

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Don't look away! Keeping the human in the loop with a interactive active learning platform

Fábio Manuel Taveira da Cunha

Master in Electrical and Computer Engineering

Supervisor: Ana Filipa Sequeira, PhD

Co-Supervisor: Jaime dos Santos Cardoso, PhD

October 31, 2023

Abstract

Machine Learning Operations (MLOps) offers excellent possibilities for automation, development, version control, testing, model training, retraining and monitoring that are perfect not only for user interaction, but for developers and Machine Learning (ML) engineers to create, update and maintain predictive models. Active Learning is a branch of ML that involves humans in the training process of ML models to increase the accuracy of predictions by requesting feedback for queries chosen by it. This document proposes a MLOps system that focus on combining these two concepts providing the user with an interactive platform where they can serve as a feedback provider during the training of ML models, while trying to improve the Replicability and Reproducibility aspects of ML research and development, often ignored or neglected, by introducing a structured and organized view of the training process, models and datasets used. Two classification problems were studied using the developed tool, revealing that active learning is not always advantageous in this specific form, leading to the need to develop more generic solutions that encompass a wider range of classification problems that deal with their inherently specificities, but also the need to investigate the effect of the Oracle's subjectivity.

Keywords: Active Learning, MLOps, Human Feedback

Resumo

Machine Learning Operations (MLOps) possibilita excelentes possibilidades para a automação, desenvolvimento, controlo de versões, testagem, treino de modelos, retreino e monitorização que são perfeitas para não só interação com utilizadores, mas também desenvolvedores e engenheiros de Machine Learning (ML) para criar, atualizar e manter modelos preditivos. Active Learning é um ramo de ML que envolve humanos no processo de treino de modelos de ML para aumentar a precisão das predições, pedindo feedback acerca de dúvidas na classificação escolhidas pelo modelo. Este documento propõe um sistema MLOps que se centra na combinação destes dois conceitos oferecendo ao utilizador uma plataforma interativa onde ele pode servir como provedor de feedback durante o treino de modelos de ML, tentando ao mesmo tempo melhorar os aspetos de Replicabilidade e Reprodutibilidade nos contextos de investigação e desenvolvimento de ML, muitas vezes ignorados ou negligenciados, introduzindo uma vista do processo de treino, modelos e dados usados estruturada e organizada. Dois problemas de classificação foram estudados usando a ferramenta desenvolvida, revelando que active learning não é sempre vantajoso nesta específica forma, levando à necessidade de desenvolver soluções mais genéricas que abranjam um maior número de problemas de classificação que lidem com as suas especificidades inerentes, mas também a necessidade de investigar os efeitos da subjetividade do Oracle.

Acknowledgements

A big thanks to my supervisor in this work, Ana Filipa Sequeira, for not only giving me the opportunity to take this project on, but also being patient and flexible to deal with the restrictions my professional life implies. The same gratitude is extended to Pedro Carneiro Neto for adapting the initial goals to others more relevant to my profile and guiding my work strategy in the right direction.

To my family. My parents, for instilling in me values worth protecting; my brother, for being a beacon 5 years ahead of me; my grandparents, that ignited the hope for a better life for their progeny.

To Olena, for, even perhaps unknowingly, reminding me of who I was and rekindling the will to be it once more.

Fábio Cunha

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Goals	2
1.3	Document Structure	2
2	Background	3
2.1	Machine Learning fundamental concepts	3
2.1.1	Convolutional Neural Networks	6
2.2	Biometric Systems	10
2.3	Human-in-the-Loop	14
2.3.1	Explainable AI	14
2.3.2	Active Learning	15
2.4	Replicability and Reproducibility	16
2.5	Machine Learning Operations	18
3	State of the Art	21
3.1	Machine Learning Operations	21
3.1.1	MLOps Architecture	24
3.1.2	ML Tools	27
3.1.3	Real-world MLOps example	32
3.2	Active Learning	35
3.2.1	Active Learning Tools	36
4	Platform Architecture	39
4.1	Data Model	39
4.2	Backend	41
4.2.1	Presentation, Service and Repository Pattern	42
4.2.2	Authentication	46
4.2.3	REST API	46
4.2.4	Concurrent Pipeline Runs Execution	51
4.2.5	Environment variables	52
4.2.6	Pipeline Steps	53
4.3	Frontend	58
4.3.1	Log In and Sign Up	59
4.3.2	Pipelines	59
4.3.3	Pipeline Edit View	59
4.3.4	Runs View	60
4.3.5	Pipeline Runs View	60

4.3.6	Run Results View	60
4.3.7	Feedback View	60
4.3.8	Single Feedback Query View	60
4.3.9	Datasets Handlers, Trainers, Testers and Trained Models Views	61
5	Automated workflow examples	69
5.1	HITL logic	69
5.2	Face Spoofing	69
5.2.1	ROSE-Youtu	70
5.2.2	Model and Training	71
5.2.3	Results	71
5.3	Pornographic content detection	72
5.3.1	PornographyXXX	72
5.3.2	Model and Training	73
5.3.3	Results	73
6	Conclusions	75
6.1	Challenges	75
6.2	Future Work	76
	References	79

List of Figures

2.1	Variability in handwritten "e" characters	3
2.2	Examples of objects from ImageNet's training dataset images used in the ImageNet Object Localization Challenge.	7
2.3	Sparse connectivity representation with a kernel width of 3	8
2.4	Typical composition of a CNN layer.	11
2.5	Example of max pooling with filter size 2 and stride size 2.	11
2.6	The three main active learning scenarios	15
2.7	Reproducible Research	18
3.1	Intersection of topics that define MLOps	21
3.2	Oravizio's MLOps Implementation	34
4.1	Backend Application Layers	42
4.2	Services Layer Part 1	44
4.3	Services Layer Part 2	45
4.4	Repository Layer Part 1	47
4.5	Repository Layer Part 2	48
4.6	Authentication tokens generation.	49
4.7	Asynq architecture	52
4.8	Step and Edge interfaces	55
4.9	Pipeline Definition Unmarshalling	56
4.10	Steps Execution Logic	62
4.11	Log In and Sign Up Pages	63
4.12	Client token handling	63
4.13	Pipelines View	63
4.14	Pipeline Edit View	64
4.15	Runs View	65
4.16	Pipeline Runs View	65
4.17	Run Results View	66
4.18	Feedback View	67
4.19	Single Query Feedback View	68
4.20	Log In and Sign Up Pages	68
4.21	Views	68
5.1	Face Spoofing HITL vs Automatic training comparision.	71
5.2	Pornographic content detection HITL vs Automatic training comparision.	73

List of Tables

3.1	Intersection between MLOps Components and Principles.	23
3.2	ML Tools by Class and License Type.	32

Abbreviations and Symbols

FR	Facial Recognition
ML	Machine Learning
MLaaS	ML as a Service
MLOps	Machine Learning Operations
CNN	Convolutional Neural Network
DL	Deep Learning
SE	Software Engineering
DE	Data Engineering
HDFS	Hadoop Distributed File System
ANN	Artificial Neural Network
GBM	Gradient Boosting Machine
FNN	Feed-forward Neural Networks
DNA	Deoxyribonucleic acid
SOTA	State of the Art
CI	Continuous Integration
CD	Continuous Delivery
CT	Continuous Training
EDA	Exploratory data analysis
RDBMS	Relational database management system
CRUD	Create, Read, Update and Delete
JSON	JavaScript Object Notation
JWT	JSON Web Token
NCI	American National Cancer Institute
HTTP	Hypertext Transfer Protocol
OS	Operating System
DB	Database
SSL	Secure Sockets Layer
Java EE	Enterprise Edition
HITL	Human in the Loop
CSV	Comma-separated values
MIT	Massachusetts Institute of Technology
UI	User Interface
CPU	Central Processing Unit
GPU	Graphics Processing Unit

Chapter 1

Introduction

This chapter presents the motivation, goals and structure for this dissertation document.

1.1 Motivation

In order to facilitate the development, integration, testing, releasing, deployment and infrastructure management of ML systems, a new philosophy, born from the idea to apply DevOps to this area of software development, is becoming popular - MLOps [1]. Using MLOps to enable to quick and efficient deployment, maintenance and retraining of ML models that fit the user's needs. At the same time, Active Learning is an evolving branch of ML that aims to overcome difficulties that an ML model has in giving accurate predictions by requesting the feedback of a human for queries generated by it [2]. Combining these two concepts into an interactive application platform is a great opportunity to leverage the advantages of workflow automation and management present in the MLOps philosophy to provide a smoother, more interactive, understandable and organized experience to the end user from whom the feedback is requested. Additionally, Replicability and Reproducibility within the field of ML has been neglected and consequently has become an increasingly bigger problem [3, 4], with researchers having difficulty to reproduce the work of others and extending the same approach to related problems with different real-world conditions. The implementation and usage of a structured tool inspired by MLOps can provide a systematic way of using ML models, contributing to an easier achievement of these two essential research work characteristics. All of these context pieces come together to provide an opportunity to develop an MLOps inspired tool that tries to answer the problems where active learning might offer performance gains while at the same time ensuring the replicability and reproducibility of the experiments ran. These problems might be related to several areas of ML research, such as Facial Recognition (FR) or video content age-related classification, as these two examples or indeed briefly studied in this work.

1.2 Goals

The main goals of combining Active Learning with MLOps within the context of this project, are:

- Goal 1 - Understand what is the ML tools and Active Learning state-of-the-art and how they overlap.
- Goal 2 - Build a basic web application platform that stores, in an organized manner, ML Pipelines and allows their scheduled or on-demand execution (Runs).
- Goal 3 - Modify the Active Learning solution proposed in [2] to enable its staggered execution, that is, enable it to be paused and await for feedback that is given by an application user, before resuming its execution.
- Goal 4 - Introduce support for generic *PyTorch* model Trainer and Tester ML algorithms, Dataset Handler (processing scripts) entities and storage and usage of Trained Models.

The code is publicly available in a GitHub repository¹.

1.3 Document Structure

The following chapters are present in this document:

- Chapter 2 - This document begins by explaining and detailing at some length essential background knowledge to understand the topics, purpose and the work in the context of the Dissertation subject. We begin by describing fundamental concepts related to ML in general, followed by a brief contextualization of how Convolutional Neural Neural Networks came to be popular in ML and how they work. Biometric systems are described next, more specifically, what defines them, their practical applications and what human characteristics can be used to power them. Then, the topic of Biometrics is presented, its constituent parts, the effect training and testing datasets have on it. To close the background section, a description of MLOps as DevOps applied to ML is made along with the differences to simple DevOps.
- Chapter 3 - Includes a State of the Art regarding the common accepted different levels of MLOps as well one relevant Human-in-the-Loop (HITL) Active Learning.
- Chapter 4 - The developed Platform's Architecture is presented, separated in two main sections - Backend and Frontend.
- Chapter 5 - Two experiments using two different datasets are described, trying to figure out if HITL affect the training process.
- Chapter 6 - A discussion of the learnings from the development work made is presented, highlighting the challenges and proposing point for future improvements and work.

¹<https://github.com/up201105402/di>

Chapter 2

Background

Pattern finding in data is a recurring problem with a long history. Its different attempts at solving spawned findings useful in many fields, from Tycho Brahe's observations in the 16th century that later allowed Johannes Kepler to find the laws of planetary motion which were fundamental for the development of classical mechanics, to something more mundane, practical and recent like the automated recognition of printed text [5, p. 1].

2.1 Machine Learning fundamental concepts

The type of problems like the latter one presents as its main challenge the variability of the input data (the images of each printed character), specially if we're talking about hand printed text. One could define a set of rules to try to uniquely identify each one of the characters on the text; but that would inevitably lead to a potentially unmanageable amount of rules in the attempt to tame the great number of slight variations on each base shape. This is where a ML approach provides significant advantages over the previously described one [5, p. 2].



Figure 2.1: Variability in handwritten "e" characters. ¹

A distinctive characteristic of a ML system is the ability to learn from data; learning, in this context, can be defined by: "A computer program is said to learn from experience E with respect to

⁰Adapted from <https://www.kaggle.com/datasets/dhruvildave/english-handwritten-characters-dataset>

some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ." [6, p. 2] In this definition, the tasks T are what the ML algorithm is trying to achieve, like identifying each handwritten character correctly in our previous example. One can identify several common types of tasks [7, p. 98-101]:

- **Classification** - assigning one of k category to an input vector $\mathbf{x}$ using a function f such that: $f : \mathbb{R}^n \rightarrow \{1, \dots, k\}$ or, alternatively, having f return a probability distribution over the categories.
- **Classification with missing inputs** - Same as the previous type but instead of one function f , several functions are used to classify the input vector $\mathbf{x}$ - one for each subset of missing values of $\mathbf{x}$.
- **Regression** - like classification but the output is real number as we're predicting a value given an input vector $\mathbf{x}$ using a function f such that: $f : \mathbb{R}^n \rightarrow \mathbb{R}$.
- **Transcription** - consists in, literally, transcribing some unstructured representation of data into text form (e.g. transcribing pieces of text present in photographs, or speech recognition on audio files and its transcription into text form).
- **Machine Translation** - literally translating a sequence of symbols in one language into another sequence of symbols in another language. This is useful to translate between human languages (e.g. English to Spanish and vice versa).
- **Structured Output** - any task that produces a vector as an output (or other data structure with multiple values) with relevant relationships between the different elements. This is a big type of tasks that contains the transcription, translation and many others.
- **Anomaly detection** - sifting through a set of events or objects and selecting the unusual or strange ones (by some criteria), like detecting credit card fraud.
- **Synthesis and sampling** - generating new data that is similar to training data. Useful when it would be expensive, time consuming or boring to manually generate lots of data (e.g. speech synthesis, that is, generating an audio file with a spoken version of a written sentence).
- **Imputation of missing values** - consists in predicting the missing x_i values from an input $\mathbf{x} \in \mathbb{R}^n$.
- **Denoising** - similar to the previous type (Imputation of missing values), but here the aim is to predict the correct $\mathbf{x}$ or $p(\mathbf{x}|\tilde{\mathbf{x}})$ before a corruption process generated the $\tilde{\mathbf{x}}$ that is inputted into the algorithm.
- **Density estimation or probability mass function estimation** - the algorithm is made to learn a probability density or mass function p_{model} such that: $p_{model} : \mathbb{R}^n \rightarrow \mathbb{R}$.

The performance measure P of a system is often related to its task T ; for classification, classification with missing inputs and transcription it is accuracy or error rate (correct or incorrect number of outputs for the total examples inputted) that we measure. In a general sense, it is how the system will perform in a real setting that is of most interest, motivating the use of *test sets* independent from the *training sets* to evaluate these performance measures [7, p. 102].

With regards to the experience E , there's two loose classes of algorithms based on how *datasets* - data collections of examples, also referred as *data points* - are experienced and its features extracted. These features are extracted by *Preprocessing* or *feature extraction*. This helps improve the performance of the whole model by transforming the input data - both the *training set* and the *testing set* - into a new variable space (e.g. scaling and cropping the character images), reducing intra-class variability and helping reveal more relevant and precise inter-class differences. Another way performance may be positively affected by this process is by reducing the overall size of input data into a smaller, more manageable, size while retaining a satisfactory ability to correctly classify new testing input samples [5, p. 2]. These two classes are: [7, p. 102-103]:

- **Unsupervised** - when the algorithm learns relevant properties belonging to the input data through features extracted from the dataset. In practise, this means learning the probability distribution $p(\mathbf{x})$ of an input vector $\mathbf{x}$ by observing several examples of it or some other relevant properties of that distribution.
- **Supervised** - when the algorithm also learns relevant properties belonging to the input data through features extracted from the dataset but in this case, each example on the dataset is also associated with a label that identifies what it is that we want to identify that is represented. In practise, this means learning to predict the vector $\mathbf{y}$ that an input $p(\mathbf{x})$ should originate in the system, by observing several examples of vector pairs; this can be done by estimating the probability: $p(\mathbf{y}|\mathbf{x})$. That is why it is called supervised: the algorithm is shown what the correct prediction based on examples and is expected to expand this to new input data.

Despite of the fact that the training data, and consequently the training datasets should in theory encompass the biggest number of examples possible, most of the times it is impossible to produce a dataset that represents all possible inputs; the ability for the algorithm to perform well when faced with inputs unrepresented in the training dataset is referred as *Generalization* [7, p. 108]. This is what separates a ML problem from a simple optimization problem, as besides defining a *training error* - an error measure for input data from training dataset - we define a *test error* or *generalization error* usually measuring the system's performance on a *test set* of data that was collected independently of the training set. Despite of this independance, one can still improve the test error if we improve the training error, which is controllable because the algorithm is trained by it. This is true if the *i.i.d. assumptions* [7, p. 109] are considered; these stand for the independent and identically distributed and it is applied to both the training and testing sets. The identically distributed aspect refers to the assumption that both sets are generated by the same probability

distribution as each other. It is still, related to both the training and test errors that two important problems can arise in ML that should be avoided [7, p. 109-110]:

- **Underfitting** - when the algorithm is not able to achieve a low enough training error.
- **Overfitting** - when the gap between the training and test errors is too big.

An algorithm's **capacity** controls its probability to underfit or overfit; a low capacity means the algorithm struggles to fit the training set but on the other hand a high capacity might mean the algorithm will overfit by memorizing the training set and not perform well with new data.

The development of Deep Learning (DL) was motivated in part by the limited ability of traditional machine learning algorithms to achieve good generalization on complex tasks such as speech and object recognition, that usually deal with data in high-dimensional spaces that often imply high computational costs. As the number of data dimensions grows, the number of possible input values for $\mathbf{x}$ grows much quicker than the examples in the number of training examples that is feasible to find; this makes the job of the algorithm much more difficult as the probability that there isn't a meaningful training example to draw from is much higher [7, p. 152-153].

2.1.1 Convolutional Neural Networks

Regression and simple classification problems usually make use of linear combinations of basis functions to produce good results; however, to adapt the same models to much bigger scale ones - like FR systems - the basis functions need to be modified appropriately [5, p. 225].

Before AlexNet [8], object recognition systems made use of machine learning methods whose higher performance for realistic input probing data depended on trying to procure larger datasets (which was a challenge as the existing ones were lacking in total number of images, typically only in the tens of thousands [9, 10, 11]) than the ones suitable for smaller scale problems like the previously described character recognition problem, or augment the already existing datasets in a way that still maintains the existing image labels (e.g. an error rate $<0.3\%$ can be obtained on the MNIST dataset [12]). That began to change with the appearance of datasets like LabelMe [13] (hundreds of thousands of images) and ImageNet [14] (15 million images, of which some are displayed in Figure 2.2); still the complexity of object recognition problems was not fully appeased by comparable datasets to ImageNet and so, AlexNet showed that importing prior knowledge can help close this flaw of not having enough training data to adequately account for every possible variation.

This is where CNNs come in as a great candidate that fulfils all the demands to produce accurate solutions for object recognition problems, ahead of the previously more successful Feed-forward Neural Networks (FNNs) [5, p. 225]. That is because when comparing CNNs to FNNs with similarly-sized layers, the first have are easier to train (less connections and parameters) while having only a slightly lower theoretically best performance [15]. Currently state-of-the-art (SOTA) FR systems [16, 17, 18] use CNNs because of the advantages just described.

²Adapted from Figure 5.5 [7, p. 109]

³Adapted from <https://www.kaggle.com/competitions/imagenet-object-localization-challenge>

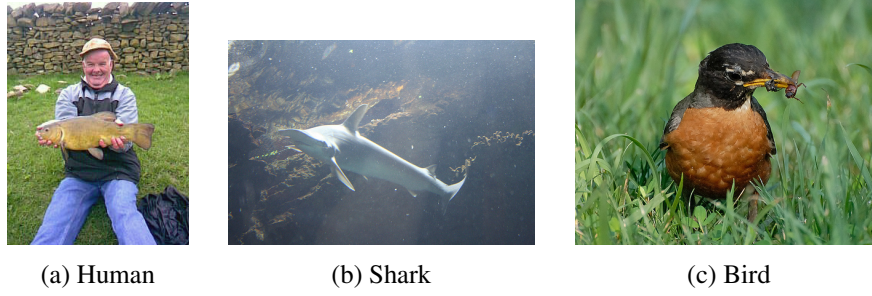


Figure 2.2: Examples of objects from ImageNet's training dataset images used in the ImageNet Object Localization Challenge. ³

CNNs are a specialized type of neural network for processing grid formatted data - like a two dimensional grid of pixels that make up images - that use **convolution** in at least one of their layers [7, p. 326]. Convolution is an operation defined as:

$$s(t) = \int x(a)w(t-a)da \quad (2.1)$$

But is usually expressed as (for continuous functions in t):

$$s(t) = (x * w)(t) \quad (2.2)$$

With the first argument - x called the **input**, the second argument - w called the **kernel** and the output - $s(t)$ - the **feature map**. For a discrete variable t , (2.1) becomes:

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \quad (2.3)$$

When convolution is applied to ML, x is usually a multidimensional array of data and w is usually, again, a multidimensional array but of parameters that are to be tuned by the algorithm; these multidimensional arrays are often referred as **tensors** [7, p. 328]. Because of the multidimensional arrays, we can extend (2.3) to account for this; for the 2 dimensional case - where I is, for example, an image - it becomes:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i-m, j-n) \quad (2.4)$$

It is common for (2.4) to be implemented as another equivalent function named **cross-correlation**, taking advantage of convolution's commutative property:

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i+m, j+n)K(m, n) \quad (2.5)$$

Convolution implementation alternatives that are more efficient computationally wise are an active subject of research; some of these established alternatives include: [7, p. 356]:

- Apply the Fourier transform to both the input and kernel, performing point-wise multiplication of the two signals and applying the inverse Fourier transform to the output.
- For kernels that can be expressed as the outer product of d vectors (separable kernels) composed convolution is faster than the naive computation (2.3).

Convolutional networks were some of the first deep models to have good enough performance to solve important commercial applications in the past [19] - check reading in 1990s, OCR and hand-written recognition - and continue to be relevant today a lot because of AlexNet [8]. Convolution makes use of important aspects within the context of ML:

- **Sparse interactions** - contrary to traditional neural networks where every output and input interact in each layers (through matrix multiplication by a matrix of parameters with a separate parameter that describes that input-output relationship), CNNs rely on sparse interactions, meaning that the kernel is smaller than the input and we need to store less parameters, saving on memory and computational resources [7, p. 330]. Figure 2.3 visually depicts the connections between the inputs and outputs for a kernel with width equal to 3. It is immediately apparent that there are fewer connections and consequently less calculations and computational requirements than if s was obtained through matrix multiplication in which case every input x_i will be connected to every output s_j .

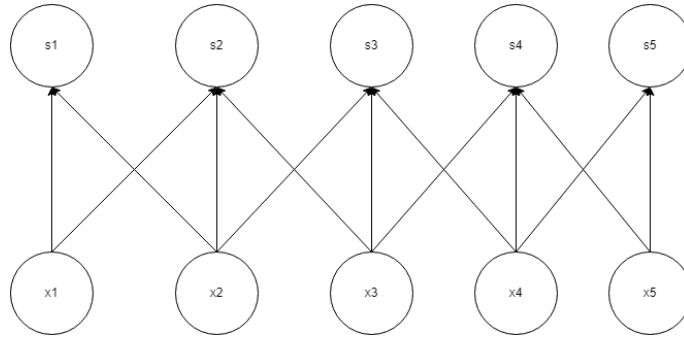


Figure 2.3: Sparse connectivity representation with a kernel width of 3. ⁴

- **Parameter sharing** - in matrix multiplication each parameter is used exactly once when computing the output of each layer. Again, conversely, parameters or members of the kernel in a CNN are used on every input, giving us another reason for not needing to learn and store as many parameters saving on their storage [7, p. 331-334].
- **Equivariant representations** - parameter sharing in convolution causes CNNs layers to show equivariance to translation, that is, a function is equivariant if the output changes the same way as the input changes [7, p. 334]:

$$f(g(x)) = g(f(x)) \quad (2.6)$$

⁴Adapted from Figures 9.2, 9.3 and 9.4 [7, p. 331-332].

Which means that if g is a function that transforms the input in some way, shifting it (but not changing its scale or rotating it [7, p. 335]), then the convolution result s is equivariant to g . To use a practical example, if there's a function g that shifts every pixel of an image I one unit to the right such that:

$$I'(x, y) = g(I) = I(x - 1, y) \quad (2.7)$$

Then:

$$(g(I) * K) = g(I' * K) \quad (2.8)$$

That is, it is equivalent to apply the shifting transformation to I and then perform the convolution or perform the convolution with I' and then apply the shifting transformation.

A CNN layer has usually 3 stages (Figure 2.4) [7, p. 335-339]:

- **Convolution stage** - where several concurrent convolution calculation are made that produce a set of linear activations. Convolution, within the context of CNNs, actually usually consists of several parallel convolution operations because different features require different kernels to be extracted appropriately [7, p. 342]. Also, it is usually not the mathematical definition of discrete convolution that is used (2.3), but some variation on it; some examples include what MATLAB calls valid, same and full convolution (different particular cases of zero padding the input), unshared convolution [20, 21], tiled convolution [22, 23].
- **Detector stage** - where the result of the convolution stage is fed to a nonlinear activation function (e.g. rectified linear activation function).
- **Pooling stage** - where the result of the detector stage is fed to a pooling function, which replaces the output of the net a certain location with a summary statistic of neighbouring outputs. It helps to make the output approximately invariant to small translations in the input, which is useful for applications where the presence of a certain feature is more important than its location within the input - for example, in a FR system it is more important to know of the existence of eyes, nose, mouth and other facial elements rather than their exact location within an image. A popular pooling function is **max pooling**, represented in Figure 2.5; this function takes the maximum value of, if we follow the typical model for CNN layer, the output after the convolution and detector stages and picks the maximum value for each region defined by the **filter size** and **stride size** - set, in the example, as 2 for both. The stride value determines the number columns or rows the filter mask of size 2x2 (as the filter size is 2 as well) skips left to right, top to bottom over the input matrix; each possible superposition of the filter mask on the input matrix given this description is a region from which we want to extract the max value. As the pooling result is smaller in dimensions than the input, the means that the next layer of the CNN will have a smaller size input, reducing storage and computational resources needs ahead. Pooling is also useful to

process and normalize inputs of different sizes. This can be accomplished by, for example, having the pooling operation output one summary value for each quadrant of an image, no matter its size. What types of pooling to use in each situation is not readily obvious, however, there has been theoretical work on it that provides some answers [24].

A **prior** is probability distribution over the parameters of a model that tells what models are reasonable before any data has been observed. An infinitely strong prior places makes some parameters impossible (zero probability) no matter how much the data might support these values. A distribution's entropy determines if it constitutes a weak or strong prior: high entropy (or high variance) means a weak prior, the opposite is true for low entropy (or low variance).

High variance distributions allow the data to move the parameters almost freely; low variance are more important in determining the parameters' values. Convolution introduces an infinitely strong prior probability distribution for the parameters of a layer and imparts in it the already described characteristics of local (sparse) interactions and equivariance to translation; in the same manner pooling is an infinitely strong prior as well and each unit should be invariant to small translations [7, p. 341]. Because of these facts, convolution and pooling can cause underfitting as with any prior they're only useful if their assumptions are reasonable. For applications where feature spatial location is important using pooling can increase the training error; pooling can be implemented on a feature to feature basis avoiding unwanted training error increases.

CNNs can output a class label for classification problems, a scalar real value for regression problems or even a tensor $\mathbf{S} = S_{i,j,k}$, representing the probability that - for an image recognition problem for example - the pixel with coordinates (j,k) belong to class i , allowing to label every pixel and outline objects present in the images [7, p. 352]. CNNs can also handle inputs of varying spatial extents, like images with a wide range of resolutions [7, p. 354].

2.2 Biometric Systems

Biometrics or biometric recognition is defined as the "automated recognition of individuals based on their biological and behavioural characteristics" [25]. Accordingly, a biometric system is defined as a "system for the purpose of the biometric recognition of individuals based on their behavioural and biological characteristics" [25]. These systems have had wide range applications, starting from law enforcement related forensics [26] to the more currently relevant automated border control [27] and general user authentication [28]. The characteristics that qualify to be used within the context of biometric recognition should satisfy each and every of the following criterion [29]:

- **Universality:** every person subject to the system must display the characteristic.
- **Distinctiveness:** any two subjects should display the same characteristic in a manner sufficiently different so that it can be used to distinguish between them.

⁵Adapted from Figure 9.7 [7, p. 336]

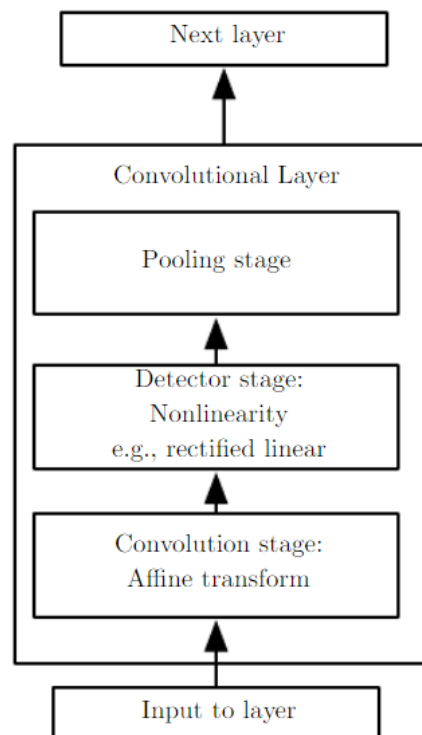
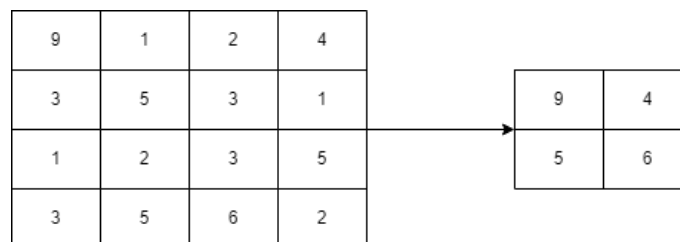
Figure 2.4: Typical composition of a CNN layer. ⁵

Figure 2.5: Example of max pooling with filter size 2 and stride size 2.

- **Permanence:** the characteristic should be displayed in a sufficiently constant state, in a manner dependant on certain distinguishing criteria, in an appropriate period of time.
- **Collectability:** the characteristic can be quantified and measured accordingly.

Additionally, biometric systems usually require more criterion to be considered when determining if a characteristic is eligible to be used [29]:

- **Performance:** the performance that the used characteristic can achieve within a biometric recognition system measured in quantities such as accuracy and speed, and how the resources and operational and environmental factors influence them.
- **Acceptability:** the willingness for people to accept the usage of the chosen characteristic in their daily lives.

- **Circumvention:** how easy it is to fool a biometric system that works based on this characteristic.

Several biometric characteristic can be used with biometric systems; no characteristic will meet all the requirements of all practical applications, each one has its advantages and disadvantages. The choice of which characteristic to use must take into account this at the same time as the previously mentioned criteria. Examples of characteristics include [29]:

- **DNA:** Deoxyribonucleic acid (DNA) is the most specific human identifier as everyone - except twins - have a unique DNA code. In a practical context though, its usage cases are limited by its easiness of forgery (one can steal a piece of another person's DNA, for example a hair, rather easily), the lengthiness of the current techniques to process and match DNA samples and the privacy concerns associated with storing and using (mainly misusing actually) a piece of such delicate and private information that DNA represents. Because of all of this, DNA biometrics is used most popularly used in the field of forensics.
- **Ear:** Although there is still work to be done, it has been posited that the human ear's shape might possess some distinctive aspects that can be used within a biometrics system.
- **Face:** Probably the most commonly used characteristic due to the non-intrusive nature of the associated biometric systems, the human face is a great candidate to be used for human identification. The most commonly approaches for FR are the using the shape, absolute and relative location of facial element like the mouth, nose, eyes, chin; and the comparison between the overall analysis of a face image and a weighted average of several canonical faces. Some limitations are usually imposed on FR systems though, namely the privacy concerns mainly regarding the collection, storing, securing and usage of facial images; the technical requirements regarding the images collected, concerning things like lighting, background, poses. The latter is an important issue with FR systems, they can struggle to recognize faces under certain lighting conditions, poses and other aspects that introduce variability into how a person's face might look like.
- **Facial, hand, and hand vein infrared thermogram:** Most of the times impractical to use in real world applications due to the prohibitively expensive nature of the technology associated required, the pattern of heat that human bodies radiate can be used as a distinguishable characteristic using infrared sensors to non-invasely capture it.
- **Fingerprint:** Probably the longest still used human characteristic for biometrics, fingerprints are still a very popular form of biometrics due its ability to provide very high identification accuracy - even twins have different fingerprint structures as they are formed during the first seven months of fetal development. Nowadays, fingerprint identification is done using methods much more advanced than in the past, like fingerprint scanners that are affordable for most applications and can even be embedded with other systems like laptops. Despite all this, fingerprint systems can require a lot of computational resources and are sometimes

not suitable for the small percentage of people whose fingerprints are not discernable or are damaged.

- **Gait:** The way or pattern a person moves their limbs while moving (e.g. walking, running) is a complex spatio-temporal biometric. Although not very distinctive, it is sufficiently distinguishable to be used in low-security applications. Because it is dependant on behaviour more than other physical characteristics it may vary over time due to many reasons (e.g. weight loss or gain, physical or brain injuries, inebriation). Like FR, it is a non-intrusive method of recognition. Because it involves processing video and analyzing the relative movement of several separate body parts, it is computationally heavy.
- **Hand and finger geometry:** The shape, size and various lengths within the human hand (e.g. finger length and width, palm dimensions) can be used for biometric systems. These systems are easy to use and relatively inexpensive, and do not appear to be affected by environmental factors like the weather or dry skin, unlike hand jewelry, challenges in dexterity and the ever changing hand appearance during the developing years of every human. The devices used for this type of biometrics are similar to the ones used for fingerprints but are usually much larger and thus less practical. Additionally, these characteristics are not distinctive enough to allow scaling up to a big number of system users.
- **Iris:** the region of the eye between the pupil and the sclera - the iris - is formed during fetal development and first two years of life and is very distinctive in each individual. It allows promising accuracy and speed on recognition systems is a good choice for large-scale (user wise) systems, as even identical twins have different irises and spoof attempts with artificial irises are easy to detect. Finally, although theses systems are still expensive to implement, they have become affordable recently.
- **Keystroke:** the particular way a person types can be used to identify them. Understandably this is literally unique to every person but it may offer sufficient discriminatory power to allow for identity verification. Similarly to a person's gait, the keystroke behaviour may vary but on the other hand it can be monitored in a non-obtrusive manner.
- **Odor:** every person exudes a distinct body odor that is unique in its chemical composition and thus can be used for recognition. It is although not clear yet if deodorant, perfume or environmental circumstances may affect the effectiveness of a system based on body odor.
- **Palmprint:** in the same line of thinking for fingerprint recognition, the human palm's patterns can be used for recognition, possibly with an even bigger degree of distinctiveness as palms are bigger than fingerprints - which in turn makes palmprint scanners more cumbersome and less practical. On the other hand the bigger surface area allows for scanners with smaller imagery resolution which lowers their cost.
- **Retinal scan:** the retinal blood vessels form a complex and unique structure in each person and each eye, so much so that it is claimed that they're the most secure characteristic to use

with biometric systems as in addition, it is difficult to replicate and or/change this structure. In addition to user authentication applications, retinal scan biometric systems can help reveal some medical conditions. However, they require from the users a more intrusive interaction as they have to allow the close examination of their eyes by a device.

- **Signature:** every person has their own way to sign their name so this can be used to identify them and are widely accepted by most people in a lot of facets of life. However, personal signatures can change over time and are susceptible to forgery.
- **Voice:** a person's voice is, a bit like their gait, the result of a mixture of physiological, behavioral and environmental factors. The physiological components are usually static - vocal tracts, mouth, nose, lips - unless they are damaged; the variability in time is mainly introduced by the behavioral aspects - emotional state, age, medical conditions. However, a person's voice is not a very distinctive characteristic and for that reason is not a good choice for large-scale systems. Additionally, these systems' performance is negatively affected by noise interference - background or other voices - and transmission channel degradation.

2.3 Human-in-the-Loop

2.3.1 Explainable AI

While the usefulness and accuracy of ML solutions has increased over the years, their sophistication and complexity has increased alongside, driven by the development of opaque decision algorithms in the field of DL such as CNNs which may contain hundreds of layers and millions of parameters that make their functioning difficult to comprehend. As these algorithms may be used to provide predictions in critical contexts with real-life consequences it's sometimes imperative to understand how the predictions are arrived at by the algorithm, because decisions that are not justifiable or legitimate can't be accepted. To achieve this, interpretability strategies are employed that explain in human terms how and what weighed on the prediction so that its recipient, an expert in the subject the ML system is inserted in, can evaluate it and decide what level of importance it warrants and help improve the model by providing feedback if the prediction is wrong or wrongly arrived at [30, 31]: this is what is meant by explainable AI (xAI). Interpretability methods can be classified in 3 categories [32]:

- Pre-model, which are applied before the ML model and try to understand the data through prototypical examples.
- In-model, which are applied in the models themselves resulting in ML algorithms that are interpretable by themselves.
- Post-model, which are applied after the ML model was built and try to explain the output, through sensitivity analysis (disturbing the input and observing the changes in the output),

saliency maps that highlight key features of the input that justify the class they belong to (commonly used in computer vision), or other strategy.

In the field of Computer Vision, explainability remains a challenge and the existing methods are still far from bulletproof even though there has been encouraging progress. This lack of reliability is illustrated by the fact that two very similar or even indistinguishable images may generate different explanations [33] and thus, the application of xAI algorithms to sensible fields that have potentially serious consequences must be a careful endeavour; still, positive outcomes have been achieved in these fields [34, 35].

2.3.2 Active Learning

In a human teaching environment, there are several methods of teaching that teachers may employ to pass on knowledge to their students. One of those is Active Learning, which invites the students to directly participate in the teaching process, by questioning, discussing, asking questions, investigate and think of solutions, all while the guidance of the teacher stays present [36]. Proponents of this teaching method claim several benefits, among them the deeper understanding of the material by the students; the easiness to correct misconceptions or errors in the students understanding; how it is easier to direct the students' minds in the right path for learning. Analogous to the Active Learning concept described above, a similar concept with the same name exists in ML, which can be summarised as how an algorithm can achieve better results, i.e. better accuracy, with less training labels than otherwise necessary, by getting feedback on unlabeled data instance from an *Oracle* - which can be a human [2, p. 4]. This concept is specially useful when the algorithm's job of labeling or classifying data is difficult - because it is time-consuming or resource intensive - but unlabeled data is easily obtainable.

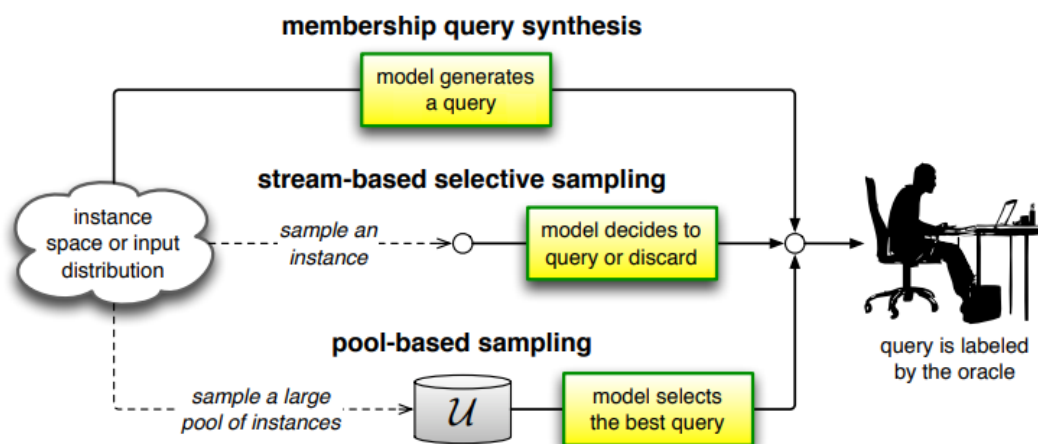


Figure 2.6: The three main active learning scenarios ¹

¹ Adapted from Figure 4 in [2, p. 9]

As can be seen in Figure 2.6, there are 3 main active learning scenarios [2, p. 8]:

- Membership query synthesis - where the learner can request labels from the *Oracle* for any unlabeled instance, including queries generated by it, rather than samples from a natural distribution.
- Stream-based selective sampling - where the learning decides if it wants either to query the *Oracle* for each unlabeled instance or discard it.
- Pool-based sampling - where there's a small set of labeled data and a large pool of unlabeled data available. Queries are chosen from the static pool, which is usually assumed to be static, according to an informativeness measure that evaluates all instances.

Active Learning implementation raise several questions though. The reliance on *Oracle* input may become a problem or of reduced utility in several scenarios [35, p. 8-9]:

- The *Oracle* may not be certain about the feedback they're providing regarding the query.
- Different people - *Oracles* - may provide different feedback for the same query, specially if it is hard to interpret.
- The *Oracle* might not have the necessary knowledge to provide accurate feedback.
- The effort put in by the *Oracle* might not be enough to offset the cost - be it time or mental - as the model's gains in learning may not be noticeable or worth it.
- Wrong feedback given by the *Oracle* might jeopardize the model's learning.

2.4 Replicability and Reproducibility

DL has assumed great importance within the topic of ML [37, 38, 39], being a major part in the software of modern systems from self-driving cars [40] to medical imaging technology [41]. This newfound popularity stems from the success DL has had in many ML areas such as natural language processing [42, 43], machine translation [44, 45] and, most relevantly, computer vision [46, 47].

One of three DL types are commonly used in Software Engineering (SE) [48, 49, 50]; the first one is the traditional artificial neural network (ANN) consisting of fully connected neural networks (e.g. multilayer perceptron [51, 52, 53], deep belief network [53]). The two others were both developed to enhance representation learning by using large sets of data and purposely designed neural networks, and are the most used in SE [54]: the convolutional neural network (CNN) [55, 56], that senses regional characteristics of a matrix data using special convolution functions and the recurrent neural network (RNN) [57, 58] that can capture features of sequential data. These DL models can be optimized for a specific SE task [59, 60], that is, train the model

parameters usually initialized with pseudo-random values using training data until the values of a loss function converge or a number of training iterations is reached.

The drawing of conclusions from the analysis of results produced by the running of an experiment that tests a previously formulated hypothesis is a foundational element of scientific inquiry. Additionally, for these conclusions to be lent credibility, the results must be repeatable provided that the experimental process is the same. These facts have been used all throughout history to consolidate the knowledge about the real-world. Along these lines, Effectiveness, Replicability and Reproducibility are all key characteristics of works within the realm of scientific research [61, 62, 63, 64, 65] which can be extended to more practical real-world implementations of ML related solutions for commercial, security or medical related purposes, to name some examples.

Despite all this, a 2016 survey found that more than 70% of researchers from a range of fields failed to reproduce another researcher's experiments, and over 50% failed to reproduce one of their own [66]. In the field of computer science, new knowledge from recent years has been increasingly obtained from practical experiments (which is true within ML research) performed under easier to control conditions as they are not almost totally determined on the natural or social real world but rely on computers built by humans. Still, researchers have had difficulty reproducing the work of others here too [3]. In the more specific research field of ML there are several reasons that explain this phenomenon [4]:

- Not readily available training data or differences in data distribution.
- ML model and/or training procedures being misspecified or under-specified.
- Not readily available necessary code to run the experiments, or errors in the code;
- Results metrics being misspecified or under-specified.
- Erroneous statistical analysis of the results, often claiming significant results when not justifiable.
- Selective results reporting and underestimation of overfitting.
- Drawing of conclusions with broader implications that what can be supported by the evidence.

Effectiveness has been shown to be achieved at higher levels when using DL algorithms over alternative ones [67, 68, 69, 70, 71], while, at the same time, saving the manual work of feature engineering [72, 38, 73, 71]. In ML, a common main objective for a model is to perform better and rank at the top on benchmarks scores. The achievement of such results are often claimed even though it is not always clear if they're actually owed to improvements in the model's design or other variables - experimental conditions, hyperparameters, random initial parameters - that might also affect them [74, 75, 76, 3].

Replicability - the ability to exactly reproduce reported experimental results given the same model and data - gives credibility and scientific merit to reported results [61, 62, 77]; this can

be a problem as for DL models as exact reproduction may be impossible because of the randomness during model initialization and optimization [77]. A wider definition for replicability can be adopted in this cases, that is, replicability is achieved if experimental results can be approximately reproduced with a high probability with the same model and data. Replicability however, does not guarantee that the results are extendable to experiments under different conditions [78, 61, 63]. A 2019 report indicated that 63.5% of the results in 255 manuscripts were successfully replicated [79]. Interestingly, it found that when help was provided by the authors 85% of results were successfully reproduced, compared to only 4% when no help was available. If we consider that there's no significant influence introduced in these values by a biased response of the authors, that is, only the ones that knew that the results would be re-obtained would respond, this goes against replication studies in other fields that did not observe such an increase in the percentage of successful replication attempts [80].

		Data	
		Same	Different
Code & Analysis	Same	Reproducible	Replicable
	Different	Robust	Generalisable

Figure 2.7: Reproducible Research. ²

Reproducibility is the requirement that must be met to achieve that, meaning that one experimental finding can also be obtained on a different experiment using the same experimental protocol, the same model, but different real-world data [63, 64]. Many SE DL studies often ignore both replicability and reproducibility considerations, making it is uncertain the extent to which their reported results can be replicated or reproduced under different conditions, such as the number of training iterations or the size and type of data. Replicability and Reproducibility are often regarded as mere threats and left to deal with on future works, even if the importance of these factors is acknowledged and thus are rarely investigated [59, 81, 82, 83].

2.5 Machine Learning Operations

DevOps is "the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving

²Adapted from <https://github.com/WhitakerLab/ReproducibleResearch>

products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market" [84]. One can apply DevOps to Machine Learning systems, which is exactly what MLOps is. MLOps aims to unify ML system development and ML system operation - respectively "Devs" and "Ops". MLOps promotes the automation and monitoring at all steps of ML system construction - integration, testing, releasing, deployment and infrastructure management [1]. The theory related to ML still apply on for MLOps, the additional challenge is to build and integrate an ML system and to continuously operate it in production and at the same time, avoid some of the pitfalls that come with this [85]. Even though an ML project can be classified as a software project, they differ in the following ways [1]:

- **Team skills:** the skills and knowledge required from the members of ML project team usually demand data scientists or ML researchers, who specialize on exploratory data analysis, model development, and experimentation; these profiles might not necessarily include knowledge and/or experience on production level software development.
- **Development:** The development process for ML is all about experimentation; different features, algorithms, modeling techniques, and parameter configurations are tried until one finds what the best solution for the problem is. This adds the challenge of tracking what worked and what didn't, so as to maintain reproducibility while maximizing code reusability.
- **Testing:** In addition to the normal unit and integration tests common in software development, testing tasks like data validation, trained model quality evaluation, and model validation are also needed.
- **Deployment:** ML systems can require the specific pipelines to perform automatic retraining and deployment of its model, adding complexity as these tasks are generally manually done by data scientists.
- **Production:** besides coding imposed performance limitations common on software projects, ML projects' performance also depend on constantly evolving data profiles. Thus, additional monitoring strategies for the production ML project in order to track its performance are needed so that it is clear when performance values deviate from the expected ones.
- **Continuous Integration (CI):** data, data schemas and models testing and validating are added to the usual code testing and validating.
- **Continuous Delivery (CD):** CD deals with the delivery of a system, that is, a ML training pipeline, not a single piece of software, that automatically deploys a model prediction service.
- **Continuous Training (CT):** CT is exclusive to ML projects and deals with automatically retraining and maintaining the models.

Chapter 3

State of the Art

ML projects in the real world often fail or come short of their objectives, never evolving into production ready applications [86], as, for the most part, focus has been more on building models but not on preparing them for real world operation and setting up the necessary tools, organization and infrastructure for it [87]. This can lead to operational issues if ML related workflows if are handled manually.

3.1 Machine Learning Operations

Due to the still recent and constantly evolving nature of MLOps, it's definition is not very precise and not widely agreed and known. However, it is understood that the term MLOps is right at the intersection of several other established and mature fields: ML, SE (in the form of DevOps) and Data Engineering [88].

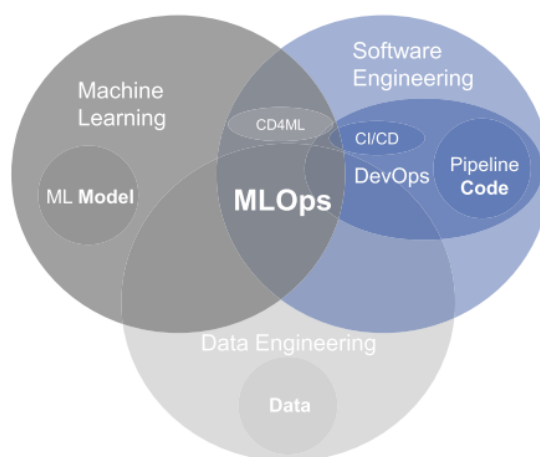


Figure 3.1: Intersection of topics that define MLOps. ¹

¹ Adapted from Figure 5 of [88]

Additionally, Kreuzberger, Kühl and Hirschl [88] define MLOps as "(...) a paradigm, including aspects like best practices, sets of concepts, as well as a development culture when it comes to the end-to-end conceptualization, implementation, monitoring, deployment, and scalability of machine learning products." It is intended to facilitate the creation of ML systems, linking the development (Dev) and operations (Ops) phases, leveraging these principles that can be totally or partially present [88]:

- P. I CI/CD automation** - responsible for the build, test, delivery and deploy steps of software project.
- P. II Workflow orchestration** - coordinates the tasks within an ML workflow, organizing their execution considering each task's dependencies and requirements.
- P. III Reproducibility** - the ability to obtain the same results when repeating the same ML experiment.
- P. IV Versioning** - assures the versioning of data, model and code to facilitate reproducibility and traceability.
- P. V Collaboration** - provides the ability to collaborate with other people on data, models and code.
- P. VI Continuous ML training & evaluation** - provides the ability to periodically and automatically retrain models with new feature data.
- P. VII ML metadata tracking/logging** - to keep track of workflow related data such as dates, execution times used to build performance metrics.
- P. VIII Continuous monitoring** - the periodic assessment of data, models and code to detect errors or changes that affect the products quality.
- P. IX Feedback loops** - the possibility to trigger operations based on the monitoring of some steps, such as scheduling retraining through the monitoring of a model's production performance.

These principles can be implemented in the system by several components which can be divided into several categories or components, each matching one or more principles [88] (Table 3.1):

- C. I CI/CD Component** - comprised of tools that implement continuous integration, continuous delivery, and continuous deployment.
- C. II Source Code Repository** - comprised of tools that implement code storing and versioning.
- C. III Workflow Orchestration** - comprised of tools that implement task orchestration of an ML workflow through directed acyclic graphs that represent the execution of the steps in the workflow.

- C. IV Feature Store System** - comprised of tools that implement central storage of commonly used features. Two databases may be typically be configured: an offline one to serve features for experimentation, and an online one to serve features with low latency for predictions in production.
- C. V Model Training Infrastructure** - comprised of the computational resources (CPUs, RAM, GPUs), that can be distributed or non-distributed, but are usually designed to be scalable.
- C. VI Model Registry** - comprised of tools that implement the storage of trained ML models together with their metadata.
- C. VII ML Metadata Stores** - comprised of tools that implement the storage and tracking of various kinds of metadata for each ML workflow pipeline task; some of these tools can be used to implement the previous component as well, as they too can encompass the handling of model metadata.
- C. VIII Model Serving** - comprised of tools that implement the serving of model produced predictions, that can be used in online inference for real-time predictions or in batch inference for predictions using large volumes of input data.
- C. IX Monitoring** - comprised of tools that implement the continuous monitoring of the model's performance, as well as the ML infrastructure, CI/CD and orchestration.

Table 3.1: Intersection between MLOps Components and Principles.

Components	Principles
CI/CD	CI/CD automation
	Continuous ML Training & Evaluation
	Feedback loops
Source Code Repository	Versioning
	Collaboration
Workflow Orchestration	Workflow orchestration
	Reproducibility
	Continuous ML training & evaluation
Feature Store System	Reproducibility
	Versioning
Model Training Infrastructure	Continuous ML training & evaluation
Model Registry	Reproducibility
	Versioning
ML Metadata Stores	Versioning
	ML metadata tracking/logging
Model Serving	CI/CD automation
Monitoring	Continuous monitoring
	Feedback loops

3.1.1 MLOps Architecture

Leveraging the advantages of the MLOps philosophy to automate the building, development, versioning, testing, deployment and monitoring of ML systems helps make these phases easier in an ML project. Applying the enumerated principles to the ML related steps to deliver a model to production, we can achieve, depending on the requirements and goals of the implementation, as close an automatic pipeline as possible. From the previous MLOps Principles and Components listed a complete, end-to-end, architecture was proposed in [88], detailing the task sequence - workflow - executed in each stage, giving ML researchers, developers and engineers guidance on the overall design of a MLOps system while remaining agnostic to what tools are responsible for what task, leaving that decision to be made along the appropriate requirements defined for each specific solution. This means that, in theory, any tool, be it simple or full fledged, open-source or commercially licensed, can be chosen to realize any step of any workflow; however, in practise, there can be limitations to the realistic options list that arise from the possible integrations or other incompatibilities or awkward combinations of tools.

1. **MLOps product initiation** - This is a preparatory stage where:
 - 1.1. A business problem is identified by a Business Stakeholder as having a possible ML solution.
 - 1.2. The overall system architecture is defined by the Solution Architect, choosing what tools are to be used after careful deliberation.
 - 1.3. The specific ML problem is derived from the initial Business Stakeholder formulation by the Data Scientist (e.g. regression, classification, etc).
 - 1.4. The data required to feed the ML solution is identified by a Data Engineer and Data Scientist, determined if it is available and can be obtained.
 - 1.5. The Data Engineer and Data Scientist then locate the raw data sources, checking its distribution and quality as well as validating that it is prepared for the designed ML system (e.g. checking if the data is properly labeled if supervised ML is being used).
2. **Requirements for feature engineering pipeline** - This is where the operations related with the ML system features are defined.
 - 2.1. A Data Engineer defines the data processing operations that produce usable data (e.g. normalization, aggregations, cleaning).
 - 2.2. A Data Scientist and Data Engineer defines the feature engineering rules that produce new or more advanced features from already existing ones. They can be iteratively adjusted based on feedback from the experimental model engineering stage or from the monitoring of the model performance.
3. **Feature engineering pipeline** - The requirements defined in the previous stage for the feature engineering pipeline are the starting point for a Data Engineer and SE to build the

feature engineering pipeline prototype where they are implemented. The task orchestration of

As a foundational requirement, the data engineer (R4) defines the code required for the CI/CD (C1) and orchestration component (C3) to ensure the task orchestration of the feature engineering pipeline. This role also defines the underlying infrastructure resource configuration.

- 3.1. The feature engineering pipeline connects to the raw data sources in whichever format or storage location it is.
 - 3.2. The data is extracted from those sources.
 - 3.3. The data is processed, that is, transformed and cleansed, and the data is made useful. The processing rules defined in the previous stage are the ones used here, subject to continuous improvement based on the feedback from phases ahead.
 - 3.4. Feature engineering is executed and new or more advanced features are produced based on other features. The feature engineering rules defined in the previous stage are the ones used here, subject to continuous improvement based on the feedback from phases ahead.
 - 3.5. The extracted features are stored into a feature store system.
4. **Experimentation** - This is where the major training logic from the training data features is executed, until a model type and its parameters are found determined to be the best for the ML problem at hand.
- 4.1. A Data Scientist connects to the feature store system or even to the raw data for an initial analysis. A feedback loop to the previous stage is established if any data adjustments are deemed to be required.
 - 4.2. Feature store data preparation and validation and train and test split dataset creation.
 - 4.3. A Data Scientist determines the most suitable algorithm for the context of the ML problem, through iterations of the following steps, until the performance metrics indicate satisfactory results; for each iteration, metadata (e.g. parameters to train the model, performance metrics results, iteration identifiers, time and date information, etc) is stored in the ML metadata. The support of Software Engineer ensures well-engineered model training code.
 - i. Model training iterations on new versioned training data, starting with the previously defined hyperparameters and algorithm in the experimental stage.
 - ii. Model evaluation iterations, testing and validating the model with several combinations of model parameters.
 - 4.4. The tuned model is exported and the code committed to the repository by a data scientist; additionally, a DevOps or ML engineer defines the ML workflow pipeline related code and commits it also.

5. **Automated ML workflow pipeline** - This stage is similar to an automated version of the experimentation stage, and it is managed by the a DevOps engineer together with a ML engineer, taking care of the runtime environments, the underlying infrastructure (hardware and other resources) and computation frameworks.

- 5.1. The features are automatically pulled from the feature store system.
- 5.2. Automated feature store data preparation and validation and train and test dataset split.
- 5.3. Automated iterations of the following steps, until the performance metrics indicate satisfactory results; for each iteration, metadata (e.g. parameters to train the model, performance metrics results, iteration identifiers, time and date information, etc) is stored in the ML metadata.
 - i. Automated model training iterations on new versioned training data, starting with the previously defined hyperparameters and algorithm in the experimental stage.
 - ii. Automated model evaluation iterations, testing and validating the model with several combinations of model parameters.
- 5.4. The final trained model is then exported.
- 5.5. The final trained model is stored in the model registry, along with its associated metadata (e.g. training data feature version, training code version), from where it can easily be fetched once it is ready for deployment in production.
- 5.6. The CI/CD component triggers the continuous deployment pipeline and the production-ready model and the associated model serving code are pulled from their storage and versioning systems. The CD pipeline builds and tests the ML model and deploys it along with serving code to deploy to production.
- 5.7. The model serving component, which can be designed by the software engineer as they are experienced in developing and deploying production applications, receives requests (typically through a REST API) and makes predictions for new data from the feature store system. It's good practice to do A/B testing, that is, deploying two competing models (one the established one and the other the challenger) to compare their real-world performance [89].
- 5.8. While the model is operational in production its performance is constantly monitored along with the model serving infrastructure; if anything triggers the need for a change, such as unsatisfactory prediction accuracy a feedback loop is used to communicate back to a previous stage.
- 5.9. The feedback loop can be used to trigger the execution of steps within previous stages, such as the experimental stage, data engineering zone, and the Automated ML workflow, always with the aim of improving the final model in production.
- 5.10. Continuous training may be triggered by the detection of concept drifts during the monitoring stage, that is, when the real-world input data deviates from the original kind

of data the training data used represented [90]; new training data or simply scheduled events can also trigger retraining.

According to the level of automation, three MLOps levels of maturity are defined [1]:

- **Level 0:** the process for building and deploying ML models is entirely manual, including the steps for data analysis, data preparation, model training and validation. This level of maturity is usually controlled by experimental and ever changing code until a workable model is produced. This means that there's a separation between the ML and operations teams, as the first only hands out a ready ML model off to the second that is responsible for the deployment, which makes release iterations more difficult and understandably, more difficult. Both CI and CD are not considered, as well as performance monitoring. This level of MLOps maturity arise might be sufficient when models are rarely changed or trained which rarely is acceptable in the real world, which is why they often break failing to adapt to environmental or data changes.
- **Level 1:** continuous training of the model is added in this level by automating the ML pipeline enabling continuous delivery of prediction service model. To automate the retraining of production models, the automated data and model validation steps are added to the pipeline, along with pipeline triggers and metadata management. Pipeline triggers enables the automation of model retraining based on the occurrence of certain events or with a certain frequency: on demand, on a schedule, on availability of new training data, on model performance degradation, on changes in the data distributions. Still, testing is usually manual assuming deployments are not frequent, as well as the deployment task itself which is often manual too. The tested source code is usually submitted and handed over to the IT team to deploy to the target environment. Level 1 is suitable when new deployments are based on new data instead of improvements on the model itself. The next level of MLOps accounts for this need of constant, reactive and rapid improvements.
- **Level 2:** This last level resolves the need for constant, reactive and rapid update of the pipelines in production with a robust automated CI/CD system. New ideas for feature engineering, model architecture, and hyperparameters are readily explored and their merit assessed through automatic building, testing, and deployment to the target environment. All of the components defined before (3.1) are included in this level of maturity.

3.1.2 ML Tools

Following the growth in popularity of AI and ML in recent years, many tools have appeared that help automate many different related processes. Several tools are presented in this subsection that provide a general, wide view of what is currently available in the MLOps universe and what features these tools have. Most of these tools are open-source in nature, as half of all IT organizations were using them in 2022 with the expectation that this proportion will grow to two thirds in 2023

[91]. The scope of the tools is varied: some are meant to be all encompassing, covering the whole MLOps workflow pipeline, others are specialized for only certain steps or components. ML related development involves trial and error, investigation and iterative execution and thus, a number of artifacts play important parts, from datasets and associated features, to models, hyperparameters, metrics, code, configuration and log files. As manual management of these ML artifacts, steps, processes and pipelines cannot be done efficiently, many tools and systems were developed in order to provide comparability, reproducibility, and traceability (collection, storage, versioning) of ML artifacts across the ML lifecycle steps and iterations. Several ML systems or applications currently exist, each with their specialization and different focuses. Classifying these systems in clear categories is not a straight forward task as boundaries are blurred between systems that simply provide functionality for the development of ML systems to ones that focus on the management, storage, and deployment of ML artifacts. Therefore, classification of these systems can be made along these general classes, which are not mutually exclusive [92]:

- **Lifecycle Management:** systems that focus on the entire MLOps architecture 3.1.1 and, going beyond specialized or limited implementation of stage steps to cover most or all of the stages (and their steps) in the ML architecture. *Microsoft Azure ML* [93], *Amazon SageMaker* [94], *Google Vertex AI* [95], *IBM Watson Studio* [96], *Comet* [97], *DataRobot AI Cloud Platform* [98] and *Cloudera Data Science Workbench* [99] are all overarching ML as a service (MLaaS) commercial solutions that are available by a provider's cloud infrastructure offering powerful configuration options such as the scaling of processing and memory resources. However, they are paid tools and its usage may not be admissible because of data protection requirements as it implies handing over access to a third party to sensible information. *Valohai* [100] is a comparable solution to the ones before, with the advantage that it can be deployed in the cloud or on-premise. Open-source solutions such as *MLflow* [101, 102], *ClearML* [103], *Polyaxon* [104] and *Hopsworks* [105], are more flexible in the sense that they're available both in the cloud and on-premise. *MLflow* focuses on capturing, storing, managing, and deploying ML artifacts; it registers experiment runs, the associated code and data dependencies, executed manually or automatically depending on what is defined on the instrumentation code. It also has a standard for packaging ML models, compatible with many serving environments; a standard for packaging reusable and reproducible project code; a model registry to store and manage models through their deployment lifecycles. *ClearML* and *Polyaxon* are also comparable in the functions they cover, adding model monitoring and resource management too. *Hopsworks* adds Hadoop Distributed File System (HDFS) together with a NewSQL database (HopsFS) for highly scalable learning, Github-like project management and an integrated feature store. A company name *Iterative* developed a collection of open-source tools, starting with a data, model, code and metrics version control tool built upon Git, appropriately named *DVC* [106], extending it with *CML* [107] - a CI/CD library for ML projects -, *DCVLive* [108] - a library for logging metrics and metadata -, *Iterative Studio* [109] - a web application for data and model management, experiment tracking, visualization and collaboration - and *MLEM* [110] - a model registry

and deployment tool. For non-ML experts, *Ease.ML* [111] guides its users along throughout its several functions: automatic data ingestion and augmentation, automatic feasibility studies, data noise debugging, data acquisition, scalable multitenant automatic training, continuous integration, and ending with continuous quality optimization. All of the previous tools are meant for general usage and thus are not focused on one specific use case, as opposed to proprietary specially developed tools to fit one specific use case, such as the internally used tools by Airbnb, Uber and LinkedIn. Airbnb uses *Bighead* for feature management, model development, execution management, lifecycle management, offline and online inference services and containerization management [112]; Uber uses *Michelangelo* to build, deploy and operate uniform and reproducible ML pipelines used in their production microservices environments, made up of a mix of open-source and internally developed tools providing feature storage, a language for feature selection and transformation, the distributed deep learning framework *Horovod* [113] and the model management system *Gallery* [114]; LinkedIn uses *Pro-ML* with scalability in mind, for data exploration and model creation with an own language for feature and model representations and a central feature marketplace, real time and batch model training, model deployment and model monitoring [115].

- **Pipeline Management:** similar to the previous class, but these tools typically do not support model deployment and monitoring (the steps 55.6., 55.7., 55.8., 55.9., 55.10.) the management of software artifacts, or both. *Velox* [116], *Vamsa* [117] and *ArangoML Pipeline* [118] are similar in the functions covered, all focusing on models and model metadata, even though their goals might differ. *Velox* provides management and orchestration of training pipelines for predeclared models, model performance evaluation and model inference; *Vamsa* provides automated provenance tracking of ML pipelines based on static analyses of Python programs; *ArangoML Pipeline* builds upon *ArangoDB* to provide artifact and metadata storage for ML pipeline tracking, auditing, reproducibility, and monitoring. *Apache SystemDS* [119] is a declarative ML pipeline system that uses a dedicated language for different ML lifecycle tasks. Both *Mltrace* [120] and *ProvDB* [121] are focused on lineage and provenance [122]: *Mltrace* is a *Python* tool for lineage and tracing of artifacts in ML pipelines; *ProvDB* is an all-in-one provenance, model lineage, and metadata management system. *TFX* [123] is a *Tensor-Flow* based tool with libraries to create ML pipelines for the data, model and operations related steps of the ML Pipeline Workflow, as well as the *MLMD* [124] library for metadata management and version control. These same workflows can be deployed in *Kubernetes* by *Kubeflow* [125]. *H2O* [126] is a powerful open-source distributed in-memory ML platform that makes easy the productionalization of ML models in enterprise environments that also provides ML lifecycle integration and ML artifact management support. *Neptune* [127], *FBLearner Flow & Predictor* [128] (Facebook's proprietary ML pipeline development and processing system), *MLCask* [129], and *Disdat* [130] additionally support software artifacts, improving reproducibility.

- Experiment Management:** tools that focus on the comparability and reproducibility of exploratory ML experiments that aim to develop models, training and optimising them; they often complement tools of the previous class providing the results from which subsequent pipelines are created and executed. *DWF* [131] allows for the tracking of experiments and training iterations and associated configurations and performance metrics artifacts on a predefined collection of models provided by *TensorFlow* [132] and *scikit-learn* [133] but trained models storage or versioning. *StudioML* [134] adds the capture of model artifacts without the necessity of modifying experiment code. *MLCube* [135] and *MLPM* [136] focus on reproducibility for the model development and deployment related steps, by providing model packaging functions. *Guild AI* [137], *Datmo* [138], *Keepsake* [139], *Runway* [140], *Sacred* [141] and *Weights & Biases* [142] further improve on the reproducibility aspect by capturing source code, dependencies, execution environment and logs. *Apache Submarine* [143] and *Determined* [144] integrate and provide functional interfaces for popular ML frameworks (e.g. *TensorFlow*, *PyTorch*, *MLflow*, *TensorBoard*) without requiring extra infrastructure knowledge for orchestration.
- Model Management:** tools that focus on the model development, deployment and monitoring steps - differing from the Experiment Management tools because they support these last two - as well as the management of models and their metadata. *ModelDB* [145], *ModelHub* [146] both focus on supporting model development, deployment, and monitoring; the first uses a relational database while the second uses Git to version models and their metadata. *ModelKB* [147] focuses on model management, experimentation, deployment, and monitoring. It collects metadata about each experiment and automatically generates source code for deployment, sharing, and reproducibility. *MMP* [148] is a model management tool designed for Industry 4.0 environments, associating models with business and domain metadata, and it has a model registry, and a central metadata store. *MISTIQUE* [149] specializes in the storage and management of model intermediates (e.g. input data) to accelerate model evaluation, performance analysis, and interpretability. *Sql4ml* [150] translates ML models expressed in SQL to Python *TensorFlow* code. *MLModelCI* [151], *ModelCI-e* [152], *Clipper* [153], *Rafiki* [154], *Overton* [155] and *CMS* [156] are tools mainly designed for model deployment and monitoring related steps (*Rafiki* supports also model training as well as *Overton* and *CMS*). *ModelHub.AI* [157] is a tool that lists a range of deep learning models for their easier dissemination.
- Dataset & Feature Management:** tools that focus on dataset and feature management capabilities as well as data pre-processing, typically acting as a complement for tools in the Model Management class. *MLdp* [158] is Apple's own tool for ML data management, that integrates different types of data, support for large volumes of raw and volatile data, version and dependency management, data provenance, and integration with major other ML frameworks. *ExDra* [159] is a tool for data acquisition, integration, and preprocessing from federated and heterogeneous raw data sources. *Pachyderm* [160] provides automated data

versioning, data lineage and provenance; provenance and explainability were also motivations for the design of *ProvLake* [161] a tool to capture, integrate, and query data across multiple programs, databases and stores. Lighter tools like *Data Provenance Library* [162] and *Shuffler* [163] are library-level solutions; *Data Provenance Library* focuses on provenance of data (using *MongoDB* as a provenance store), supporting data reduction, augmentation and transformation operations; *Shuffler* focuses on data preparation for computer vision. *Feast* [164] is a feature store that aims to feed ML models with features enabling a DevOps philosophy for the lifecycle of features.

All of these tools are collected in Table 3.2.

Table 3.2: ML Tools by Class and License Type ²

Class	License Type	Tools
Lifecycle Management	Free	<i>MLflow</i> [101, 102], <i>ClearML</i> [103], <i>Polyaxon</i> [104], <i>Hopsworks</i> [105], <i>Iterative Tools</i> [106, 107, 108, 109]
	Non-Free	<i>Microsoft Azure ML</i> [93], <i>Amazon SageMaker</i> [94], <i>Google Vertex AI</i> [95], <i>IBM Watson Studio</i> [96], <i>Comet</i> [97], <i>DataRobot AI Cloud Platform</i> [98], <i>Cloudera Data Science Workbench</i> [99], <i>Valohai</i> [100], <i>Bighead</i> [112], <i>Michelangelo</i> [114], <i>Pro-ML</i> [115]
Pipeline Management	Free	<i>Velox</i> [116], <i>ArangoML Pipeline</i> [118], <i>Apache SystemDS</i> [119], <i>MLtrace</i> [120], <i>TFX</i> [123], <i>Kubeflow</i> [125], <i>H2O</i> [126], <i>Disdat</i> [130]
	Non-Free	<i>Vamsa</i> [117], <i>ProvDB</i> [121], <i>Neptune</i> [127], <i>FBLearner Flow & Predictor</i> [128], <i>ML-Cask</i> [129]
Experiment Management	Free	<i>DWF</i> [131], <i>StudioML</i> [134], <i>MLCube</i> [135], <i>MLPM</i> [136], <i>Guild AI</i> [137], <i>Datmo</i> [138], <i>Keepsake</i> [139], <i>Sacred</i> [141], <i>Apache Submarine</i> [143], <i>Determined</i> [144]
	Non-Free	<i>Runway</i> [140], <i>Weights & Biases</i> [142]
Model Management	Free	<i>ModelDB</i> [145], <i>Sql4ml</i> [150], <i>Clipper</i> [153], <i>Rafiki</i> [154], <i>ModelHub.AI</i> [157]
	Non-Free	<i>ModelHub</i> [146], <i>ModelKB</i> [147], <i>MMP</i> [148], <i>MISTIQUE</i> [149], <i>MLModelCI</i> [151], <i>ModelCI-e</i> [152], <i>Overton</i> [155], <i>CMS</i> [156]
Dataset & Feature Management	Free	<i>Pachyderm</i> [160], <i>ProvLake</i> [161], <i>Data Provenance Library</i> [162], <i>Shuffler</i> [163], <i>Feast</i> [164]
	Non-Free	<i>MLdp</i> [158], <i>ExDra</i> [159]

3.1.3 Real-world MLOps example

Implementing ML in real-world applications is a challenge when the focus is systems integration and scaling. Due to the still novel nature of MLOps there are very few use cases available in the literature that clearly describe a MLOps implementation from the problem identification and understanding, to model continuous training, deployment, delivery, and monitoring [165]. One of these rare examples is the use case of Oravizio [166], a software product providing information on patient-incurred risks related to hip about patient-level risks related to hip (total hip arthroplasty)

²Adapted from Figure 2 of [92]

and knee joint replacement surgery (total knee arthroplasty). It helps the interaction between a surgeon and a patient helping to make the well informed decisions and consent regarding an eventual operation. Oravizio was developed in cooperation between two different organization, a hospital specializing in joint replacement surgery, and Solita, a software company headquartered in Finland. The hospital had a 10 years worth accumulated volume of data, used as an asset that might contain risk factors related to joint replacement surgeries down to an individual patient's level.

Three different prediction models are provided:

- Risk of infection within one year from surgery;
- Risk of revision within two years from surgery;
- Risk of death within two years from surgery;

These models are fed with data from patients who have undergone surgery, collected over the years, totalling 30000 medical records. This number is too large to be processed manually by a surgeon during or in preparation of each appointment, thus a risk determining model that predicts the outcome of the surgery is used. This data however, was not well suited for post-processing as various formats and computer systems were used to gather and store it, some of which were already been retired. Extra effort was thus required to first consolidate and only then pre-process it to give it uniformity and utility. After that, the objective is to have an explanatory ML model for each of the listed risks to allow for validation and ensure regulatory compliance, which is made easy by the already known relations between the risk in joint replacement surgery and selected explanatory values from previous clinical research. XGBoost was selected as the implementation of the Gradient Boosting Machine (GBM) [167], to calculate the predictions required and the final model was built, which is deterministic, meaning that it can be validated with test data in a test environment not needing to perform any validation in the production environment.

A version control system (VCS) was used to store all the developed artifacts with the exception of the model itself, due to privacy restrictions related to strict regulations on patient records use. Because of this, the final model was deployed in the production environment by handover from data scientists working within the hospital's computational environment to software developers working on the Solita's production environment using a network drive.

This separation of environment owners, makes frequent changes like retraining of the model more difficult adding to the fact they were not anticipated as needed. However, re-training was later performed with more 45000 patient records somewhat improving the model's performance, showing the benefits of model re-training. As is apparent in this implementation example, the model is a key artifact in a MLOps system; in this particular case, it serves as the interface between the hospital and service provider. If needed or simply desired, a more accurate model can be re-created in pre-determined intervals or in an ad-hoc fashion, spawning a new model version for each iteration cycle.

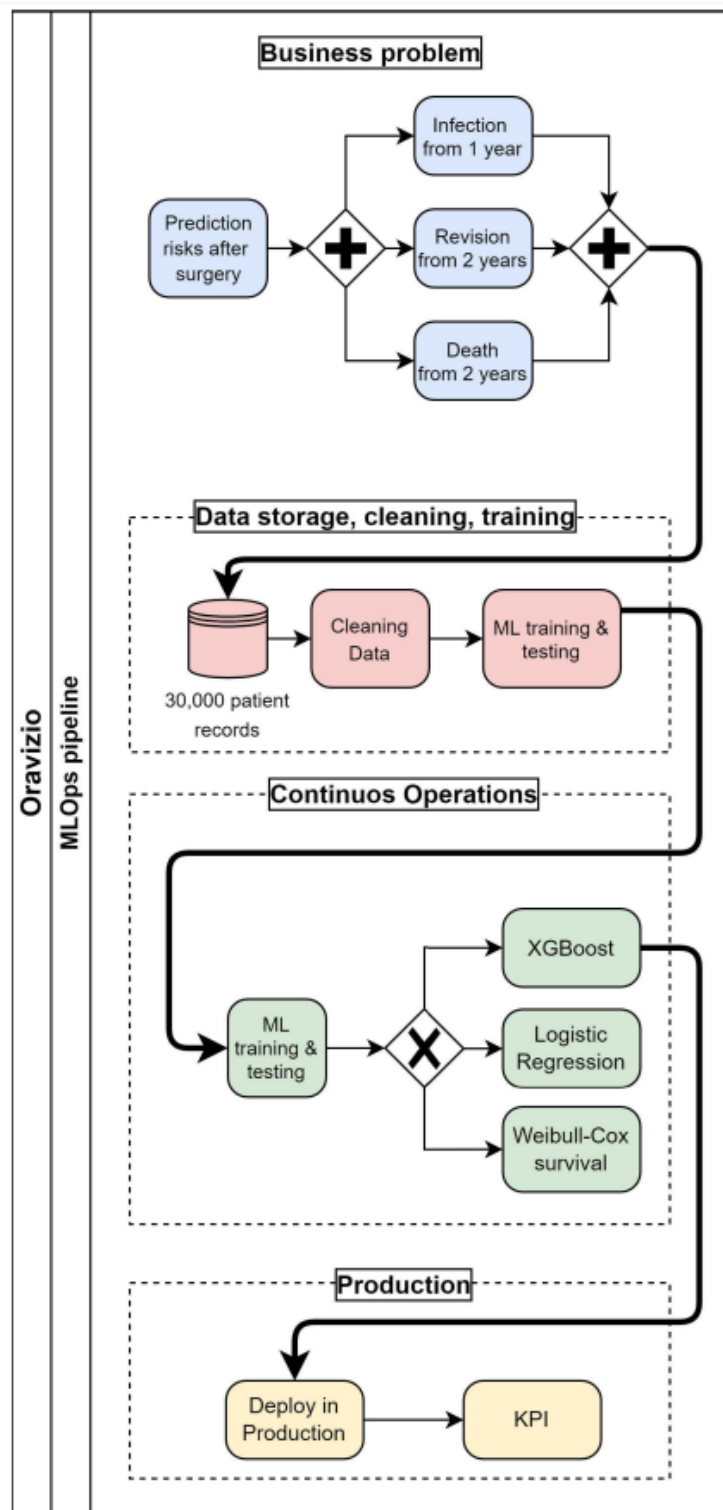


Figure 3.2: Oravizio's MLOps Implementation. ³

³Adapted from Figure 3 of [165]

Due to the two different organizations involved in Oravizio's solution being isolated from one another, the model's monitoring process was designed with this in mind and has an effect on the detection of bias and other problems. The most prominent related design decision is the selection of a model that gives deterministic results, not evolving over time, producing the same results continuously, not giving rise to any results drift and thus, no retraining is needed unless a new patient records dataset can be produced, in which case the retraining process is triggered manually after the accordingly human decision is made.

Finally, the model was not intended to give prescriptive results but only helpful advice related to the possible risks associated with a surgical operation, leaving the final decision, and all the responsibility and consequences inherent with it, to an expert that can determine if the advice is relevant and correct or if it is wrong, infeasible and the result of the model's malfunctioning.

3.2 Active Learning

When an ML model makes a wrong prediction, it is common to provide to retrain it with more data related to said predictions; for example, in a classification problem, if the model has problems distinguishing between two classes to apply to test data, further training data with instances of said classes is fed to retrain the model. This may not always be the better solution as said data may be hard to find, or retraining may be costly in time and resources. An alternate solution is to involve a human component into the process, by having a person with the right knowledge profile be asked by the model to provide feedback. Such a solution is described in [35]; it uses a neural network for a classification task, with the model training and testing process split into several steps:

1. After predictions are generated by the model, data points for which said predictions are given are chosen as they represent the images with the highest entropy.
2. Explanations for each sampled image in the step before are generated, returning a tensor with the size of the image, that can be thought to represent of how much the network is focusing on each pixel.
3. Average pooling is applied to the tensor, with a certain window and stride, splitting the tensor into small rectangular regions, generalizing the importance levels within them, making it easy for a human eye to form a general idea of where the model is focusing more.
4. Said squares are drawn over the original image and given to the *Oracle*, a human, that chooses which rectangles are relevant to a correct classification of the query. This choice is fed to the model by a weights tensor of zeros, except filled with ones where image was selected, which is then multiplied by the gradients of the network, and from the mean of the result an extra member of the loss is obtained, which is added to the already calculated cross-entropy loss.

This solution allows the *Oracle* to impact the training of the model, modifying the model loss function to penalize wrong predictions further. This can increase the value of the information of

each data point by decreasing the entropy [168] and, also, decreasing the amount of data required to train a more accurate neural network.

3.2.1 Active Learning Tools

Many tools have been designed and built over recent years to fulfill the several tasks executed in the steps within the MLOps Architecture 3.1.1 (specially as the popularity of ML and AI has increased in recent years), be it in the form of open-source and free software typically covering only a subset of those steps or in a full featured commercial solution form that cover all or a wide range of them. While initially these tools were used in more traditional ML ways, supplying data to a model and analysing the results, drawing conclusions to improve next training iterations, they eventually started to consider collaboration with users, in the form of active learning [169] with users in the role of oracles to be queried when there was a need to label new data. In these tools, execution control rests on the learning algorithms, which determine when it is necessary to query the oracles; this dynamic can be modified to a shared control philosophy, classified as interactive machine learning (IML) [170], where interactions between algorithms and are more focused, frequent and incremental.

Tools of the first kind can request feedback from users at different stages: preparation (collecting data to build the model around), execution (labeling data when queried by the algorithm - oracles) or validation (checking the performance of the model when needed). Examples are:

- AliPy [171], a tool to evaluate, compare and analyze the performance of active learning algorithms implemented within it.
- Libact [172], a tool that implements several AL algorithms and allows being extended using provided interfaces.
- modAL [173], a tool suitable for rapid prototyping.
- NEXT [174], a crowdsourcing tool for label collecting.

Tools of the second kind provide greater interactivity with humans, making the interface design more important as it is responsible for the bidirectional feedback between users and model [175]. IML is still a recent research field and there are not many general purpose tools dedicated to it, the most notable being AnchorViz [176], a visualization tool that focuses on the discovery of prediction errors and previously unseen concepts through humandriven semantic data exploration. Custom-made, ad-hoc, IML related developments are more common though, more restricted in goals, aimed at solving specific problems, such as giving additional structure to unstructured data, that is, data that has no identifiable structure, does not have a predefined model, or does not fit into relational databases, such as images, video, audio files and certain types of text documents [177]. Examples of this include:

- Crayons [178], a tool that creates image classifiers.

- *CueFlik* [179], a web image search tool in which users classify images based on created rules giving examples and counterexamples;
- *ilastik* [180] that uses image segmentation, object classification, counting and tracking allowing non-expert users to interactively provide annotations;
- *AIDE* [181], an image annotation tool for ecological surveys that links users and ML models;
- *JAABA* [182] adds labels to video frames to identify animal behaviors;
- *InteractML* [183], a tool for users working with movement interaction in immersive media;
- *Wekinator* [184], a tool that enables the application of ML music information retrieval to real-time musical performance;
- *BeatBox* [185], a tool that enables user creation of custom beatbox recognizers;

IML can also be used as a way of promoting xAI: *CAIPI* [186] allows the ML model to query users while at the same time explaining that query; both the query but the explanation can be give feedback by the user, that is, the query is answered but the explanation can also be corrected.

Chapter 4

Platform Architecture

This chapter describes the architecture for a new Experiment Management ML tool. Even though the usage of one or several already existing tools to achieve the active learning and reproducibility and replicability previously states goals would possibly be feasible, it was deemed that their adaptation to fit these would be more difficult and take up more time than creating an all new application designed and built with them specifically in mind. It is worth mentioning though, that the advantages related to the simplicity of installation and usage, may be overshadowed by the lack of other more powerful features present in other tools, depending on the specific application; on the other hand, when these are not a requirement and the ML problem to be solved fits the tool, a simple tool is more convenient and desirable. The developed platform, as most web applications, can be divided into two main parts:

- **Backend** - developed from scratch using Go for the web server implementation, *Redis* [187] for in-memory data storage and PostgreSQL [188] for in-disk RDBMS data storage.
- **Frontend** - developed using the Javascript framework Vue 3 [189], from an existing MIT-licensed template called Admin One [190], along with Tailwind [191] for CSS. Vite [192] was used as a development server and for tooling.

First, the Data Model and the created entities in the chosen RDBMS and then design and functioning of the platform will be described, having in mind these two main categories of software development - Backend and Frontend - that will be . The main purpose of the overall design was to achieve a level of automation for Active Learning ML models based on the one described by [35] but modified to fit the pipeline execution logic described in 4.2.6.

4.1 Data Model

The RDBMS chosen to store permanent data was PostgreSQL as it is open-source, well supported and has a long history of reliability. The backend web server performs CRUD operations using the GORM [193] library. This adds three extra columns to every entity:

- *created_at*
- *updated_at*
- *deleted_at*

Which contain a timestamp representation of the moment in time each entity was created, last updated and deleted at, respectively.

There are 13 entities:

- **Users** - represents a system user identified by its unique ID, containing a unique username and a hashed password.
- **Pipelines** - represents a ML pipeline, uniquely identified by its ID, containing a name, the user it belongs to and, most importantly, its definition, which is made up of the steps to be executed described in a stringified JSON format.
- **Runs** - represents a Pipeline's execution, uniquely identified by its ID. It stores information about the execution status, the last time it was run (if ever), the pipeline definition at the time the run was created and the step awaiting feedback (if any) and the error message (if the execution stopped due to an error).
- **Pipeline Schedules** - represents the a scheduled, future pipeline execution, uniquely identified by its ID. It can either be a single execution event - in which case the *unique_occurrence* column will have a date representation value - or a recurring event described by a *cron* expression.
- **Run Statuses** - represents the possible values for the status of a run, uniquely identified by their respective ID. Five possible statuses are defined initially: *Not Run*, *Executing*, *Error*, *Success* and *Waiting for Feedback*.
- **Run Step Statuses** - represents the status of each step on a pipeline's execution's - run - definition, mapped through the step ID. The possible statuses are the same as the overall run statuses described above.
- **Human Feedback Query** - represents the *queries* - or feedback requests - a run's step may need. It contains the relevant step ID, Epoch, Query ID, Run ID and Query Status.
- **Query Statuses** - represents the possible statuses for a *Human Feedback Query*, which are initially defined as *Unresolved*, *Submitted* or *Resolved*.
- **Human Feedback Rect** - represents the regions - in the form of rectangles - within a *query*, identified by a step and in need of selection among them by a human who can discern which ones are relevant for a better prediction. It contains the rectangle's coordinates within the image and a flag indicating if it was selected as relevant or not by a user.

- **Dataset Handlers** - represents a custom *PyTorch* dataset processing class. Uniquely identified by an ID contains a name, a owner user and a path where the python script containing said class is stored. The script file must implement the following methods and classes (where *fraction* gives the fraction value of data to use for training and testing, respectively):

```
# returns a tuple with the data classes
def get_data_classes():
    ...

# returns a class that implements torch.utils.data.Dataset
class TrainDataset(torch.utils.data.Dataset):
    def __init__(self, fraction=1):
        ...

# returns a class that implements torch.utils.data.Dataset
class ValidationDataset(torch.utils.data.Dataset):
    def __init__(self, fraction=1):
        ...
```

This script is copied into the relative path *datasets/handler.py* within a run's work files directory.

- **Trainers** - represents a custom *PyTorch* based trainer script. It should look for a dataset handler in the relative path *datasets/handler.py* on its directory or implement a dataset handler itself. Uniquely identified by an ID, contains a name, a owner user and a path to the python script that implements the trainer. Any training logic, including custom models, loss functions, performance metrics can be used, from any installed *Python* library. If feedback is needed during the its execution - during a model's training - it should be marked as staggered. It must generate a *PyTorch* model, extending the
- **Testers** - represents a custom *PyTorch* based tester script. It must look for a dataset handler in the relative path *datasets/handler.py* on its directory or implement a dataset handler itself. Uniquely identified by an ID, contains a name, a owner user and a path to the python script that implements the trainer.
- **Trained Model** - represents a previously trained *PyTorch* model weights whose *state_dict* was saved to a file with the extension *.pt*. Uniquely identified by an ID, contains a name, a owner user and a path to the file.

4.2 Backend

The backend was developed in Go [194] using the Gin Web Framework [195] to implement the Web Server, providing static files and resources and a REST API responsible for the getting, updating and deleting the stored data, as well as requesting and scheduling processing requiring tasks.

4.2.1 Presentation, Service and Repository Pattern

The web server is designed along the lines of a Controller, Service and Repository design pattern as it is widely used by the software developer community (so much so that it is the de facto standard for Java EE applications) [196]. It consists of organizing the application's components within horizontal layers that take up different roles and responsibilities, abstracting the work they perform from the other layers, leading to independence between them. The layers need only to communicate between them using the interface each component implements, without worrying about the specifics of what each component does. Although there isn't a set number of layers, the backend applications is organized into the 4 most common layers: Presentation, Service, Repository and Database (Figure 4.1).

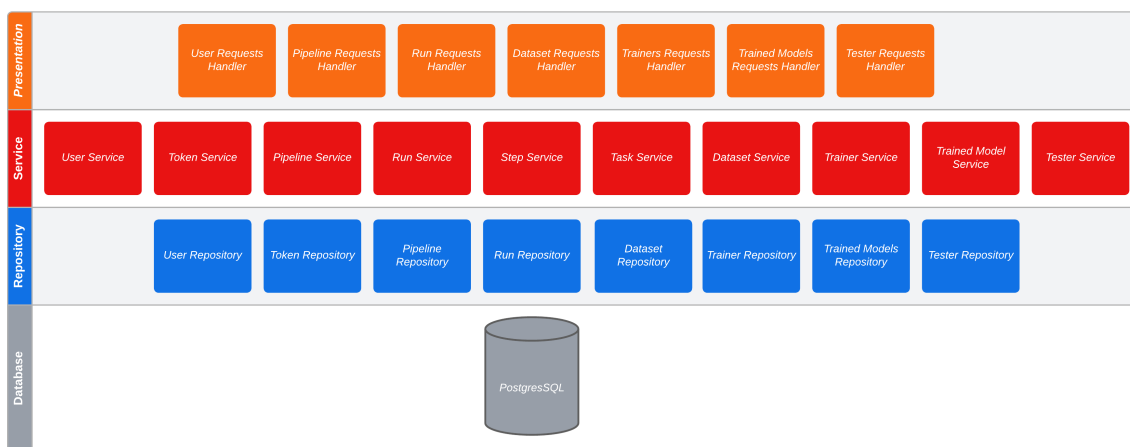


Figure 4.1: Backend Application Layers

Each of these layers contains the following components:

- **Presentation Layer**

- **User Requests Handler** - Handles user related requests from the frontend application, specifically signup, log in, log out and new token requests.
- **Pipeline Requests Handler** - Handles pipeline related requests from the frontend application, specifically fetching, creating, updating and deleting.
- **Run Requests Requests Handler** - Handles run related requests from the frontend application, specifically fetching, creating, running and resuming. Run results fetching is also handle here.
- **Dataset Requests Handler** - Handles dataset handler related requests from the frontend application, specifically fetching, creating, updating and deleting.
- **Trainer Requests Handler** - Handles trainer related requests from the frontend application, specifically fetching, creating, updating and deleting.

- **Trained Models Requests Handler** - Handles trained models related requests from the frontend application, specifically fetching, creating, updating and deleting.
- **Trainer Requests Handler** - Handles trainer related requests from the frontend application, specifically fetching, creating, updating and deleting.

- **Service Layer**

- **User Service** - Provides the methods listed in Figure 4.2a for user creation, editing and authentication, which contain the relevant business logic.
- **Token Service** - Provides the methods listed in Figure 4.2b for token creation and deletion which contain the relevant business logic.
- **Pipeline Service** - Provides the methods listed in Figure 4.2c for pipeline fetching, creation, editing, run scheduling and deletion which contain the relevant business logic.
- **Run Service** - Provides the methods listed in Figure 4.2d for run fetching, creation, run tasks creation, execution, resuming and deletion which contain the relevant business logic.
- **Step Service** - Provides the methods listed in Figure 4.2e for step and edge (that is, the connections between steps) creation which contain the relevant business logic.
- **Task Service** - Provides the sole method listed in Figure 4.3a to setup the Async server worker, as the business logic for task creation and handling stayed with RunService to avoid cyclic imports.
- **Dataset Service** - Provides the methods listed in Figure 4.3b for dataset handlers fetching, creation, editing and deletion, which contain the relevant business logic.
- **Trainer Service** - Provides the methods listed in Figure 4.3c for trainer scripts fetching, creation, editing and deletion, which contain the relevant business logic.
- **Trained Model Service** - Provides the methods listed in Figure 4.3d for trained models fetching, creation, editing and deletion, which contain the relevant business logic.
- **Tester Service** - Provides the methods listed in Figure 4.3e for tester scripts fetching, creation, editing and deletion, which contain the relevant business logic.

- **Repository Layer**

- **User Repository** - Provides the methods listed in Figure 4.4a necessary for user creation, editing and authentication, that interact directly with the PostgreSQL database.
- **Token Repository** - Provides the methods listed in Figure 4.4b necessary token creation and deletion, that interact directly with the Redis in-memory database.
- **Pipeline Repository** - Provides the methods listed in Figure 4.4c for pipeline fetching, creation, editing, run scheduling and deletion, that interact directly with the PostgreSQL database.

 <i>UserService</i>
● Get(id uint) (*model.User, error) ● GetByUsername(username string) (*model.User, error) ● Signup(username string, password string) error ● Signin(user *model.User) error ● UpdateDetails(user *model.User) error

(a) User Service Interface

 <i>TokenService</i>
● NewFirstPairFromUser(ctx context.Context, u *model.User) (*model.TokenPair, error) ● NewPairFromUser(ctx context.Context, u *model.User, refreshToken model.RefreshToken) (*model.TokenPair, error) ● Signout(ctx context.Context, uid uint) error ● ValidateIDToken(tokenString string) (*model.User, error) ● ValidateRefreshToken(refreshTokenString string) (*model.RefreshToken, error)

(b) Token Service Interface

 <i>PipelineService</i>
● SyncAsyncTasks() ● Get(id uint) (*model.Pipeline, error) ● GetPipelineSchedule(id uint) (*model.PipelineSchedule, error) ● GetPipelineSchedules(id uint) ([]model.PipelineSchedule, error) ● GetByOwner(ownerId uint) ([]model.Pipeline, error) ● Create(userId uint, name string, definition string) error ● CreatePipelineSchedule(pipelineID uint, uniqueOccurrence time.Time, cronExpression string) error ● Update(pipeline *model.Pipeline) error ● Delete(id uint) error ● DeletePipelineSchedule(id uint) error

(c) Pipeline Service Interface

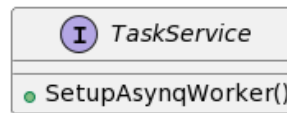
 <i>RunService</i>
● Get(id uint) (*model.Run, error) ● GetByPipeline(pipelineID uint) ([]model.Run, error) ● FindRunStepStatusesByRun(runID uint) ([]model.RunStepStatus, error) ● FindHumanFeedbackQueriesByRunID(runID uint) ([]model.HumanFeedbackQuery, error) ● FindHumanFeedbackQueryByID(queryID uint) (*model.HumanFeedbackQuery, error) ● FindHumanFeedbackQueriesByStepID(runID uint, runStepStatusID uint) ([]model.HumanFeedbackQuery, error) ● FindHumanFeedbackRectsByHumanFeedbackQueryID(humanFeedbackQueryID uint) ([]model.HumanFeedbackRect, error) ● FindHumanFeedbackQueryStatusByID(queryStatusID uint) (*model.QueryStatus, error) ● Create(pipeline model.Pipeline) (model.Run, error) ● CreateRunStepStatus(runID uint, stepID int, stepName string, runStatusID uint, errorMessage string) error ● CreateHumanFeedbackQuery(epoch uint, runID uint, stepID int, queryID uint, rects []uint) error ● Execute(runID uint) error ● Resume(runID uint) error ● Update(run *model.Run) error ● UpdateRunStepStatus(run *model.RunStepStatus) error ● UpdateHumanFeedbackQuery(query *model.HumanFeedbackQuery) error ● UpdateHumanFeedbackRects(rects []model.HumanFeedbackRect) error ● Delete(id uint) error ● DeleteRunStepStatus(id uint) error ● NewRunPipelineTask(pipelineID uint, runID uint, graph string) (*async.Task, error) ● NewResumeRunPipelineTask(pipelineID uint, runID uint, graph string, stepID int) (*async.Task, error) ● HandleRunPipelineTask(ctx context.Context, t *async.Task) error ● HandleScheduledRunPipelineTask(ctx context.Context, t *async.Task) error ● UpdateRunStatus(runID uint, statusID uint, stepWaitingFeedback int, errorMessage string) error

(d) Run Service Interface

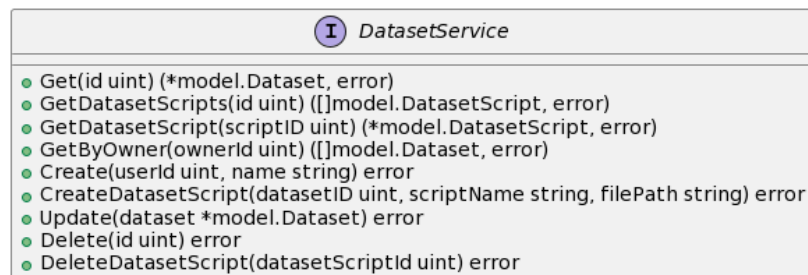
 <i>StepService</i>
● NewStepInstance(pipelineID uint, runID uint, nodeType string, nodeDescription model.NodeDescription) (*steps.Step, error) ● NewEdgeInstance(pipelineID uint, runID uint, edgeType string, nodeDescription model.NodeDescription) (*steps.Edge, error)

(e) Step Service Interface

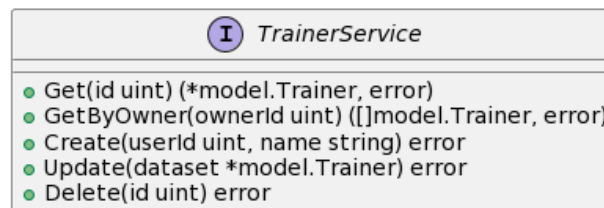
Figure 4.2: Services Layer Part 1



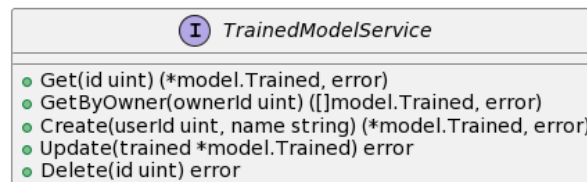
(a) Task Service Interface



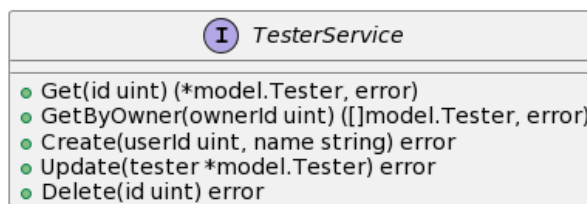
(b) Dataset Service Interface



(c) Trainer Service Interface



(d) Trained Model Service Interface



(e) Tester Service Interface

Figure 4.3: Services Layer Part 2

- **Run Repository** - Provides the methods listed in Figure 4.4d for run fetching, creation, updating, execution, resuming and deletion that interact directly with the PostgreSQL database.
- **Dataset Repository** - Provides the methods listed in Figure 4.5a for dataset handlers fetching, creation, editing and deletion, that interact directly with the PostgreSQL database.
- **Trainer Repository** - Provides the methods listed in Figure 4.5b for trainer scripts fetching, creation, editing and deletion, that interact directly with the PostgreSQL database.
- **Trained Model Repository** - Provides the methods listed in Figure 4.5c for trained models fetching, creation, editing and deletion, that interact directly with the PostgreSQL database.
- **Tester Repository** - Provides the methods listed in Figure 4.5d for tester scripts fetching, creation, editing and deletion, that interact directly with the PostgreSQL database.

4.2.2 Authentication

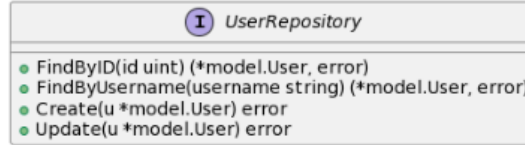
User authentication is handled with claims represented by JWTs [197] and signed using RS256, one of the recommended algorithms in the specification for JWS [198], which is used by JWT for encryption. Claims issuing follows the logic bellow (Figure 4.6):

1. A user is registered or logs in.
2. A pair of JWT Claims Sets [197] is generated, one with a smaller expiration date called id/login token, the other with a longer expiration date called refresh token.
3. The refresh token ID and user ID are stored in *Redis*.
4. The id/login and refresh token structures are signed using RS256 with a private key from a pair of public and private keys generated beforehand.
5. These signed structures are returned to the user.

4.2.3 REST API

All the requests - apart for the POSTs to `/api/user/login` and `/api/user/signup` - must contain authentication information in the *Auhtorization* header (a signed JWT token provided by the web server after logging in). The following endpoints and methods are provided by the web server:

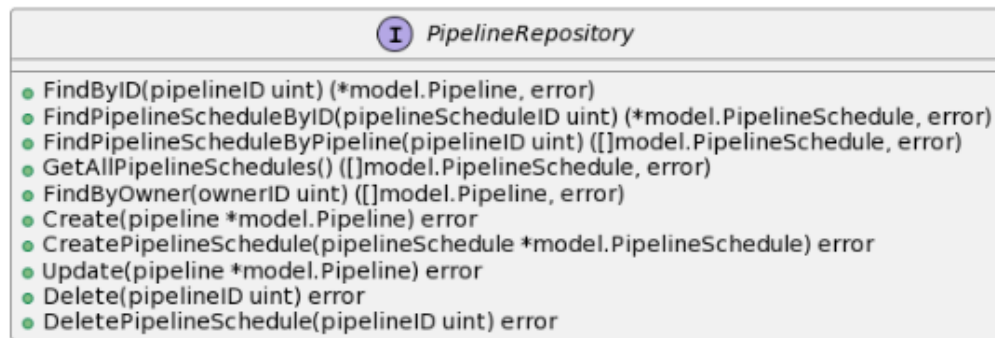
- **User API**
 - **POST `/api/user/`** - Edits user information.



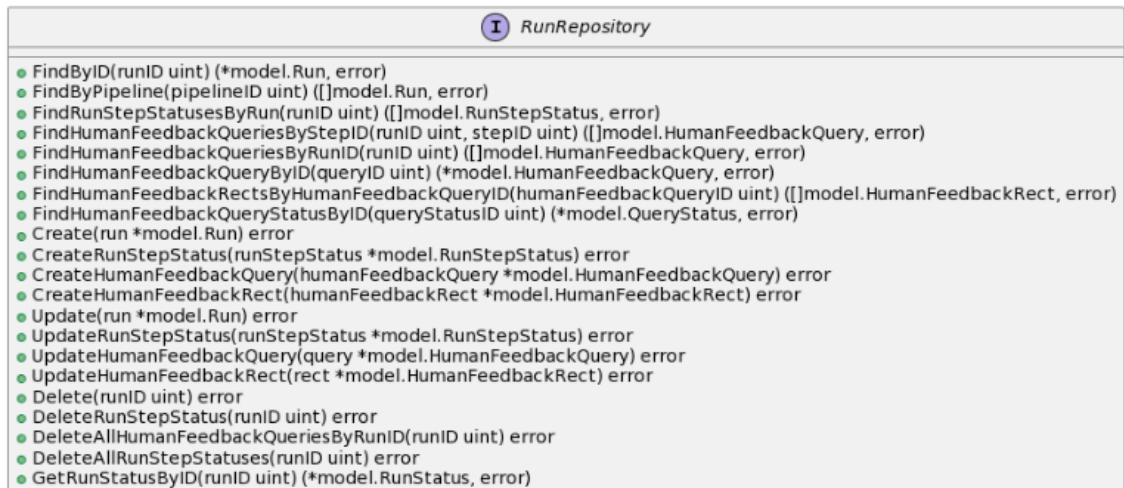
(a) User Repository Interface



(b) Token Repository Interface

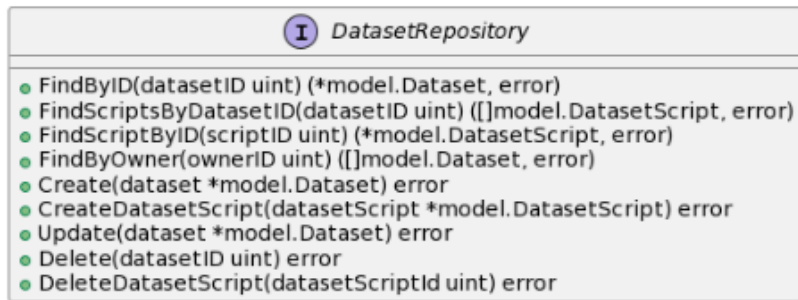


(c) Pipeline Repository Interface

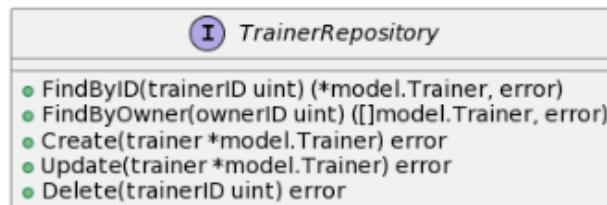


(d) Run Repository Interface

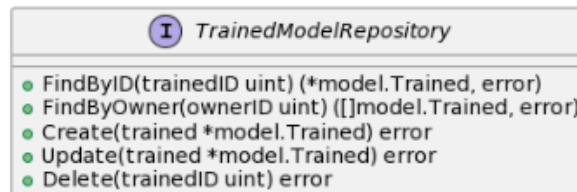
Figure 4.4: Repository Layer Part 1



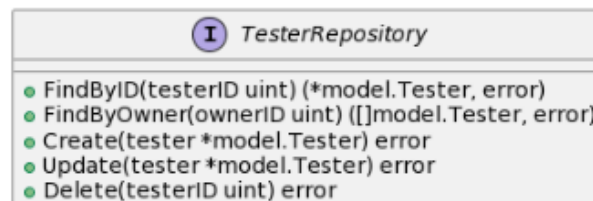
(a) Dataset Repository Interface



(b) Trainer Repository Interface



(c) Trained Model Repository Interface



(d) Tester Repository Interface

Figure 4.5: Repository Layer Part 2

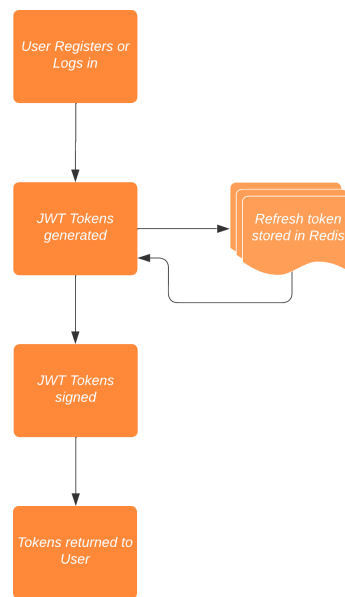


Figure 4.6: Authentication tokens generation

- **POST /api/user/login** - Logs in a user and generates a new pair of login and refresh tokens.
- **POST /api/user/signup** - Registers a new user.
- **POST /api/user/tokens** - Gets a new login token if provided with a still valid refresh token.
- **POST /api/user/signout** - Signs out a user and removes its tokens from memory.

• Pipeline API

- **GET /api/pipeline/** - Gets all the pipelines for the currently logged in user.
- **POST /api/pipeline/** - Creates a new pipeline.
- **DELETE /api/pipeline/** - Delete a pipeline.
- **GET /api/pipeline/:id** - Gets a single pipeline, identified by its *id*.
- **POST /api/pipeline/:id** - Updates a the pipeline information, most importantly its definition, given its *id*.
- **GET /api/pipeline/:id/schedule** - Gets all the scheduled pipeline execution records for a given pipeline identified by its *id*, past and present.
- **POST /api/pipeline/:id/schedule** - Creates a new scheduled pipeline run record, which can be a single occurrence or a repeatable event described by a cron expression.
- **DELETE /api/pipeline/:id/schedule** - Deletes an existing scheduled pipeline execution.

- **POST /api/pipeline/:id/file** - Uploads a file to the web server, associating it to a given step within a pipeline.

- **Run API**

- **GET /api/run/** - Gets all the runs for the currently logged in user.
- **GET /api/run/:id** - Gets all the runs for the pipeline identified by the *id*.
- **POST /api/run/:id** - Creates a run for the pipeline identified by the *id*.
- **POST /api/run/execute/:runID** - Executes a run identified by the *id*, resetting its state and deleting any previously associated information.
- **POST /api/run/resume/:runID** - Executes a run identified by the *id*, picking up its execution from the state and step it was suspended at.

- **Run Results API**

- **GET /api/runresults/:id** - Gets all the information related to the current results of run (identified by its *id*) - the overall and step statuses, the feedback queries, the tail end of the log file (along with the URL to the full log).
- **GET /api/runresults/:id/log** - Gets just the tail end of the log file (along with the URL to the full log) associated with a run (identified by its *id*).

- **Feedback API**

- **GET /api/feedback/:id** - Get not resolved feedback query information for a run identified by *id*.
- **GET /api/feedback/:id** - Submits feedback information for not yet resolved queries, for a run identified by *id*.
- **GET /api/feedback/:id/query/:queryId** - Gets a specific feedback query with id *queryId* for a run identified by *id*.

- **Dataset API**

- **GET /api/dataset/** - Gets all the dataset handlers for a specific user.
- **POST /api/dataset/** - Creates a dataset handler for a specific user, given a name.
- **GET /api/dataset/:id** - Gets a specific dataset handler identified by its *id*.
- **POST /api/dataset/:id/file** - Updates a dataset handler identified by its *id*, uploading a *python* file that implements it.
- **DELETE /api/dataset/** - Deletes a specific dataset handler.

- **Trainer API**

- **GET /api/trainer/** - Gets all the trainers for a specific user.

- **POST /api/trainer/** - Creates a trainer for a specific user, given a name.
- **GET /api/trainer/:id** - Gets a specific trainer identified by its *id*.
- **POST /api/trainer/:id/file** - Updates a trainer identified by its *id*, uploading a *python* file that implements it.
- **DELETE /api/trainer/** - Deletes a specific trainer.

- **Tester API**

- **GET /api/tester/** - Gets all the testers for a specific user.
- **POST /api/tester/** - Creates a tester for a specific user, given a name.
- **GET /api/tester/:id** - Gets a specific tester identified by its *id*.
- **POST /api/tester/:id/file** - Updates a tester identified by its *id*, uploading a *python* file that implements it.
- **DELETE /api/tester/** - Deletes a specific tester.

- **Trained Model API**

- **GET /api/tester/** - Gets all the trained models for a specific user.
- **POST /api/tester/** - Creates a trained model for a specific user, given a name.
- **GET /api/tester/:id** - Gets a specific trained model identified by its *id*.
- **POST /api/tester/:id/file** - Updates a trained model identified by its *id*, uploading a *.pt PyTorch* file that implements it.
- **DELETE /api/tester/** - Deletes a specific trained model.

4.2.4 Concurrent Pipeline Runs Execution

While concurrent execution of HTTP requests is assured by the Gin Framework, as it uses the Goroutines [199] dependent *http.Serve* function from Go's standard lib behind the scenes - concurrent execution of pipeline run's is still needed. This is assured by using *Redis* as a message broker for a task queue. The Go Asynq library [200] provides just this.

Figure 4.7 represents the basic architecture behind Asynq: a client creates tasks that are submitted and queued in the message broker in queues with different priorities. These tasks are then picked up by worker threads on the server and processed. In reality, the http threads that serve run execution requests are the clients that enqueue tasks into a single queue, as all run execution tasks have the same priority, while a separate Goroutine on the web server, that sets up the asynq server during startup, polls for awaiting tasks and spawns up to 10 concurrent task processing threads. These task can be one of two types:

- **Execute Run** - Executes a pipeline run from the start, removing any previous working data and results

¹ Adapted from Example use case in [200]

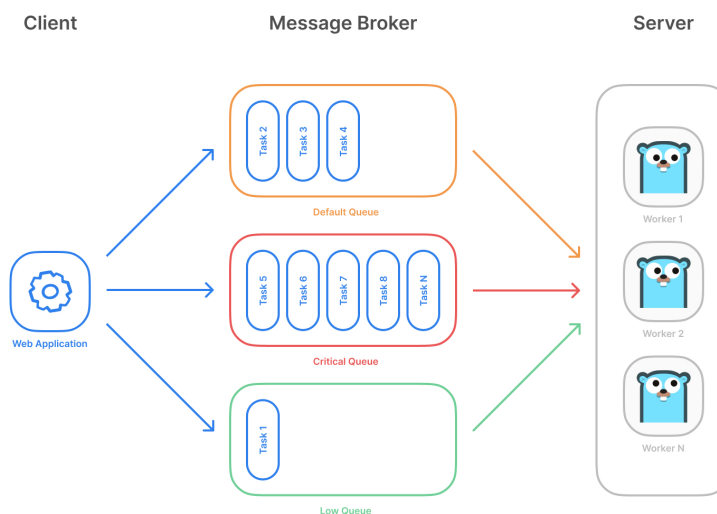


Figure 4.7: Asynq architecture ¹

- **Resume Run** - Resumes execution of a pipeline run that had been paused awaiting for feedback; previously associated working data and results are kept as they are needed to continue the execution until a final state - Error or Success - is obtained.

Within the first type, there are two sub-types:

- Immediate execution tasks - or, more precisely, executed as soon as there's a worker thread available. They are executed against an already existing run entity.
- Scheduled execution tasks - that are queued as tasks in *Redis* but only executed at a set time in the future. They create a new run entity object when picked up by the service worker thread. If they're recurrent events, only the first event is queued as a task and when it is picked up for execution by the server the next occurrence will be queue and so on. However, their execution will be compromised if *Redis* is shut down after they are queued. In that case, the web server must be restarted, as during startup the scheduled tasks queued in *Redis* are synchronized: all of the unique occurrence and cron scheduled tasks are fetched from the database and compared to the *Redis* scheduled tasks; if any of them is missing from the *Redis* queue, it is queued up.

4.2.5 Environment variables

The backend is configured mainly through environment variables divided into 3 groups:

- **Web Server** (path related variables should contain paths for which the OS user running the web server go process has read and write permissions for):

- *WEB_SERVER_PORT* - the port the webserver will be listening to HTTP connections and requests on.
- *ID_TOKEN_DURATION* - how long an id/login token will be valid for, in seconds.
- *REFRESH_TOKEN_DURATION* - how long an refresh token will be valid for, in seconds.
- *PIPELINES_WORK_DIR* - the directory onto which pipeline execution related files will be created on or copied.
- *FILE_UPLOAD_DIR* - the directory onto which files uploaded through the REST API will be created.
- *RUN_LOGS_DIR* - the directory onto which run execution logs will be created.
- *RUN_LOG_FILE_NAME* - the file name for the a run's execution log.

- **PostgreSQL Connection:**

- *POSTGRES_HOST* - the hostname or ip address of the host where the PostgreSQL DB is installed.
- *POSTGRES_PORT* - the PostgreSQL DB port.
- *POSTGRES_USER* - the PostgreSQL user.
- *POSTGRES_PASSWORD* - the PostgreSQL user password.
- *POSTGRES_DB* - the PostgreSQL database.
- *POSTGRES_SSL* - whether to enable SSL mode.
- *POSTGRES_TIMEZONE* - the PostgreSQL timezone.

- **Redis:**

- *REDIS_HOST* - the hostname or ip address of the host where *Redis* is installed.
- *REDIS_PORT* - the port through which to connect to the *Redis*.

4.2.6 Pipeline Steps

When a Run is created for a Pipeline, the latter's definition is copied onto the Run entity object, capturing it as it was at that moment in time, so that updates to the Pipeline's definition won't affect the run. A Pipeline's definition consists in the steps that make it up as well as the connections between them; this is represented in a JSON array object; taking advantage of the fact the frontend pipeline steps editor is based on Vue Flow [201] which produces one already, a few more fields are added for the backend. A pipeline definition example would be:

```
[
  {
    "type": "checkoutRepo",
    ...
  }
]
```

```

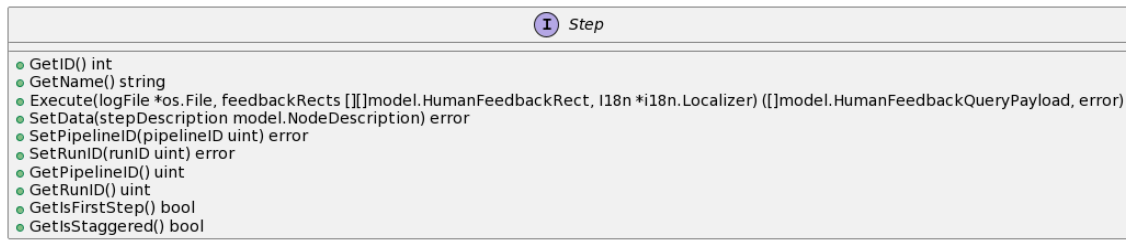
    "data": {
      "id": "0",
      "nameAndType": {
        ...
      },
      "stepConfig": {
        ...
      },
    },
    ...
  },
  {
    "type": "humanFeedbackNN",
    ...
    "data": {
      "id": "1",
      "nameAndType": {
        ...
      },
      "stepConfig": {
        ...
      },
    },
    ...
  },
  {
    ...
    "type": "smoothstep",
    "source": "1",
    "target": "0",
    ...
  }
]

```

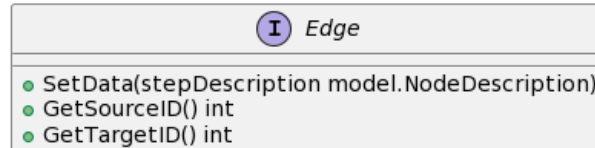
Storing this JSON array as a string in the database avoids the need to perform marshalling and unmarshalling operations every time we need to fetch or edit a pipeline's definition. However, unmarshalling does occur just before executing a run, because the step JSON representations need to be converted into Go struct objects that implement a step interface (Figure 4.8a) allowing them to have type specific individual logic while the top layer of code that calls their execution remains agnostic to it.

After unmarshalling (Figure 4.10):

1. The Steps are collected into a linked list, which in fact is a graph in which the vertices have only both one input and output edges. The decision to use a graph was to allow future



(a) Step Interface



(b) Edge Interface

Figure 4.8: Step and Edge interfaces

changes to the Steps definition to support a graph like structure instead of the simpler linked list one.

2. With the Steps collected into a single structure, the execution can start by navigating through the Steps. If we're executing a run, previously created Step Statuses and run related files are deleted from the database and disk, respectively, if any exist. New directories for the run log and files are then created, at *RUN_LOGS_DIR/pipelines/<pipelineID>/<runID>* and *PIPELINES_WORK_DIR/<pipelineID>/<runID>*, respectively. If we're resuming a run, this is skipped.
3. The Run Status is updated to "Executing".
4. The starting Step to execute is fetched. If we're executing a run, that is the first Step in the definition; if we're resuming a run, that's the Step that has a Step Status with the value "Waiting for Feedback" and Human Feedback Queries submitted.
5. The Step Status is updated to "Executing".
6. The Step is Executed.
7. If the Step execution failed: item
 - The Step Status is updated to "Error"
 - The Run Status is also updated to "Error".

If the Step succeeded:

- If there's feedback required by the Step, Human Feedback Query entities are created, the Step Status is updated to "Waiting for Feedback" along with the Run Status.

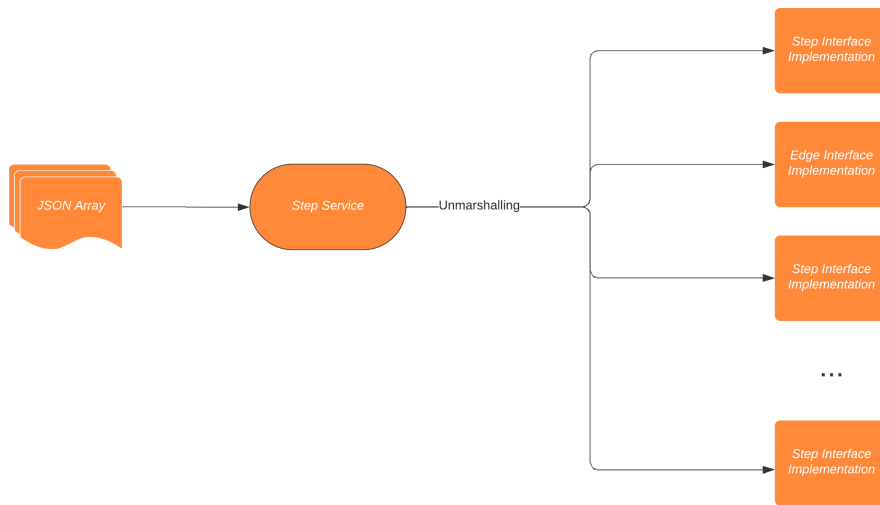


Figure 4.9: Pipeline Definition Unmarshalling

- If there's no feedback required by the Step, the Step Status is updated to "Success" and the next Step in the list is fetched; if there are no Steps left, the Run Status is updated to "Success", otherwise we execute the next step and repeat the cycle.

There are several types of steps, divided into 6 categories:

- **General** - these are steps that perform generic or utilitarian tasks, not necessarily related with ML.
 - **Checkout Repository** - Checkout a Git [202] repository into the run's local working directory.
 - **Shell Script** - Copies a shell script written by the user into the run's local working directory and runs it.
 - **Python Script** - Copies a Python script written by the user into the run's local working directory and runs it.
- **HITL** - these are steps that perform built-in HITL algorithm tasks.
 - **Neural Network Training with Human Feedback** - Uses the training and testing algorithm described in [35], adapted to conform to the logic described in Figure 4.10. It can be used with the pre-packaged datasets handlers for *ISIC 2017* [203], *APTOS 2019* [204], American National Cancer Institute (NCI) cervigrams, *ROSE-Youtu* [205] and PornographyXXX. Several model variations can be trained on of these datasets, with the following *PyTorch* implementations to choose from:
 - * **AlexNet** [206] - *alexnet*
 - * **ConvNeXt** [207] - *convnext_tiny*, *convnext_small*, *convnext_base*, *convnext_large*
 - * **DenseNet** [208] - *densenet121*, *densenet161*, *densenet169*, *densenet201*

- * **EfficientNet** [209] - *efficientnet_b0, efficientnet_b1, efficientnet_b2, efficientnet_b3, efficientnet_b4, efficientnet_b5, efficientnet_b6, efficientnet_b7*
- * **EfficientNetV2** [210] - *efficientnet_v2_s, efficientnet_v2_m, efficientnet_v2_l*
- * **GoogLeNet** [211] - *googlenet*
- * **Inception V3** [212] - *inception_v3*
- * **MaxVit** [213] - *maxvit_t*
- * **MNASNet** [214] - *mnasnet0_5, mnasnet0_75, mnasnet1_0, mnasnet1_3*
- * **MobileNet V2** [215] - *mobilenet_v2*
- * **MobileNet V3** [216] - *mobilenet_v3_large, mobilenet_v3_small*
- * **RegNet** [217] - *regnet_y_400mf, regnet_y_800mf, regnet_y_1_6gf, regnet_y_3_2gf, regnet_y_8gf, regnet_y_16gf, regnet_y_32gf, regnet_y_128gf, regnet_x_400mf, regnet_x_800mf, regnet_x_1_6gf, regnet_x_3_2gf, regnet_x_8gf, regnet_x_16gf, regnet_x_32gf*
- * **ResNet** [218] - *resnet18, resnet34, resnet50, resnet101, resnet152*
- * **ResNeXt** [219] - *resnext50_32x4d, resnext101_32x8d, resnext101_64x4d*
- * **ShuffleNet V2** [220] - *shufflenet_v2_x0_5, shufflenet_v2_x1_0, shufflenet_v2_x1_5, shufflenet_v2_x2_0*
- * **SqueezeNet** [221] - *squeezenet1_0, squeezenet1_1*
- * **SwinTransformer** [222] - *swin_t, swin_s, swin_b*
- * **SwinTransformer V2** [223] - *swin_v2_t, swin_v2_s, swin_v2_b*
- * **VGG** [224] - *vgg11, vgg11_bn, vgg13, vgg13_bn, vgg16, vgg16_bn, vgg19, vgg19_bn*
- * **VisionTransformer** [225] - *vit_b_16, vit_b_32, vit_l_16, vit_l_32, vit_h_14*
- * **Wide ResNet** [226] - *wide_resnet50_2, wide_resnet101_2*

Similarly, several parameter optimization algorithms can be chosen:

- * **Adadelta** [227]
- * **Adagrad** [228]
- * **Adam** [229]
- * **AdamW** [230]
- * **SparseAdam** [229]
- * **Adamax** [229]
- * **ASGD** [231]
- * **SGD** [232]
- * **RAdam** [233]
- * **Rprop** [234]
- * **RMSprop** [235]
- * **NAdam** [236]
- * **LBFGS** [237]

- **Datasets** - Created based on existing dataset handlers; they use their assigned *PyTorch* dataset handler script, copying it into the local run files directory, onto the *datasets/handler.py* path. They should implement a *Python* a method named *get_data_classes()* that returns a tuple with the possible data label classes, a class named *TrainDataset* and a class named *ValidationDataset* that both extend the *PyTorch torch.utils.data.Dataset* (just a convention so that the HITL, Trainer and Tester scripts can easily access it).
- **Trained Models** - Created based on existing trained models weights; they use their assigned *PyTorch .pt* file, copying it into the local run files directory, onto the *base_models/trained_model.pt* path. Testers and Trainers that want to use these models should locate them under this path.
- **Trainers** - Created based on existing trainer scripts; they use their assigned *PyTorch* trainer script, copying it into the local run files directory and running it. If human interaction through feedback is required, they should implement a *Python* the same basic logic that a Neural Network Training with Human Feedback Step does, that is, after each epoch, if queries are generated, they must create a CSV file for each one at *epochs/n/query_i_rects.csv* along with the respective image file at *epochs/n/query_i_image.png* (*n* and *i* being the epoch number and the query number, respectively) and store in the disk any data it needs to resume its execution after feedback is given in the web application. For that, the script file must support the usage of the *resume_epoch* argument that indicates the epoch the execution was paused at, previously.
- **Testers** - Created based on existing model testing scripts; they use their assigned *PyTorch* tester script, copying it into the local run files directory and running it.

4.3 Frontend

The frontend web application was developed using the Javascript framework Vue 3 [189], starting from an existing MIT licensed template (Admin One [190]), along with Tailwind [191] for CSS in a single page application design with Vue Router [238] serving the different views. Vite [192] was used as a development server and for tooling. Data fetching, creation, updating and deletion is done through the endpoints listen in 4.2.3.

There are 14 different views:

- */login* - Log In View
- */signup* - Sign Up View
- */pipelines* - Pipelines View
- */pipelines/edit/:id* - Pipeline Edit View
- */runs* - Runs View

- /pipelines/runs/:id - Pipeline Runs View
- /runresults/:id - Run Results View
- /feedback/:id - Feedback View
- /feedback/:id/query/:queryId - Single Feedback Query View
- /datasets - Datasets Handlers View
- /trainers - Trainers View
- /testers - Testers View
- /trained - Trained Models View

4.3.1 Log In and Sign Up

In order to use the application, users must register and choose a username and password with between 5 and 30 characters long. After log in or user registration is completed (using the screens in Figures 4.11a and 4.11b), the user is redirected to the Pipelines page and authenticated requests are made by passing the signed id/login token on the *Authorization* header. Before using them though, the client should check if the locally stored tokens are still valid. Depending on their validity, one of three scenarios may happen (Figure 4.12):

1. Id/login and refresh tokens are both valid: authenticated requests to the server are made using the *Authorization* header with the value of the signed id/login token.
2. The id/login token is expired but the refresh token is valid: a request for a new id/login token is sent to the server, passing the old id/login and refresh tokens.
3. Both the id/login token and refresh tokens are expired: the user is sent to the login page.

4.3.2 Pipelines

Accessible through the side menu. Pipelines are listed in a table. New ones can be created, existing one can be deleted or edited (Figure 4.13).

4.3.3 Pipeline Edit View

Accessible when clicking in one of the Edit buttons in 4.3.2. Pipelines are edited in this screen; new steps can be added, existing steps can be edited by double clicking on them or deleted by clicking the "X" popup icon when they're selected (Figure 4.13). Steps can also be moved freely on the Vue Flow [201] canvas for better user viewing. The scheduled executions of the pipeline are also displayed in a table manner and new ones can be added and existing ones can be deleted. Changes can be saved by clicking the "Save" button or discarded by clicking the "Cancel" button.

4.3.4 Runs View

Accessible through the side menu clicking on the "Runs" option. Every Pipeline and its Runs are listed in a table with nested rows. New Runs can be created, existing ones can be executed, resumed, given feedback (depending on their status) or get their results checked (Figure 4.15).

4.3.5 Pipeline Runs View

Accessible when clicking in one of the View buttons in 4.3.4. Shows the same and the same actions can be performed as in the Runs View but for a single chosen Pipeline (Figure 4.16).

4.3.6 Run Results View

Accessible when clicking in a View Run button in 4.3.4 or 4.3.5. Shows the current results for a Run.

- If a run is Waiting for Feedback, a button to the Feedback View is added so that Feedback can be submitted.
- Each Step displayed on the canvas has a tag with their status.
- The latest 25 lines in the Run log are displayed and updated every 5 seconds. A button to show the full logs is also present.
- A table showing all the Run generated feedback queries is displayed.

4.3.7 Feedback View

Users can access this view by clicking the "Give Feedback" button in 4.3.6. It shows the current not Resolved Feedback Queries for a Run, linked to the last Step that made the run's execution pause. User's can select the rectangular areas that are most relevant for the purpose of training algorithm, either by clicking those areas on the image (which will make the corresponding rectangle turn green) or by checking the corresponding rows in the table below (Figure 4.18). Clicking the submit button submits the feedback given, making the Run ready to be resumed again.

4.3.8 Single Feedback Query View

Users can access this view by clicking the View button in Feedback table in 4.3.6. It shows only the query selected. Similarly to 4.3.7, users can select and submit the relevant rectangular areas (Figure 4.19), but only if the query is not in the Resolved status.

4.3.9 Datasets Handlers, Trainers, Testers and Trained Models Views

Users can access these view by clicking the respective button in the side bar. All of them present a table representation of Datasets Handlers, Trainers, Testers and Trained Models (Figure 4.20). New records can be created, existing ones may be deleted or edited, by uploading a different *.pt* file for a Trained Model or a *.py* file for the rest.

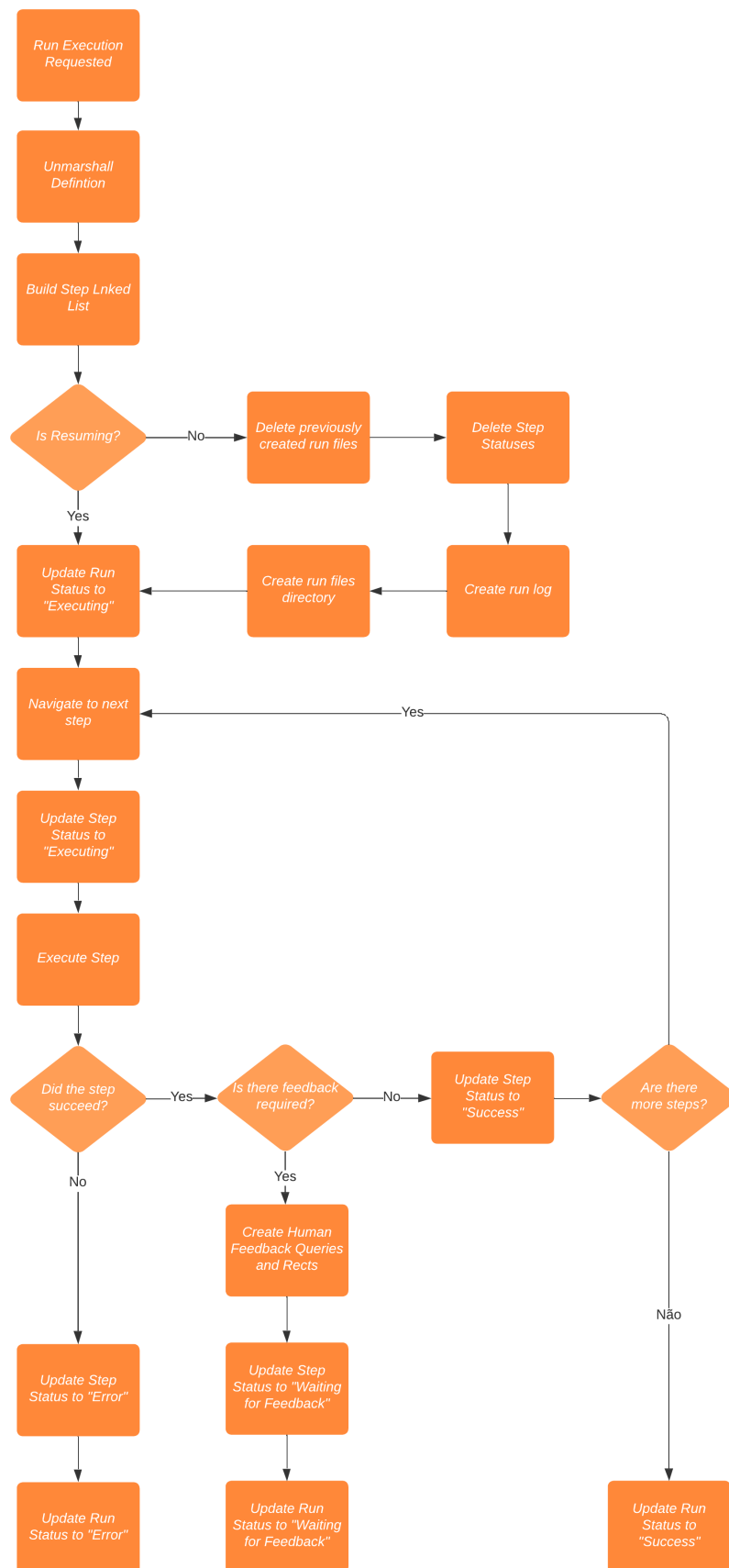


Figure 4.10: Steps Execution Logic

Login

Username

user1

Please enter your username

Password

*

Please enter your password

Submit
Sign Up

Sign Up

Username

user1

Please enter a username

Password

*

Please enter a password

Submit
Cancel

(a) Log In
(b) Sign Up

Figure 4.11: Log In and Sign Up Pages

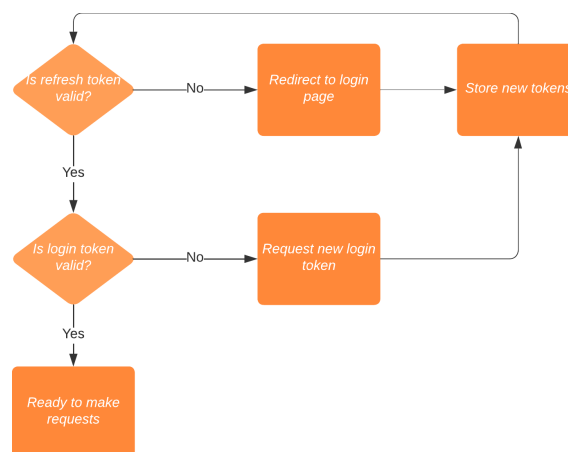


Figure 4.12: Client token handling

Pipelines						
ID	Name	Modified	Created			
21	Pipeline 5	July 28th 2023, 7:24:38 pm	July 28th 2023, 7:15:58 pm			
20	Pipeline 5	July 25th 2023, 12:17:14 pm	July 25th 2023, 12:17:14 pm			
19	Pipeline 4	July 23rd 2023, 4:30:05 pm	July 23rd 2023, 4:30:05 pm			
18	Pipeline 3	July 23rd 2023, 1:24:05 am	July 23rd 2023, 1:10:08 am			
2	Pipeline 2	July 18th 2023, 9:52:32 pm	April 26th 2023, 10:23:14 pm			

1
Page 1 of 1

Figure 4.13: Pipelines View

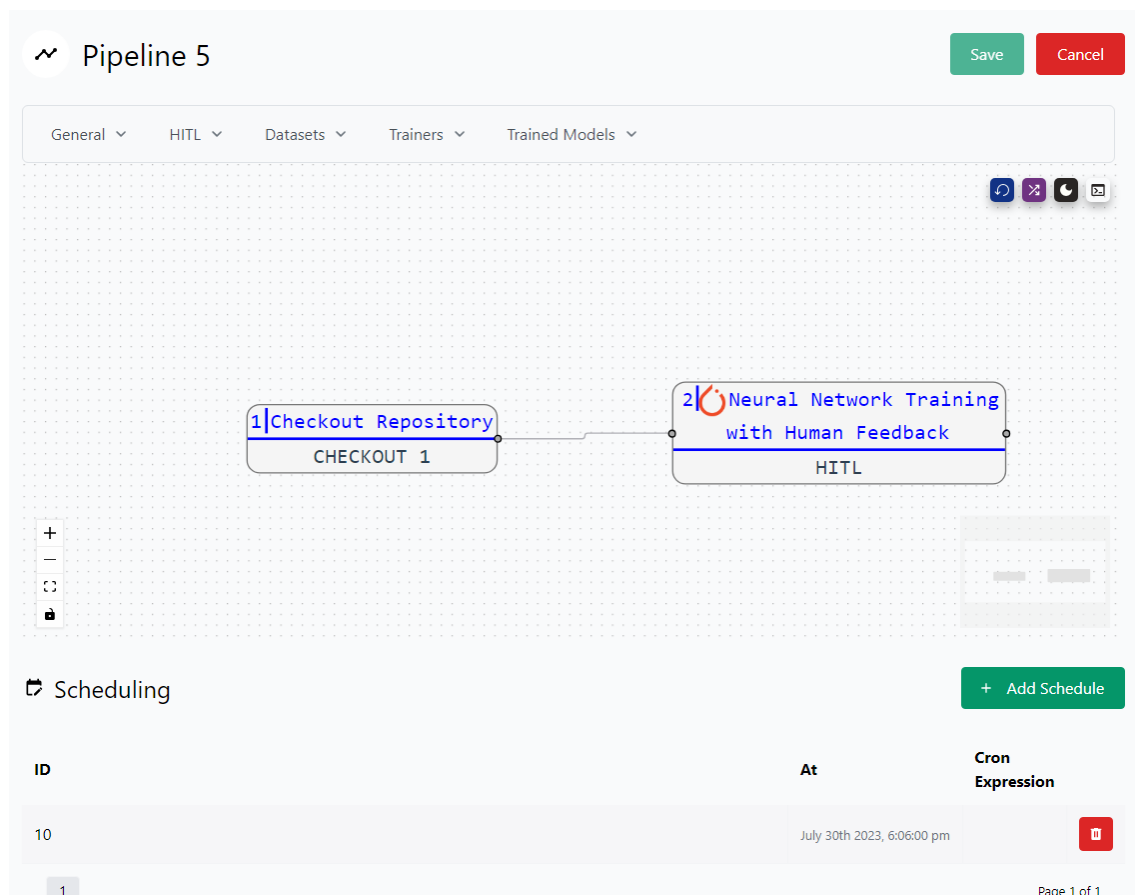


Figure 4.14: Pipeline Edit View

Runs

Name	Last Run	
> Pipeline 5	-	
> Pipeline 5	-	
> Pipeline 4	-	
▼ Pipeline 3	July 24th 2023, 11:38:26 am	

ID	Status	Created	Last Run	
43	Waiting Feedback	July 24th 2023, 11:25:47 am	July 24th 2023, 11:38:26 am	
42	Waiting Feedback	July 24th 2023, 10:46:25 am	July 24th 2023, 10:59:41 am	
40	Executing	July 23rd 2023, 9:14:02 am	July 23rd 2023, 7:01:30 pm	
39	Executing	July 23rd 2023, 8:56:03 am	July 23rd 2023, 8:56:12 am	
38	Waiting Feedback	July 23rd 2023, 2:14:45 am	July 23rd 2023, 2:14:49 am	

1

2

Page 1 of 2

Figure 4.15: Runs View

Pipeline 18 Runs

ID	Status	Created	Last Run	
43	Waiting Feedback	July 24th 2023, 11:25:47 am	July 24th 2023, 11:38:26 am	
42	Waiting Feedback	July 24th 2023, 10:46:25 am	July 24th 2023, 10:59:41 am	
40	Executing	July 23rd 2023, 9:14:02 am	July 23rd 2023, 7:01:30 pm	
39	Executing	July 23rd 2023, 8:56:03 am	July 23rd 2023, 8:56:12 am	
38	Waiting Feedback	July 23rd 2023, 2:14:45 am	July 23rd 2023, 2:14:49 am	

1

2

Page 1 of 2

Figure 4.16: Pipeline Runs View

Pipeline 3 - Run 43

Waiting Feedback

Give Feedback

1

Neural Network Training with Human Feedback

FDSAFSDFS

WAITING FEEDBACK

+

-

↺

🔒

Full log

```

plt.show()
/usr/src/dl/hidl/train.py:258: UserWarning: Matplotlib is currently using agg, which is a non-GUI backend, so cannot show the figure.
plt.show()
/usr/src/dl/hidl/train.py:274: UserWarning: Matplotlib is currently using agg, which is a non-GUI backend, so cannot show the figure.
plt.show()
Lowest entropy predictions after 4 epochs:
0.7733062
previous pred: tensor(0) new pred: 0 torch.Size([1])
query_pred: 0 query_label tensor(1)
avg_patches: torch.Size([1, 8, 8])
ranks shape: torch.Size([1, 8, 8])
set()
Successfully saved at: /usr/src/dl/work/pipelines/18/43/epochs/3/efficientNet_b1_10.0p_5e_low_entropy_epoch_3.pt
plot saved
plot saved
2023/07/24 10:38:26.382460 /usr/src/dl/backend/service/run_service.go:1076: Step with id 0 executed with success.
2023/07/24 10:38:26.525145 /usr/src/dl/backend/service/run_service.go:1167: Run with id 43 executed with success.

```

Human Feedback Queries

ID	Status	Created	Updated	Step ID	
14	Unresolved	July 24th 2023, 11:38:25 am	July 24th 2023, 11:38:25 am	-	

Figure 4.17: Run Results View

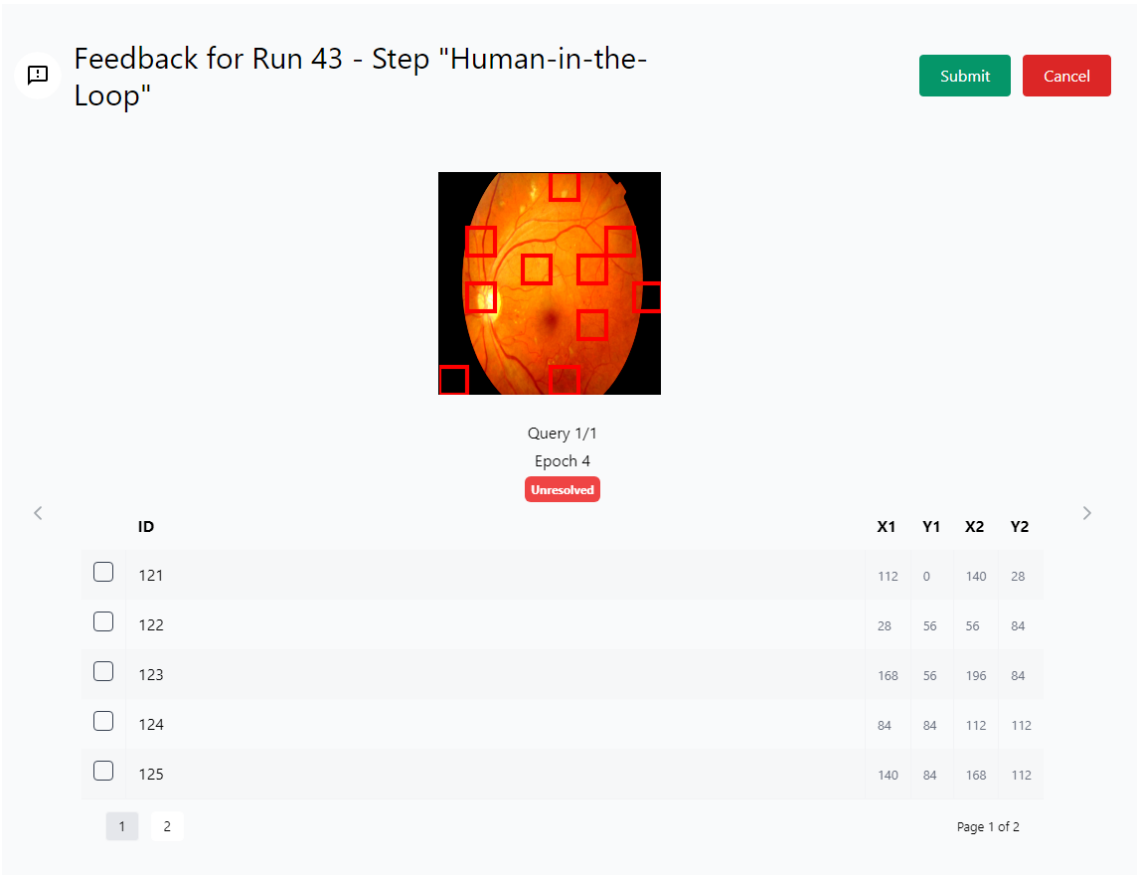


Figure 4.18: Feedback View

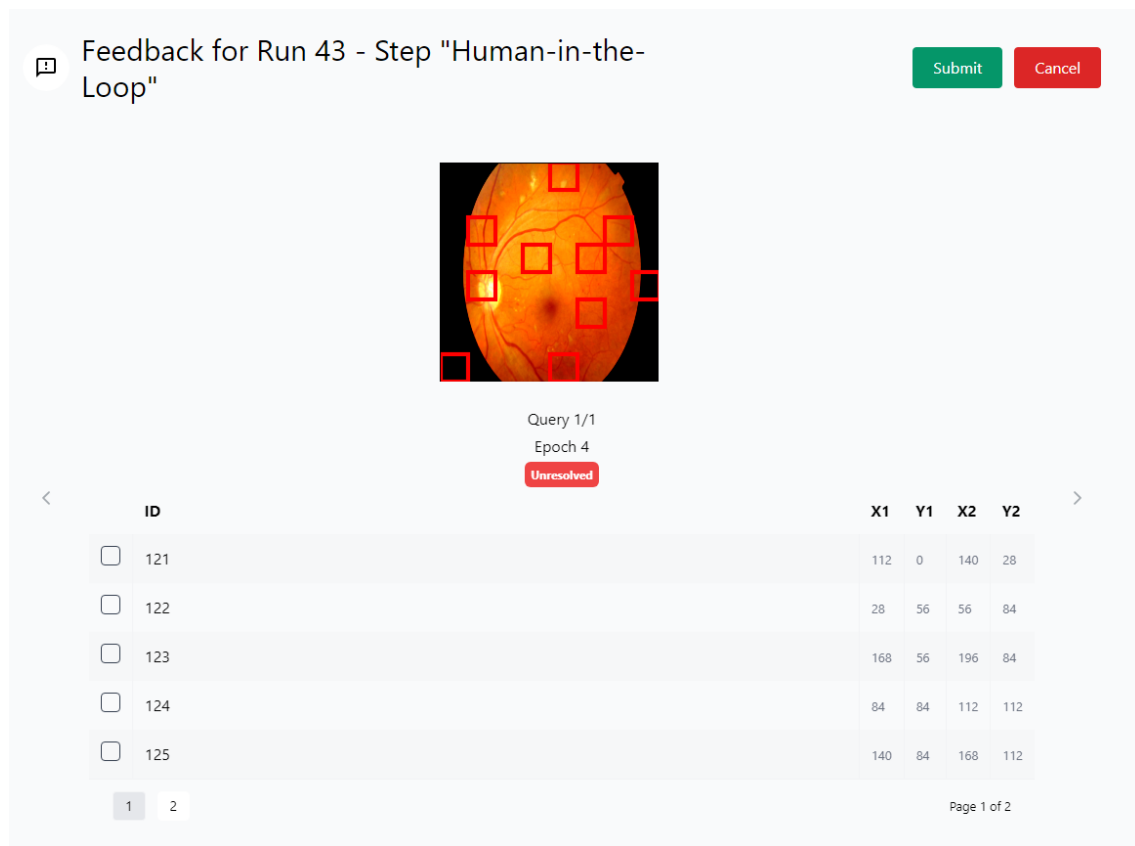


Figure 4.19: Single Query Feedback View

Dataset Handlers					Trainers				
ID	Name	File	Updated	Created	ID	Name	File	Updated	Created
2	APTOS 2019	b.py	July 25th 2023, 10:16:03 pm	July 24th 2023, 9:59:40 pm	1	A	b.py	July 25th 2023, 4:08:12 pm	July 25th 2023, 4:07:06 pm

(a) Dataset Handlers View

(b) Trainers View

Figure 4.20: Log In and Sign Up Pages

Testers					Trained Models				
ID	Name	File	Updated	Created	ID	Name	File	Updated	Created
1	Tester		July 25th 2023, 6:46:28 pm	July 25th 2023, 6:46:28 pm	1	Trained Model		July 25th 2023, 7:40:39 pm	July 25th 2023, 7:40:39 pm

(a) Testers View

(b) Trained Models View

Figure 4.21: Views

Chapter 5

Automated workflow examples

This chapter describes two examples of pipelines ran on the application described in 4, describing the datasets, models and training methods used.

5.1 HITL logic

The human oracle queries for both of the pipelines were generated using the logic proposed in [35] for each training epoch:

1. The model generates a prediction for the training images;
2. Predictions are selected by calculating their associated entropy values, sorting them in descending order and choosing the first x .
3. A xAI algorithm (*DeepLIFT* [239], even though other options that might have been better suited were not searched for) is run for the images associated with each prediction selected, returning a tensor with the pixel attributions which can be read as how much each pixel the model is focusing on each pixel.
4. Average pooling is applied to the attribution tensor, producing rectangular (more specifically, squared) regions that translates it to a visual interpretation on the image.
5. These rectangular regions are drawn over the images and presented to the human oracle which then selects the ones most relevant for the correct classification.
6. The selected rectangles are then translated into ones in a tensor otherwise full of zeros, multiplied by the gradients of the network and from the mean of the result an extra loss member is obtained, that is added to the already calculated cross-entropy loss.

5.2 Face Spoofing

Face spoofing consists of fooling a face recognition ML systems with a fake video or image, for example a paper print out of a face or a face picture displayed on a screen. The experiment

described in this section was inspired by previous work on it, specifically the binary classification task and the effect of background elements on the predictions which can be leveraged to provide a more informed decision by the model[31].

5.2.1 ROSE-Youtu

Commonly used datasets used for related research were *Idiap REPLAY-ATTACK* [240], *CASIA Face AntiSpoofing* [241] and *MSU mobile face spoofing* [242]. *Idiap REPLAY-ATTACK* [240] has 1200 videos taken on a MacBook webcam that represent 2 conditions: the first is uniform background and lighting and the second is complex background and natural lighting; spoofing attack videos were shot on a Canon PowerShot camera of iPad and iPhone 3GS displayed videos and paper printed faces. *CASIA Face AntiSpoofing* [241] has 600 videos with more face acquisition devices and different quality levels, with spoofing types including warping, cutting and replaying attacks. *MSU* has 280 videos of genuine and fake faces captured on a Laptop and Android phone camera, with two types spoofing attacks, printed photo attack and replay video attack. It is also divided into training and testing data.

ROSE-Youtu [205] was introduced after the already named datasets and focused on:

- face samples captured in real-world scenarios;
- the mediums used for spoofing are able to fool a system without anti-spoofing detection;
- several cameras, quality levels and illumination used to capture videos;
- several types of spoofing attacks;

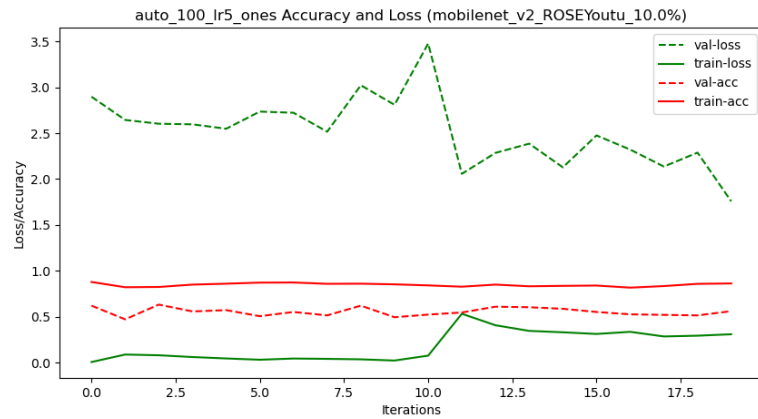
It contains more than 4000 videos of 25 subjects, more than *Idiap REPLAY-ATTACK*, *CASIA Face AntiSpoofing* and *MSU mobile face spoofing*, even though only 3350 videos of 20 subjects are available in its public version. There's a total of between 150 to 200 videos for each subject, captured on 5 mobile devices at a distance between 30 and 50 centimeters (using front-facing cameras with different camera resolutions) in 5 lighting conditions, with an average duration of 10 seconds. Besides videos of a genuine subject, there are 7 other types of videos representing attacks: 2 paper printed attacks, 2 replay attacks (of videos played on a Lenovo and Apple laptops) and 3 paper mask attacks (introducing three-dimensional information in the dataset). This dataset was divided into two parts: a training dataset and a testing dataset made up of 4 equally spaced frames extracted from each of the videos of the first 10 and the last 10 subjects, respectively; each frame was labeled according to the video it was extracted from: if the video was of a genuine subject it was labeled so, if it was from a video of any type attack it was labeled simply as an attack. There are thus two possible classes of data:

- **Class 0:** Genuine subject
- **Class 1:** Face spoofing attack

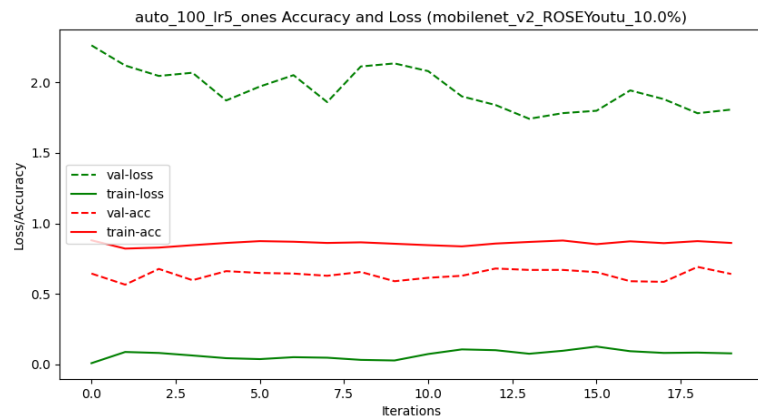
5.2.2 Model and Training

A MobileNet v2 [215] model with two outputs values which are activated with softmax was used. The optimization of the weights is done using the Adam algorithm [229] and the learning rate was set at 1×10^{-5} . A base model trained for 20 epochs on 10% of the training data (700 images) was created starting from a pre-trained MobileNetv2 on the ImageNet dataset (the default pre-trained option for MobileNetv2 offered in *PyTorch*), using also 10% of the testing data. Then, this base model was trained for 20 more epochs with oracle queries set up to start at the 10th. To compare the HITL results, an automatic training pipeline was also ran on the same base model for 20 epochs.

5.2.3 Results



(a) HITL



(b) Auto

Figure 5.1: Face Spoofing HITL vs Automatic training comparison

There was some difficulties when choosing the relevant areas in which the model should focus; of the offered choices few were clear cut indications of either a real or face attack image. This

is specially true for the attacks that most closely resemble real faces even for human observers, the replay attacks, as their most distinguishable characteristic is the overall foggy, less defined appearance and not specific regions of the image, unless the edges of the screen the image is displayed or a finger of the hand that holds the screen are visible, which is an obvious tell it is a face spoofing attack. The choosing of these regions was a focus not for these first queries, which proved to be a mistake; instead, the choosing of regions with bright spots of incident light on the screen were preferred. Additionally, almost all of these first queries were of the replay attack kind, which restricted feedback opportunities for images of the other attacks as well as real face images. Thus, all this might have lead to some choices that guided the model in the wrong direction for classification of the next epoch's testing and training data. These are all possible explanations to the unremarkable effects on the mitigation of the overfitting of the model on the training data; additionally the testing accuracies evolved similarly for both the HITL and auto training pipelines.

5.3 Pornographic content detection

Video content classification is widely used in all its distribution channels, be it television, movie theaters, internet streaming, to direct and try to restrict the access of said video contents to only the intended or legally allowed age group public. This classification is made manually, by people that watch and review the images showed and determine what type of audience should be recommended or allowed to watch them.

5.3.1 PornographyXXX

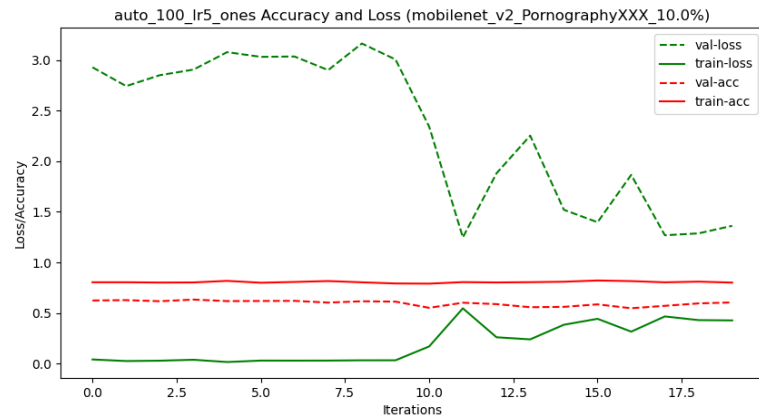
PornographyXXX is a dataset compiled of several extracted video frames from original content ranging from mostly innocuous (instructional videos, live music performance, live sports, product advertising, news shows), passing through age restricted and/or containing people with less clothing (tv shows with people in swimsuits in the pool or on the beach, sporting wrestling or fighting sports), to pornographic films. These constitute a scale of seemingly increasing difficulty when it comes to distinguishing between pornographic and non pornographic content. A total of 16727 frames are present, of which 6785 are categorized as easily identifiable non-pornographic, 3555 are categorized as difficult to identify as non-pornographic and 6387 are categorized as pornographic. For the purposes of the model's classification though, this data was labeled as simply pornographic or non-pornographic, making its task a binary classification one. There are thus two possible classes of data:

- **Class 0:** Non-pornographic
- **Class 1:** Pornographic

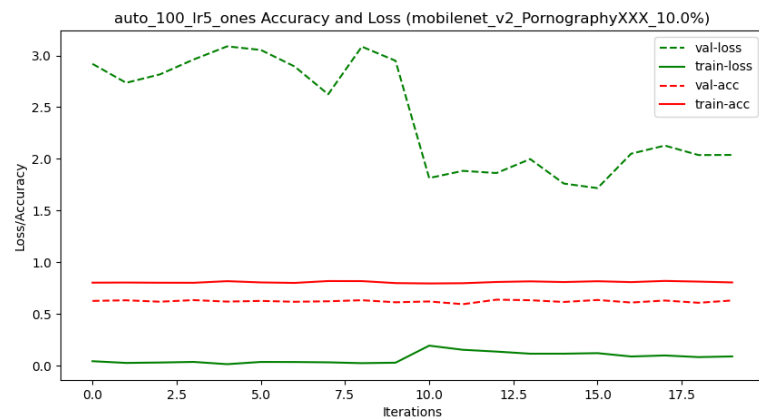
5.3.2 Model and Training

A MobileNet v2 [215] model with two outputs values which are activated with softmax was used. The optimization of the weights is done using the Adam algorithm [229] and the learning rate was set at 1×10^{-5} . A base model trained for 100 epochs on 10% of the training data (1672 images) was created starting from a pre-trained MobileNetv2 on the ImageNet dataset (the default pre-trained option for MobileNetv2 offered in *PyTorch*), using also 10% of the testing data. This is quite a lot more training epochs than the Face Spoofing example; indeed the model achieved a stable training accuracy of around 80% at around the 50 epochs mark, but a decision was made to continue training to try to improve this score. Then, this base model was trained for 20 more epochs with oracle queries set up to start at the 10th. To compare the HITL results, an automatic training pipeline was also ran on the same base model for 20 epochs.

5.3.3 Results



(a) HITL



(b) Auto

Figure 5.2: Pornographic content detection HITL vs Automatic training comparison

Similarly to the Face Spoofing detection there were some difficulties when choosing the relevant areas in which the model should focus. For non-pornographic images, it is a tough decision to select on what the model should focus more, due to the wide range of images present in the dataset and indeed possible in the real-world. A decision was made to avoid picking any rectangle in these cases, implying that the absense of special focus on any area is in itself a reason to classify the image in the non-pornographic class. This does not necessarily translate well in the HITL algorithm, as it the selection of the areas of special focus that push the model into a specific category and, in the absense of any selection, the effect is the same as if no feedback were provided in this cases which is the same as automatic training. For the cases in which the images should be labeled as pornographic, the clear cut indications are areas where nudity (more specifically, genitals and other private areas) is displayed; however, these were not always available to chose. Additionally, images belonging to pornographic videos may not always display these clear indications as they can be frames from moments in the video where they are not present, either because they're off-frame, it's from a moment with a black screen, an introductory screen or other of the kind. Again, these are all possible explanations to the unremarkable effects on the mitigation of the overfitting of the model on the training data; additionally the testing accuracies evolved similarly for both the HITL and auto training pipelines.

Chapter 6

Conclusions

The recent crescent popularity of MLOps provides an excellent opportunity to better incorporate ML with any software project relieving developers and engineers from the cumbersome specific ML associated tasks that can be automated, while at the same time making the more familiar generic pipelines - development, integration, testing, code and data maintenance, deployment and monitoring much more efficient, controllable and useful when it comes to improving the performance of the prediction models. It is therefore evident that there is an excellent opportunity to leverage the advantages of MLOps and apply it to the context of efficient ML. There are a wide range of MLOps systems that also cover a wide range of applications, some in more detail and with more features than others.

6.1 Challenges

During the development of the MLOps tool, it was apparent that some problems arise when designing such a system, specially one with the quite limited scope and size as this one:

- **Active Learning Feedback** - having a one size fits all approach for an active learning system is difficult, not only because different datasets have different types of data, and constitute data points for different kinds of problems. Consequently, the way ML training algorithms interact with humans to request and receive feedback is problem specific. That means that the approach followed in [35] and embedded in the implementend system is not generic enough to accomodate all classification tasks, let alone all ML tasks. Not only the visual interactive way - selection of regions marked by rectangles - but also the criteria for when to ask the user for queries are subject of research and experimentation. Additionally, the *Oracle's* subjectivity plays a part as well, as feedback choices may vary between users for various reasons, from individual expertise to the fact that different people make different choices in situations where the answer is not obvious.
- **Dataset & Feature Management** - handling and storing datasets is a challenge. That is because a lot of the time they're many gigabytes in size disk space a precious commodity; their data is often organized and listed differently, calling for different pre-processing

logic which ; there's not one (or indeed, neither a small set) definite, centralized, agreed upon, public repository where they're originally hosted and can be reliably fetched from. This means that a fully automated, UI-driven dataset and feature management is difficult to achieve and compromises have to be made, such as providing only a very generic way to pre-process already existing datasets stored in disk, by having the users creating their own *Python* scripts with only loose restrictions to their structure, in order to accomodate as wide range of needs as possible.

- **Server Resources** - ML algorithms are often costly in resources for the host system in which they run. This remains and increases to even a bigger challenge when choosing the host server of where to install an MLOps system, as several ML models may be trained or tested simultaneously, multiplying manifold concerns for CPU, GPU availability and capability. In the system described in 4, a maximum of 10 pipelines may be running at a given time which can be a stretch for powerful system even.
- **Different ML Tasks** - The ML tool described in this work has the limiation of only being able to handle classification tasks of the kind described in 5, meaning that it will fit a fairly limited set within the universe of possible ML problems; this is further emphasized by the already described "one size fits all" approach of the active learning logic, specifically with the usage of rectangles that are not appropriate to provide feedback for all classification problems.

6.2 Future Work

Given the limited features that were implemented in the designed system, it is not comprehensive when comparing to the MLOps universe, as it is more geared to human interaction through active learning. Nevertheless, some improvements and additions can be made to make it a more useful and powerful system, such as:

- **Active Learning Feedback Improvements** - the restrictions imposed to the interaction with users, expressed in the shape of the regions to choose and the interface points with the system defined algorithms can be somewhat lifted by using a more encompassing design. Regions for selection don't have to be always rectangles, as they may not accurately represent a reasonable representation for queries to be presented to a user, depending on the dataset chosen, but also on the ML problem at hand. Similarly, the creation and reading of CSV files and saving image files in pre-determined directory, forces the users to write algorithms that have to conform to these simplistic impositions, rather than a well established and designed interface protocol.
- **Dataset & Feature Management** - the limited tools to manage datasets and feature extraction within the system, makes their management cumbersome with a big manual component. The development of a features bank, with tracking and versioning would gear the system in

way to overcome this and avoid the need to move the feature extract logic to the Trainers defined, which leads to longer execution times and resource heavier ML workflows, than if features were already extracted and readily available to be used in future pipelines.

- **Framework support beyond *PyTorch*** - the self made imposition to use *PyTorch* as the ML framework comes from the fact that the previous work this system is based on [35] uses it as well. However, other frameworks such as scikit-learn [133] may possibly be used with the Dataset Handlers, Trainers, Testers and Trained Models steps by lifting this restriction and making the necessary changes to the code and testing accordingly. This would provide more flexibility and useability to the system.

References

- [1] Google, “MLOps: Continuous delivery and automation pipelines in machine learning,” July 2020, [Online; accessed 08-January-2022]. [Online]. Available: <https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>
- [2] B. Settles, “Active learning literature survey,” 2009. [Online]. Available: <https://api.semanticscholar.org/CorpusID:324600>
- [3] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, “Deep reinforcement learning that matters,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. [Online]. Available: <https://ojs.aaai.org/index.php/AAAI/article/view/11694>
- [4] J. Pineau, P. Vincent-Lamarre, K. Sinha, V. Larivière, A. Beygelzimer, F. d’Alché Buc, E. Fox, and H. Larochelle, “Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program),” 2020.
- [5] C. Bishop, *Pattern Recognition and Machine Learning*, ser. Information Science and Statistics. Springer, 2006. [Online]. Available: <https://books.google.pt/books?id=qWPwnQEACAAJ>
- [6] T. Mitchell, *Machine Learning*, ser. McGraw-Hill International Editions. McGraw-Hill, 1997. [Online]. Available: <https://books.google.pt/books?id=EoYBngEACAAJ>
- [7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, no. 6, p. 84–90, may 2017. [Online]. Available: <https://doi.org/10.1145/3065386>
- [9] Y. LeCun, F. J. 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. CVPR 2004.*, vol. 2, 2004, pp. II–104 Vol.2.
- [10] L. 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*, 2004, pp. 178–178.
- [11] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” University of Toronto, Toronto, Ontario, Tech. Rep. 0, 2009.

- [12] D. C. Ciresan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” *CoRR*, vol. abs/1202.2745, 2012. [Online]. Available: <http://arxiv.org/abs/1202.2745>
- [13] B. Russell, A. Torralba, K. Murphy, and W. Freeman, “Labelme: A database and web-based tool for image annotation,” *International Journal of Computer Vision*, vol. 77, 05 2008.
- [14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.
- [15] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, “What is the best multi-stage architecture for object recognition?” in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 2146–2153.
- [16] J. Deng, J. Guo, J. Yang, N. Xue, I. Cotsia, and S. P. Zafeiriou, “ArcFace: Additive angular margin loss for deep face recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021. [Online]. Available: <https://doi.org/10.1109%2Ftpami.2021.3087709>
- [17] Q. Meng, S. Zhao, Z. Huang, and F. Zhou, “Magface: A universal representation for face recognition and quality assessment,” *CoRR*, vol. abs/2103.06627, 2021. [Online]. Available: <https://arxiv.org/abs/2103.06627>
- [18] Y. Huang, Y. Wang, Y. Tai, X. Liu, P. Shen, S. Li, J. Li, and F. Huang, “Curricularface: Adaptive curriculum learning loss for deep face recognition,” 2020. [Online]. Available: <https://arxiv.org/abs/2004.00288>
- [19] Y. LeCun, K. Kavukcuoglu, and C. Farabet, “Convolutional networks and applications in vision,” in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 2010, pp. 253–256.
- [20] Y. LeCun, “Learning processes in an asymmetric threshold network,” in *Disordered systems and biological organization, Les Houches, France*, E. Bienenstock, F. Fogelman-Soulie, and G. Weisbuch, Eds. Springer-Verlag, 1986, pp. 233–240.
- [21] LeCun, “Generalization and network design strategies,” in *Connectionism in Perspective*, R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, Eds. Zurich, Switzerland: Elsevier, 1989, an extended version was published as a technical report of the University of Toronto.
- [22] K. Gregor and Y. LeCun, “Emergence of complex-like cells in a temporal product network with local receptive fields,” 2010. [Online]. Available: <https://arxiv.org/abs/1006.0448>
- [23] J. Ngiam, Z. Chen, D. Chia, P. Koh, Q. Le, and A. Ng, “Tiled convolutional neural networks,” in *Advances in Neural Information Processing Systems*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., vol. 23. Curran Associates, Inc., 2010. [Online]. Available: <https://proceedings.neurips.cc/paper/2010/file/01f78be6f7cad02658508fe4616098a9-Paper.pdf>
- [24] Y.-L. Boureau, J. Ponce, and Y. LeCun, “A theoretical analysis of feature pooling in visual recognition,” in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML’10. Madison, WI, USA: Omnipress, 2010, p. 111–118.

- [25] ISO/IEC JTC 1/SC 37 Biometrics, “ISO/IEC 2382-37:2022 Information technology — Vocabulary — Part 37: Biometrics,” International Organization for Standardization, Tech. Rep., 2022.
- [26] C. Champod and M. Tistarelli, *Biometric Technologies for Forensic Science and Policing: State of the Art*. Cham: Springer International Publishing, 2017, pp. 1–15. [Online]. Available: https://doi.org/10.1007/978-3-319-50673-9_1
- [27] J. Deng, J. Guo, J. Yang, N. Xue, I. Cotsia, and S. P. Zafeiriou, “ArcFace: Additive angular margin loss for deep face recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021. [Online]. Available: <https://doi.org/10.1109%2Ftpami.2021.3087709>
- [28] N. Reddy, A. Rattani, and R. Derakhshani, “Comparison of deep learning models for biometric-based mobile user authentication,” in *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, 2018, pp. 1–6.
- [29] A. Jain, A. Ross, and S. Prabhakar, “An introduction to biometric recognition,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 1, pp. 4–20, 2004.
- [30] A. B. Arrieta, N. Díaz-Rodríguez, J. D. Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, “Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai,” 2019.
- [31] P. C. Neto, A. F. Sequeira, and J. S. Cardoso, “Myope models – are face presentation attack detection models short-sighted?” 2021.
- [32] W. Silva, K. Fernandes, M. Cardoso, and J. Cardoso, *Towards Complementary Explanations Using Deep Neural Networks: First International Workshops, MLCN 2018, DLF 2018, and iMIMIC 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 16-20, 2018, Proceedings*, 09 2018, pp. 133–140.
- [33] E. Tjoa and C. Guan, “A survey on explainable artificial intelligence (XAI): Toward medical XAI,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 11, pp. 4793–4813, nov 2021. [Online]. Available: <https://doi.org/10.1109%2Ftnnls.2020.3027314>
- [34] S. Raghunath, A. E. Ulloa Cerna, L. Jing, D. P. vanMaanen, J. Stough, D. N. Hartzel, J. B. Leader, H. L. Kirchner, M. C. Stumpe, A. Hafez, A. Nemani, T. Carbonati, K. W. Johnson, K. Young, C. W. Good, J. M. Pfeifer, A. A. Patel, B. P. Delisle, A. Alsaïd, D. Beer, C. M. Haggerty, and B. K. Fornwalt, “Prediction of mortality from 12-lead electrocardiogram voltage data using a deep neural network,” *Nature Medicine*, vol. 26, no. 6, pp. 886–891, Jun 2020. [Online]. Available: <https://doi.org/10.1038/s41591-020-0870-z>
- [35] P. Serrano e Silva, R. Cruz, A. S. M. Shihavuddin, and T. Gonçalves, “Interpretability-guided human feedback during neural network training,” in *Pattern Recognition and Image Analysis*, A. Pertusa, A. J. Gallego, J. A. Sánchez, and I. Domingues, Eds. Cham: Springer Nature Switzerland, 2023, pp. 276–287.
- [36] Cornell University, “Active learning - center for teaching innovation,” [Online; accessed 26-July-2023]. [Online]. Available: <https://teaching.cornell.edu/teaching-resources/active-collaborative-learning/active-learning>

- [37] J. Han, E. Shihab, Z. Wan, S. Deng, and X. Xia, “What do programmers discuss about deep learning frameworks,” *Empirical Software Engineering*, vol. 25, 07 2020.
- [38] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. [Online]. Available: <https://doi.org/10.1038/nature14539>
- [39] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, jan 2015. [Online]. Available: <https://doi.org/10.1016%2Fj.neunet.2014.09.003>
- [40] Y. Tian, K. Pei, S. Jana, and B. Ray, “Deeptest: Automated testing of deep-neural-network-driven autonomous cars,” 2018.
- [41] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. van Ginneken, and C. I. Sánchez, “A survey on deep learning in medical image analysis,” *Medical Image Analysis*, vol. 42, pp. 60–88, dec 2017. [Online]. Available: <https://doi.org/10.1016%2Fj.media.2017.07.005>
- [42] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, “Natural language processing (almost) from scratch,” 2011.
- [43] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, “The Stanford CoreNLP natural language processing toolkit,” in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Baltimore, Maryland: Association for Computational Linguistics, Jun. 2014, pp. 55–60. [Online]. Available: <https://aclanthology.org/P14-5010>
- [44] T. Deselaers, S. Hasan, O. Bender, and H. Ney, “A deep learning approach to machine transliteration,” in *Proceedings of the Fourth Workshop on Statistical Machine Translation*. Athens, Greece: Association for Computational Linguistics, Mar. 2009, pp. 233–241. [Online]. Available: <https://aclanthology.org/W09-0438>
- [45] A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A. N. Gomez, S. Gouws, L. Jones, Łukasz Kaiser, N. Kalchbrenner, N. Parmar, R. Sepassi, N. Shazeer, and J. Uszkoreit, “Tensor2tensor for neural machine translation,” 2018.
- [46] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.
- [47] R. Szeliski, *Computer Vision: Algorithms and Applications*, ser. Texts in Computer Science. Springer London, 2010. [Online]. Available: <https://books.google.pt/books?id=bXzAlkODwa8C>
- [48] H. Ha and H. Zhang, “Deepperf: Performance prediction for configurable software with deep sparse neural network,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 1095–1106.
- [49] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, “A novel neural source code representation based on abstract syntax tree,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 783–794.
- [50] D. Zhao, Z. Xing, C. Chen, X. Xia, and G. Li, “Actionnet: Vision-based workflow action recognition from programming screencasts,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 350–361.

- [51] J. Y. Choi and C.-H. Choi, “Sensitivity analysis of multilayer perceptron with differentiable activation functions,” *IEEE Transactions on Neural Networks*, vol. 3, no. 1, pp. 101–107, 1992.
- [52] T. Fine, *Feedforward Neural Network Methodology*, ser. Information Science and Statistics. Springer New York, 2006. [Online]. Available: <https://books.google.pt/books?id=s-PIBwAAQBAJ>
- [53] A. Goh, “Back-propagation neural networks for modeling complex systems,” *Artificial Intelligence in Engineering*, vol. 9, no. 3, pp. 143–151, 1995. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/095418109400011S>
- [54] C. Liu, C. Gao, X. Xia, D. Lo, J. Grundy, and X. Yang, “On the reproducibility and replicability of deep learning in software engineering,” *ACM Transactions on Software Engineering and Methodology*, vol. 31, no. 1, pp. 1–46, oct 2021. [Online]. Available: <https://doi.org/10.1145%2F3477535>
- [55] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, “A convolutional neural network for modelling sentences,” 2014.
- [56] S. Lawrence, C. Giles, A. C. Tsoi, and A. Back, “Face recognition: a convolutional neural-network approach,” *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98–113, 1997.
- [57] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, “Recurrent neural network based language model,” vol. 2, 01 2010, pp. 1045–1048.
- [58] T. Mikolov, S. Kombrink, L. Burget, J. Černocký, and S. Khudanpur, “Extensions of recurrent neural network language model,” in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011, pp. 5528–5531.
- [59] S. Bhatia, P. Kohli, and R. Singh, “Neuro-symbolic program corrector for introductory programming assignments,” in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 60–70.
- [60] J. Guo, J. Cheng, and J. Cleland-Huang, “Semantically enhanced software traceability using deep learning techniques,” in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017, pp. 3–14.
- [61] J. E. Boylan, P. Goodwin, M. Mohammadipour, and A. A. Syntetos, “Reproducibility in forecasting research,” *International Journal of Forecasting*, vol. 31, no. 1, pp. 79–90, 2015. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0169207014001009>
- [62] A. Branco, K. B. Cohen, P. Vossen, N. Ide, and N. Calzolari, “Replicability and reproducibility of research results for human language technology: introducing an Ire special section,” *Language Resources and Evaluation*, vol. 51, no. 1, pp. 1–5, Mar 2017. [Online]. Available: <https://doi.org/10.1007/s10579-017-9380-0>
- [63] N. Juristo and O. S. Gomez, *Replication of Software Engineering Experiments*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 60–88. [Online]. Available: https://doi.org/10.1007/978-3-642-25231-0_2

- [64] Z. Mahmood, D. Bowes, T. Hall, P. C. Lane, and J. Petrić, “Reproducibility and replicability of software defect prediction studies,” *Information and Software Technology*, vol. 99, pp. 148–163, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950584917304202>
- [65] A. A. Neto, “A strategy to support replications of controlled experiments in software engineering,” *SIGSOFT Softw. Eng. Notes*, vol. 44, no. 3, p. 23, oct 2020. [Online]. Available: <https://doi.org/10.1145/3356773.3356796>
- [66] M. Baker, “1,500 scientists lift the lid on reproducibility,” *Nature*, vol. 533, no. 7604, pp. 452–454, May 2016. [Online]. Available: <https://doi.org/10.1038/533452a>
- [67] Y. Ge, M. Chen, C. Liu, F. Chen, S. Huang, and H. Wang, *Deep Metric Learning for Software Change-Proneness Prediction: 8th International Conference, IScIDE 2018, Lanzhou, China, August 18–19, 2018, Revised Selected Papers*, 08 2018, pp. 287–300.
- [68] C. Liu, D. Yang, X. Zhang, B. Ray, and M. M. Rahman, “Recommending github projects for developer onboarding,” *IEEE Access*, vol. 6, pp. 52 082–52 094, 2018.
- [69] Y. Wan, J. Shu, Y. Sui, G. Xu, Z. Zhao, J. Wu, and P. Yu, “Multi-modal attention network learning for semantic source code retrieval,” in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 13–25.
- [70] Y. Wan, Z. Zhao, M. Yang, G. Xu, H. Ying, J. Wu, and P. S. Yu, “Improving automatic source code summarization via deep reinforcement learning,” in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 397–407. [Online]. Available: <https://doi.org/10.1145/3238147.3238206>
- [71] P. Zhou, J. Liu, X. Liu, Z. Yang, and J. Grundy, “Is deep learning better than traditional approaches in tag recommendation for software information sites?” *Inf. Softw. Technol.*, vol. 109, no. C, p. 1–13, may 2019. [Online]. Available: <https://doi.org/10.1016/j.infsof.2019.01.002>
- [72] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, “Software engineering challenges of deep learning,” in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, aug 2018. [Online]. Available: <https://doi.org/10.1109%2Fseaa.2018.00018>
- [73] C. Liu, X. Zhang, H. Hu, J. Barson, and B. Ray, “A recommender system for developer onboarding,” 05 2018, pp. 319–320.
- [74] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, “Are gans created equal? a large-scale study,” 2018.
- [75] G. Melis, C. Dyer, and P. Blunsom, “On the state of the art of evaluation in neural language models,” 2017.
- [76] X. Bouthillier, C. Laurent, and P. Vincent, “Unreproducible research is reproducible,” in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 725–734. [Online]. Available: <https://proceedings.mlr.press/v97/bouthillier19a.html>

- [77] L. Li and A. Talwalkar, “Random search and reproducibility for neural architecture search,” 2019.
- [78] B. C. Anda, D. I. Sjøberg, and A. Mockus, “Variability and reproducibility in software engineering: A study of four companies that developed the same system,” *IEEE Transactions on Software Engineering*, vol. 35, no. 3, pp. 407–429, 2009.
- [79] E. Raff, “A step toward quantifying independently reproducible machine learning research,” 2019.
- [80] R. A. Klein, C. L. Cook, C. R. Ebersole, C. Vitiello, B. A. Nosek, J. Hilgard, P. H. Ahn, A. J. Brady, C. R. Chartier, C. D. Christopherson, S. Clay, B. Collisson, J. T. Crawford, R. Cromar, G. Gardiner, C. L. Gosnell, J. Grahe, C. Hall, I. Howard, J. A. Joy-Gaba, M. Kolb, A. M. Legg, C. A. Levitan, A. D. Mancini, D. Manfredi, J. Miller, G. Nave, L. Redford, I. Schlitz, K. Schmidt, J. L. M. Skorinko, D. Storage, T. Swanson, L. M. Van Swol, L. A. Vaughn, D. Vidamuerte, B. Wiggins, and K. A. Ratliff, “Many Labs 4: Failure to Replicate Mortality Salience Effect With and Without Original Author Involvement,” *Collabra: Psychology*, vol. 8, no. 1, p. 35271, 04 2022. [Online]. Available: <https://doi.org/10.1525/collabra.35271>
- [81] X. Gu, H. Zhang, and S. Kim, “Deep code search,” in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 933–944. [Online]. Available: <https://doi.org/10.1145/3180155.3180167>
- [82] A. LeClair, S. Jiang, and C. McMillan, “A neural model for generating natural language summaries of program subroutines,” in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE ’19. IEEE Press, 2019, p. 795–806. [Online]. Available: <https://doi.org/10.1109/ICSE.2019.00087>
- [83] M. Tufano, J. Pantiuchina, C. Watson, G. Bavota, and D. Poshyvanyk, “On learning meaningful code changes via neural machine translation,” 2019.
- [84] Amazon, “What is DevOps?” July 2020, [Online; accessed 08-January-2022]. [Online]. Available: https://aws.amazon.com/devops/what-is-devops/?nc1=h_ls
- [85] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, and M. Young, “Machine learning: The high interest credit card of technical debt,” in *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*, 2014.
- [86] Rob van der Meulen & Thomas McCall Gartner, “Gartner says nearly half of cios are planning to deploy artificial intelligence,” [Online; accessed 26-July-2023]. [Online]. Available: <https://www.gartner.com/en/newsroom/press-releases/2018-02-13-gartner-says-nearly-half-of-cios-are-planning-to-deploy-artificial-intelligence>
- [87] A. Posoldova, “Machine learning pipelines: From research to production,” *IEEE Potentials*, vol. 39, no. 6, pp. 38–42, 2020.
- [88] D. Kreuzberger, N. Kühl, and S. Hirschl, “Machine learning operations (mlops): Overview, definition, and architecture,” *IEEE Access*, vol. 11, pp. 31 866–31 879, 2023.
- [89] N. Antonio, A. de Almeida, and L. Nunes, “Predicting hotel bookings cancellation with a machine learning classification model,” in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2017, pp. 1049–1054.

- [90] L. Baier, T. Schlör, J. Schöffner, and N. Kühl, “Detecting concept drift with neural network model uncertainty,” 2022.
- [91] G. Symeonidis, E. Nerantzis, A. Kazakis, and G. A. Papakostas, “Mlops – definitions, tools and challenges,” 2022.
- [92] M. Schlegel and K.-U. Sattler, “Management of machine learning lifecycle artifacts: A survey,” *SIGMOD Rec.*, vol. 51, no. 4, p. 18–35, jan 2023. [Online]. Available: <https://doi.org/10.1145/3582302.3582306>
- [93] Microsoft Corporation, “Azure Machine Learning - ML as a Service | Microsoft Azure,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://azure.microsoft.com/en-us/products/machine-learning>
- [94] Amazon.com, Inc., “Amazon SageMaker - Machine Learning - AWS,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://aws.amazon.com/sagemaker/>
- [95] Google Inc., “Vertex AI | Google Cloud,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://cloud.google.com/vertex-ai>
- [96] IBM, “IBM Watson Studio,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://www.ibm.com/products/watson-studio>
- [97] Comet ML, Inc., “Comet ML - Build better models faster,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://www.comet.com/>
- [98] DataRobot, Inc., “DataRobot AI Platform,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://www.datarobot.com/platform/>
- [99] Cloudera, Inc., “Data Science Workbench (CDSW),” [Online; Accessed 28-08-2023]. [Online]. Available: <https://www.cloudera.com/products/data-science-and-engineering/data-science-workbench.html>
- [100] Valohai, “Valohai | Take ML places it’s never been,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://valohai.com/>
- [101] M. A. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar, “Accelerating the machine learning lifecycle with mlflow,” *IEEE Data Eng. Bull.*, vol. 41, pp. 39–45, 2018.
- [102] A. Chen, A. Chow, A. Davidson, A. DCunha, A. Ghodsi, S. A. Hong, A. Konwinski, C. Mewald, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, A. Singh, F. Xie, M. Zaharia, R. Zang, J. Zheng, and C. Zumar, “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*, ser. DEEM’20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: <https://doi.org/10.1145/3399579.3399867>
- [103] Allegro AI, “ClearML | The Continuous Machine Learning Company,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://clear.ml/>
- [104] Polyaxon, “Open source Machine Learning at scale with Kubernetes - MLOps Lifecycle for data-scientists & machine-learning engineers - Model & Data Centric AI - Polyaxon,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://polyaxon.com/>

- [105] Hopsworks, “Hopsworks - Batch and Real-time ML Platform,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://www.hopsworks.ai/>
- [106] Iterative AI, “Data Version Control · DVC,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://dvc.org/>
- [107] —, “CML · Continuous Machine Learning,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://cml.dev/>
- [108] —, “DVCLive | Data Version Control · DVC,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://dvc.org/doc/dvclive>
- [109] —, “Iterative Studio: Collaboration for Machine Learning Teams,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://studio.iterative.ai/>
- [110] —, “Simplifying Machine Learning Model Deployment,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://mlem.ai/>
- [111] L. A. Melgar, D. Dao, S. Gan, N. M. Gürel, N. Hollenstein, J. Jiang, B. Karlas, T. Lemmin, T. Li, Y. Li, X. Rao, J. Rausch, C. Renggli, L. Rimanic, M. Weber, S. Zhang, Z. Zhao, K. Schawinski, W. Wu, and C. Zhang, “Ease.ml: A lifecycle management system for machine learning,” in *Conference on Innovative Data Systems Research*, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:232328184>
- [112] E. Brumbaugh, M. Bhushan, A. Cheong, M. G.-Q. Du, J. Feng, N. Handel, A. Hoh, J. Hone, B. Hunter, A. Kale, A. Luque, B. Nooraei, J. Park, K. Puttaswamy, K. Schiller, E. Shapiro, C. Shi, A. Siegel, N. Simha, M. Sbrocca, S.-J. Yao, P. Yoon, V. Zanoian, X.-H. T. Zeng, and Q. Zhu, “Bighead: A framework-agnostic, end-to-end machine learning platform,” in *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2019, pp. 551–560.
- [113] A. Sergeev and M. D. Balso, “Horovod: fast and easy distributed deep learning in tensor-flow,” 2018.
- [114] C. Sun, N. Azari, and C. Turakhia, “Gallery: A machine learning model management system at uber,” in *International Conference on Extending Database Technology*, 2020, pp. 474–485.
- [115] LinkedIn Corporation, “Scaling Machine Learning Productivity at LinkedIn | LinkedIn Engineering,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://engineering.linkedin.com/blog/2019/01/scaling-machine-learning-productivity-at-linkedin>
- [116] D. Crankshaw, P. Bailis, J. E. Gonzalez, H. Li, Z. Zhang, M. J. Franklin, A. Ghodsi, and M. I. Jordan, “The missing piece in complex analytics: Low latency, scalable model management and serving with velox,” 2014.
- [117] M. H. Namaki, A. Floratou, F. Psallidas, S. Krishnan, A. Agrawal, Y. Wu, Y. Zhu, and M. Weimer, “Vamsa: Automated provenance tracking in data science scripts,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, aug 2020. [Online]. Available: <https://doi.org/10.1145%2F3394486.3403205>

- [118] ArangoDB, Inc, “arangoml/arangopipe: ArangoML Pipeline is a common and extensible Metadata Layer for Machine Learning Pipelines based on ArangoDB.” [Online; Accessed 28-08-2023]. [Online]. Available: <https://github.com/arangoml/arangopipe>
- [119] M. Boehm, I. Antonov, S. Baunsgaard, M. Dokter, R. Ginthoer, K. Innerebner, F. Klezin, S. Lindstaedt, A. Phani, B. Rath, B. Reinwald, S. Siddiqi, and S. B. Wrede, “Systemds: A declarative machine learning system for the end-to-end data science lifecycle,” 2020.
- [120] Shreya Shankar, “loglabs/mltrace: Coarse-grained lineage and tracing for machine learning pipelines.” [Online; Accessed 28-08-2023]. [Online]. Available: <https://github.com/loglabs/mltrace>
- [121] H. Miao, A. Chavan, and A. Deshpande, “Provdb: A system for lifecycle management of collaborative analysis workflows,” 2016.
- [122] H. Miao, “Provenance management for collaborative data science workflows,” Ph.D. dissertation, University of Maryland, College Park, 2018.
- [123] A. N. Modi, C. Y. Koo, C. Y. Foo, C. Mewald, D. M. Baylor, E. Breck, H.-T. Cheng, J. Wilkiewicz, L. Koc, L. Lew, M. A. Zinkevich, M. Wicke, M. Ispir, N. Polyzotis, N. Fiedel, S. E. Haykal, S. Whang, S. Roy, S. Ramesh, V. Jain, X. Zhang, and Z. Haque, “Tfx: A tensorflow-based production-scale machine learning platform,” in *KDD 2017*, 2017.
- [124] Google Inc., “ML Metadata | TFX | TensorFlow,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://www.tensorflow.org/tfx/guide/mlmd>
- [125] The Kubeflow Authors, “Kubeflow,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://www.kubeflow.org/>
- [126] H2O.ai, “H2O Open Source | H2O.ai,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://h2o.ai/platform/ai-cloud/make/h2o/>
- [127] Neptune Labs, “neptune.ai | The MLOps stack component for experiment tracking,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://neptune.ai/>
- [128] Meta, “Introducing FBLeaRner Flow: Facebook’s AI backbone - Engineering at Meta,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://engineering.fb.com/2016/05/09/core-data/introducing-fblearner-flow-facebook-s-ai-backbone/>
- [129] Z. Luo, S. H. Yeung, M. Zhang, K. Zheng, L. Zhu, G. Chen, F. Fan, Q. Lin, K. Y. Ngiam, and B. C. Ooi, “Mlcask: Efficient management of component evolution in collaborative data analytics pipelines,” 2021.
- [130] K. Yocum, S. Rowan, J. Lunt, and T. M. Wong, “Disdat: Bundle data management for machine learning pipelines,” in *2019 USENIX Conference on Operational Machine Learning (OpML 19)*. Santa Clara, CA: USENIX Association, May 2019, pp. 35–37. [Online]. Available: <https://www.usenix.org/conference/opml19/presentation/yocum>
- [131] R. Ferenc, T. Viskok, T. Aladics, J. Jász, and P. Hegedűs, “Deep-water framework: The swiss army knife of humans working with machine learning models,” *SoftwareX*, vol. 12, p. 100551, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2352711019303772>

- [132] studio.ml, “studioml/studio: Studio: Simplify and expedite model building process,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://www.tensorflow.org/>
- [133] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Édouard Duchesnay, “Scikit-learn: Machine learning in python,” *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: <http://jmlr.org/papers/v12/pedregosa11a.html>
- [134] Google Inc., “TensorFlow,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://github.com/studioml/studio>
- [135] MLCommons, “MLCube | MLCommons,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://mlcommons.org/en/mlcube/>
- [136] X. Yao, “Mlpm: Machine learning package manager,” 02 2020.
- [137] TensorHub, Inc, “Guild AI - Experiment tracking, ML developer tools ,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://guild.ai/>
- [138] Acusense Technologies, Inc, “Datmo - Productivity for Data Science & AI,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://www.datmo.com/>
- [139] Replicate, “Keepsake – Version control for machine learning,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://keepsake.ai/>
- [140] J. Tsay, T. Mummert, N. Bobroff, A. Braz, P. Westerink, and M. Hirzel, “Runway: machine learning model experiment management tool,” 02 2018.
- [141] K. Greff, A. Klein, M. Chovanec, F. Hutter, and J. Schmidhuber, “The sacred infrastructure for computational research,” in *SciPy*, 2017. [Online]. Available: <https://api.semanticscholar.org/CorpusID:42114474>
- [142] Weights & Biases, “Weights & Biases – Developer tools for ML,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://keepsake.ai/>
- [143] K.-H. Chen, H.-P. Su, W.-C. Chuang, H.-C. Hsiao, W. Tan, Z. Tang, X. Liu, Y. Liang, W.-C. Lo, W. Ji, B. Hsu, K. Hu, H. Jian, Q. Zhou, and C.-M. Wang, “Apache submarine: A unified machine learning platform made simple,” 2021.
- [144] Determined AI, “Distributed Deep Learning and Hyperparameter Tuning Platform | Determined AI,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://www.determined.ai/>
- [145] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia, “Modeldb: A system for machine learning model management,” in *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, ser. HILDA '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: <https://doi.org/10.1145/2939502.2939516>
- [146] H. Miao, A. Li, L. S. Davis, and A. Deshpande, “Modelhub: Towards unified data and lifecycle management for deep learning,” 2016.

- [147] G. Gharibi, V. Walunj, S. Rella, and Y. Lee, “Modelkb: Towards automated management of the modeling lifecycle in deep learning,” in *2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, 2019, pp. 28–34.
- [148] C. Weber and P. Reimann, “Mmp - a platform to manage machine learning models in industry 4.0 environments,” in *2020 IEEE 24th International Enterprise Distributed Object Computing Workshop (EDOCW)*, 2020, pp. 91–94.
- [149] M. Vartak, J. M. F. da Trindade, S. Madden, and M. Zaharia, “Mistique: A system to store and query model intermediates for model diagnosis,” in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1285–1300. [Online]. Available: <https://doi.org/10.1145/3183713.3196934>
- [150] N. Makrynioti, R. Ley-Wild, and V. Vassalos, “sql4ml a declarative end-to-end workflow for machine learning,” 2019.
- [151] H. Zhang, Y. Li, Y. Huang, Y. Wen, J. Yin, and K. Guan, “Mlmodelci: An automatic cloud platform for efficient mlaas,” in *Proceedings of the 28th ACM International Conference on Multimedia*, ser. MM ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 4453–4456. [Online]. Available: <https://doi.org/10.1145/3394171.3414535>
- [152] Y. Huang, H. Zhang, Y. Wen, P. Sun, and N. B. D. TA, “Modelci-e: Enabling continual learning in deep learning serving systems,” 2021.
- [153] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, “Clipper: A low-latency online prediction serving system,” in *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI’17. USA: USENIX Association, 2017, p. 613–627.
- [154] W. Wang, S. Wang, J. Gao, M. Zhang, G. Chen, T. K. Ng, and B. C. Ooi, “Rafiki: Machine learning as an analytics service system,” 2018.
- [155] C. Ré, F. Niu, P. Gudipati, and C. Srisuwananukorn, “Overton: A data system for monitoring and improving machine-learned products,” 2019.
- [156] K. Li and N. Gui, “Cms: A continuous machine-learning and serving platform for industrial big data,” *Future Internet*, vol. 12, no. 6, 2020. [Online]. Available: <https://www.mdpi.com/1999-5903/12/6/102>
- [157] A. Hosny, M. Schwier, C. Berger, E. P. Örneek, M. Turan, P. V. Tran, L. Weninger, F. Isensee, K. H. Maier-Hein, R. McKinley, M. T. Lu, U. Hoffmann, B. Menze, S. Bakas, A. Fedorov, and H. J. Aerts, “Modelhub.ai: Dissemination platform for deep learning models,” 2019.
- [158] P. Agrawal, R. Arya, A. Bindal, S. Bhatia, A. Gagneja, J. Godlewski, Y. Low, T. Muss, M. M. Paliwal, S. Raman, V. Shah, B. Shen, L. Sugden, K. Zhao, and M.-C. Wu, “Data platform for machine learning,” in *Proceedings of the 2019 International Conference on Management of Data*, ser. SIGMOD ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1803–1816. [Online]. Available: <https://doi.org/10.1145/3299869.3314050>

- [159] S. Baunsgaard, M. Boehm, A. Chaudhary, B. Derakhshan, S. Geißelsöder, P. M. Grulich, M. Hildebrand, K. Innerebner, V. Markl, C. Neubauer, S. Osterburg, O. Ovcharenko, S. Redyuk, T. Rieger, A. Rezaei Mahdiraji, S. B. Wrede, and S. Zeuch, “Exdra: Exploratory data science on federated raw data,” in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2450–2463. [Online]. Available: <https://doi.org/10.1145/3448016.3457549>
- [160] Pachyderm, “Home Page | Pachyderm,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://www.pachyderm.com/>
- [161] IBM, “ProvLake - IBM,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://ibm.biz/provlake>
- [162] A. Chapman, P. Missier, G. Simonelli, and R. Torlone, “Capturing and querying fine-grained provenance of preprocessing pipelines in data science,” *Proc. VLDB Endow.*, vol. 14, no. 4, p. 507–520, dec 2020. [Online]. Available: <https://doi.org/10.14778/3436905.3436911>
- [163] E. Toropov, P. A. Buitrago, and J. M. F. Moura, “Shuffler,” in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning)*. ACM, jul 2019. [Online]. Available: <https://doi.org/10.1145%2F3332186.3333046>
- [164] Feast Authors, “Feast: Feature Store for Machine Learning,” [Online; Accessed 28-08-2023]. [Online]. Available: <https://feast.dev/>
- [165] M. Testi, M. Ballabio, E. Frontoni, G. Iannello, S. Moccia, P. Soda, and G. Vessio, “Mlops: A taxonomy and a methodology,” *IEEE Access*, vol. 10, pp. 63 606–63 618, 2022.
- [166] T. Granlund, V. Stirbu, and T. Mikkonen, “Towards regulatory-compliant mlops: Oravizio’s journey from a machine learning experiment to a deployed certified medical product,” *SN Computer Science*, vol. 2, no. 5, p. 342, Jun 2021. [Online]. Available: <https://doi.org/10.1007/s42979-021-00726-1>
- [167] T. Chen and C. Guestrin, “XGBoost,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2016. [Online]. Available: <https://doi.org/10.1145%2F2939672.2939785>
- [168] C. E. Shannon, “A mathematical theory of communication,” *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1948.tb01338.x>
- [169] R. Monarch, R. Munro, and C. Manning, *Human-in-the-Loop Machine Learning: Active Learning and Annotation for Human-centered AI*. Manning, 2021. [Online]. Available: <https://books.google.pt/books?id=LCh0zQEACAAJ>
- [170] S. Amershi, M. Cakmak, W. B. Knox, and T. Kulesza, “Power to the people: The role of humans in interactive machine learning,” *AI Magazine*, vol. 35, no. 4, pp. 105–120, 2014. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1609/aimag.v35i4.2513>
- [171] Y.-P. Tang, G.-X. Li, and S.-J. Huang, “Alipy: Active learning in python,” 2019.
- [172] Y.-Y. Yang, S.-C. Lee, Y.-A. Chung, T.-E. Wu, S.-A. Chen, and H.-T. Lin, “libact: Pool-based active learning in python,” 2017.

- [173] T. Danka and P. Horvath, “modal: A modular active learning framework for python,” 2018.
- [174] K. G. Jamieson, L. Jain, C. Fernandez, N. J. Glattard, and R. Nowak, “Next: A system for real-world development, evaluation, and application of active learning,” in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/89ae0fe22c47d374bc9350ef99e01685-Paper.pdf
- [175] J. J. Dudley and P. O. Kristensson, “A review of user interface design for interactive machine learning,” *ACM Trans. Interact. Intell. Syst.*, vol. 8, no. 2, jun 2018. [Online]. Available: <https://doi.org/10.1145/3185517>
- [176] J. Suh, S. Ghorashi, G. Ramos, N.-C. Chen, S. Drucker, J. Verwey, and P. Simard, “Anchorviz: Facilitating semantic data exploration and concept discovery for interactive machine learning,” *ACM Trans. Interact. Intell. Syst.*, vol. 10, no. 1, aug 2019. [Online]. Available: <https://doi.org/10.1145/3241379>
- [177] E. Mosqueira-Rey, E. H. Pereira, D. Alonso-Ríos, and J. Bobes-Bascarán, “A classification and review of tools for developing and interacting with machine learning systems,” in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, ser. SAC ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1092–1101. [Online]. Available: <https://doi.org/10.1145/3477314.3507310>
- [178] J. A. Fails and D. R. Olsen, “Interactive machine learning,” in *Proceedings of the 8th International Conference on Intelligent User Interfaces*, ser. IUI ’03. New York, NY, USA: Association for Computing Machinery, 2003, p. 39–45. [Online]. Available: <https://doi.org/10.1145/604045.604056>
- [179] J. Fogarty, D. Tan, A. Kapoor, and S. Winder, “Cueflik: Interactive concept learning in image search,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’08. New York, NY, USA: Association for Computing Machinery, 2008, p. 29–38. [Online]. Available: <https://doi.org/10.1145/1357054.1357061>
- [180] S. Berg, D. Kutra, T. Kroeger, C. N. Straehle, B. X. Kausler, C. Haubold, M. Schiegg, J. Ales, T. Beier, M. Rudy, K. Eren, J. I. Cervantes, B. Xu, F. Beuttenmueller, A. Wolny, C. Zhang, U. Koethe, F. A. Hamprecht, and A. Kreshuk, “ilastik: interactive machine learning for (bio)image analysis,” *Nature Methods*, vol. 16, no. 12, pp. 1226–1232, Dec 2019. [Online]. Available: <https://doi.org/10.1038/s41592-019-0582-9>
- [181] B. Kellenberger, D. Tuia, and D. Morris, “Aide: Accelerating image-based ecological surveys with interactive machine learning,” *Methods in Ecology and Evolution*, vol. 11, no. 12, pp. 1716–1727, 2020. [Online]. Available: <https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.13489>
- [182] M. Kabra, A. A. Robie, M. Rivera-Alba, S. Branson, and K. Branson, “Jaaba: interactive machine learning for automatic annotation of animal behavior,” *Nature Methods*, vol. 10, no. 1, pp. 64–67, Jan 2013. [Online]. Available: <https://doi.org/10.1038/nmeth.2281>
- [183] C. Hilton, N. Plant, C. González Díaz, P. Perry, R. Gibson, B. Martelli, M. Zbyszynski, R. Fiebrink, and M. Gillies, “Interactml: Making machine learning accessible for creative practitioners working with movement interaction in immersive media,” in *Proceedings of*

- the 27th ACM Symposium on Virtual Reality Software and Technology*, ser. VRST '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: <https://doi.org/10.1145/3489849.3489879>
- [184] R. Fiebrink and P. Cook, “The wekinator: A system for real-time, interactive machine learning in music,” *Proceedings of The Eleventh International Society for Music Information Retrieval Conference (ISMIR 2010)*, 01 2010.
- [185] K. Hipke, M. Toomim, R. Fiebrink, and J. Fogarty, “Beatbox: End-user interactive definition and training of recognizers for percussive vocalizations,” in *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces*, ser. AVI '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 121–124. [Online]. Available: <https://doi.org/10.1145/2598153.2598189>
- [186] S. Teso and K. Kersting, “Explanatory interactive machine learning,” in *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, ser. AIES '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 239–245. [Online]. Available: <https://doi.org/10.1145/3306618.3314293>
- [187] Redis Ltd., “Redis,” [Online; accessed 26-July-2023]. [Online]. Available: <https://redis.io/>
- [188] The PostgreSQL Global Development Group, “PostgreSQL: The world’s most advanced open source database,” [Online; accessed 26-July-2023]. [Online]. Available: <https://www.postgresql.org/>
- [189] Evan You, “Vue.js - The Progressive JavaScript Framework | Vue.js,” [Online; accessed 26-July-2023]. [Online]. Available: <https://vuejs.org/>
- [190] JustBoil, “Admin One — Free Vue 3.x Tailwind 3.x Admin Dashboard with dark mode,” [Online; accessed 26-July-2023]. [Online]. Available: <https://github.com/justboil/admin-one-vue-tailwind>
- [191] Tailwind Labs, “Tailwind CSS is a utility-first CSS framework for rapidly building modern websites without ever leaving your HTML.” [Online; accessed 26-July-2023]. [Online]. Available: <https://tailwindcss.com/>
- [192] Evan You & Vite Contributors, “Vite | Next Generation Frontend Tooling,” [Online; accessed 26-July-2023]. [Online]. Available: <https://vitejs.dev/>
- [193] Jinzhu, “GORM - The fantastic ORM library for Golang, aims to be developer friendly.” [Online; accessed 26-July-2023]. [Online]. Available: <https://gorm.io/>
- [194] Google Inc., “The Go Programming Language,” [Online; accessed 26-July-2023]. [Online]. Available: <https://go.dev/>
- [195] Gin Team, “Gin Web Framework,” [Online; accessed 26-July-2023]. [Online]. Available: <https://gin-gonic.com/>
- [196] M. Richards, *Software Architecture Patterns*. O’Reilly Media, Inc., 2015.
- [197] M. B. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” RFC 7519, May 2015. [Online]. Available: <https://www.rfc-editor.org/info/rfc7519>

- [198] M. B. Jones, “JSON Web Algorithms (JWA),” RFC 7518, May 2015. [Online]. Available: <https://www.rfc-editor.org/info/rfc7518>
- [199] Google Inc., “Goroutines,” [Online; accessed 26-July-2023]. [Online]. Available: <https://go.dev/tour/concurrency/1>
- [200] JustBoil, “Simple, reliable & efficient distributed task queue in Go,” [Online; accessed 26-July-2023]. [Online]. Available: <https://github.com/hibiken/asyncq>
- [201] Burak Cakmakoglu, “Vue Flow | Vue Flow,” [Online; accessed 26-July-2023]. [Online]. Available: <https://vueflow.dev/>
- [202] Linus Torvalds, “Git,” [Online; accessed 26-July-2023]. [Online]. Available: <https://git-scm.com/>
- [203] N. C. F. Codella, D. Gutman, M. E. Celebi, B. Helba, M. A. Marchetti, S. W. Dusza, A. Kalloo, K. Liopyris, N. Mishra, H. Kittler, and A. Halpern, “Skin lesion analysis toward melanoma detection: A challenge at the 2017 international symposium on biomedical imaging (isbi), hosted by the international skin imaging collaboration (isic),” 2018.
- [204] S. D. Karthik, Maggie, “Aptos 2019 blindness detection,” 2019. [Online]. Available: <https://kaggle.com/competitions/aptos2019-blindness-detection>
- [205] H. Li, W. Li, H. Cao, S. Wang, F. Huang, and A. C. Kot, “Unsupervised domain adaptation for face anti-spoofing,” *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 7, pp. 1794–1809, 2018.
- [206] A. Krizhevsky, “One weird trick for parallelizing convolutional neural networks,” 2014.
- [207] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, “A convnet for the 2020s,” 2022.
- [208] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” 2018.
- [209] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” 2020.
- [210] —, “Efficientnetv2: Smaller models and faster training,” 2021.
- [211] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” 2014.
- [212] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” 2015.
- [213] Z. Tu, H. Talebi, H. Zhang, F. Yang, P. Milanfar, A. Bovik, and Y. Li, “Maxvit: Multi-axis vision transformer,” 2022.
- [214] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “Mnasnet: Platform-aware neural architecture search for mobile,” 2019.
- [215] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

- [216] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, “Searching for mobilenetv3,” 2019.
- [217] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, “Designing network design spaces,” 2020.
- [218] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.
- [219] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” 2017.
- [220] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, “Shufflenet v2: Practical guidelines for efficient cnn architecture design,” 2018.
- [221] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size,” 2016.
- [222] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, “Swin transformer: Hierarchical vision transformer using shifted windows,” 2021.
- [223] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei, and B. Guo, “Swin transformer v2: Scaling up capacity and resolution,” 2022.
- [224] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2015.
- [225] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” 2021.
- [226] S. Zagoruyko and N. Komodakis, “Wide residual networks,” 2017.
- [227] M. D. Zeiler, “Adadelta: An adaptive learning rate method,” 2012.
- [228] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011. [Online]. Available: <http://jmlr.org/papers/v12/duchi11a.html>
- [229] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017.
- [230] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” 2019.
- [231] B. T. Polyak and A. B. Juditsky, “Acceleration of stochastic approximation by averaging,” *SIAM J. Control Optim.*, vol. 30, no. 4, p. 838–855, jul 1992. [Online]. Available: <https://doi.org/10.1137/0330046>
- [232] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. [Online]. Available: <https://proceedings.mlr.press/v28/sutskever13.html>
- [233] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, “On the variance of the adaptive learning rate and beyond,” 2021.

- [234] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: the rprop algorithm," in *IEEE International Conference on Neural Networks*, 1993, pp. 586–591 vol.1.
- [235] A. Graves, "Generating sequences with recurrent neural networks," 2014.
- [236] T. Dozat, "Incorporating nesterov momentum into adam," 2016. [Online]. Available: <https://api.semanticscholar.org/CorpusID:70293087>
- [237] M. Schmidt, "minfunc: unconstrained differentiable multivariate optimization in matlab," *Software available at http://www.cs.ubc.ca/~schmidtm/Software/minFunc.htm*, 2005.
- [238] Evan You, Eduardo San Martin Morote, "Vue Router | The official Router for Vue.js," [Online; accessed 26-July-2023]. [Online]. Available: <https://router.vuejs.org/>
- [239] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," 2019.
- [240] I. Chingovska, A. Anjos, and S. Marcel, "On the effectiveness of local binary patterns in face anti-spoofing," 01 2012.
- [241] Z. Zhang, J. Yan, S. Liu, Z. Lei, D. Yi, and S. Li, "A face antispoofing database with diverse attacks," 03 2012, pp. 26–31.
- [242] D. Wen, H. Han, and A. K. Jain, "Face spoof detection with image distortion analysis," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 4, pp. 746–761, 2015.