task2Vec embeddings. Task2Vec [3] is another method that we implement to obtain node features. Unlike domain similarity, Task2Vec also takes into account the labels of the dataset and learns embeddings for different tasks with pre-trained networks. The main formula to retrieve the Task2Vec embeddings involves computing the diagonal Fisher Information Matrix of the network filter parameters for a given task:

$$\tilde{E}_k = F, \quad F = \mathbb{E}_{x,y\ \tilde{p}(x)p_w(y|x)[\nabla_w log p_w(y|x) \nabla_w log p_w(y|x)^T]} \tag{4.4}$$

, where F is the diagonal Fisher Information Matrix (FIM). The Task2Vec embedding can then be obtained by averaging the FIM for all weights in each filter of the probe network. This results in a fixed-size vector representation for each task that captures its complexity and semantic similarity to other tasks. The norm of this embedding correlates with the complexity of the task, while the distance between embeddings captures semantic similarities between tasks.

DATASET SIMILARITY

A model with good performance on the source task is likely to have good fine-tuning performance when the target task is similar [183]. We use ϕ to represent the dataset similarity. The similarity between datasets is measured by the similarity between dataset representations with Euclidean distance. We expect a higher similarity between semantically

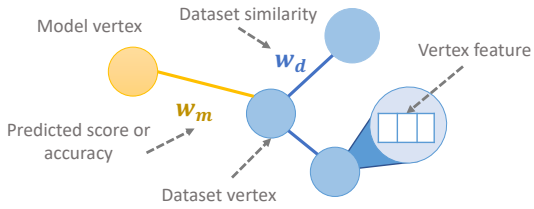


Figure 4.4: Graph properties

similar datasets. For example, a dataset of flowers shall be more similar to a dataset of plants than airplanes.

4.4.3. OTHER FEATURES

Existing works such as Model2Vec [3] and attribution map [168] have investigated to obtain model features for transfer learning. Future work can investigate using model features as an additional type of feature for predicting the model performance.

4.5. GRAPH CONSTRUCTION AND LEARNING

The metadata and dataset features mentioned in Section 4.4 characterize the datasets and models from a high-level perspective. When the metadata information and dataset features are similar, distinguishing between them becomes challenging, leading to difficulties in predicting model performance. In order to obtain more subtle features of models and datasets, we aim to explore the intrinsic relationships between models and datasets. For example, whether a model's proficiency on one dataset implies good performance on a similar dataset, or whether models pre-trained on diverse datasets exhibit distinct performance on a given target dataset.

We introduce a graph-based approach to capture and leverage the relationships between models and datasets. Utilizing graph learners, we seek to exploit the rich relational information embedded in the graph structure. The subsequent section will detail how we construct this graph, tailored for the purpose of model selection with a model zoo. In addition, we introduce the representative graph learning algorithms that capture the information of the constructed graph.

4.5.1. GRAPH CONSTRUCTION

To assign attributes to nodes and edges, it is crucial to identify entities and relationships. In Figure 4.4, we present an overview of the graph structure, where vertices and edges may carry distinct semantic meanings.

VERTICES

A vertex in the constructed graph can be either a model or a dataset. The vertices are connected to each other, embedded with model-dataset relationships or dataset-dataset relationships. Usually, model zoos contain models trained on overlapping publicly available (benchmark) datasets, making the number of models exceed the number of datasets in a model zoo.

VERTEX FEATURES

A vertex can be embedded with features. Some graph learners, e.g., GraphSAGE [73], GAT [181], can capture the vertex features and use them to initiate the learning process. We introduce dataset features earlier in Section 4.4.1. We can embed the dataset features as the features of the dataset vertices.

EDGES AND EDGE ATTRIBUTES

The edges are constructed in three ways: i) edges between datasets indicating the similarity between datasets, ii) model performance on datasets as edges between models and datasets, iii) predicted scores obtained from other feature-based model selection strategies as another type of edges between models and datasets.

Dataset-Dataset (D-D) edge attributes. The construction of D-D edges is achieved by evaluating the similarity of dataset representations. The dataset similarity is denoted as ϕ. The computation encompasses all possible dataset pairs, with the resulting similarity scores employed as edge attributes.

Model-Dataset (M-D) edge attributes. The edges between datasets and models are associated with different meanings. A model can connect to a dataset with training performance or predicted score. For example, if we can obtain the pre-trained performance of $m_{Resnet50}$ on the dataset *cifar100* with an accuracy of, e.g., 95%, the vertices between $m_{Resnet50}$ and *cifar100* has an edge with an attribute of 0.95. We can also embed the fine-tuning results if they are available. In addition, the predicted scores obtained from other model selection strategies can also embed meaningful information between models and datasets.

4.5.2. GRAPH LEARNING

In the context of model selection, we formulate the graph structure to address a link prediction task, evaluating the likelihood of a model exhibiting high performance on the target dataset. The connectivity between the model and dataset vertices is established if the model is anticipated to yield favorable outcomes.

For effective resolution of the link prediction problem, it is imperative to distinguish positive edges from negative ones. In our pursuit of identifying high-performing models, we designate relationships where a model demonstrates good performance (e.g., with accuracy higher than 60%) on the dataset as positive edges, while those with lower accuracy are categorized as negative edges (e.g., lower than 50%).

We employ diverse graph learning algorithms for the acquisition of knowledge from the constructed graph. These algorithms consider a variety of information, e.g., link structure and edge attributes. In essence, graph learning algorithms demonstrate the capability to capture intrinsic knowledge within a graph by assimilating neighborhood information.

RANDOM-WALK-BASED GRAPH LEARNING ALGORITHMS

Graph learning algorithms based on random walks do not incorporate the features of vertices; instead, they focus on learning the graph's link structure. This paper specifically explores Node2Vec [65] and its variant, Node2Vec+ [116].

Node2Vec. Node2Vec [65] introduces a probability model where the random walk has a certain probability, $1/p$, to revisit nodes being traversed. Additionally, it employs an in-out parameter, q, to control the exploration of the global structure. When the return parameter, p, is small, the random walk may become trapped in a loop, focusing on the local structure. Conversely, when the in-out parameter, q, is small, the random walk resembles a depth-first-sampling strategy more closely, capable of preserving the global structure in the embedding space.

Node2Vec+. Node2Vec+ [116] is a variant of Node2Vec. Different from Node2Vec traversing the graph with parameters, p and q, Node2Vec+ takes into account the edge weights. When it constructs walks in the graph, the probability of visiting the next neighbor is associated with the edge weights.

NEURAL-NETWORK-BASED LEARNING METHODS

Different graph neural networks can learn different kinds of information from the graph. All of them capture the edges in the graph. Some also learn from the edge attributes, or node features.

GraphSAGE. GraphSAGE [73] employs a sampling and aggregation method to perform inductive node embedding, utilizing node features such as text attributes, node profiles, and more. The model trains a set of aggregation functions that integrate features from the local neighbors and pass them to the target node, denoted as v_i. Subsequently, the hidden state of the node v_i is updated by:

$$h_i^{(k+1)} = ReLU\left(W^{(k)} h_i^{(k)}, \sum_{n \in N(i)} (ReLU(Q^{(k)} h_n^{(k)}))\right) \tag{4.5}$$

4.5.3. ATTENTION GRAPH EMBEDDING

We also consider another type of graph learning method, using attention mechanisms in the learning process. The attention mechanisms enable graph learning to concentrate on specific parts of a graph that are more relevant to a given task. One advantage of applying attention to graphs is the ability to filter out the noisy components, thereby increasing the signal-to-noise ratio in information processing. In this line of work, we adopt Graph attention networks (GAT) in this paper.

GAT. GAT [181] employs masked self-attentional layers to address the limitations of prior graph convolutional-based methods. The layers aim to compute attention coefficients.

$$\alpha_{ij} = \frac{exp(LeakyReLU(\overrightarrow{a}^T [W\overrightarrow{h_i} || W\overrightarrow{h_j}]))}{\sum_{k \in N_i} exp(LeakyReLU(\overrightarrow{a}^T [W\overrightarrow{h_i} || W\overrightarrow{h_j}]))}, \tag{4.6}$$

W is the weight matrix of the initial linear transformation. The transformed information for each neighbor's feature is then concatenated to derive the new hidden state. This new hidden state undergoes a *LeakyReLU* activation function, a widely utilized rectifier. The attention mechanism described above constitutes a single-layer feed-forward neural network, parameterized by the weight vector mentioned earlier.

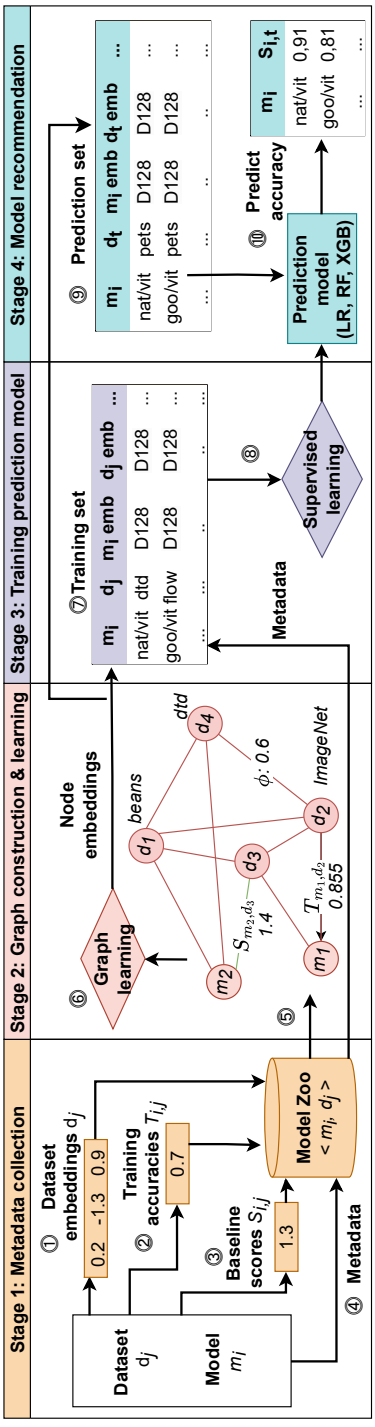


Figure 4.5: An overview of TransferGraph on model selection for fine-tuning, including model zoo construction (stage 1), training (stage 2-3) and model selection (stage 4).

4.6. THE FRAMEWORK OF TRANSFERGRAPH

We propose *TransferGraph*, a framework that performs model selection via a graph learning process. To achieve this, there are a few steps for the graph-based model selection process, as shown in Figure 4.5. The processes are divided into four main steps:

4.6.1. METADATA AND FEATURE COLLECTION

We first collect all the information needed, as described in Section 4.4. Step {①-④} indicate the collection process of different features and metadata used for the subsequent steps. Step ① obtains the dataset representations which can be further applied to compute the similarity between datasets. Step ② encapsulates the training performance of models across different datasets, while step ③ represents the predicted scores of model performance, which can be obtained from existing works, e.g., LogME [194]. Step ④ collects the metadata of models and datasets. All the collected information will be returned to the model zoo and stored as preparatory data for further processes.

4.6.2. GRAPH CONSTRUCTION AND LEARNING

With the collected information, we continue to construct a graph in step ⑤, embedding relationships between models and datasets, and other attributes. The graph construction details can refer to Section 4.5.

We expect that graph learning can help capture the intrinsic structure of the graph. If we consume all the information and construct a graph, the graph would be fully connected, hence no significance information can be learned. The mechanism of graph learning is to map the "connected" nodes close in the vector space, while keep the "disconnected" nodes further away. By saying "connected", the nodes can be reached out within the same path by traversing the graph.

For this problem, our goal is to identify the models that can achieve good performance on the downstream tasks, i.e., target dataset. We set a fine-tune accuracy threshold as {0.6} and {0.5} for transferability score. For model-dataset pair with accuracy lower than 0.6 and transferability score lower than 0.5, we will prune the edges from the graph. We use the remaining graph for representation learning. With the edges being pruned, we obtain a graph for training the graph learners. The properties are shown in Table 4.2.[4]

We further use one of the graph learners, e.g., Node2Vec, presented in Section 4.5 to capture the information in the graph, e.g., link structure or vertex features, as in step ⑥. The graph learner is trained for a link prediction task. With the trained graph learner, we can extract the embeddings for each node.

4.6.3. TRAINING PREDICTION MODEL FOR PERFORMANCE PREDICTION

As a learning-based strategy, we learn from the training history to predict the model performance on an unseen dataset as a regression task. In step ⑦, we construct a training set for the supervised learning as a regression task. The training features are constructed by metadata of models and datasets, as well as graph features. The label is the training performance of a model on a dataset, which we learn to predict. For each model and dataset pair, if there exists training history, we identify the metadata of the model and

[4] * indicates that the number would change if the target dataset is different.

Graph property	Value
graph type	homogenous
Threshold on transferability score for edge pruning	0.5
Threshold on accuracy for edge pruning	0.6
Threshold of negative edge identification on accuracy	0.5
number of nodes	265
average node degree*	20.1
number of dataset-dataset edge*	5256
number of model-dataset edge with accuracy weight*	1753
number of model-datset edge with transferability weight*	916

Table 4.2: Statistics of the graph properties

dataset. Through graph learning, we obtain the learned representations of the corresponding models and datasets. The training set can be represented as tabular data, and we can learn a prediction model, e.g., linear regression, random forest, as shown in step ⑧.

4.6.4. MODEL RECOMMENDATION FOR FINE-TUNING

We construct a prediction set ⑨ similarly to the training set construction. Specifically, the dataset embeddings only belong to the target dataset d_t where we will fine-tune models. The metadata of the dataset also adjust with the target dataset. We include all the models, since we would like to predict performance of the models in the model zoo.

Given the trained prediction model, we obtain a score for each model and target dataset pair. We apply these predicted scores as an indicator to rank and select models for fine-tuning.

Linear regression. One of the prediction model we use is linear regression. We use the linear regression model to learn various features, e.g., meta features and graph features. Linear regression fits a straight line or surface that minimizes the discrepancies between predicted and actual output values.

Random forest. Random forest is also a highly adopted model due to its simplicity and explainability. We set the number of trees as 100, max depth as 5.

XGBoost. XGBoost (eXtreme Gradient Boosting) is one of the ensemble learning methods and is particularly effective in structured and tabular data scenarios [37]. XGBoost is an ensemble of decision trees and minimizes the objective function with gradient descent. We set the number of trees as 500, and maximum depth as 5.

4.7. EVALUATION

4.7.1. EXPERIMENT SETUPS

Datasets. We collected 12 public image datasets, which are often used for image classification tasks and are listed in Table 4.3. Among them, 11 datasets are from the VTAB-1k benchmark suite [198]. Additionally, we use another common public image dataset,

Dataset	caltech101 [53]	cifar100 [103]	diabetic_ret [85]	dtd [40]	eurosat [75]	flowers [139]
Samples	3060	50000	35126	1880	27000	1020
Classes	101	100	5	47	10	10
Dataset	kitti [60]	pets[200]	smallnorb_elevation[106]	stanfordcars [100]	svhn [136]	
Samples	6347	3680	24300	8144	73257	
Classes	4	37	18	196	10	

Table 4.3: The target tasks used for evaluation

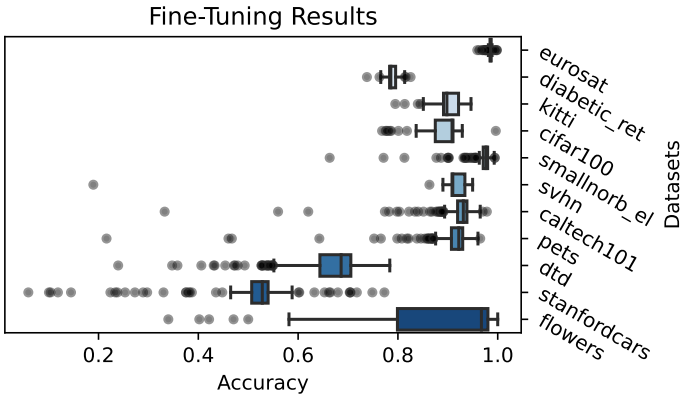


Figure 4.6: Fine-tuning performance of models over different datasets sorted by standard deviation

Stanford Cars [100].

Models. We include 185 heterogeneous models, with different architectures, such as ViT [52], Swin-Transformer [118] and ConvNeXT[119], and pre-trained on diverse datasets. We use public image classification models from HuggingFace[5]. Different from the setup in the previous works [110, 194], we do not constrain the model to be trained on a certain dataset, e.g., ImageNet, which is the case in [110].

Ground truth. A pre-trained deep learning model consists of two components: a *feature extractor* and a *classifier*. During fine-tuning, the model is initiated with the pre-trained weights, coupled with a classifier layer that is randomly initialized. Subsequently, this new model is retrained on the target dataset. To determine the actual fine-tuning accuracy, we fine-tune all models on our target datasets. For optimization, we employ *stochastic gradient descent*. We use a cyclical learning rate scheduler, which avoids the need for a redundancy test or expensive hyperparameter tuning [167]. Using this strategy in transfer learning has been validated as effective in [114]. We optimize for 30 epochs, using a momentum of 0.9. We present the fine-tuning performance of models across different datasets, as in Figure 4.6. Notably, in certain datasets, the performance variance is small. For example, in the case of *eurosat*, where the standard deviation is only 0.005, model selection is not necessary. In the following experiments, we only report results on datasets where model performance exhibit variation. The datasets are

[5]https://huggingface.co/models

ordered by the standard deviation of the performance.

Baselines. We compare our work with the two baselines below:

- LR (Amazon LR [110]) is the state-of-the-art approach for model selection for model zoos. It exploits the meta-features of models and datasets, and uses these features to train a linear regression model to predict the fine-tuned accuracy. The metadata of datasets includes data size, number of classes, etc. The metadata of models consists of the model architecture, input size, pre-trained domain, etc.

- LogME [194] is one of the most representative works that measure the transferability of a model to a target dataset. Transferability is a score that assesses a model's transfer learning performance to a new task (see Section 4.8 for explanations). The mechanism of LogME is to estimate the maximum value of label evidence $p(y|R)$ (R is the representations extracted by a model) given features extracted by pre-trained models.

Evaluation. To validate the effectiveness of our approach, we adopt a "leave-one-out" (LOO) mechanism for evaluation. This is a standard setting in related works of model selection, such as [110]. At each time, we learn from the training history of models trained on the existing datasets while excluding the target dataset. When constructing the graph in our proposed method, we remove all the edges of models connected to the target dataset node, i.e., the target dataset while maintaining the edges between datasets. Then, with the learned GNN, we identify the node representations of models and the target dataset, and use them as the graph features.

For baseline comparison, we apply a evaluation metric: **Pearson correlation**. Existing methods for model selection mostly predict a score, i.e., model selection score, for each pair of a model and target dataset. The Pearson correlation measures the correlation between the predicted scores and the ground-truth results, i.e., accuracy. A model selection method is considered better, with a higher correlation between its predicted score and the ground truth.

4.7.2. EVALUATION ON HETEROGENEOUS MODEL ZOO

We construct a model zoo with 185 heterogeneous pre-trained models. These models vary in terms of various aspects, e.g., pre-trained dataset, architectures, and various other metadata features. The downstream tasks are from the VTAB benchmark, where the domains include animals, 3D objects, plants, remote sensing, etc.

SUMMARY OF OUR PROPOSED GRAPH-LEARNING-BASED STRATEGY

There are a few design choices with our proposed methods.

Prediction model. We include three prediction models: linear regression (`LR`), random forest (`RF`), XGBoost (`XGB`).

Graph learners. The graph learning algorithms include GraphSAGE [73], GAT [181], Node2Vec [65] and Node2Vec+ [116]. In particular, `N2V(+)` is short for Node2Vec(+) in the plots.

Additional features for supervised learning. Along with the graph features, we also include additional features as inputs for supervised learning when predicting the training results. We take into account features including all the metadata of models and datasets,

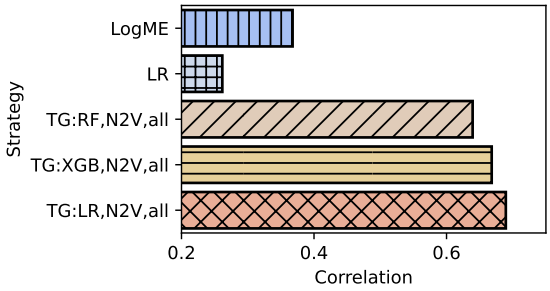


Figure 4.7: Comparison of different model selection strategies, i.e., feature-based, learning-based, and our proposed graph-learning-based.

as in Section 4.4.1. In addition, we include the distance between the source dataset and target as another type of features for supervised learning.

The variants of our proposed strategies begin with `TG`. For example, `TG:LR,N2V,all` indicates that we use a linear regression model to learn from *all* supervised features along with the graph features obtained by Node2Vec.

RESULTS

In Figure 4.7, we report the average Pearson correlation between the predicted score and the fine-tuning results over 8 downstream datasets. We compare our graph-learning-based strategy with other strategies mentioned in Section 4.7.1, i.e., `LogME`, and `LR`. `LogME` is feature-based and does not take into account of meta features nor the source dataset representations. The rest are all learning-based model selection strategies. They learn from the training history and predict the model performance on a target dataset. We present graph-featured-based strategies, the names beginning with `TG`. Each strategy differs in terms of the prediction model being used.

Figure 4.7 shows that our proposed graph-feature-based strategies significantly improve the model selection performance compared to baselines `LogME` and `LR`. We use three kinds of prediction models, i.e., linear regression model `LR`, random forest model `RF`, and XGBoost `XGB` model. Compared only using (meta) features (`LR`), the graph features can improve the capability of the prediction model and achieve a higher correlation between the predicted scores and the fine-tuning accuracy. It shows that the intrinsic relationships between models and datasets revealed via graph learning are important to predict the performance of models on a new dataset. In addition, we notice that `LogME`, without any information on the training history, can outperform a learning-based strategy, i.e., `LR`. This, on the hand, proves the significance of the graph-based features.

4.7.3. ABLATION STUDY

In this experiment, we conduct an ablation study where we investigate the effect of different features, i.e., i) with only (meta) features, ii) with only graph-based features, and iii) with (meta) features, dataset distance and graph-features. We use the same prediction model, `LR` to train on the features.

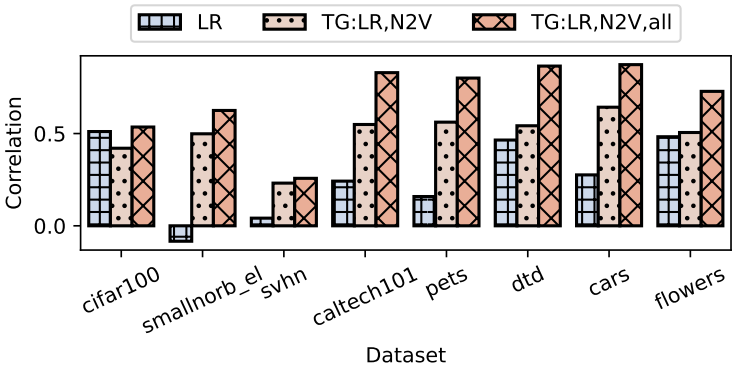


Figure 4.8: Ablation study when including various features.

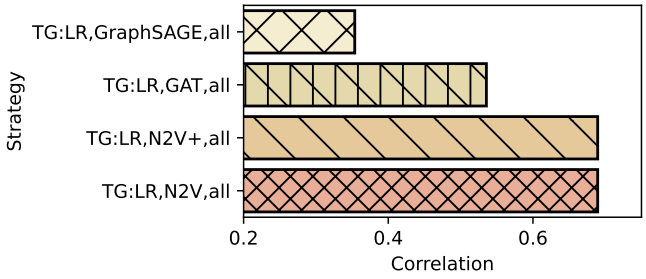


Figure 4.9: Performance of model selection strategies using different graph learners

As seen from Figure 4.8, in most cases except *cifar100*, including more features results in better performance. When only using graph features, learned from Node2Vec, we can already achieve a higher correlation than the baseline LR, which only includes the basic metadata. We also note that when LR fails to learn, e.g., *smallnorb_evaluation*, the strategies using the graph features can successfully predict the model performance. Among all, the most effective strategy is to include all the features, (meta) features, dataset distance and graph features.

4.7.4. EFFECT OF GRAPH LEARNING METHODS

In the previous study, we investigate the effect of different features. We move forward to verify the effectiveness of different graph learning methods. In the following, we compare the average performance using different graph learning algorithms to extract the graph features. All the strategies included in this experiment use a LR model to learn the graph features, as well as other (meta) features and dataset distance. The graph features are extracted by four graph learners: i) *GraphSAGE* [73], ii) *GAT* [181], iii) *Node2Vec+* [116], and iv) *Node2Vec* [65].

Figure 4.9 presents the correlation results when using different graph features ob-

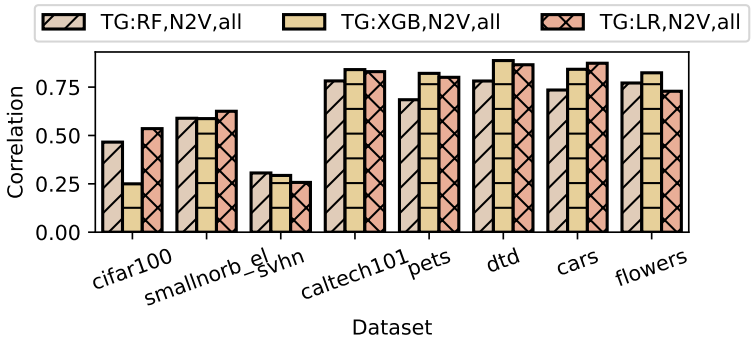


Figure 4.10: Ablation study when including various features.

tained by various graph learners. The strategies learning features from the Node2Vec series, i.e., *Node2Vec* and *Node2Vec+* outperform the ones using *GraphSAGE* and *GAT*. Each graph learners consume different graph properties. *Node2Vec* only learns the link structure. Besides the link structure, *Node2Vec+* also takes into account the edge attributes in the graph. While *GraphSAGE* and *GAT* obtain not only the link structure, edge attributes, but also the node features, each updating the node representations in different mechanisms.

The graph neural networks usually work well on large graph, e.g., *Citation data* containing 302,424 nodes, and *Reddit* with 232,965 edges [73]. *GraphSAGE* and *GAT* do not perform well in our context because the constructed graph is relatively small compared to those graph datasets. The graph used in this paper has only 265 nodes and thousands of edges. The computation overhead of obtaining such large-scale graph dataset is extremely expensive. While the *Node2Vec* series of graph learners can perform well on various size of graph dataset. It is noted that we do not explore the hyperparameter space of these graph learners, e.g., walk length, number of neighbors sampled by each node, window size, etc. Complementary work can identify the best hyperparameter candidate for each graph learners, and also investigate which graph learner to apply given different setting scenarios, e.g., graph size, link structure and node/edge features.

4.7.5. EFFECT AND CAPABILITY OF PREDICTION MODEL

The prediction models are used to learn features and predict the fine-tuning scores. In this experiment, we investigate the effect when applying different prediction models. In Figure 4.10, we present the correlation performance when applying different prediction models. We observe that there is no dominant prediction model that can obtain the best results among all the datasets, and the performance of the prediction models on a dataset is similar, except *cifar100*, which indicates that the feature selection is more important than prediction model selection. Yet we do not fully tune the prediction models on each dataset. Further study can be done to identify the most appropriate prediction model based on varying dataset characteristics.

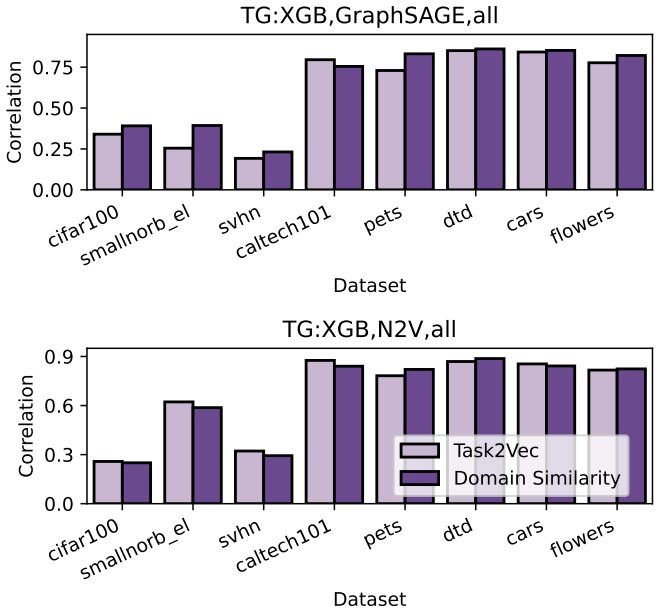


Figure 4.11: Correlation results affected by different dataset representations.

4.7.6. EFFECT OF DATASET REPRESENTATIONS

We extract dataset representations using proposed methods from [3]. The representations are used in two ways: i) to compute the distance between datasets, and ii) used as initial node features for datasets. In this experiment, we aim to investigate the effect of different dataset representations. The dimension of a Task2vec embedding is 13842, and the one of a domain-similarity embedding is 1024, depending on the extraction layer of the reference model.

In Figure 4.11, we present the results of two of our proposed strategies, i.e., `TG:XGB, GraphSAGE,all` and `TG:XGB,N2V+,all`, using GraphSAGE and Node2Vec+ as graph learner respectively. We observe only slight differences in the performance on most of the datasets between using *Task2Vec* representations and the ones of *Domain Similarity*. For Nove2Vec+, the embeddings are only used to compute the dataset distance. The small differences in the dataset distance do not affect the final results. However, in the case of using GraphSAGE, where the representations are used for both similarity computation and the vertex features. We observe that in most cases, *Task2Vec* representations do not show advantages when using GraphSAGE. One reason is that the *Task2Vec* embeddings have really high dimension, while the graph in general is not big. We suggest that future work can delve into this and identify better representations for a graph learner to learn for the model selection problem.

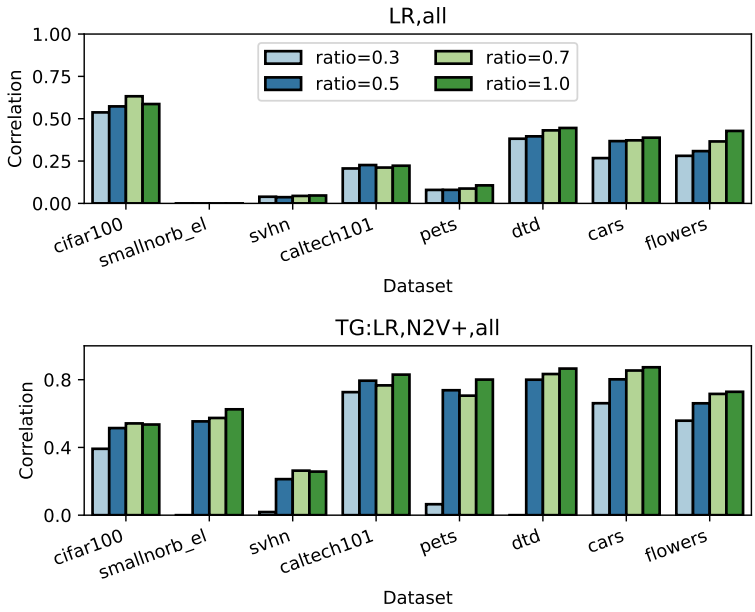


Figure 4.12: Effect of inputs ratio

4.7.7. EFFECT OF INPUT RATIO

We investigate the effect of the input size of the training history on the performance. The entire training history ($ratio = 1.0$) include the training results of all the model and dataset pairs, excluding the target dataset and model pairs. We experiment on training with different ratios of the training history: {0.3, 0.5, 0.7, 1.0}. The strategy training would be much more efficient with lower input ratio, because the feature collection can be expensive though it can be performed offline. This experiment aims to answer the question: how the amount of information affects the prediction performance.

We compare two main categories, i.e., a strategy training without graph features (`LR,all`) and another strategy training with graph features (`TG:LR,N2V+,all`). As in Figure 4.12, the performance of both strategy can be affected by the input ratio. `LR,all` is more robust even when limited training history is used to train the strategy. While graph-feature-based strategy is more sensitive to the input ratio, especially with low input ratio. When we set training history as $ratio$=0.3, `TG:LR,N2V+,all` fails to predict the performance. The reason is that with a small input ratio, the constructed graph may have a large number of disconnected components. The graph learner fails to capture the global information by traversing the graph.

4.7.8. DISCUSSION

Through comprehensive experiments, we have shown the efficacy of graph-based features in addressing the model selection problem with a model zoo. Our most competitive model selection strategy incorporates both graph-based features and additional meta-

data of models and datasets. It is noted that, in this chapter, we use image classification task and visual models as illustrative scenarios. However, our proposed model selection strategy can be applied to diverse cases on various modalities. For example, it can seamlessly extend to managing a model zoo with large language models for textual datasets. Below we discuss the limitations and directions that can be investigated in future research.

Graph construction. We incorporate different information in a graph, e.g., dataset distance, model performance, dataset representations, etc. Yet, we do not discuss the contribution and importance of each type of features embedded in a graph. We apply a simple threshold-based edge pruning process to maintain the graph structure. Future work can investigate a more advanced graph construction method and make it adapt to the capability of different graph learning models.

Efficiency. The collection of the relevant features for the prerequisite works are not trivial, though this process can be achieved off-line. Future work can investigate the most impactful features and make the preparation process more efficient.

Graph learner selection. We investigate four types of graph learner to obtain graph features. In the graph community, the performance of the graph learner may depend on the graph properties. Further work can pursue to identify good candidate of graph learner (with tuned hyper-parameters) for the graph generated from each specific model zoo.

4.8. RELATED WORK

4.8.1. TRANSFER LEARNING

Traditional machine learning techniques have seen significant progress in various knowledge engineering areas such as classification, regression, clustering and data mining. Despite these advancements, real-world applications frequently encounter limitations. Unfortunately, in many scenarios, obtaining sufficient and representative training data can be a costly and time-consuming effort. Transfer learning has been very successful in combatting these problems, especially in the domain of deep learning, where the *data dependence* is even greater [172].

The process of transfer learning typically begins with selecting an *upstream* or *pre-trained* model from a repository containing models trained on different source datasets and architectures. Subsequently, one or multiple selected models are then fine-tuned using the users' target dataset, and the user can select the fine-tuned model with the best characteristics for their downstream task. There are various available fine-tuning strategies identified by [172]. We adopt the most popular *network-based deep transfer learning* in this work. Network-based deep transfer learning refers to reusing the partial network that pre-trained in the source domain and retraining the deep neural network which used in target domain.

4.8.2. GRAPH LEARNING

Graph learning broadly refers to machine learning on data structured as a graph. It is gaining more and more attention, as many complex real-world data can be expressed as graphs. Graph learning can be separated into four different methods [189]: 1) *graph signal processing*, 2) *matrix factorization*, 3) *random walk* and 4) *neural network*. We

focus on the latter two methods, as those are mainly used in graph learning-based recommender systems [183].

RANDOM-WALK-BASED GRAPH LEARNING ALGORITHMS

These types of algorithms sample random walks by traversing the graph. Given a walk length, i.e., number of steps, a random vertex is selected as the starting point and a neighbor vertex would be selected with probability as the next step in the walk. These walks indicate the context of connected vertices. The randomness of walks gives the ability to explore the graph and capture both the global and the local structural information by walking through neighboring vertices. After the walks are built, probability models, such as skip-gram [128], can be applied to learn the representations. The mechanism of the random-walk-based graph learning is aiming to make the representations of connected nodes in the vector space closer to each other while disconnected ones further away. In such a way, the representations capture the intrinsic structure of the graph.

NEURAL-NETWORK-BASED LEARNING METHODS

This line of works were inspired by the success of neural network models, RNNs and CNNs. Graph learning methods using RNNs resemble walks sampled from a graph as words, and use natural language processing models to learn representation of vectors. Another family of neural-network-based methods adopt CNN models. The input can be walks sampled from a graph or the entire graph itself. In this work, we only discuss CNN-based learning methods in this category. Representative works include GraphSAGE [73], GCN [94].

4.9. CONCLUSION

We pioneer the exploration of a graph-learning-based model selection strategy within the model zoo framework and introduce a comprehensive framework to address the intricate model selection problem. Predicting model performance proves to be challenging, given no dominant model excels across all datasets. Extensive experiments have shown that effectiveness of leveraging the intrinsic relationships between models and datasets along with metadata of models and datasets for predicting the model performance. The most competitive variant of our model selection strategy gains 21.8% increase in measuring the correlation of the predicted model performance and the fine-tuning results. Furthermore, the graph-learning-based model selection strategy can continuously be improved with more metadata and training history in the model zoo.

5

CONCLUSION

This dissertation discusses the significant roles that metadata could play in the realm of machine learning (ML). Through an in-depth examination of metadata management and its critical influence on enhancing ML workflows, this work shows the benefits that structured and comprehensive metadata imparts in optimizing the processes of model training, inference, and fine-tuning. Our analysis of the literature and of existing solutions for models management reveals a notable deficiency in the provision of exhaustive metadata within the existing public model zoos and ML systems. This shortfall limits the utility and effectiveness of the platforms for practitioners, particularly in the realms of model selection and the various ML applications aiming for efficiency and effectiveness.

This thesis addresses questions concerning the identification, organisation, and exploitation of metadata in ML workflows. We show that a meticulously organized and queryable metadata repository not only augments the manageability and accessibility of ML models but also equips practitioners with the necessary tools to make informed, data-driven decisions for model selection and model reuse. In this conclusion chapter, we revisit the research questions addressed in this thesis and will further discuss the implications for future work.

5.1. ANSWERS TO RESEARCH QUESTIONS

The thesis tackled the following main research question: ***How to represent and utilize metadata within ML model zoos with the aim of enhancing the processes of model training, inference, and fine-tuning?***

We approached the question by exploring two main directions: the *design* of a metadata representation for ML models (and related resources), and the *use* of such metadata. We tackled, in particular, the case of model selection, for both model inference and fine-tuning. Given the context of a model zoo, where thousands of models are available, selecting models for model inference and fine-tuning is challenging given that models are diverse and with heterogeneous properties. Yet, good model selection can lead to higher effectiveness and efficiency.

The work in this thesis is organised around the following three research questions.

Q1: How to represent metadata in a model zoo in a structured and queryable fashion?

In the first research question, we identify a significant gap in the domain of machine learning (ML) model repositories (model zoos), namely the lack of rich, structured, machine-readable, and therefore queryable metadata describing ML models and related artifacts. We therefore proposed a novel approach to metadata representation. Our representation has been used as a representation backbone for *Macaroni*, an example of a model zoo offering advanced functionalities enabled by our metadata model. In essence, the metamodel paves the way for a more dynamic, accessible, and efficient use of model zoos. By enhancing the interpretability, reproducibility, and overall utility of these repositories, we contribute to the broader goal of democratizing ML practices and fostering innovation in this rapidly evolving field.

With the metadata being represented, our research pivoted to the application of this metadata in enhancing various ML workflows. The first applications is to exploit the metadata for accelerating a ML inference/training pipeline in the context of data distributed in different data silos. Additionally, we explored the use of metadata in optimizing ML inference queries. Regarding these two distinct applications, our findings reveal that the judicious use of metadata can significantly improve both the effectiveness and efficiency of these processes. This enhancement is contingent upon the proper utilization of metadata, underscoring its potential as a critical resource in the advancement of ML workflows.

Q2: How can the metadata be leveraged to enhance the efficiency and effectiveness of ML inference queries?

We continue to exploit metadata for ML inference queries, where predicates in the queries are ML models. In this scenario, we consider user-imposed constraints on multiple aspects, such as execution cost and accuracy performance. We formulate the ML inference query optimization problem as a Mixed Integer Programming (MIP) and the subsequent development of a MIP-based optimizer. This optimizer jointly optimizes model assignment and predicate ordering, and leverages the selectivity of ML-model-based predicates to determine their execution order. In addition, we explore the scenario where multiple objectives are considered and constrained. Similarly, we formulate the multi-objective ML inference query optimization problem as Multiple-objective Mixed Integer Programming (MOMIP) that optimizes for accuracy, execution cost, and memory footprint.

The efficacy of our proposed optimizer is demonstrated through extensive experiments. Our proposed optimizer outperforms the other baselines and can achieve higher query accuracy or shorter execution time, especially given very diverse model zoos with different execution time and accuracy trade-offs. Additionally, our research sheds light on the complexity of this optimization problem. We observe that the time required for optimization escalates exponentially with the increase in the number of predicates within queries.

These findings contribute to the advancement of ML pipeline engineering, especially in scenarios involving the processing of complex ML inference queries over varied datasets and unstructured content streams, all while adhering to specific constraints.

Our approach not only tackles the intricate task of constructing optimal query plans within these parameters but also introduces a novel problem setting in the realm of optimizing ML inference queries. The insights and solutions presented herein have the potential to reshape approaches to query optimization in machine learning contexts.

Q3: How can metadata, along with other representations be employed to predict the performance (i.e., accuracy) of models without undergoing the fine-tuning process?

Finally, we investigate the problem of model selection for fine-tuning with a model zoo. Predicting model performance proves to be challenging, given no dominant model excels across all datasets. We explore the use of a graph-learning-based model selection strategy within the model zoo framework and introduce a comprehensive framework to address the model selection problem. We propose a novel framework, *TransferGraph*, that transforms the model selection problem into a graph learning challenge, representing a departure from traditional methods that rely solely on basic model and dataset information, or similarity between model features and target data features.

With our experimental analysis, we find the efficacy of exploiting the intrinsic relationships between models and datasets in predicting model performance. The most effective strategy identified through our research is when including both graph-based features and supplementary metadata of models and datasets. This variant of our model selection strategy achieves a 21.8% improvement in correlating the predicted performance of models with their actual fine-tuning outcomes. Moreover, this graph-learning-based approach to model selection offers the potential for continuous enhancement through the integration of additional metadata and training history available within the model zoo.

5.2. LIMITATIONS AND RESEARCH OPPORTUNITIES

After the summary of our works above and the analysis of our findings, we conclude this thesis by discussing the limitations of our research for future research efforts. We divide the discussion into three sessions, corresponding to the three research questions, regarding metadata representation, ML inference query optimization, and model selection for fine-tuning respectively.

5.2.1. METAMODEL AND THE MAINTENANCE OF METADATA

Our work makes a significant contribution by proposing a metamodel that encapsulates metadata of various artifacts in a more structured and systematic way. Below, we outline several limitations of our current work and offer insights into potential enhancements.

Refinement of the Metamodel. In the context of evolving data management paradigms, particularly with the increasing prominence of array databases and various data embeddings or representations, it becomes imperative to contemplate the inclusion of multivariate metadata types. These types would not only encompass the characteristics of models or datasets but also extend to their respective representations. A critical area for future enhancement involves broadening the scope to identify a more diverse range of interrelationships. This expansion should consider relationships not only between datasets but also among models themselves, and crucially, the interplay between datasets

and models. Moreover, an overlooked aspect in the current framework is the provenance of datasets and models. Future iterations of the metamodel should integrate mechanisms that consider this provenance, potentially leveraging existing tools to support an array of functionalities.

Acquisition and crawling the metadata from the Web utilizing LLMs. Our current methodology for acquiring metadata, as implemented in *Macaroni*, predominantly relies on the extraction of publicly accessible and straightforward information from the web, such as Markdown files and APIs. Nevertheless, this approach often results in a scarcity of metadata, primarily due to the frequent omission of information by model providers. The scarcity of metadata complicates its application across various platforms. Hence, there is a pressing call for practitioners to meticulously document extensive metadata related to their developed models, configurations, and dataset specifics, and develop tools to manage the metadata. This situation underscores the need for more sophisticated metadata acquisition techniques. Recent advancements in large language models (LLMs) present a viable path for gleaning refined and essential information from descriptive texts.

5

5.2.2. OPTIMIZING ML INFERENCE QUERIES

In this study, we tackle the challenge of optimizing ML inference queries, focusing on achieving high accuracy within specified execution time constraints, and vice versa. Overall, our research contributes to the advancement of query optimization techniques in ML by proposing and evaluating several optimization strategies. Below, we highlight a few limitations and propose potential solution for future investigations.

Problem Complexity and Computational Challenges. Through our empirical analysis, we could appreciate the intrinsic computational complexity of the problem. Particularly, it was noted that as the number of predicates in an inference query increases to thresholds such as 32 or 64, the time required to generate an optimized plan increases substantially. As a future avenue of research, there lies an opportunity to explore and develop approximate mechanisms. Such mechanisms could potentially offer more computationally feasible solutions while maintaining an acceptable level of accuracy and efficiency, especially in scenarios involving a high number of predicates.

Adaptation to Dynamic Environments. In the current scope of our work, the generation of an optimized plan leverages the dataset distribution and predicate statistics. This approach, while effective, assumes a relatively static environment in terms of data distribution and predicate characteristics. However, real-world applications often involve dynamic environments where data distributions and predicate statistics can change over time. Future research should, therefore, focus on enhancing the adaptability of the optimization process to dynamically evolving environments. This could involve the development of mechanisms that can update or modify the optimization plan in response to changes in data distribution or predicate statistics, ensuring continued relevance and efficacy of the plan.

Correlation between predicates. Another aspect that warrants further investigation is the correlation between query predicates. In our current methodology, predicates are considered independently, without an in-depth analysis of potential correlations between them. While existing research [192] has shed light on the impacts of correlated

predicates, these studies primarily pertain to simple query types, such as conjunctive queries, and do not take various constraints into account. Understanding and integrating these correlations into the optimization process could lead to more sophisticated and effective optimization strategies. Future studies could focus on developing models or algorithms that identify and utilize these correlations, potentially improving the optimization outcomes. This would not only enhance the accuracy of the optimization plan but also could uncover deeper insights into the underlying structure of the problem.

5.2.3. MODEL SELECTION FOR FINE-TUNING

Our work addresses the problem of model selection for fine-tuning, an area previously fraught with challenges due to the fact that no model dominant across diverse datasets. The effectiveness of leveraging intrinsic relationships between models and datasets in predicting model performance marks a significance stride in ML practices. There are a few aspects could benefit from further improvement.

Computation overhead in preparation step. Introducing a new dataset into the system requires certain computational resource. It needs gathering relevant information about the dataset, including dataset representations and its transferability scores to other models. Although our findings indicate that partial information (such as transferability scores for 50% of the models) can also achieve good performance, this process still incurs notable computational overhead. Future research should aim to streamline this preparatory step, making it more efficient, and identify the most critical set of information required for effective model performance prediction.

Graph Construction Methodology. In our approach, we integrate various types of information into a graph, such as dataset distance, model performance, and dataset representations. However, the specific contribution and significance of each feature type within the graph have not been thoroughly examined. Our current method employs a basic threshold-based edge pruning process to maintain the graph's structure. Future investigations could explore more sophisticated graph construction methods, tailoring them to the unique capabilities of different graph learning models.

Optimization of Graph Learner Selection. In our study, we explored four different types of graph learners to extract graph features. However, within the graph learning community, it is recognized that the performance of a graph learner might vary depending on the specific properties of the graph. Therefore, subsequent research should aim to determine the most suitable graph learner, complete with optimally tuned hyper-parameters, for the unique graphs generated from each specific model zoo. This pursuit will not only refine the model selection process but also contribute to a deeper understanding of the interplay between graph properties and learner performance.

5.3. IMPLICATION AND FUTURE WORK

After discussing the limitations and direct research opportunities, we continue by discussing the implication of the thesis and needs for research extending beyond the scope of this thesis.

More metadata in ML. In our research, we have delineated a structured approach to

representing metadata. The advancements in metadata representation that we propose facilitate a more nuanced analysis and exploitation of the data inherent within model zoos.

Metadata can also improve model interpretability, reproducibility, and governance (ensuring ML practices compliant with laws like GDPR [182]). Metadata offers a deeper understanding and increased interpretability of ML models by providing detailed insights into model architectures, training algorithms, and hyper-parameters. Such transparency is especially critical in highly regulated industries where interpretability is as important as performance.

Furthermore, metadata serves as a critical informant to automated systems, conveying the performance attributes of various models. Metadata allows for the autonomous selection of the most fitting models for particular tasks, thereby reducing the need for manual selection processes. This thesis presents two distinct applications of metadata usage in the realm of model selection, pertinent to both ML inference and fine-tuning tasks. Beyond that, metadata can also play a pivotal role in directing the data preprocessing steps and automating the training process, thereby enhancing the quality and specificity of data for particular ML tasks. Our investigation predominantly focuses on unstructured data, primarily images. However, in practical scenarios, data is heterogenous, encompassing both unstructured forms, such as texts, images, and videos, and structured types like tabular data and traditional databases. It is noteworthy that structured data still predominates in corporate usage. We acknowledge that data preprocessing profoundly influences model performance [6, 38, 130]. An astute application of metadata has the potential to automate the training process and augment the efficiency of ML workflows significantly.

The scope of opportunities that metadata presents to improve ML practices is vast. Beyond the above-mentioned benefits, it also ranges from enhancing the discoverability of data and models to bolstering collaborative efforts within the ML community. Given the broad spectrum of opportunities that metadata presents, there is a pressing need for further investigation and research in this domain. Further research could unlock new paradigms in ML practices, leading to more automated, transparent, and collaborative processes.

The use of ML models in a broader sense. Training of new ML models is increasingly easier, mostly thanks to advanced tools and systems. However, the practical usage of these extremely large amount of models remains an underexplored area. While the capability exists to train and learn new models as required, practical constraints such as the scarcity of training data and the time-intensive nature of retraining models frequently arise. Therefore, it could be advised to leverage the existing collections of pre-trained or fine-tuned models.

The advent of Bert-series language models has spurred the popularity of Large Language Models (LLMs) and foundation models (FMs), with the field experiencing a surge following the launch of ChatGPT in November 2022. Despite the capability of FMs and LLMs to perform a multitude of tasks, practitioners are faced with the task of meticulously selecting models that align with their specific goals or constraints (e.g. a consultancy may necessitate a model ingrained with Dutch legal domain knowledge). For ex-

ample, Amazon Bedrock[1] provides dozens of foundation models from industry providers. Users need to choose the model that is best suited to achieving their unique goals. Currently, we find ourselves in an era where no single model outperform across all dimensions—efficiency, effectiveness, and domain expertise. It is anticipated that this scenario will persist into the foreseeable future, necessitating the careful selection of models tailored to specific requirements.

Research on multi-modality has been advancing [17, 137, 190], yet the integration of these modalities often remains confined to pairs, such as visual with text, audio with text, or audio with visual. Our understanding of the world is multifaceted, encompassing sight, sound, movement, touch, and even smell. To date, we have not succeeded in developing a foundational model that encapsulates all these modalities. In constructing applications that necessitate a synthesis of multiple modalities, such as robotics, a calculated optimization of several multimodal FMs may be required. On the other hand, from an efficiency standpoint, both cost and time are critical considerations. Efforts are underway to condense or distill large models to enhance efficiency. In parallel, optimizing the use of these more compact yet efficient models remains a priority for the field.

The current landscape of ML model utilization emphasizes the need for strategic deployment rather than just model training. With the abundance of diverse, pre-trained models at our disposal, the challenge shifts to effectively selecting and employing these resources to meet specific objectives and constraints.

DB4ML and ML4DB. The convergence of databases and ML research presents an ongoing and dynamic area of study. In general, there are two distinct yet interconnected directions [204]: *DB4ML* (Database for Machine Learning) and *ML4DB* (Machine Learning for Database). DB4ML focuses on the application of database techniques to optimize ML processes, and building systems or tools for ML applications, e.g., SystemDS [22], DuckDB [149]. While ML4DB harnesses learned-based methodologies address traditional database challenges, such as query optimization and cardinality estimation. This thesis mainly investigate within the realm of DB4ML, showcasing two applications, i.e., model selection for model inference and model fine-tuning. Beyond that, there is a multitude of research directions aimed at enhancing the efficiency, scalability, and functionality of ML tasks when applied data management techniques, e.g., training and inference acceleration using database techniques [169], data cleaning/discovery/labeling for ML tasks [55, 102, 162]. Unlike traditional query optimization, optimizing ML operators demands the consideration of multiple objectives, such as accuracy, efficiency, cost, and hardware compatibility (CPUs, TPUs, or GPUs).

Despite advancements, there remains a notable divide between ML tools and data management systems like databases. Practitioners often find themselves transferring data between platforms for model training and application, a process that can be inefficient and cumbersome. While improvements have been made in in-database ML, challenges persist in supporting ML training within databases, particularly concerning security, privacy, model storage and updating, parallel training, adapting to dynamic environments, and etc. Innovative projects like Amalur [69] have begun to explore the convergence of data integration systems with model zoos, pointing towards a future where data integration systems and ML seamlessly integrate.

[1] https://aws.amazon.com/bedrock

BIBLIOGRAPHY

[1] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. "Profiling relational data: a survey". In: *The VLDB Journal* 24 (2015), pp. 557–581.

[2] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the web: from relations to semistructured data and XML*. Morgan Kaufmann, 2000.

[3] Alessandro Achille et al. "Task2Vec: Task Embedding for Meta-Learning". In: Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019, pp. 6430–6439. URL: https://openaccess.thecvf.com/content_ICCV_2019/html/Achille_Task2Vec_Task_Embedding_for_Meta-Learning_ICCV_2019_paper.html (visited on 11/16/2023).

[4] Robert Adolf et al. "Fathom: Reference workloads for modern deep learning methods". In: *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE. 2016, pp. 1–10.

[5] Pulkit Agrawal et al. "form for machine learning". In: *Proceedings of the 2019 International Conference on Management of Data*. 2019, pp. 1803–1816.

[6] Saqib Alam and Nianmin Yao. "The impact of preprocessing steps on the accuracy of machine learning algorithms in sentiment analysis". In: *Computational and Mathematical Organization Theory* 25 (2019), pp. 319–335.

[7] Yazeed Alaudah et al. "A machine-learning benchmark for facies classification". In: *Interpretation* 7.3 (2019), SE175–SE187.

[8] Maria Joao Alves and João Climaco. "A review of interactive methods for multiobjective integer and mixed-integer programming". In: *European Journal of Operational Research* 180.1 (2007), pp. 99–115.

[9] Michael R Anderson et al. "Physical representation-based predicate optimization for a visual analytics database". In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE. 2019, pp. 1466–1477.

[10] Mohammad Asghari et al. "Transformation and Linearization Techniques in Optimization: A State-of-the-Art Survey". In: *Mathematics* 10.2 (2022), p. 283.

[11] Rob Ashmore, Radu Calinescu, and Colin Paterson. "Assuring the machine learning lifecycle: Desiderata, methods, and challenges". In: *ACM Computing Surveys (CSUR)* 54.5 (2021), pp. 1–39.

[12] Žiga Avsec et al. "Kipoi: accelerating the community exchange and reuse of predictive models for genomics". In: *BioRxiv* (2018), p. 375345.

[13] Brian Babcock and Surajit Chaudhuri. "Towards a robust query optimizer: a principled and practical approach". In: *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 2005, pp. 119–130.

[14] Arjun Balasubramanian et al. “Accelerating deep learning inference via learned caches”. In: *arXiv preprint arXiv:2101.07344* (2021).

[15] Agathe Balayn, Christoph Lofi, and Geert-Jan Houben. “Managing bias and unfairness in data for decision support: a survey of machine learning and data engineering approaches to identify and mitigate bias and unfairness within data management and analytics systems”. In: *The VLDB Journal* 30.5 (2021), pp. 739–768.

[16] Agathe Balayn et al. “What do you mean? Interpreting image classification with crowdsourced concept extraction and analysis”. In: *Proceedings of the Web Conference 2021*. 2021, pp. 1937–1948.

[17] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. “Multimodal machine learning: A survey and taxonomy”. In: *IEEE transactions on pattern analysis and machine intelligence* 41.2 (2018), pp. 423–443.

[18] Francesco Barbieri and et. al. et. “Tweeteval: Unified benchmark and comparative evaluation for tweet classification”. In: *arXiv preprint arXiv:2010.12421* (2020).

[19] Philip A Bernstein. “Applying Model Management to Classical Meta Data Problems”. In: *CIDR*. 2003.

[20] Sagar Bharadwaj et al. “Discovering related data at scale”. In: *VLDB* 14.8 (2021), pp. 1392–1400.

[21] Ekaba Bisong and Ekaba Bisong. “Kubeflow and kubeflow pipelines”. In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners* (2019), pp. 671–685.

[22] Matthias Boehm et al. “SystemDS: A declarative machine learning system for the end-to-end data science lifecycle”. In: *arXiv preprint arXiv:1909.02976* (2019).

[23] Daniel Bolya, Rohit Mittapalli, and Judy Hoffman. “Scalable Diverse Model Selection for Accessible Transfer Learning”. In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 19301–19312. URL: https://proceedings.neurips.cc/paper_files/paper/2021/hash/a1140a3d0df1c81e24ae Abstract.html (visited on 11/15/2023).

[24] Karen L Boyd. “Datasheets for Datasets help ML Engineers Notice and Understand Ethical Issues in Training Data”. In: *Proceedings of the ACM on Human-Computer Interaction* 5.CSCW2 (2021), pp. 1–27.

[25] Eli Brumbaugh et al. “Bighead: a framework-agnostic, end-to-end machine learning platform”. In: *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2019, pp. 551–560.

[26] Erin M Buchanan et al. “Getting started creating data dictionaries: How to create a shareable data set”. In: *Advances in Methods and Practices in Psychological Science* 4.1 (2021), p. 2515245920928007.

[27] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. “Why and where: A characterization of data provenance”. In: *Database Theory—ICDT 2001: 8th International Conference London, UK, January 4–6, 2001 Proceedings 8*. Springer. 2001, pp. 316–330.

[28] Zhaowei Cai, Mohammad Saberian, and Nuno Vasconcelos. “Learning complexity-aware cascades for pedestrian detection”. In: *IEEE transactions on pattern analysis and machine intelligence* 42.9 (2019), pp. 2195–2211.

[29] Jiashen Cao et al. “THIA: Accelerating Video Analytics using Early Inference and Fine-Grained Query Planning”. In: *arXiv preprint arXiv:2102.08481* (2021).

[30] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. “Creating embeddings of heterogeneous relational datasets for data integration tasks”. In: *SIGMOD*. 2020, pp. 1335–1349.

[31] Chengliang Chai et al. “Data management for machine learning: A survey”. In: *IEEE Transactions on Knowledge and Data Engineering* (2022).

[32] Jae-Young Chang and Sang-goo Lee. “An optimization of disjunctive queries: Union-pushdown”. In: *Proceedings of COMPSAC)*. IEEE. 1997, pp. 356–361.

[33] Ryan Chard et al. “DLHub: Model and data serving for science”. In: *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2019, pp. 283–292.

[34] Surajit Chaudhuri. “An overview of query optimization in relational systems”. In: *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. 1998, pp. 34–43.

[35] Andrew Chen et al. “Developments in mlflow: A system to accelerate the machine learning lifecycle”. In: *Proceedings of the fourth international workshop on data management for end-to-end machine learning*. 2020, pp. 1–4.

[36] L. Chen et al. “Towards Linear Algebra over Normalized Data”. In: *PVLDB* 10.11 (2017).

[37] Tianqi Chen and Carlos Guestrin. “Xgboost: A scalable tree boosting system”. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[38] Eunnuri Cho, Tai-Woo Chang, and Gyusun Hwang. “Data preprocessing combination to improve the performance of quality classification in the manufacturing process”. In: *Electronics* 11.3 (2022), p. 477.

[39] KR1442 Chowdhary et al. “Natural language processing”. In: *Fundamentals of artificial intelligence* (2020), pp. 603–649.

[40] Mircea Cimpoi et al. “Describing Textures in the Wild”. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Columbus, OH, USA: IEEE, June 2014, pp. 3606–3613. (Visited on 12/04/2023).

[41] *ClearML - Auto-Magical Suite of tools to streamline your ML workflow*. https://clear.ml/docs/latest. Accessed: 2023-09-10.

[42] Cody Coleman et al. “Analysis of dawnbench, a time-to-accuracy machine learning performance benchmark”. In: *ACM SIGOPS Operating Systems Review* 53.1 (2019), pp. 14–25.

[43] Cody Coleman et al. “Dawnbench: An end-to-end deep learning benchmark and competition”. In: *Training* 100.101 (2017), p. 102.

[44] *Comat - Less friction, mode ML*. https://www.comet.com/site/. Accessed: 2023-09-10.

[45] Daniel Crankshaw et al. "Clipper: A {Low-Latency} Online Prediction Serving System". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 2017, pp. 613–627.

[46] Yin Cui et al. "Large Scale Fine-Grained Categorization and Domain-Specific Transfer Learning". In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018, pp. 4109–4118. (Visited on 11/15/2023).

[47] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009 IEEE Conference on Computer Vision and Pattern Recognition. ISSN: 1063-6919. June 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848. URL: https://ieeexplore.ieee.org/document/5206848 (visited on 12/04/2023).

[48] A. Deshpande et al. "A linearized framework and a new benchmark for model selection for fine-tuning". In: *ArXiv* (Jan. 29, 2021). URL: https://www.semanticscholar.org/paper/A-linearized-framework-and-a-new-benchmark-for-for-Deshpande-Achille/38e7fbb3b64405e0a44148cb233869905bcce92e (visited on 12/04/2023).

[49] Aditya Deshpande et al. "A linearized framework and a new benchmark for model selection for fine-tuning". In: *arXiv preprint arXiv:2102.00084* (2021).

[50] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[51] A. Doan, A. Halevy, and Z. Ives. *Principles of data integration*. Elsevier, 2012.

[52] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=YicbFdNTTy (visited on 12/03/2023).

[53] Li Fei-Fei, R. Fergus, and P. Perona. "Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories". In: 2004 Conference on Computer Vision and Pattern Recognition Workshop. June 2004, pp. 178–178. (Visited on 12/04/2023).

[54] R. C. Fernandez et al. "Aurum: A Data Discovery System". In: *ICDE*. 2018, pp. 1001–1012.

[55] Raul Castro Fernandez et al. "Aurum: A data discovery system". In: *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE. 2018, pp. 1001–1012.

[56] Ian Foster et al. "Chimera: A virtual data system for representing, querying, and automating data derivation". In: *Proceedings 14th International Conference on Scientific and Statistical Database Management*. IEEE. 2002, pp. 37–46.

[57] Karl Anton Froeschl. "A metadata approach to statistical query processing". In: *Statistics and Computing* 6 (1996), pp. 11–29.

[58] Apurva Gandhi et al. “The Tensor Data Platform: Towards an AI-centric Database System”. In: *CIDR* (2023).

[59] Timnit Gebru et al. “Datasheets for datasets”. In: *Communications of the ACM* 64.12 (2021), pp. 86–92.

[60] A. Geiger, P. Lenz, and R. Urtasun. “Are we ready for autonomous driving? The KITTI vision benchmark suite”. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Providence, RI: IEEE, June 2012, pp. 3354–3361. (Visited on 12/04/2023).

[61] Gharib Gharibi et al. “Automated end-to-end management of the modeling lifecycle in deep learning”. In: *Empirical Software Engineering* 26.2 (2021), pp. 1–33.

[62] Tarleton Gillespie. “Content moderation, AI, and the question of scale”. In: *Big Data & Society* 7.2 (2020), p. 2053951720943234.

[63] Joan Giner-Miguelez, Abel Gómez, and Jordi Cabot. “DescribeML: a tool for describing machine learning datasets”. In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 2022, pp. 22–26.

[64] Stefan Grafberger et al. “Mlinspect: A data distribution debugger for machine learning pipelines”. In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 2736–2739.

[65] Aditya Grover and Jure Leskovec. “node2vec: Scalable Feature Learning for Networks”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. New York, NY, USA: Association for Computing Machinery, Aug. 13, 2016, pp. 855–864. (Visited on 11/28/2023).

[66] Peizhen Guo, Bo Hu, and Wenjun Hu. “Sommelier: Curating DNN Models for the Masses”. In: *Proceedings of the 2022 International Conference on Management of Data*. 2022, pp. 1876–1890.

[67] R. Hai, S. Geisler, and C. Quix. “Constance: An Intelligent Data Lake System”. In: *SIGMOD*. ACM. 2016, pp. 2097–2100.

[68] Rihan Hai et al. “Amalur: Data Integration Meets Machine Learning”. In: *Proceeding of the 39th IEEE International Conference on Data Engineering (ICDE)* (2023).

[69] Rihan Hai et al. “Amalur: Data integration meets machine learning”. In: *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE. 2023, pp. 3729–3739.

[70] Rihan Hai et al. “Data Lakes: A Survey of Functions and Systems”. In: *TKDE* (2023).

[71] Rihan Hai et al. “Data Lakes: A Survey of Functions and Systems”. In: *IEEE Transactions on Knowledge and Data Engineering* 35.12 (2023), pp. 12571–12590. DOI: 10.1109/TKDE.2023.3270101.

[72] Alon Y Halevy, Flip Korn, Steven Euijong Whang, et al. “Managing Google’s data lake: an overview of the Goods system”. In: *IEEE Data Eng. Bull.* 39.3 (2016), pp. 5–14.

[73] William L. Hamilton, Rex Ying, and Jure Leskovec. “Inductive representation learning on large graphs”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Red Hook, NY, USA: Curran Associates Inc., Dec. 4, 2017, pp. 1025–1035. ISBN: 978-1-5108-6096-4. (Visited on 11/21/2023).

[74] Kaiming He et al. “Identity mappings in deep residual networks”. In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer. 2016, pp. 630–645.

[75] Patrick Helber et al. “EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification”. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 12.7 (July 2019), pp. 2217–2226. (Visited on 12/04/2023).

[76] Ciarán Hogan and Ganesh Sistu. “Automatic Vehicle Ego Body Extraction for Reducing False Detections in Automated Driving Applications”. In: *Irish Conference on Artificial Intelligence and Cognitive Science*. Springer. 2022, pp. 264–275.

[77] Andrew G Howard et al. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861* (2017).

[78] Jonathan Huang et al. “Speed/accuracy trade-offs for modern convolutional object detectors”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7310–7311.

[79] Long-Kai Huang et al. “Frustratingly Easy Transferability Estimation”. In: *Proceedings of the 39th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, June 28, 2022, pp. 9201–9225. URL: https://proceedings.mlr.press/v162/huang22d.html (visited on 11/15/2023).

[80] Hiroshi Iida et al. “Tabbie: Pretrained representations of tabular data”. In: *arXiv preprint arXiv:2105.02584* (2021).

[81] Ihab F Ilyas and Xu Chu. *Data cleaning*. Morgan & Claypool, 2019.

[82] Yannis E Ioannidis. “Query optimization”. In: *ACM Computing Surveys (CSUR)* 28.1 (1996), pp. 121–123.

[83] Shashank Mohan Jain. “Hugging Face”. In: *Introduction to Transformers for NLP*. Springer, 2022, pp. 51–67.

[84] Junchen Jiang et al. “Chameleon: scalable adaptation of video analytics”. In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 2018, pp. 253–266.

[85] Pan Junjun et al. “Diabetic Retinopathy Detection Based on Deep Convolutional Neural Networks for Localization of Discriminative Regions”. In: 2018 International Conference on Virtual Reality and Visualization (ICVRV). Qingdao, China: IEEE, Oct. 2018, pp. 46–52. (Visited on 12/04/2023).

[86] Daniel Kang et al. “Noscope: optimizing neural network queries over video at scale”. In: *arXiv preprint arXiv:1703.02529* (2017).

[87] Konstantinos Karanasos et al. “Extending relational query processing with ML inference”. In: *CIDR* (2020).

[88] Yağız Kargın. “Turning scientists into data explorers”. In: *Proceedings of the 2013 SIGMOD/PODS Ph. D. symposium*. 2013, pp. 25–30.

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

[90] A. Kemper et al. “Optimizing Disjunctive Queries with Expensive Predicates”. In: *ACM SIMGOD*. SIGMOD ’94. Minneapolis, Minnesota, USA, 1994, pp. 336–347.

[91] Alfons andothers Kemper. “Optimizing disjunctive queries with expensive predicates”. In: *ACM SIGMOD Record* 23.2 (1994), pp. 336–347.

[92] Aamod Khatiwada, Roee Shraga, and Renée J Miller. “DIALITE: Discover, Align and Integrate Open Data Tables”. In: *Companion of the 2023 International Conference on Management of Data*. 2023, pp. 187–190.

[93] Ralph Kimball and Margy Ross. *The data warehouse toolkit: the complete guide to dimensional modeling*. John Wiley & Sons, 2011.

[94] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. Feb. 22, 2017. DOI: 10.48550/arXiv.1609.02907. arXiv: 1609.02907[cs,stat]. URL: http://arxiv.org/abs/1609.02907 (visited on 11/21/2023).

[95] Janis Klaise et al. “Alibi explain: Algorithms for explaining machine learning models”. In: *The Journal of Machine Learning Research* 22.1 (2021), pp. 8194–8200.

[96] Phokion G Kolaitis. “Schema mappings, data exchange, and metadata management”. In: *PODS*. Baltimore, MD,USA: ACM, 2005, pp. 61–75. ISBN: 1-59593-062-0.

[97] Hanna Köpcke, Andreas Thor, and Erhard Rahm. “Evaluation of entity resolution approaches on real-world match problems”. In: *VLDB* 3.1-2 (2010), pp. 484–493.

[98] Christos Koutras, George Siachamis, Asterios Katsifodimos, et al. “Valentine: Evaluating matching techniques for dataset discovery”. In: *ICDE*. IEEE. 2021, pp. 468–479.

[99] Peter Kraft et al. “Willump: A statistically-aware end-to-end optimizer for machine learning inference”. In: *Proceedings of Machine Learning and Systems* 2 (2020), pp. 147–159.

[100] Jonathan Krause et al. “3D Object Representations for Fine-Grained Categorization”. In: 2013 IEEE International Conference on Computer Vision Workshops. Dec. 2013, pp. 554–561. DOI: 10.1109/ICCVW.2013.77. URL: https://ieeexplore.ieee.org/document/6755945 (visited on 11/28/2023).

[101] Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. “Machine learning operations (mlops): Overview, definition, and architecture”. In: *IEEE Access* (2023).

[102] Sanjay Krishnan, Jiannan Wang, Ken Goldberg, et al. “Activeclean: Interactive data cleaning for statistical modeling”. In: *VLDB* 9.12 (2016), pp. 948–959.

[103] A. Krizhevsky. “Learning Multiple Layers of Features from Tiny Images”. In: 2009. URL: https://www.semanticscholar.org/paper/Learning-Multiple-Layers-of-Features-from-Tiny-Krizhevsky/5d90f06bb70a0a3dced62413346235c0 (visited on 12/04/2023).

[104] Arun Kumar et al. “Cerebro: A Layered Data Platform for Scalable Deep Learning”. In: *11th Annual Conference on Innovative Data Systems Research (CIDR'21)*. 2021.

[105] Hai Lan, Zhifeng Bao, and Yuwei Peng. “A survey on advancing the dbms query optimizer: Cardinality estimation, cost model, and plan enumeration”. In: *Data Science and Engineering* 6 (2021), pp. 86–101.

[106] Y. LeCun, Fu Jie Huang, and L. Bottou. “Learning methods for generic object recognition with invariance to pose and lighting”. In: Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. Vol. 2. Washington, DC, USA: IEEE, pp. 97–104. ISBN: 978-0-7695-2158-9. DOI: 10.1109/CVPR.2004.1315150. URL: http://ieeexplore.ieee.org/document/1315150/ (visited on 12/04/2023).

[107] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (Nov. 1998). Conference Name: Proceedings of the IEEE, pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791. URL: https://ieeexplore.ieee.org/document/726791 (visited on 12/03/2023).

[108] Wilfried Lemahieu, Seppe vanden Broucke, and Bart Baesens. *Principles of database management: the practical guide to storing, managing and Analyzing big and small Data*. Cambridge University Press, 2018.

[109] Roman Levin et al. “Transfer Learning with Deep Tabular Models”. In: *ICLR*. 2022.

[110] Hao Li et al. “Guided Recommendation for Model Fine-Tuning”. In: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Vancouver, BC, Canada: IEEE, June 2023, pp. 3633–3642. ISBN: 9798350301298. DOI: 10.1109/CVPR52729.2023.00354. URL: https://ieeexplore.ieee.org/document/10204061/ (visited on 12/04/2023).

[111] Ziyu Li et al. “Macaroni: Crawling and Enriching Metadata from Public Model Zoos”. In: *International Conference on Web Engineering*. Springer. 2023, pp. 376–380.

[112] Ziyu Li et al. “Metadata Representations for Queryable ML Model Zoos”. In: *arXiv preprint arXiv:2207.09315* (2022).

[113] Ziyu Li et al. “Optimizing machine learning inference queries for multiple objectives”. In: *ICDEW*. IEEE. 2023, pp. 74–78.

[114] Kunsen Lin et al. “Deep convolutional neural networks for construction and demolition waste classification: VGGNet structures, cyclical learning rate, and knowledge transfer”. In: *Journal of Environmental Management* 318 (Sept. 15, 2022), p. 115501. ISSN: 0301-4797. DOI: 10.1016/j.jenvman.2022.115501. URL: https://www.sciencedirect.com/science/article/pii/S030147972201074X (visited on 11/29/2023).

[115] Tsung-Yi Lin et al. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[116] Renming Liu, Matthew Hirn, and Arjun Krishnan. “Accurately modeling biased random walks on weighted networks using node2vec+”. In: *Bioinformatics* 39.1 (2023). Publisher: Oxford University Press, btad047.

[117] Wei Liu et al. “Ssd: Single shot multibox detector”. In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[118] Ze Liu et al. “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows”. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021 IEEE/CVF International Conference on Computer Vision (ICCV). Montreal, QC, Canada: IEEE, Oct. 2021, pp. 9992–10002. ISBN: 978-1-66542-812-5. DOI: 10.1109/ICCV48922.2021.00986. URL: https://ieeexplore.ieee.org/document/9710580/ (visited on 12/03/2023).

[119] Zhuang Liu et al. “A ConvNet for the 2020s”. In: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). New Orleans, LA, USA: IEEE, June 2022, pp. 11966–11976. ISBN: 978-1-66546-946-3. DOI: 10.1109/CVPR52688.2022.01167. URL: https://ieeexplore.ieee.org/document/9879745/ (visited on 12/03/2023).

[120] Yao Lu et al. “Accelerating machine learning inference with probabilistic predicates”. In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 1493–1508.

[121] Volker Markl, Guy M Lohman, and Vijayshankar Raman. “LEO: An autonomic query optimizer for DB2”. In: *IBM Systems Journal* 42.1 (2003), pp. 98–106.

[122] R Timothy Marler and Jasbir S Arora. “Survey of multi-objective optimization methods for engineering”. In: *Structural and multidisciplinary optimization* 26.6 (2004), pp. 369–395.

[123] Peter Mattson et al. “Mlperf training benchmark”. In: *Proceedings of Machine Learning and Systems* 2 (2020), pp. 336–349.

[124] Hui Miao, Amit Chavan, and Amol Deshpande. “Provdb: Lifecycle management of collaborative analysis workflows”. In: *Proceedings of the 2nd Workshop on Human-in-the-Loop Data Analytics*. 2017, pp. 1–6.

[125] Hui Miao et al. “Towards unified data and lifecycle management for deep learning”. In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE. 2017, pp. 571–582.

[126] Milagros Miceli et al. “Documenting computer vision datasets: an invitation to reflexive data practices”. In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. 2021, pp. 161–172.

[127] Pietruszka Michał, Michał Turski, Garncarek Łukasz, et al. “STable: Table Generation Framework for Encoder-Decoder Models”. In: *NeurIPS TRL*. 2022.

[128] Tomás Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2013. URL: http://arxiv.org/abs/1301.3781.

[129] Lester James Miranda. “Towards data-centric machine learning: a short review”. In: *ljvmiranda921. github. io* (2021).

[130] Puneet Misra and Arun Singh Yadav. “Impact of preprocessing methods on healthcare predictions”. In: *Proceedings of 2nd International Conference on Advanced Computing and Software Engineering (ICACSE)*. 2019.

[131] Margaret Mitchell et al. “Model cards for model reporting”. In: *Proceedings of the conference on fairness, accountability, and transparency*. 2019, pp. 220–229.

[132] *ML Metadata - TFX - TensorFlow*. https://www.tensorflow.org/tfx/guide/mlmd. Accessed: 2023-09-10.

[133] Dinithi Nallaperuma et al. “Online incremental machine learning platform for big data-driven smart traffic management”. In: *IEEE Transactions on Intelligent Transportation Systems* 20.12 (2019), pp. 4679–4690.

[134] Fatemeh Nargesian et al. “Data lake management: challenges and opportunities”. In: *Proceedings of the VLDB Endowment* 12.12 (2019), pp. 1986–1989.

[135] Fatemeh Nargesian et al. “Organizing Data Lakes for Navigation”. In: *SIGMOD*. 2020, pp. 1939–1950.

[136] Yuval Netzer et al. “Reading Digits in Natural Images with Unsupervised Feature Learning”. In: 2011. URL: https://www.semanticscholar.org/paper/Reading-Digits-in-Natural-Images-with-Unsupervised-Netzer-Wang/02227c94dd41fe0b439e050d377b0beb5d427cda (visited on 12/04/2023).

[137] Jiquan Ngiam et al. “Multimodal deep learning”. In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011, pp. 689–696.

[138] Cuong Nguyen et al. “LEEP: A New Measure to Evaluate Transferability of Learned Representations”. In: *Proceedings of the 37th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, Nov. 21, 2020, pp. 7294–7305. URL: https://proceedings.mlr.press/v119/nguyen20b.html (visited on 11/15/2023).

[139] Maria-Elena Nilsback and Andrew Zisserman. “Automated Flower Classification over a Large Number of Classes”. In: *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*. Image Processing (ICVGIP). Bhubaneswar, India: IEEE, Dec. 2008, pp. 722–729. DOI: 10.1109/ICVGIP.2008.47. URL: http://ieeexplore.ieee.org/document/4756141/ (visited on 12/04/2023).

[140] Randal S Olson et al. “A system for accessible artificial intelligence”. In: *Genetic programming theory and practice XV*. Springer, 2018, pp. 121–134.

[141] Randal S Olson et al. “PMLB: a large benchmark suite for machine learning evaluation and comparison”. In: *BioData mining* 10.1 (2017), pp. 1–13.

[142] *OpenVino - Open Model Zoo*. https://github.com/openvinotoolkit/open_model_zoo/tree/master. Accessed: 2023-06-10.

[143] Christos H. Papadimitriou and Mihalis Yannakakis. "Multiobjective Query Optimization". In: *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS '01. Santa Barbara, California, USA: Association for Computing Machinery, 2001, pp. 52–59. ISBN: 1581133618. DOI: 10.1145/375551.375560. URL: https://doi.org/10.1145/375551.375560.

[144] *Papers with Code*. https://paperswithcode.com/sota. Accessed: 2023-06-10.

[145] Pietro Pinoli et al. "Metadata management for scientific databases". In: *Information Systems* 81 (2019), pp. 1–20.

[146] Neoklis Polyzotis et al. "Data lifecycle challenges in production machine learning: a survey". In: *ACM SIGMOD Record* 47.2 (2018), pp. 17–28.

[147] *PyTorch Hub*. https://pytorch.org/hub/. Accessed: 2023-06-10.

[148] C. Quix, R. Hai, and I. Vatov. "Metadata Extraction and Management in Data Lakes With GEMMS". In: *CSIMQ* 9 (2016), pp. 67–83.

[149] Mark Raasveldt and Hannes Mühleisen. "Duckdb: an embeddable analytical database". In: *Proceedings of the 2019 International Conference on Management of Data*. 2019, pp. 1981–1984.

[150] Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018).

[151] Vijay Janapa Reddi et al. "Mlperf inference benchmark". In: *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2020, pp. 446–459.

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

[153] Cedric Renggli et al. "SHiFT: an efficient, flexible search engine for transfer learning". In: *Proceedings of the VLDB Endowment* 16.2 (2022), pp. 304–316. ISSN: 2150-8097. DOI: 10.14778/3565816.3565831. URL: https://dl.acm.org/doi/10.14778/3565816.3565831 (visited on 11/15/2023).

[154] Albert Reuther et al. "Survey and benchmarking of machine learning accelerators". In: *2019 IEEE high performance extreme computing conference (HPEC)*. IEEE. 2019, pp. 1–9.

[155] Jenn Riley. "Understanding metadata". In: *Washington DC, United States: National Information Standards Organization* 23 (2017), pp. 7–10.

[156] Francisco Romero et al. "Optimizing video analytics with declarative model relationships". In: *Proceedings of the VLDB Endowment* 16.3 (2022), pp. 447–460.

[157] Francisco Luque Sánchez et al. "Revisiting crowd behaviour analysis through deep learning: Taxonomy, anomaly detection, crowd emotions, datasets, opportunities and prospects". In: *Information Fusion* 64 (2020), pp. 318–335.

[158] Monica Scannapieco, Ahmed K Elmagarmid, et al. “Privacy preserving schema and data matching”. In: *SIGMOD*. ACM, 2007, pp. 653–664.

[159] Sebastian Schelter et al. “Automatically tracking metadata and provenance of machine learning experiments”. In: *Machine Learning Systems Workshop at NIPS*. 2017, pp. 27–29.

[160] Sebastian Schelter et al. “On Challenges in Machine Learning Model Management”. In: (2018).

[161] Marius Schlegel and Kai-Uwe Sattler. “Management of machine learning lifecycle artifacts: A survey”. In: *ACM SIGMOD Record* 51.4 (2023), pp. 18–35.

[162] Burr Settles. “Active learning literature survey”. In: (2009).

[163] Haichen Shen et al. “Fast video classification via adaptive cascading of deep models”. In: *Proceedings of the IEEE CVPR*. 2017, pp. 3646–3654.

[164] Abraham Silberschatz, Henry F Korth, and Shashank Sudarshan. “Database system concepts”. In: (2011).

[165] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *International Conference on Learning Representations*. 2015.

[166] Gurmeet Singh, Shishir Bharathi, Laura Pearlman, et al. “A metadata catalog service for data intensive applications”. In: *ICS*. 2003, p. 33.

[167] Leslie N. Smith. “Cyclical Learning Rates for Training Neural Networks”. In: 2017 IEEE Winter Conference on Applications of Computer Vision (WACV). Santa Rosa, CA, USA: IEEE, Mar. 2017, pp. 464–472. ISBN: 978-1-5090-4822-9. DOI: 10.1109/WACV.2017.58. URL: http://ieeexplore.ieee.org/document/7926641/ (visited on 11/29/2023).

[168] Jie Song et al. “Deep Model Transferability from Attribution Maps”. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/hash/e94fe9ac8dc10dd8b9a239e6abee2848-Abstract.html (visited on 11/15/2023).

[169] Wenbo Sun, Asterios Katsifodimos, and Rihan Hai. “Accelerating Machine Learning Queries with Linear Algebra Query Processing”. In: *Proceedings of the 35th International Conference on Scientific and Statistical Database Management*. SSDBM ’23. New York, NY, USA: Association for Computing Machinery, 2023. ISBN: 9798400707469. DOI: 10.1145/3603719.3603726. URL: https://doi.org/10.1145/3603719.3603726.

[170] Alexey Svyatkovskiy, Julian Kates-Harbeck, and William Tang. “Training distributed deep recurrent neural networks with mixed precision on GPU clusters”. In: *Proceedings of the Machine Learning on HPC Environments*. 2017, pp. 1–8.

[171] Jonti Talukdar et al. “Transfer learning for object detection using state-of-the-art deep neural networks”. In: *2018 5th International Conference on Signal Processing and Integrated Networks (SPIN)*. IEEE. 2018, pp. 78–83.

[172] Chuanqi Tan et al. “A Survey on Deep Transfer Learning”. In: *Artificial Neural Networks and Machine Learning – ICANN 2018*. Ed. by Věra Kůrková et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 270–279. ISBN: 978-3-030-01424-7. DOI: 10.1007/978-3-030-01424-7_27.

[173] *Tensorflow Hub*. https://www.tensorflow.org/hub. Accessed: 2023-06-10.

[174] Ignacio G Terrizzano et al. “Data Wrangling: The Challenging Yourney from the Wild to the Lake.” In: *CIDR*. Asilomar. 2015.

[175] Anh T. Tran, Cuong V. Nguyen, and Tal Hassner. “Transferability and Hardness of Supervised Classification Tasks”. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019, pp. 1395–1405. URL: https://openaccess.thecvf.com/content_ICCV_2019/html/Tran_Transferability_and_Hardness_of_Supervised_Classification_Tasks_ICCV_2019_paper.html (visited on 11/15/2023).

[176] Immanuel Trummer and Christoph Koch. “Multi-Objective Parametric Query Optimization”. In: *SIGMOD Record* 45.1 (2016), pp. 24–31.

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

[178] Jason Tsay et al. “Runway: machine learning model experiment management tool”. In: *Conference on Systems and Machine Learning (SysML)*. 2018.

[179] Tom Van Der Weide et al. “Versioning for end-to-end machine learning pipelines”. In: *Proceedings of the 1st Workshop on Data Management for End-to-End Machine Learning*. 2017, pp. 1–9.

[180] Manasi Vartak et al. “ModelDB: a system for machine learning model management”. In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. 2016, pp. 1–3.

[181] Petar Veličković et al. *Graph Attention Networks*. Feb. 4, 2018. arXiv: 1710.10903[cs, stat]. URL: http://arxiv.org/abs/1710.10903 (visited on 11/21/2023).

[182] Paul Voigt and Axel Von dem Bussche. “The eu general data protection regulation (gdpr)”. In: *A Practical Guide, 1st Ed., Cham: Springer International Publishing* 10.3152676 (2017), pp. 10–5555.

[183] Zirui Wang et al. “Characterizing and Avoiding Negative Transfer”. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Long Beach, CA, USA: IEEE, June 2019, pp. 11285–11294. ISBN: 978-1-72813-293-8. DOI: 10.1109/CVPR.2019.01155. URL: https://ieeexplore.ieee.org/document/8953565/ (visited on 12/04/2023).

[184] *Weights & Biases, The AI developer Platform*. https://wandb.ai/site. Accessed: 2023-09-10.

[185] Dave Wells. *Introduction to data catalogs*. 2019.

[186] Thomas Wolf et al. “Transformers: State-of-the-art natural language processing”. In: *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*. 2020, pp. 38–45.

[187] Yongji Wu et al. “Serving and Optimizing Machine Learning Workflows on Heterogeneous Infrastructures”. In: ().

[188] Zhenqin Wu et al. “MoleculeNet: a benchmark for molecular machine learning”. In: *Chemical science* 9.2 (2018), pp. 513–530.

[189] Feng Xia et al. “Graph Learning: A Survey”. In: *IEEE Transactions on Artificial Intelligence* 2.2 (Apr. 2021), pp. 109–127. ISSN: 2691-4581. DOI: 10.1109/TAI.2021.3076021. URL: https://ieeexplore.ieee.org/document/9416834/ (visited on 11/21/2023).

[190] Peng Xu, Xiatian Zhu, and David A Clifton. “Multimodal learning with transformers: A survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023).

[191] Chengrun Yang et al. “Oboe: Collaborative filtering for automl initialization”. In: *arXiv preprint arXiv:1808.03233* (2018).

[192] Zhihui Yang et al. “Optimizing machine learning inference queries with correlative proxy models”. In: *Proceedings of the VLDB Endowment* 15.10 (2022), pp. 2032–2044.

[193] Jason Yosinski et al. “How transferable are features in deep neural networks?” In: *NeurIPS* 27 (2014).

[194] Kaichao You et al. “LogME: Practical Assessment of Pre-trained Models for Transfer Learning”. In: *Proceedings of the 38th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, July 1, 2021, pp. 12133–12143. URL: https://proceedings.mlr.press/v139/you21b.html (visited on 11/15/2023).

[195] Mireia Yurrita et al. “Disentangling Fairness Perceptions in Algorithmic Decision-Making: The Effects of Explanations, Human Oversight, and Contestability”. In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. CHI ’23. Hamburg, Germany: Association for Computing Machinery, 2023. ISBN: 9781450394215. DOI: 10.1145/3544548.3581161. URL: https://doi.org/10.1145/3544548.3581161.

[196] Matei Zaharia et al. “Accelerating the machine learning lifecycle with MLflow.” In: *IEEE Data Eng. Bull.* 41.4 (2018), pp. 39–45.

[197] Amir R. Zamir et al. “Taskonomy: Disentangling Task Transfer Learning”. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Salt Lake City, UT: IEEE, June 2018, pp. 3712–3722. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00391. URL: https://ieeexplore.ieee.org/document/8578489/ (visited on 12/04/2023).

[198] Xiaohua Zhai et al. “A Large-scale Study of Representation Learning with the Visual Task Adaptation Benchmark”. In: *arXiv: Computer Vision and Pattern Recognition* (Oct. 1, 2019). URL: https://www.semanticscholar.org/paper/A-Large-scale-Study-of-Representation-Learning-with-Zhai-Puigcerver/85b9e68eb27069e87181050035f40b79438dd220 (visited on 11/28/2023).

[199] Haoyu Zhang et al. “Live video analytics at scale with approximation and delay-tolerance”. In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 2017, pp. 377–392.

[200] Hui Zhang et al. “0/1 Deep Neural Networks via Block Coordinate Descent”. In: *CoRR* abs/2206.09379 (2022). DOI: 10.48550/ARXIV.2206.09379. arXiv: 2206.09379. URL: https://doi.org/10.48550/arXiv.2206.09379.

[201] Y. Zhang and Z. G. Ives. “Finding Related Tables in Data Lakes for Interactive Data Science”. In: *SIGMOD*. 2020, pp. 1951–1966.

[202] Yang Zhang et al. “DataLab: a version data management and analytics system”. In: *Proceedings of the 2nd International Workshop on BIG Data Software Engineering*. 2016, pp. 12–18.

[203] Zhao Zhang, Evan R Sparks, and Michael J Franklin. “Diagnosing machine learning pipelines with fine-grained lineage”. In: *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. 2017, pp. 143–153.

[204] Xuanhe Zhou et al. “Database meets artificial intelligence: A survey”. In: *IEEE Transactions on Knowledge and Data Engineering* (2020).

[205] Fuzhen Zhuang et al. “A comprehensive survey on transfer learning”. In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.

List of Figures

LIST OF TABLES

SUMMARY

Over the last two decades, the machine learning (ML) field has witnessed a dramatic expansion, propelled by burgeoning data volumes and the advancement of computational technologies. Deep learning (DL) in particular has demonstrated remarkable success across a wide range of domains, including healthcare, mobility, life sciences, and energy systems. This success has been further accelerated by the availability and efficiency of open-source ML frameworks like TensorFlow and PyTorch, making ML methodologies more accessible than ever.

However, this rapid growth has brought its own set of challenges. The proliferation of ML models and related artifacts, such as datasets, have brought abundant information during the ML lifecycle. The descriptive and property information of these artifacts is referred as metadata. Yet current practices, such as model cards used in public model zoos and tools to track metadata within scripts, cannot fully captured the metadata of these artifacts, let alone a standardized approach for their management, and access. In addition, the prevailing practice of managing ML/DL scripts via traditional software repositories, while adequate for software engineering, falls short in addressing the unique needs of ML workflows, such as model reuse and comparative analysis. These practices hinder the effective use of structured and comprehensive metadata representation. This disconnect points to a pressing need for improved methodologies and tools in the ML field.

In response to these challenges, this thesis delves into the development and exploitation of structured metadata representations within ML model zoos. In Chapter 2, we first propose a metamodel that represent different types of metadata, thus transforming the metadata from being merely descriptive to being queryable and machine-readable. The structured nature of our metamodel allows for more efficient querying and retrieval of information, which is a substantial improvement over the traditional, text-based descriptions.

Additionally, the thesis explores the use of metadata to optimize various ML processes, particularly in the selection of appropriate models for specific tasks, i.e., model inference and fine-tuning. In Chapter 3, we investigate the optimization of ML inference queries in heterogeneous model zoos using a Mixed-Integer-Programming-based optimizer. This optimizer, which considers multiple objectives such as accuracy and inference speed, provides a robust framework for model selection and execution planning. In Chapter 4, the research extends to model selection for fine-tuning. We investigate on predicting model performance, particularly accuracy, in scenarios where data domains shift, thus negating the need for constant model fine-tuning. By selectively choosing only the most promising candidates, this method substantially lowers the computational burden and associated costs of extensive model fine-tuning.

Overall, this thesis investigates the representation and application of metadata. The insights and methodologies presented not only improve the efficiency and effectiveness

of ML workflows but also pave the way for further exploration in the integration of metadata within ML practices, highlighting the continual development and potential for advancements in ML.

SAMENVATTING

In de afgelopen twee decennia heeft het vakgebied van machine learning (ML) een dramatische uitbreiding meegemaakt, aangedreven door de groeiende hoeveelheid gegevens en de vooruitgang van computationele technologieën. Deep Learning (DL) in het bijzonder heeft opmerkelijke successen getoond in een breed scala aan domeinen, waaronder gezondheidszorg, mobiliteit, levenswetenschappen en energiesystemen. Dit succes is verder versneld door de beschikbaarheid en efficiëntie van open-source ML frameworks zoals TensorFlow en PyTorch, waardoor ML-methodologieën toegankelijker zijn dan ooit tevoren.

Deze snelle groei heeft echter ook zijn eigen uitdagingen met zich meegebracht. De woekering van ML-modellen en gerelateerde artefacten, zoals datasets, heeft tijdens de ML-levenscyclus een overvloed aan informatie gebracht. De beschrijvende en eigenschapinformatie van deze artefacten wordt aangeduid als metadata. Huidige praktijken, zoals modelkaarten die in openbare model verzamelingen worden gebruikt en tools om metadata binnen scripts bij te houden, kunnen de metadata van deze artefacten echter niet volledig vastleggen, laat staan een gestandaardiseerde aanpak voor hun beheer en toegang. Bovendien schiet de gangbare praktijk van het beheren van ML/DL-scripts via traditionele softwareopslagplaatsen tekort bij het aanpakken van de unieke behoeften van ML-workflows, zoals modelhergebruik en vergelijkende analyse. Deze praktijken kunnen niet volledig het potentieel van gestructureerde en uitgebreide metadatarepresentatie benutten. Dit wijst op een dringende behoefte aan verbeterde methodologieën en hulpmiddelen op het gebied van ML.

Als reactie op deze uitdagingen verdiept deze scriptie zich in de ontwikkeling en exploitatie van gestructureerde metadatavoorstellingen binnen ML model verzamelingen. In Hoofdstuk 2 stellen we eerst een metamodel voor dat verschillende soorten metadata vertegenwoordigt. Deze vooruitgang transformeert de metadata van slechts beschrijvend naar opvraagbaar en machine-leesbaar. De gestructureerde aard van ons metamodel maakt efficiëntere opvraging en terugwinning van informatie mogelijk, wat een aanzienlijke verbetering is ten opzichte van de traditionele, op tekst gebaseerde beschrijvingen.

Daarnaast verdiept de scriptie zich in het verbeteren van het nut en de toegankelijkheid van metadata om diverse ML-processen te optimaliseren, met name bij de selectie van geschikte modellen voor specifieke taken, zoals modelinferentie en fine-tuning. In Hoofdstuk 3 onderzoeken we de optimalisatie van ML-inferentievragen in heterogene model verzamelingen met behulp van een op Mixed-Integer-Programming gebaseerde optimizer. Deze optimizer, die meerdere doelstellingen in overweging neemt zoals nauwkeurigheid en inferentiesnelheid, biedt een robuust raamwerk voor modelselectie en uitvoeringsplanning. Verder, in Hoofdstuk 4, breidt het onderzoek zich uit tot modelselectie voor fine-tuning. We onderzoeken de voorspelling van modelprestaties, met name nauwkeurigheid, in scenario's waarbij datadomeinen verschuiven, waardoor

de noodzaak voor constante model fine-tuning wordt tenietgedaan. Door selectief alleen de meest veelbelovende kandidaten te kiezen, vermindert deze methode aanzienlijk de computationele eisen en kosten die gepaard gaan met uitgebreide model fine-tuning.

Over het algemeen onderzoekt deze scriptie de representatie en toepassing van metadata. De inzichten en methodologieën die worden gepresenteerd, verbeteren niet alleen de efficiëntie en effectiviteit van ML-workflows, maar banen ook de weg voor verdere verkenning in de integratie van metadata binnen ML-praktijken. Hiermee wordt de voortdurende ontwikkeling en het potentieel voor vooruitgang in ML benadrukt.

ACKNOWLEDGEMENTS

This journey of pursuing my Ph.D. has been an amalgam of challenges, learning, and invaluable experiences, made possible by the support and guidance of many. I am profoundly grateful to each one who contributed to this significant phase of my academic and personal growth.

Firstly, I would like to express my profound appreciation to my promotors, Alessandro Bozzon and Geert-Jan Houben. To Alessandro, who joined my master thesis defense and offered me the opportunity to embark on this PhD journey, I am immensely grateful. Your perspectives have been invaluable in shaping the direction and integrity of this study. I would also like to thank Geert-Jan for the continuous support to members of the WIS group. Your dedication in monitoring our individual progress and providing guidance from various angles is commendable. Your efforts in fostering a cohesive group dynamic have greatly contributed to our collective and individual growth.

I am also immensely thankful to my daily supervisors, Asterios Katsifodimos. Asterios, your day-to-day mentorship, and hands-on approach in the lab (or field, as applicable) have been crucial in navigating the practical challenges of my research. Your patience and dedication have greatly contributed to my personal and professional growth. Your critical feedback and unwavering support have been instrumental in refining my research and academic skills. I would give special thanks to Rihan Hai, the hours you spent reviewing drafts, and providing encouragement have been vital to the completion of this project. Your attention to detail and commitment to excellence have left a lasting impact on my work ethic and research quality.

I would also like to express my appreciation to the members of my thesis committee, Prof. George Smaragdakis, Prof. Cesare Pautasso, Prof. Yannis Velegrakis, and Prof. David Abbink for their valuable time, insightful comments, and constructive criticisms during the course of this research.

My heartfelt thanks go to the WIS group, where an environment of collaboration and intellectual rigor has provided an ideal setting for pursuing my research. I am deeply appreciative of the constant support from our Delta team. The discussions in the office, the exchange of ideas, and our gatherings both within and outside the Netherlands have been invaluable to my journey. My special thanks to Marios, Andra, Christos, Danning, George (Georgios), Kyriakos and Wenbo. You are like brothers and sister in my PhD journey, offering a supportive network for dialogue and interaction. I am grateful for the support of my fellow researchers and colleagues, especially Avishek, Christoph, Jie, Sole, Ujwal, George (Giorgos), Venktesh, Sarah, Aditya, Agathe, Alisa, Anne, Arthur, Esra, Gaole, Garrett, Petros, Lijun, Lorenzo, Marcus, Nirmal, Peide, Philip, Sara, Shahin, and Shreyan, whose shared passion and insightful exchanges have enriched my PhD journey. The engaging conversations in the office, over lunch, near the coffee machine, and in other various occasions have been both enjoyable and intellectually stimulating.

I also like to offer special thanks Daphne and Nadia for your unwavering support and dedication.

I extend my sincere thanks to all my collaborators, Lydia from Distributed System group, Macro Bambila from Politecnico di Milano, Megha from Multimedia Computing group, Yan Kang from WeBank, Mariette, Henk, Hilco, Jiaqi, and Wenjie. Your support and collaboration have been instrumental in enhancing the quality of my work.

I am deeply indebted to Cognizant, whose financial support made this research possible. Thanks to Vivek and Vijay for their regular feedback and support.

Finally, I would like to dedicate this achievement to my family - my parents - for their unwavering love, understanding, and support throughout my academic pursuits. Special thanks to Mingliang. Your support, from shared moments in cooking, bouldering, traveling, etc., has been a source of immense joy and comfort, providing a constant presence throughout the ups and downs of this path. Your belief in me has been my constant source of motivation and strength. Of course, I'm not forgetting my dearest pal, Fajie, who accompanies me almost throughout the entire PhD journey. Together, we have grown - in more ways than one (knowledge and experience).

Curriculum Vitæ

Ziyu Li

Professional Experience

2024.2 – 2024.7	Postdoc Delft University of Technology Delft, the Netherlands
2018.5 – 2018.8	Data Science Intern R&D department, Corbion Amsterdam, the Netherlands

Education

2019 – 2024	PhD. Computer Science Delft University of Technology Delft, the Netherlands
2017 – 2019	Master in Computer Science Delft University of Technology Delft, the Netherlands
2013 – 2017	Bachelor in Computer Engineering South China University of Technology Guangzhou, China

LIST OF PUBLICATIONS

11. **Ziyu Li**, Wenjie Zhao, Asterios Katsifodimos and Rihan Hai. "LLM-PQA: LLM-enhanced Prediction Query Answering." *In Proceedings of the 33rd Conference on Information and Knowledge Management*, 2024.

10. Tim Littau, **Ziyu Li**, Rihan Hai, "Quantum Data Structures for Enhanced Database Performance." In VLDB Workshops, vol. 2024.

9. **Ziyu Li**, Hilco van der Wilk, Danning Zhan, Megha Kloska, Alessandro Bozzon, and Rihan Hai. "Model Selection with Model Zoo via Graph Learning." *In 2024 IEEE 40th International Conference on Data Engineering (ICDE)*, IEEE, 2024.

8. **Ziyu Li***[2], Wenbo Sun*, Danning Zhan, Yan Kang, Lydia Chen, Alessandro Bozzon, and Rihan Hai. "Amalur: Data integration meets machine learning." *IEEE Transactions on Knowledge and Data Engineering* (2024).

7. **Ziyu Li**, Henk Kant, Rihan Hai, Asterios Katsifodimos, Marco Brambilla, and Alessandro Bozzon. "Metadata Representations for Queryable Repositories of Machine Learning Models." *IEEE Access* (2023).

6. **Ziyu Li***, Mariette Schönfeld*, Wenbo Sun, Marios Fragkoulis, Rihan Hai, Alessandro Bozzon, and Asterios Katsifodimos. "Optimizing ML Inference Queries Under Constraints." *In International Conference on Web Engineering*, pp. 51-66. Cham: Springer Nature Switzerland, 2023.

5. **Ziyu Li**, Henk Kant, Rihan Hai, Asterios Katsifodimos, and Alessandro Bozzon. "Macaroni: Crawling and Enriching Metadata from Public Model Zoos." *In International Conference on Web Engineering*, pp. 376-380. Cham: Springer Nature Switzerland, 2023.

4. Hai, Rihan, Christos Koutras, Andra Ionescu, **Ziyu Li**, Wenbo Sun, Jessie Van Schijndel, Yan Kang, and Asterios Katsifodimos. "Amalur: Data integration meets machine learning." *In 2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pp. 3729-3739. IEEE, 2023.

3. **Ziyu Li**, Mariette Schönfeld, Rihan Hai, Alessandro Bozzon, and Asterios Katsifodimos. "Optimizing machine learning inference queries for multiple objectives." *In 2023 IEEE 39th International Conference on Data Engineering Workshops (ICDEW)*, pp. 74-78. IEEE, 2023.

2. **Ziyu Li**, Rihan Hai, Alessandro Bozzon, and Asterios Katsifodimos. "Metadata Representations for Queryable ML Model Zoos." In ICML, DataPerf Benchmarking Data for Data-Centric AI. 2022.

1. **Ziyu Li**, Asterios Katsifodimos, Alessandro Bozzon, and Geert Jan Houben. "Complex Event Processing on Real-time Video Streams." *In PhD@ VLDB. 2020.*

[2] * Authors have contributed equally to this work

SIKS DISSERTATION SERIES

2016
01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
04 Laurens Rietveld (VUA), Publishing and Consuming Linked Data
05 Evgeny Sherkhonov (UvA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
07 Jeroen de Man (VUA), Measuring and modeling negative emotions for virtual training
08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
09 Archana Nottamkandath (VUA), Trusting Crowdsourced Information on Cultural Artefacts
10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
11 Anne Schuth (UvA), Search Engines that Learn from Their Users
12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
13 Nana Baah Gyan (VUA), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
16 Guangliang Li (UvA), Socially Intelligent Autonomous Agents that Learn from Human Reward
17 Berend Weel (VUA), Towards Embodied Evolution of Robot Organisms
18 Albert Meroño Peñuela (VUA), Refining Statistical Data on the Web
19 Julia Efremova (TU/e), Mining Social Structures from Genealogical Data
20 Daan Odijk (UvA), Context & Semantics in News & Web Search
21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
22 Grace Lewis (VUA), Software Architecture Strategies for Cyber-Foraging Systems
23 Fei Cai (UvA), Query Auto Completion in Information Retrieval
24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach

25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
26 Dilhan Thilakarathne (VUA), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
30 Ruud Mattheij (TiU), The Eyes Have It
31 Mohammad Khelghati (UT), Deep web content monitoring
32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
33 Peter Bloem (UvA), Single Sample Statistics, exercises in learning from just one example
34 Dennis Schunselaar (TU/e), Configurable Process Trees: Elicitation, Analysis, and Enactment
35 Zhaochun Ren (UvA), Monitoring Social Media: Summarization, Classification and Recommendation
36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
40 Christian Detweiler (TUD), Accounting for Values in Design
41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
42 Spyros Martzoukos (UvA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
44 Thibault Sellam (UvA), Automatic Assistants for Database Exploration
45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
46 Jorge Gallego Perez (UT), Robots to Make you Happy
47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
48 Tanja Buttler (TUD), Collecting Lessons Learned
49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
50 Yan Wang (TiU), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

2017 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime
02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
05 Mahdieh Shadi (UvA), Collaboration Behavior
06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
08 Rob Konijn (VUA), Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
10 Robby van Delden (UT), (Steering) Interactive Play Behavior
11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
12 Sander Leemans (TU/e), Robust Process Mining with Guarantees
13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
14 Shoshannah Tekofsky (TiU), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
15 Peter Berck (RUN), Memory-Based Text Correction
16 Aleksandr Chuklin (UvA), Understanding and Modeling Users of Modern Search Engines
17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
18 Ridho Reinanda (UvA), Entity Associations for Search
19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
22 Sara Magliacane (VUA), Logics for causal inference under uncertainty
23 David Graus (UvA), Entities of Interest — Discovery in Digital Traces
24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning
25 Veruska Zamborlini (VUA), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
28 John Klein (VUA), Architecture Practices for Complex Contexts
29 Adel Alhuraibi (TiU), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"

30 Wilma Latuny (TiU), The Power of Facial Expressions
31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
35 Martine de Vos (VUA), Interpreting natural science spreadsheets
36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
37 Alejandro Montes Garcia (TU/e), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
38 Alex Kayal (TUD), Normative Social Applications
39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
46 Jan Schneider (OU), Sensor-based Learning Support
47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
48 Angel Suarez (OU), Collaborative inquiry-based learning

2018 01 Han van der Aa (VUA), Comparing and Aligning Process Representations
02 Felix Mannhardt (TU/e), Multi-perspective Process Mining
03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
05 Hugo Huurdeman (UvA), Supporting the Complex Dynamics of the Information Seeking Process
06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
10 Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology

11 Mahdi Sargolzaei (UvA), Enabling Framework for Service-oriented Collaborative Networks
12 Xixi Lu (TU/e), Using behavioral context in process mining
13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
14 Bart Joosten (TiU), Detecting Social Signals with Spatiotemporal Gabor Filters
15 Naser Davarzani (UM), Biomarker discovery in heart failure
16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
17 Jianpeng Zhang (TU/e), On Graph Sample Clustering
18 Henriette Nakad (UL), De Notaris en Private Rechtspraak
19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
20 Manxia Liu (RUN), Time and Bayesian Networks
21 Aad Slootmaker (OU), EMERGO: a generic platform for authoring and playing scenario-based serious games
22 Eric Fernandes de Mello Araújo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
26 Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
27 Maikel Leemans (TU/e), Hierarchical Process Mining for Scalable Software Analysis
28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
29 Yu Gu (TiU), Emotion Recognition from Mandarin Speech
30 Wouter Beek (VUA), The "K" in "semantic web" stands for "knowledge": scaling semantics to the web

---

2019 01 Rob van Eijk (UL),Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
03 Eduardo Gonzalez Lopez de Murillas (TU/e), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
04 Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
05 Sebastiaan van Zelst (TU/e), Process Mining with Streaming Data
06 Chris Dijkshoorn (VUA), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
07 Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
09 Fahimeh Alizadeh Moghaddam (UvA), Self-adaptation for energy efficiency in software systems

10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
12 Jacqueline Heinerman (VUA), Better Together
13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
16 Guangming Li (TU/e), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
17 Ali Hurriyetoglu (RUN),Extracting actionable information from microtexts
18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
21 Cong Liu (TU/e), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
22 Martin van den Berg (VUA),Improving IT Decisions with Enterprise Architecture
23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
24 Anca Dumitrache (VUA), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
25 Emiel van Miltenburg (VUA), Pragmatic factors in (automatic) image description
26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
27 Alessandra Antonaci (OU), The Gamification Design Process applied to (Massive) Open Online Courses
28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
29 Daniel Formolo (VUA), Using virtual agents for simulation and training of social skills in safety-critical circumstances
30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
31 Milan Jelisavcic (VUA), Alive and Kicking: Baby Steps in Robotics
32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
33 Anil Yaman (TU/e), Evolution of Biologically Inspired Learning in Artificial Neural Networks
34 Negar Ahmadi (TU/e), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
35 Lisa Facey-Shaw (OU), Gamification with digital badges in learning programming

36 Kevin Ackermans (OU), Designing Video-Enhanced Rubrics to Master Complex Skills
37 Jian Fang (TUD), Database Acceleration on FPGAs
38 Akos Kadar (OU), Learning visually grounded and multilingual representations

---

2020 01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
04 Maarten van Gompel (RUN), Context as Linguistic Bridges
05 Yulong Pei (TU/e), On local and global structure mining
06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
07 Wim van der Vegt (OU), Towards a software architecture for reusable game components
08 Ali Mirsoleimani (UL),Structured Parallel Programming for Monte Carlo Tree Search
09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
10 Alifah Syamsiyah (TU/e), In-database Preprocessing for Process Mining
11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data Augmentation-Methods for Long-Tail Entity Recognition Models
12 Ward van Breda (VUA), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
15 Konstantinos Georgiadis (OU), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
17 Daniele Di Mitri (OU), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences
18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
20 Albert Hankel (VUA), Embedding Green ICT Maturity in Organisations
21 Karine da Silva Miras de Araujo (VUA), Where is the robot?: Life as it could be
22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
24 Lenin da Nóbrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots
25 Xin Du (TU/e), The Uncertainty in Exceptional Model Mining

	26	Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization
	27	Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context
	28	Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
	29	Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
	30	Bob Zadok Blok (UL), Creatief, Creatiever, Creatiefst
	31	Gongjin Lan (VUA), Learning better – From Baby to Better
	32	Jason Rhuggenaath (TU/e), Revenue management in online markets: pricing and online advertising
	33	Rick Gilsing (TU/e), Supporting service-dominant business model evaluation in the context of business model innovation
	34	Anna Bon (UM), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development
	35	Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production
2021	01	Francisco Xavier Dos Santos Fonseca (TUD),Location-based Games for Social Interaction in Public Space
	02	Rijk Mercuur (TUD), Simulating Human Routines: Integrating Social Practice Theory in Agent-Based Models
	03	Seyyed Hadi Hashemi (UvA), Modeling Users Interacting with Smart Devices
	04	Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
	05	Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
	06	Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
	07	Armel Lefebvre (UU), Research data management for open science
	08	Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
	09	Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play
	10	Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
	11	Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
	12	Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
	13	Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
	14	Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
	15	Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
	16	Esam A. H. Ghaleb (UM), Bimodal emotion recognition from audio-visual cues
	17	Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks

18 Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
19 Roberto Verdecchia (VUA), Architectural Technical Debt: Identification and Management
20 Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems
21 Pedro Thiago Timbó Holanda (CWI), Progressive Indexes
22 Sihang Qiu (TUD), Conversational Crowdsourcing
23 Hugo Manuel Proença (UL), Robust rules for prediction and description
24 Kaijie Zhu (TU/e), On Efficient Temporal Subgraph Query Processing
25 Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
26 Benno Kruit (CWI/VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
27 Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You
28 Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs

---

2022 01 Judith van Stegeren (UT), Flavor text generation for role-playing video games
02 Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey
03 Ali el Hassouni (VUA), A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare
04 Ünal Aksu (UU), A Cross-Organizational Process Mining Framework
05 Shiwei Liu (TU/e), Sparse Neural Network Training with In-Time Over-Parameterization
06 Reza Refaei Afshar (TU/e), Machine Learning for Ad Publishers in Real Time Bidding
07 Sambit Praharaj (OU), Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics
08 Maikel L. van Eck (TU/e), Process Mining for Smart Product Design
09 Oana Andreea Inel (VUA), Understanding Events: A Diversity-driven Human-Machine Approach
10 Felipe Moraes Gomes (TUD), Examining the Effectiveness of Collaborative Search Engines
11 Mirjam de Haas (UT), Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring
12 Guanyi Chen (UU), Computational Generation of Chinese Noun Phrases
13 Xander Wilcke (VUA), Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge
14 Michiel Overeem (UU), Evolution of Low-Code Platforms

	15	Jelmer Jan Koorn (UU), Work in Process: Unearthing Meaning using Process Mining
	16	Pieter Gijsbers (TU/e), Systems for AutoML Research
	17	Laura van der Lubbe (VUA), Empowering vulnerable people with serious games and gamification
	18	Paris Mavromoustakos Blom (TiU), Player Affect Modelling and Video Game Personalisation
	19	Bilge Yigit Ozkan (UU), Cybersecurity Maturity Assessment and Standardisation
	20	Fakhra Jabeen (VUA), Dark Side of the Digital Media - Computational Analysis of Negative Human Behaviors on Social Media
	21	Seethu Mariyam Christopher (UM), Intelligent Toys for Physical and Cognitive Assessments
	22	Alexandra Sierra Rativa (TiU), Virtual Character Design and its potential to foster Empathy, Immersion, and Collaboration Skills in Video Games and Virtual Reality Simulations
	23	Ilir Kola (TUD), Enabling Social Situation Awareness in Support Agents
	24	Samaneh Heidari (UU), Agents with Social Norms and Values - A framework for agent based social simulations with social norms and personal values
	25	Anna L.D. Latour (UL), Optimal decision-making under constraints and uncertainty
	26	Anne Dirkson (UL), Knowledge Discovery from Patient Forums: Gaining novel medical insights from patient experiences
	27	Christos Athanasiadis (UM), Emotion-aware cross-modal domain adaptation in video sequences
	28	Onuralp Ulusoy (UU), Privacy in Collaborative Systems
	29	Jan Kolkmeier (UT), From Head Transform to Mind Transplant: Social Interactions in Mixed Reality
	30	Dean De Leo (CWI), Analysis of Dynamic Graphs on Sparse Arrays
	31	Konstantinos Traganos (TU/e), Tackling Complexity in Smart Manufacturing with Advanced Manufacturing Process Management
	32	Cezara Pastrav (UU), Social simulation for socio-ecological systems
	33	Brinn Hekkelman (CWI/TUD), Fair Mechanisms for Smart Grid Congestion Management
	34	Nimat Ullah (VUA), Mind Your Behaviour: Computational Modelling of Emotion & Desire Regulation for Behaviour Change
	35	Mike E.U. Ligthart (VUA), Shaping the Child-Robot Relationship: Interaction Design Patterns for a Sustainable Interaction
2023	01	Bojan Simoski (VUA), Untangling the Puzzle of Digital Health Interventions
	02	Mariana Rachel Dias da Silva (TiU), Grounded or in flight? What our bodies can tell us about the whereabouts of our thoughts
	03	Shabnam Najafian (TUD), User Modeling for Privacy-preserving Explanations in Group Recommendations
	04	Gineke Wiggers (UL), The Relevance of Impact: bibliometric-enhanced legal information retrieval

05 Anton Bouter (CWI), Optimal Mixing Evolutionary Algorithms for Large-Scale Real-Valued Optimization, Including Real-World Medical Applications
06 António Pereira Barata (UL), Reliable and Fair Machine Learning for Risk Assessment
07 Tianjin Huang (TU/e), The Roles of Adversarial Examples on Trustworthiness of Deep Learning
08 Lu Yin (TU/e), Knowledge Elicitation using Psychometric Learning
09 Xu Wang (VUA), Scientific Dataset Recommendation with Semantic Techniques
10 Dennis J.N.J. Soemers (UM), Learning State-Action Features for General Game Playing
11 Fawad Taj (VUA), Towards Motivating Machines: Computational Modeling of the Mechanism of Actions for Effective Digital Health Behavior Change Applications
12 Tessel Bogaard (VUA), Using Metadata to Understand Search Behavior in Digital Libraries
13 Injy Sarhan (UU), Open Information Extraction for Knowledge Representation
14 Selma Čaušević (TUD), Energy resilience through self-organization
15 Alvaro Henrique Chaim Correia (TU/e), Insights on Learning Tractable Probabilistic Graphical Models
16 Peter Blomsma (TiU), Building Embodied Conversational Agents: Observations on human nonverbal behaviour as a resource for the development of artificial characters
17 Meike Nauta (UT), Explainable AI and Interpretable Computer Vision – From Oversight to Insight
18 Gustavo Penha (TUD), Designing and Diagnosing Models for Conversational Search and Recommendation
19 George Aalbers (TiU), Digital Traces of the Mind: Using Smartphones to Capture Signals of Well-Being in Individuals
20 Arkadiy Dushatskiy (TUD), Expensive Optimization with Model-Based Evolutionary Algorithms applied to Medical Image Segmentation using Deep Learning
21 Gerrit Jan de Bruin (UL), Network Analysis Methods for Smart Inspection in the Transport Domain
22 Alireza Shojaifar (UU), Volitional Cybersecurity
23 Theo Theunissen (UU), Documentation in Continuous Software Development
24 Agathe Balayn (TUD), Practices Towards Hazardous Failure Diagnosis in Machine Learning
25 Jurian Baas (UU), Entity Resolution on Historical Knowledge Graphs
26 Loek Tonnaer (TU/e), Linearly Symmetry-Based Disentangled Representations and their Out-of-Distribution Behaviour
27 Ghada Sokar (TU/e), Learning Continually Under Changing Data Distributions

28 Floris den Hengst (VUA), Learning to Behave: Reinforcement Learning in Human Contexts
29 Tim Draws (TUD), Understanding Viewpoint Biases in Web Search Results

2024 01 Daphne Miedema (TU/e), On Learning SQL: Disentangling concepts in data systems education
02 Emile van Krieken (VUA), Optimisation in Neurosymbolic Learning Systems
03 Feri Wijayanto (RUN), Automated Model Selection for Rasch and Mediation Analysis
04 Mike Huisman (UL), Understanding Deep Meta-Learning
05 Yiyong Gou (UM), Aerial Robotic Operations: Multi-environment Cooperative Inspection & Construction Crack Autonomous Repair
06 Azqa Nadeem (TUD), Understanding Adversary Behavior via XAI: Leveraging Sequence Clustering to Extract Threat Intelligence
07 Parisa Shayan (TiU), Modeling User Behavior in Learning Management Systems
08 Xin Zhou (UvA), From Empowering to Motivating: Enhancing Policy Enforcement through Process Design and Incentive Implementation
09 Giso Dal (UT), Probabilistic Inference Using Partitioned Bayesian Networks
10 Cristina-Iulia Bucur (VUA), Linkflows: Towards Genuine Semantic Publishing in Science
11 Mahmoud Shokrollahi-Far (TiU), Computational Reliability of Quranic Grammar
12 Peide Zhu (TUD), Towards Robust Automatic Question Generation For Learning
13 Enrico Liscio (TUD), Context-Specific Value Inference via Hybrid Intelligence
14 Larissa Capobianco Shimomura (TU/e), On Graph Generating Dependencies and their Applications in Data Profiling
15 Ting Liu (VUA), A Gut Feeling: Biomedical Knowledge Graphs for Interrelating the Gut Microbiome and Mental Health
16 Arthur Barbosa Câmara (TUD), Designing Search-as-Learning Systems
17 Razieh Alidoosti (VUA), Ethics-aware Software Architecture Design
18 Laurens Stoop (UU), Data Driven Understanding of Energy-Meteorological Variability and its Impact on Energy System Operations
19 Azadeh Mozafari Mehr (TU/e), Multi-perspective Conformance Checking: Identifying and Understanding Patterns of Anomalous Behavior
20 Ritsart Anne Plantenga (UL), Omgang met Regels
21 Federica Vinella (UU), Crowdsourcing User-Centered Teams
22 Zeynep Ozturk Yurt (TU/e), Beyond Routine: Extending BPM for Knowledge-Intensive Processes with Controllable Dynamic Contexts
23 Jie Luo (VUA), Lamarck's Revenge: Inheritance of Learned Traits Improves Robot Evolution
24 Nirmal Roy (TUD), Exploring the effects of interactive interfaces on user search behaviour
25 Alisa Rieger (TUD), Striving for Responsible Opinion Formation in Web Search on Debated Topics

26 Tim Gubner (CWI), Adaptively Generating Heterogeneous Execution Strategies using the VOILA Framework
27 Lincen Yang (UL), Information-theoretic Partition-based Models for Interpretable Machine Learning
28 Leon Helwerda (UL), Grip on Software: Understanding development progress of Scrum sprints and backlogs
29 David Wilson Romero Guzman (VUA), The Good, the Efficient and the Inductive Biases: Exploring Efficiency in Deep Learning Through the Use of Inductive Biases
30 Vijanti Ramautar (UU), Model-Driven Sustainability Accounting
31 Ziyu Li (TUD), On the Utility of Metadata to Optimize Machine Learning Workflows
32 Vinicius Stein Dani (UU), The Alpha and Omega of Process Mining