uc3m | Universidad Carlos III de Madrid

Master's Degree in Computer Science and Technology

2022-2023

Master's Thesis

Continually Adaptive Machine Learning applied to a Healthcare Platform for the Detection of Tuberculosis

Simón E. Sánchez Viloria

Jesús Carretero PhD.

Lara Visuña Perez

Leganés, September 2023

ABSTRACT

The advent of machine learning (ML) has brought significant advancements to virtually every domain, with healthcare being no exception. However, due to the big risk of deploying ML models to make or aid in making high-stake decisions, together with the dynamic and data-rich nature of healthcare data, it becomes of special importance to design more robust and reliable systems for these applications.

Integrating techniques that allow ML models to continually adapt to the data they are continuously fed to presents a potential to mitigate these risks. However, it is important to evaluate how well these techniques perform and how they can be integrated into existing workflows ensuring that the utility of the models is not compromised in the process.

In this work, we combine ideas from self-adaptive system design and continual machine-learning literature to propose a framework for the continual adaptation of ML models. We implement our solution into a healthcare platform for the detection of Tuberculosis (TB), which is meant to be integrated into a data portal in the context of the ERA4TB project, a European initiative to accelerate the development of new drugs to treat TB. The platform includes a web-based interface for the annotation of medical images, a data management system, and a pipeline for the continual training and evaluation of ML models.

We evaluated our solution on a pre-trained deep-learning model (DETR), adapting it to reliably identify TB bacteria in sputum-smear microscopy images. We trained this model incrementally, applying active learning tactics to select the most informative samples for each next training round. Our experiments were conducted using as a baseline a DETR model trained on 202 images and achieved an AP@50 score of 0.824 on the test set.

With the proposed techniques, we were able to train a model that achieved comparable performance to the baseline while being trained on 40% fewer images. We discuss the challenges and potential of using continual learning techniques in this context, and how certain limitations can be addressed in future work.

Keywords: Machine Learning, Continual Learning, Tuberculosis, Computer Vision, Self-Adaptive Systems, System Design, Healthcare

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

As health information becomes increasingly digitized, machine learning (ML) algorithms play a crucial role in deriving meaningful insights from complex, multi-modal data that is often difficult to interpret by humans. Conventional machine-learning approaches, however, may be inadequate in the face of rapidly evolving health technology, shifting patient needs, and the increasing availability of large amounts of uncertain and noisy data that is deemed too unreliable for use in clinical settings.

Modern techniques aim to address these limitations by refining models in a continual loop, prioritizing data acquisition, labeling, and feedback from experts to dynamically adapt to the data stream reliably. A field of research known in the relevant literature as *continual learning* (CL) **parisi_continual_2019** This work explores the potential of these techniques in a medical context.

In particular, we consider the problem of developing a machine-learning platform to aid in the research of *tuberculosis* (TB), an infectious disease that affects millions of people worldwide, we incorporate continual and active learning methods into this platform to improve the performance of computer vision models used to detect TB.

The platform proposed here is meant to be integrated into a Healthcare and Data Portal that is being developed as part of an ongoing European project for the research of Tuberculosis, and it might serve as a tool to test and validate the use of machine learning models for its diagnosis.

As such, in an effort to research the competence of continual adaptation methods in that context, the platform incorporates a system that is designed to continually adapt to new data as it becomes available and to ease labeling efforts by prioritizing the acquisition of the most informative data samples, including a front-end interface for the visualization of the detection, labeling, and interaction with the models.

We evaluate the system's performance on a real dataset and compare it to a baseline model that doesn't use adaptation techniques. Our results show that the use of these methods outperforms the baseline in terms of robustness and sample efficiency while

maintaining a similar level of accuracy on the test set. Furthermore, the system proposed can automatically adapt to new data and improve its performance over time.

These results demonstrate the potential of incorporating these techniques into designing real-world systems for healthcare applications and other high-stake domains. The final chapter of this work includes a discussion about possible future work and improvements to the system that might facilitate its integration into other projects and a broader discussion about the potential of these techniques, going beyond the scope of this thesis.

1.1. Context and Motivation

Tuberculosis (TB) is an infectious pulmonary disease that has affected humankind for well over 4,000 years **cdctb_world_2023** and still affects millions of people worldwide. According to the World Health Organization, until the arrival of the COVID-19 pandemic, TB was the leading cause of widespread death from a single infectious agent, even above HIV **who_global_2022**.

This work is done in the context of the European Regimen Accelerator for Tuberculosis (ERA4TB). ERA4TB is a public-private initiative that started in 2020 and aims to create an open European platform to accelerate the development of new treatment regimens for tuberculosis (TB). The project is integrated by over 31 organizations from the European Union and the United States, including academic institutions, research centers, non-profit organizations, and other public and private entities **noauthor_era4tb**.

ERA4TB's mission statement aligns itself with (and is in response to) the United Nation's (UN) Sustainable Development Goals (SGD) to end the TB epidemic by 2030 **world_health_organization_regional_office_for_europe_tuberculosis_2017**. The project's website reads, 'The goal of ERA4TB is to deliver an innovative and differentiated combination regimen for the treatment of TB, which can play a key role in the TB elimination agenda' **noauthor_era4tb**.

To address some of the challenges of TB drug development and clinical trial design, one of the project's objectives is developing a data-science-specific platform to enable the efficient use of machine learning methods from the collaborative platform. The platform is meant to be used by researchers and other project stakeholders to facilitate the use,

development, and evaluation of ML models that can aid in the research of TB.

Thus, the motivation behind this work comes from the idea of incorporating novel methods into this data-science-specific platform to support researchers and collaborators in their mission to end TB by 2030. The techniques described in this work are designed to improve the performance of supervised models while prioritizing their overall robustness and reliability, aspects that are crucial in the context of healthcare applications.

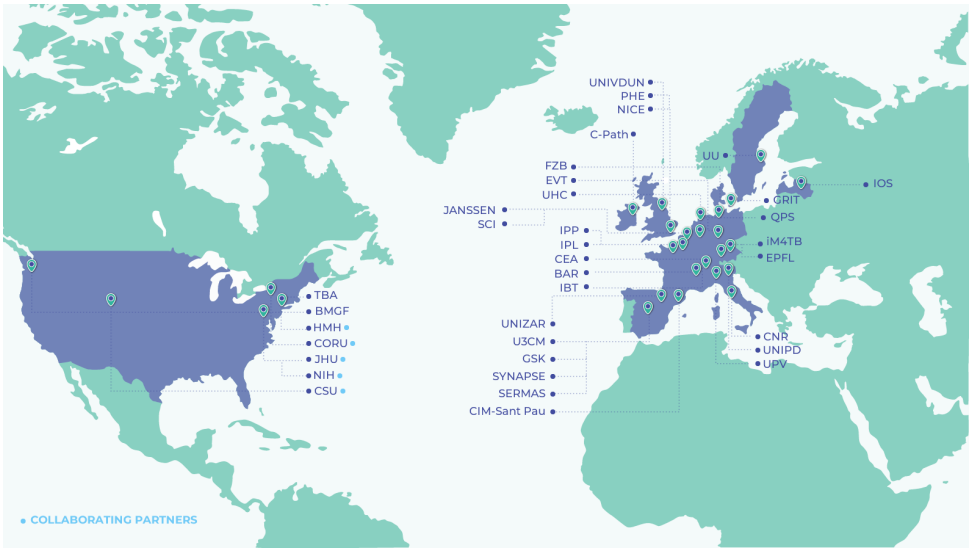

Fig. 1.1. ERA4TB's Consortium of Partners and Collaborators. Source: **noauthor_era4tb**

1.2. Background Concepts

The following section provides a brief introduction to some of the concepts and techniques that are relevant to this work. It is meant to provide the reader with the necessary high-level information to understand the context behind the ideas proposed that were used to inform every design decision and experiment conducted. For a more in-depth overview of the same concepts, the reader is referred to the literature review in Chapter **??**.

1.2.1. Tuberculosis Treatment and Diagnosis

Tuberculosis is caused by the bacillus (bacteria) *Mycobacterium tuberculosis* (MTB), which is transmitted when people who are sick with TB expel the bacteria into the air by coughing, sneezing, or spitting. The disease is preventable with the administration of

a vaccine and curable with the use of antibiotics over a significant period (although drug-resistant strains of the bacteria are becoming increasingly common) **who_tuberculosis_2023**.

Nonetheless, the disease is often underdiagnosed and undertreated, especially in low-resource settings (i.e., developing countries, rural areas, and marginalized/impoverished communities), where the disease is more prevalent, calling for the development of more efficient and cost-effective methodologies to diagnose and treat the disease **who_global_2022**, **imi_era4tb_2020**, **who_tuberculosis_2023**.

Some of the diagnosis techniques to detect TB include chest X-rays, sputum smear microscopy, and molecular tests. *Chest X-rays* are a common procedure to diagnose any signs of tuberculosis due to the wide availability of radiology devices, but they are often inconclusive and require expert radiologists to interpret the results **escalante_tuberculosis_2009**.

Sputum smear microscopy is another widely available technique that requires a trained clinician to identify the bacteria under images taken with a microscope of a patient's sputum (a mixture of saliva and mucus from the respiratory tract).

This technique is relatively inexpensive but requires a high concentration of bacteria in the sample to be effective. Additionally, studies argue that in conditions with limited resources and a significant number of samples, there have been reports of poor sample observation and quality control measures, which can result in false-negative results **desikan_sputum_2013**.

Molecular tests based on *nucleic acid amplification* (NAATs), similar to the ones popularized to detect COVID-19, are another technique to diagnose TB **cdc_tb_2016**. These tests identify the presence of bacilli by amplifying the genetic material of the bacteria in a patient's sputum sample (if any is present) and using a chemical solution to react to it.

NAATs are by far the most reliable method to diagnose TB and have the advantage of being rapid and fully automated. However, they are also expensive to produce and require specialized equipment, making them less accessible in low-resource settings **albert_development_2016**, **maclean_advances_2020**. Indeed, a study conducted in 2018 showed that the ratio of smear microscopy tests to NAATs in countries with a high burden of TB was 6:1 **cazabon_market_2018**, **maclean_advances_2020**.

Recently, there have been efforts to develop machine-learning models that can aid in

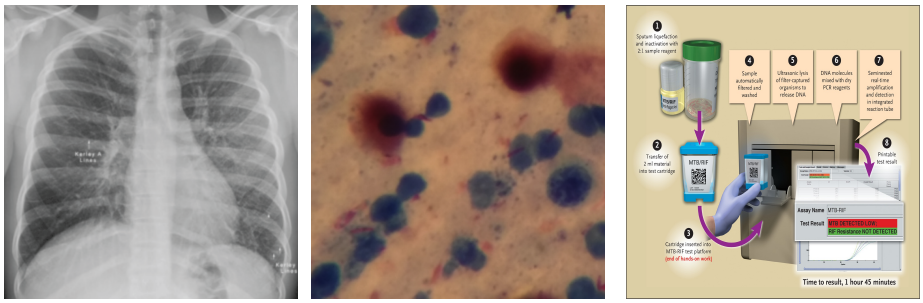

Fig. 1.2. Examples of common techniques to diagnose Tuberculosis. Left: Chest X-ray of a patient with TB **ubaidi_radiological_nodate**. Middle: Sputum smear with tuberculosis bacilli **shah_ziehlneelsen_2017**. Right: Example of a molecular test for MTB **boehme_rapid_2010**.

the diagnosis of TB as a way to reduce the need (or provide a first/second opinion) for expert clinicians in the process and improve the speed and/or accuracy of the diagnosis [1].

1.2.2. Supervised and Semi-Supervised Machine-Learning

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) that studies the design of algorithms that can learn from data and make predictions based on it. ML algorithms are fed a set of data samples (often called the training set) and learn a function that maps the input data to a desired output. The goal is then to learn a function that can generalize well to unseen data and make accurate predictions.

Supervised learning is a paradigm of ML where the goal is to fit a function $f : x \rightarrow \hat{y}$ that maps a given input, x, to a 'prediction' output, $\hat{y}$, based on an available finite set of input-output pairs (x_i, y_i) that are passed to a learnable model as 'training' data. The function is learned by minimizing a loss function $L(y, \hat{y})$ that measures the difference between the 'predicted' output and the actual one and returns a value that represents the error of the prediction, which is then used to update the model's parameters.

The most common supervised learning tasks are classification and regression, where the goal is to predict discrete and continuous outputs, respectively. Conversely, the aforementioned loss function is often defined based on the task at hand and the type of data available (e.g., cross-entropy loss for classification tasks, mean squared error for regression tasks, etc.).

The presence of a known output (or label) for each input is what makes this paradigm

[1] A literature review on the topic of TB diagnosis using ML methods is presented in section **??**

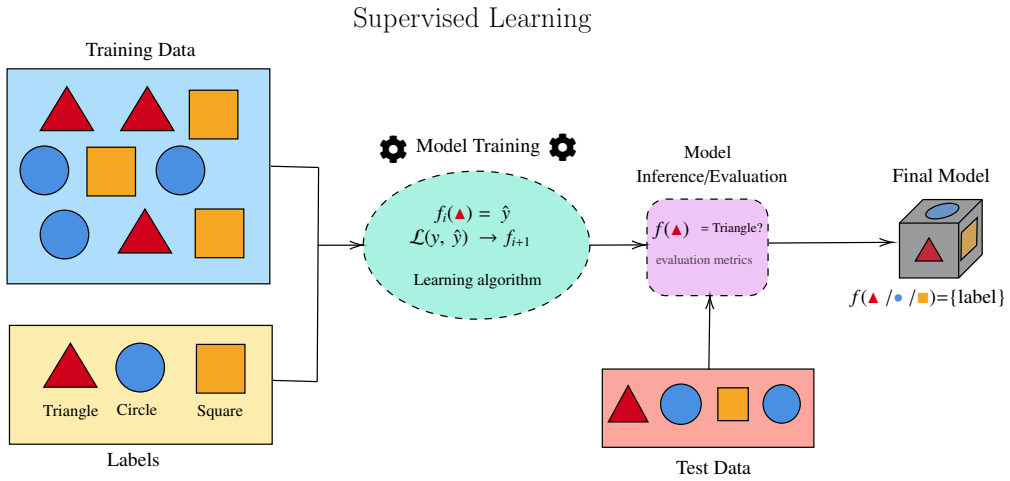

Fig. 1.3. Illustrative example of a supervised machine-learning pipeline.

'supervised'. For example, in healthcare, supervised learning might be used to predict the presence of a disease or condition based on a set of features extracted (or learned) from the patient's data. A model can be trained to predict the presence or severity of a disease based on a set of symptoms, clinical history, or imaging data.

Besides its many benefits, the most significant disadvantage of supervised learning is that it often needs large amounts of labeled samples to produce accurate and robust results **lecun_deep_2015**, approaches such as *semi-supervised learning* (sometimes also called weak supervision) aim to address this issue by leveraging both labeled and unlabeled data to train the model **zhu_semi-supervised_2008**.

Semi-supervised learning works by using unlabeled samples to learn an intermediary representation of the data that can be used as a first step to train a supervised model. This approach is especially useful when the unlabeled data is abundant and easy to obtain, but their labels are scarce and expensive. This is often the case in some healthcare applications, where the labeling task can only be performed by professional workers, making it a costly and time-consuming task **yakimovich_labels_2021**, **chen_study_2015**, **figueroa_predicting_2012**.

In the last decade, one set of algorithms that have enabled significant advances in Machine Learning, allowing to solve very difficult problems, is **deep learning**. Deep learning is a subfield of ML that studies the design of algorithms that can learn complex representations of data by hierarchically composing simpler functions in an architecture inspired by the structure of the human brain known as 'Deep Neural Networks' (DNN)

lecun_deep_2015.

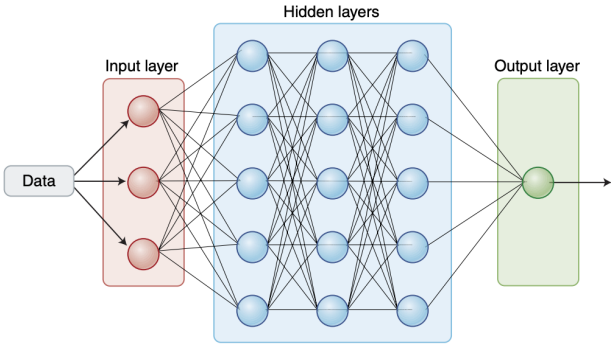

Fig. 1.4. Example of a simplified neural network architecture taken from Topol et. al (2019) **topol_high-performance_2019**.

1.2.3. Challenges and Potential of Adopting ML-enabled Systems in Healthcare

ML techniques have seen their adoption in many applications, from malware and spam detection to self-driving cars and environmental modeling. The healthcare field is no exception. In the 20 years between 1995 and 2015, the FDA had approved fewer than 30 algorithms for medical use. In contrast, only in the last 5 years, the total count of new approvals has reached over *10 times* that amount (see Figure **??**).

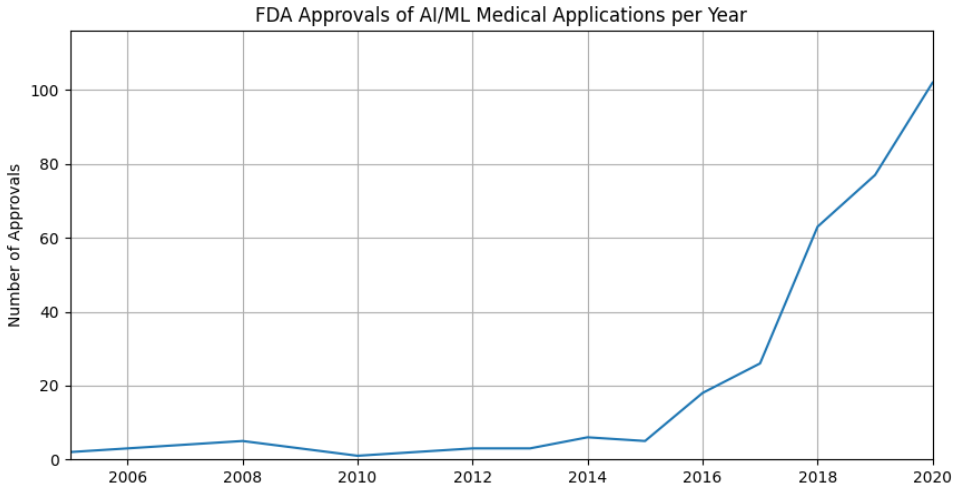

Fig. 1.5. Number of FDA-approved AI applications per year since 2005. Data Source: **health_artificial_2022**

Indeed, the recent availability to store and process ever larger amounts of data through the use of Big Data technologies and the development of more powerful hardware and algorithmic techniques have made it possible to train models that can perform complex

medical tasks with enough high accuracy and robustness to be considered for use in that industry **topol_high-performance_2019**.

Of such techniques, some that have recently disrupted the medical field are those based on **computer vision** (CV). CV methods try to develop algorithms that enable computers to solve visual tasks. It is a broad field that has been applied to problems like image classification, object detection, and image segmentation and has seen significant advances in the last decade thanks to the adoption of Deep Learning algorithms **lecun_deep_2015**.

In the context of healthcare, CV techniques have been primarily used in radiology to aid with the diagnosis of diseases and other tasks through the analysis of medical images (e.g., X-rays, CT scans, MRIs, microscopy, etc.) **esteva_deep_2021**. Indeed, over 75% of all FDA-cleared AI applications have been for radiology use cases **health_artificial_2022**.

One study made in 2018 trained a *Convolutional Neural Network* (CNN), a common DL model popular for computer vision tasks, to detect pneumonia from X-ray images at an Indian hospital. The researchers compared the performance of the algorithm with the findings of four expert radiologists and concluded that the algorithm was comparable to - sometimes even *outperformed* - the radiologist in most cases **wang_chestx-ray8_2017**.

Company	Year	Usecase	Panel
Apple	2022	Atrial Fibrillation Detection via Apple Watch	Cardiovascular
Arterys	2022	Liver and Lung Cancer Detection	Radiology
Philips Healthcare	2022	Philips Incisive CT Reconstruction	Radiology
GE Healthcare	2021	Deep Learning Image Reconstruction	Radiology
Siemens	2021	AI-Rad Companion for CT Interpretation	Radiology
Icometrix	2018	Brain MRI Analysis	Radiology
23&Me	2017	Genetic Testing for Hereditary Thrombophilia	Hematology

Table 1.1. Examples of FDA-approved AI applications for medical use.
Source: **health_artificial_2022**

Other AI systems such as Google Deepmind's *AlphaFold* **yang_alphafold2_2023** have been shown to predict the 3D structure of proteins with high accuracy, solving a problem that had been considered to be one of the most challenging in computational biology for over 50 years. Such breakthrough is thought to have a significant impact in applications like drug discovery in the near future **jumper_highly_2021**.

Results like these shine a light on the potential that AI techniques have in the med-

ical field. However, the adoption of such technologies doesn't come with new challenges. Experts have emphasized the importance of improving aspects of these models like their lack of interpretability, robustness to unseen data, and the difficulty of integrating them into existing workflows before they can be widely adopted in clinical settings **esteva_deep_2021**, **topol_high-performance_2019**.

Thus, the design of ml-enabled systems must be mindful of the limitations of such models. Research in the healthcare and AI fields must pave the way to develop workflows that can be trusted by every stakeholder alike to improve the quality of care/research, reduce costs, and overall increase the efficiency of the healthcare system.

1.2.4. Continual Learning and Self-Adaptive Systems

To address some of the challenges of adopting ML-enabled systems in healthcare, we can take a look at the idea of incorporating *continual learning* and *self-adaptive systems* (SAS) in their design. Continual learning refers to the ability of an ML system to learn continually from new data, adapting to novel changes in the data stream that it may have not been exposed to before **parisi_continual_2019**.

SAS has a similar definition, but it generalizes to any software systems that continually monitor faults in their operating environment using a closed feedback loop. The system can then modify its behavior at runtime by the execution of so-called **tactics** in an attempt to fix them, thereby reducing human efforts in the interaction. Nowadays, self-adaptivity is considered a classical concept with ample literature in fields like software engineering and robotics **macias-escrivaSelfadaptiveSystemsSurvey2013**, **gheibiApplyingMachineLearning2020**, **casimiro_self-adaptive_2022**.

The two ideas are closely related, with *continual* (or *lifelong*) learning often used to refer specifically to ML systems while *SAS* to any software that adapts itself to disruptive changes. The core idea of both is the same, though, the ability of a system to 'survive' variations in its environment with or without human intervention and continue to perform its intended function **macias-escrivaSelfadaptiveSystemsSurvey2013**.

In this work, we use the term **continually adaptative machine learning** to refer to this idea applied to ML models - drawing inspiration from both concepts.

Consider the scenario of a machine-learning model that has been trained with pictures

from an X-ray machine of a particular hospital. The model is then deployed in a different hospital where the X-ray machines are of another and produce images with different characteristics (a particular noise, artifacts, different resolution, etc.).

Because the data distribution is different from the one the model was trained with, the model's performance may likely be affected. However, under a continual learning setting, the model would be able to **adapt to the new data** and improve its performance over time.

On the other side, the design of such systems in practice presents unique challenges that must be addressed. Some experts argue that the questionable robustness of the adaptation process **vokingerContinualLearningMedical2021** is one of the reasons why **the FDA has never approved a medical application based on continual learning**.

Indeed, some known problems like **catastrophic forgetting** (when the model forgets how to perform a task after learning a new one) and **bias drift** (when the model's predictions suddenly become biased towards a particular class or group of samples) are among the most studied phenomena in continual learning.

Still, the fact that ML models are often deployed as static components in a dynamic environment with the assumption that the data used to train them is representative of the data it will encounter in the real world is another source of ML degradation **vokingerContinualLearningMedica**

In Table **??** we describe some relevant causes of degradation of ML systems. It gives an overview of different phenomena that can affect a model's performance and the challenges that must be addressed in the design of an adaptive system.

One important thing to note is that a model experiencing one of the degradation causes listed doesn't mean that it will necessarily fail or that it should be addressed. For example, a model might suffer from catastrophic forgetting after learning new information but that might not affect the performance of the current task in any significant way.

This is why a crucial aspect of designing such systems is that of defining the *adaptation criteria* that should be monitored in order to trigger the right *tactics* that will allow the system to adapt to the specific changes in its environment that cause it to degrade.

In chapter **??** we address some state-of-the-art techniques that can be used to deal with the causes of degradation listed in the table.

Cause	Description	Example
Data Drift	**Covariate shift:** When the input distribution $P(X)$ that the model was trained with differs significantly from the one in the inference environment (i.e. the input changes over time, but the model remains the same).	A model trained on chest X-ray images from a particular dataset is deployed in a hospital where the X-ray machines are of a different brand and produce images with different characteristics.
	Label shift: When the model is trained on a dataset whose class proportions are substantially different from the ones in the inference environment ($P(Y)$).	A model trained on a dataset where the proportion of positive cases is 50% is deployed in a hospital where the proportion is 10%.
Model Unfairness	**Model bias:** The model misrepresents or produces an erroneous causal relationship between its input features and the target output, often caused when the training data is not fully representative of the real-word.	A model trained with data from patients of a hospital in one neighborhood performs poorly after being deployed in another neighborhood with very different demographics.
Learning Plasticity	**Catastrophic forgetting:** The model no longer performs well in an old task after training on new information.	A model is trained to detect a specific disease, but when adapted to detect a different disease, it 'forgets' how to detect the first.
	Loss of Plasticity: The model is unable to learn new information after training on a specific task (DNNs are often prone to this phenomenon **parisi_continual_2019**).	A model is trained to detect a specific disease, but can't be adapted to detect a different one.
Adversarial Attacks	**Vulnerability to adversarial attacks:** The model is vulnerable to small perturbations to the input that cause it to make very different predictions than it would otherwise.	A bad actor makes imperceptible changes to individual pixels in an X-ray image that cause the model to misclassify the presence of a disease.

Table 1.2. Some causes of Degradation of an ML System and their characteristics.

1.3. Objectives

Main objective

The main objective of this work is to research the most relevant techniques on continual adaptation methods in Machine Learning, make a comparative analysis of the most relevant techniques and their relevance for health applications, and design and implement a system for the diagnosis of tuberculosis that incorporates these techniques into its design that can be integrated into the ERA4TB platform.

The platform should allow its users to incorporate machine-learning models into the platform, facilitate their use, and allow collaboration between researchers and other stakeholders.

Specific objectives

Auxiliary to the main objective, the following specific objectives have been defined to guide the development of the work and evaluate its success:

1. The system should be capable of automatically triggering the continual learning process when new data is available or when the model's performance degrades based on a predefined metric.
2. The platform must implement a feedback loop between the data annotation process and model training that prioritizes the acquisition of the most informative data samples to improve the model's performance (Active Learning).
3. Develop a front-end interface that allows users to interact with the machine-learning models by selecting or submitting new data samples and visualizing the model's predictions.
4. Consider the limitations of the proposed system and the ethical implications of its use and present a well-founded outline of necessary future work to address these limitations or improve the system in a way that aligns with the project's mission statement.
5. Evaluate possible future research directions that could be explored in the area of continual and dynamic adaptation in Machine Learning, highlighting the contributions that have a higher potential for impact in healthcare or other high-stake domains.

1.4. Structure of the Work

This work is divided into five chapters, including this introduction. Chapter **??** describes the context and motivation of this work and its objectives and provides the necessary background information to understand the concepts and techniques used.

Chapter **??** describes relevant work and state-of-the-art techniques. It also provides a literature review of related work in tuberculosis detection and adaptive machine learning systems, highlighting the most important contributions and their limitations and showing examples of their use in healthcare applications.

Chapter **??** describes the design of the system we propose. It aims to provide a high-level understanding of its architecture, the algorithms used, as well as the aspects we considered in the design of the solution, and the rationale behind them without going into the technical details of the implementation.

Meanwhile, chapter **??** describes the methodology used to implement the technical solution applied to our specific problem. Here we give a detailed description of the data, models, techniques, and specific tools used to come up with develop the platform, experiments conducted, and relevant metrics to evaluate the system's performance.

Chapter **??** presents the results of the experiments described in the previous chapter, analyzing the performance of the system, and comparing the results with the baseline metrics.

Finally, Chapter **??** presents the conclusions of the work. It discusses the implications of the results obtained in the context of the project and the limitations of the proposed system. It also proposes possible future work that could be explored to improve the system and its integration into the ERA4TB platform and discusses emergent directions in the areas of research discussed in this work, highlighting those that have a higher potential for impact or that we consider to be of special interest.

Additionally, there are two extra chapters at the end where we discuss the regulatory framework, ethical considerations, and the socioeconomic implications of the use of the proposed system (or similar systems) in the context of the project and the healthcare field in general.

2. STATE OF THE ART

The following chapter provides an overview of the current environment and state of the art in the topics related to this thesis. The goal is to provide a better understanding of the techniques that can be used to address the problem introduced in the previous chapter, highlighting emerging methods and their relevance to the problem.

We begin with a review of prior studies relevant to the topics of this thesis that have historically achieved state-of-the-art performance or have done similar work as the one proposed here. We go into detail about the implementation of the techniques used in each study, their overall contributions to the field, and how they relate to our work.

Then, in section **??**, we give a high-level description of known machine learning paradigms and techniques that have been proposed in the literature to address continual adaptation and the degradation of ML systems that we consider relevant to our solution, highlighting their application in the healthcare domain.

2.1. Literature Review

This section provides a detailed overview of the most relevant work in the literature related to the topics of this thesis. It is meant for readers who want to dive more into the details of the techniques studied, how they achieved state-of-the-art performance, their overall contributions, and how they relate to the problems posed in this work.

2.1.1. Computer Vision and DL-Based Object-Detection Techniques

This thesis focuses on continual learning applied to computer vision (CV) techniques to detect Tuberculosis. Thus, we consider it important to first review the CV literature with the goal that the discussion here serves as a good reference for the proposed solution and the techniques that will be used in the upcoming chapters.

Note that CV techniques cover a vast field, since the problem in this work relates mainly to image classification and object detection using Deep Learning (DL) algorithms (to identify Tuberculosis), we will limit our scope to the most relevant work in those areas.

First, it's important to highlight the importance of DL techniques in CV. The immense popularity that DL has gained in the area of CV can be traced back to 2012 when Convolutional Neural Network (CNN) architectures like AlexNet **dengImageNetLargescaleHierarchical2009** started showing breakthrough performance for solving image classification tasks in recognized competitions like the ImageNet Large Scale Visual Recognition Challenge (ISLVR) **krizhevskyImageNetClassificationDeep2012**.

CNNs are a type of neural network that uses *convolutional layers*, which can be thought of as a set of functions that learn image filters through the use of convolutional operations, to extract features from an input image [**goodfellowDeepLearning2016**]. CNNs were first introduced by Yann LeCun in 1989 (30 years before ImageNet), and they have since been used to achieve SOTA performance on a wide range of CV tasks, including object-detection **lecunBackpropagationAppliedHandwritten1989**, **zouObjectDetection202023b**, **goodfellowDeepLearning2016**, **lecun_deep_2015**.

Thus, bringing the focus specifically to Deep Learning for object detection tasks - and ignoring others like image segmentation, captioning, or image generation where CNNs have also achieved SOTA - we identify from the literature two main approaches to dealing with the problem: *two-stage* and *one-stage* object detectors.

Two-stage object detection methods first propose a set of candidate regions in the image (the *region proposal* stage) and then classify each region as either containing an object or not (the *classification* stage). **One-stage object detection** methods, on the other hand, directly predict the bounding boxes and class labels of the objects in the image **zouObjectDetection202023b**. Figure **??** shows a comparison between the two approaches.

Generally, the advantage of two-stage detectors is that they tend to be more accurate than one-stage detectors, as the region proposal stage allows them to focus on a smaller set of candidate regions that can be then individually discriminated by the classification stage. However, this comes at the cost of being slower than one-stage detectors since they require two inference steps through the network **zouObjectDetection202023b**.

The SOTA status of two-stage object detection methods precedes that of their one-stage counterpart. The popularity of CNNs in the CV field in the early 2010s led to the development of **RCNN** (2014). This architecture first proposes image regions by

selective search **uijlingsSelectiveSearchObject2013**, which are then rescaled and fed into a CNN to extract features to finally use a linear classifier to predict their label **girshickRichFeatureHierarchies2014**.

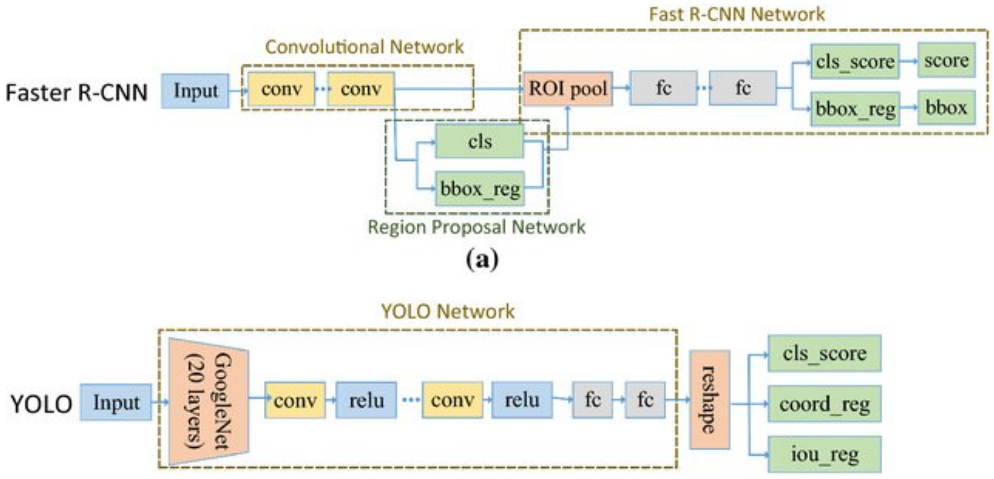

Fig. 2.1. (a) A two-stage Faster R-CNN object detector. (b) A one-stage YOLO object detector. Figure from Li et al. (2019) **liEnhancedBirdDetection2019**.

The success of RCNN for object detection tasks was followed by SPP-Net (2014), which introduced a Spatial Pyramid Pooling (SPP) layer to allow the model to process images of arbitrary sizes without rescaling them **heSpatialPyramidPooling2014**.

Then came **Fast RCNN** and **Faster R-CNN** in 2015, which respectively improved on RCNN by training the detector and bounding box regressor jointly, and by using a Region Proposal Network (RPN) to replace the much slower selective search algorithm used by previous models **girshickFastRCNN2015**, **renFasterRCNNRealTime2016**.

But while two-stage detectors were achieving state-of-the-art performance in terms of accuracy, they were not fast enough to be used for real-time applications or on embedded devices such as smartphones. The demand for methods that work for these devices led to the research on faster object detection methods that could be used for such purposes **zouObjectDetection202023b**.

It wasn't until 2016 that one-stage object detectors reached this milestone with the proposal of the **YOLO** (You Only Look Once) model **redmonYouOnlyLook2016**. Unlike RCNN-based approaches, YOLO poses the task as a regression problem, where the model directly predicts the bounding boxes and class probabilities of the objects in one

evaluation, allowing the system to be optimized in an end-to-end fashion. (Figure **??** shows YOLO's architecture).

YOLO marked a turning point in the field, a surge in the popularity of one-stage detectors led to more methods being proposed in the following years, from more adequate loss functions like RetinaNet's Focal Loss **linFocalLossDense2018** to subsequent developments to YOLO's architecture like YOLO9K (2016), YOLOv3 (2018), and YOLOv8 (2023) **redmonYOLO9000BetterFaster2016**, **redmonYOLOv3IncrementalImprovement2018**, **Jocher_YOLO_by_Ultralytics_2023** that improved on the original model's performance, allowing it to retain SOTA performance.

In this age, single-stage object detectors dominate the general benchmarks (e.g., ISLVR, COCO **linMicrosoftCOCOCommon2015**, PASCAL VOC **everinghamPascalVisualObject2010**) in both accuracy and speed. More recent methods like Meta's **DETR** uses a Transformer architecture **vaswaniAttentionAllYou2017** to overcome the limitations of traditional CNNs in terms of parallelization and locally restrictive receptive field, showing that abandoning convolutions in favor of an attention-only approach can also achieve SOTA **carionEndtoEndObjectDetection2020**, **zhuDeformableDETRDeformable2021**.

2.1.2. Tuberculosis Detection using Machine Learning Methods

While general object detection methods have achieved outstanding performance for more common object detection tasks, like detecting cars, people, or animals from images that could be taken in a wide range of conditions (i.e., those that most people would be able to identify), they tend to perform poorly when applied to more domain-specific tasks.

This problem is because models like those described in the previous subsection are trained on big datasets of images that can be commonly found on the internet and are easier to annotate, which tend to exclude more specialized images that are hard to obtain and even harder to annotate.

Furthermore, two-stage object detections like YOLO tend to perform poorly when applied to images containing small objects (e.g., cells, bacteria, etc.) because their region proposal mechanism favors larger objects with a clear distinction from the background.

These latter issues present a problem with the type of datasets usually available for medical applications, where images are typically taken in a more controlled environment

and contain objects that only experts in the specific domain can identify.

Furthermore, it is likely for objects of interest in medical images to be small and difficult to distinguish, as the need for specialized devices - often noise-prone and with low spatial resolution - is a common thing in the field.

This is the case of the kind of dataset used for TB detection, which is of interest in this work. **In sputum-microscopy images**, for example, the objects of interest are the bacilli that cause tuberculosis, tiny bacteria that are small and challenging to distinguish from other organic materials that surround them **osman_tuberculosis_2011**.

This makes the detection of tuberculosis a task that requires more specialized techniques to achieve good performance.

But even though the models that achieve SOTA in general benchmarks tend to fail for medical applications right out of the box, that doesn't mean that they cannot be adapted to more specialized datasets. In fact, the same methods have also been shown to perform well for medical applications when **adapted** to the task, and TB detection is no exception.

Some of the earliest examples we can find in the literature of the use of DL methods for detecting TB come from the work of Osman et al. in the 2010s **osmanDetectionMycobacteriumTuberculos** **osman_tuberculosis_2011**, **ahmadGeneticAlgorithmArtificialNeural2010**. Osman studied the use of techniques like multilayer perceptron (MPL) networks, K-Means clustering, and genetic algorithms, combined with more classical CV techniques like color thresholding and morphological operations, to detect and segment TB in **Ziehl-Neelsen (ZN) stained sputum smear** microscopy images.

More recently, Lakhani and Sundaram (2017) **lakhaniDeepLearningChest2017** used CNNs to classify Tuberculosis from chest X-ray images automatically. The authors used a deidentified dataset composed of 1007 posteroanterior chest radiographs, of which 15% were used for testing. The model achieved an AUC of 0.99 with an ensemble of AlexNet **krizhevskyImageNetClassificationDeep2012** and GoogLeNet **szegedyGoingDeeperConvolutions2014** models.

Roy et al. (2020) **royDeepLearningClassification2020** presented a deep learning-based method for the assisted diagnosis of **COVID-19** markers from lung ultrasonogra-

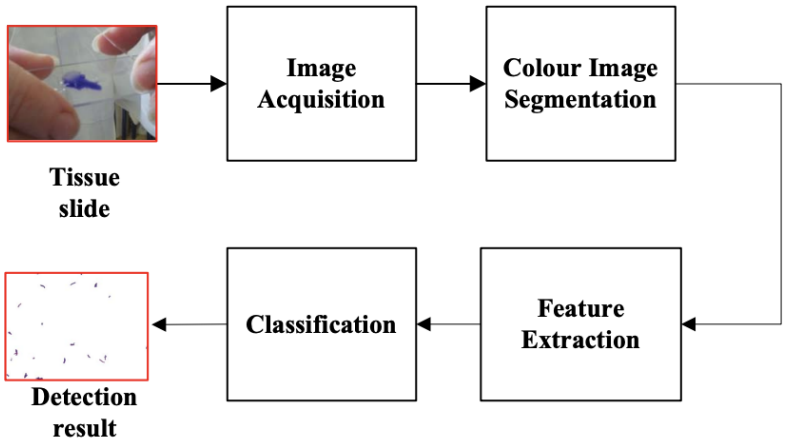

Fig. 2.2. Block diagram of the method proposed by Osman et al. (2010) **osmanDetectionMycobacteriumTuberculosis2010** for automated TB bacilli detection from sputum-smear microscopy images.

phy (LUS) images. The researchers collected data from six Italian hospitals and used it to train a CNN architecture derived from Spatial Transformers **jaderbergSpatialTransformerNetworks2016** (not related to Attention-based Transformers) to classify LUS images as either healthy or pathological and segment the affected area. The model was trained on 77 LUS videos from 35 patients for a total of 58,924 image frames and achieved a semantic segmentation accuracy of 96%.

The study most close to our work comes from Visuña et al. (2023) **visuna_novel_2023**, which presented a DL-based technique to localize tuberculosis present on sputum smear microscopy images. The author used a **one-stage object detection method** with a Convolutional Neural Network backbone to detect the presence of bacilli in the images.

Visuña first fragmented the image into patches of 80x80 pixels and then **classified each patch as either containing bacilli or not** for maximum spatial coverage. This study is very relevant to our use case since it uses the same dataset that we will study in this work. Their model was trained on 200 microscopy stain images and, using a 70/30 train/test split, achieved a 99.49%

2.1.3. Continually Adaptive Systems

Continually adaptive systems are systems that can adapt to changes in their environment instantly or over time, they are often used in applications where the environment is constantly changing, such as in robotics or autonomous vehicles **gheibiApplyingMachineLearning2020**,

casimiro_self-adaptive_2022. The thesis of this work is about considering their use case in a healthcare setting. Thus, we consider it appropriate to review some of the relevant literature in the topic.

Casimiro et al. (2022) discusses the challenges and opportunities of self-adaptive systems (SAS) in machine learning. The authors propose a framework for the development of SAS that rely on ML components. This framework is based on the concept of **MAPE-K** loops **kephartVisionAutonomicComputing2003**, which consists of a set of modules that allow the system to **M**onitor, **A**nalyze, **P**lan, and **E**xecute changes to itself (*tactics*) with the help of a **K**nowledge base that tracks the system's behavior (a MAPE-K loop is depicted in Figure **??**).

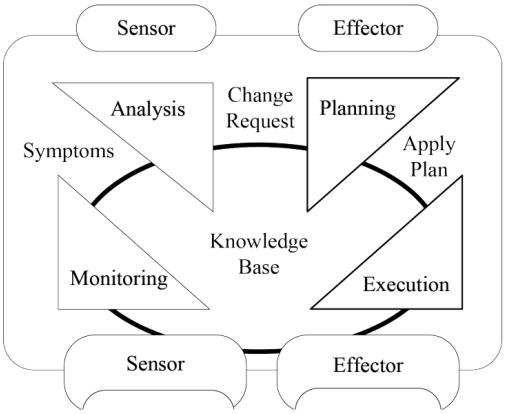

Fig. 2.3. The original MAPE-K Loop, as first introduced by IBM in 2004. Source: **redbooks_practical_2004**

Casimiro discusses the required changes to the traditional MAPE-K loop and the challenges associated with developing such systems for ML applications. They motivate their ideas by presenting a case study of SAS in the enterprise (ES) and cyber-physical (CPS) system domains. Table **??** shows an example of the problems they identified in these domains along with the learning-based tactics considered to solve them.

Note how rather than focusing on how to obtain a plan to adapt the system, the authors approach is to identify first the specific **causes of degradation** within each domain to then characterize the appropriate **adaptation tactics**.

Similar works such as Gheibi et al. (2020) **gheibiApplyingMachineLearning2020**, which makes a review of the literature on machine learning applied to self-adaptive systems, also focuses on the use of MAPE-K loops to develop such systems. However, the

Problem	Domain	Example Situation	Applicable Tactics
Covariate shift	ES	Transaction patterns change Adversaries poison data	• Component replacement • Unlearning
	CPS	Noise/uncertainty in sensors	• Transfer learning
		Different lighting conditions for face recognition	• Component replacement
		Adversaries manipulate smart meter data	• Human-based labeling • Unlearning
Label shift	ES	Variable fraud rate	• Human-based labeling
	CPS	Unknown command for voice controller	• Human-based labeling
Concept shift	ES	New fraud strategies	• Transfer learning
	CPS	Inhabitant's living patterns	• Retrain • Unlearning

Table 2.1. Casimiro et al. (2022) **casimiro_self-adaptive_2022**: Example of problems of Learning Systems within each domain and tactics to solve them.

scope of the study is about using ML to support SAS rather than using the latter to support the learning process.

Note that the difference between the two approaches is subtle but important. In the first case, the problem is about using learning algorithms as a means to support the adaptive system. The task of the ML model, then, is to improve one or more functions of the MAPE loop. For example, to predict the best tactic to use in a given situation (Planning) or detecting anomalies in the behavior of the system (Monitoring), without considering the tasks that it executes.

The second problem - the one we concern about in this work - is about using the MAPE-K loop as a means to **improve the learning process** of an ML model. That is, we are putting the learning process at the center of the system, and the task of the MAPE-K loop is to support it with any tactic (learning-based or not) that can help improve its performance and/or efficiency.

The idea of bringing MAPE-K loops to the context of this work is that much like traditional software systems, ML can also benefit from the study of SAS techniques (and the ample literature that exists on the topic) to improve their performance and robustness.

In her book 'Designing Machine Learning Systems' **huyen_designing_2022**, Chip Huyen presents the challenges and patterns of deploying ML Systems. Like Casimiro et al., she stresses the importance of monitoring a model's behavior to identify causes of degradation and continually update them. She also discusses how having **humans in the loop** who can provide feedback to the system is essential for their reliability in high-stakes

domains.

Vokinger et al. (2021) makes further discussion about the reliability of continually-adapted systems in healthcare, arguing that the risk that such systems pose in such a domain is likely the reason why the FDA has not approved any continual-learning systems for medical use **vokingerContinualLearningMedical2021**. The authors highlight problems like **catastrophic forgetting** and **bias induction** as two inherent risks of continual learning systems to address.

Adaptive systems that have been deployed for health uses can be more commonly found in the remote sensing domain, where typically a set of sensors is used to monitor signals from the user's body to obtain fitness-related insights. Jha et al. (2021) **jhaContinualLearningSensorbased2021**, for example, makes an empirical analysis of continual learning applied to Human-Activity Recognition models that can learn to recognize new activities over time different from the ones they were initially trained on.

A different approach to self-adaptivity comes from the authors of MsO-KELM **haoTechnologyOriented** a system that takes a kernel extreme learning machine (KELM) model **xiaEvolvingKernelExtreme2022** and introduces a swarm intelligence algorithm to optimize its hyperparameters in a self-adaptive manner. The authors show that their system can achieve better performance than other KELM-based models. This approach, however, suffers from the lack of flexibility MAPE-K loops provide, as it is not clear how this system would adapt to other problems or models.

In the next section, we will characterize some adaptation techniques and learning paradigms proposed in the literature that could be used as *tactics* of a continual-learning system based on MAPE-K loops that tackle the problems with ML systems that have been described.

2.2. Adaptation Techniques and Learning Paradigms

Throughout this section, we consider a set of novel techniques proposed in the literature related to ML model adaptation. One of the critical aspects in selecting which methods to include in this section - besides their relevance to the topic - was the technical feasibility of implementing them as *tactics* (as described above) in a self-adaptive system.

The idea is that some of the techniques described here may be used as the basis for the adaptation process of an ML model after it has been detected (through specific monitorization) that some form of degradation has occurred. The goal would be to then use some of these techniques (or a combination of them) as a way to improve performance.

2.2.1. Continual Learning

Continual learning refers to the concept of constantly updating a model as new information arrives, allowing it to adapt to changing data **huyen_designing_2022**. This is, of course, at the core of the problem we are trying to solve in this work. Rather than a specific method or technique, continual learning is a framework that encompasses a set of techniques that allow an ML system to learn continually from a data stream.

In its most basic form, continual learning is about updating the model when new data becomes available. However, this is not as simple as it sounds. The key to a successful continual learning process is that the model is updated such that it performs well on the new data without hurting it in the task it was designed for. While obvious, this is not a trivial problem, and it is a particularly known weak point of current ML algorithms. Often called the **stability-plasticity dilemma mermillodStabilityplasticityDilemmaInvestigating2013**.

Richard Sutton, one of the pioneers of the field of reinforcement learning (and author of the infamous article 'The Bitter Lesson' **suttonBitterLesson2019**), has very recently studied this problem in *deep supervised learning models*, a subject that is very relevant to this work. In an August 2023 paper (and seminar), he makes the big claim that 'Deep learning does not work for continual learning' **dohareLossPlasticityDeep2023**.

This statement is a bit exaggerated (by Sutton's own admission). What he argues about really is the reality that DNNs tend to eventually become very slow to learn from new data, eventually leading to a catastrophic loss of performance.

Problems with DNN plasticity have been studied before, Ash et. al's (2019) **ashWarmStartingNeuralN** was a very influential paper that discussed the failure of warm-starting neural networks, and proposed a regularization method consisting of shrinking and perturbing slightly the weights of the network at every optimization step, which improved significantly the performance of continual learning tasks.

Related to plasticity is **catastrophic forgetting**, when models forget previously learned

information when trained on new data, which is another well-studied problem with ML algorithms **mccloskeyCatastrophicInterferenceConnectionist1989**, **huyen_designing_2022**, **parisiContinualLifelongLearning2019**.

Beyond that, there are other myriads of issues to face when implementing a continual learning setting: how to select the optimal samples to train the model with, dealing with class imbalance, improving hardware efficiency, how to store and manage the data and evaluate performance, to name a few.

The work in this thesis is all about adopting continual learning for healthcare applications. We'll continue this section with techniques that can be used to implement or improve a continual learning system to face some of the challenges described above.

2.2.2. Transfer Learning and Domain Adaptation

Transfer learning is a technique that aims to improve the performance of a model by 'transferring' the knowledge of a pre-trained model to a new task. This is achieved by first training the model on a large dataset - that was presumably easy to obtain - from which it can learn general features and patterns of the data modality, and then **fine-tuning** it on a smaller dataset specific to the new task. Avoiding the need to train the model from scratch on the new dataset allows the model to achieve better performance with less data and training time **panSurveyTransferLearning2010**.

Nowadays, transfer learning has been adopted widely by the Deep Learning community in parts because it allows researchers and practitioners in the industry alike to effectively 'recycle' models that have been trained and shared previously on large datasets and apply them for their own purposes. Thus saving the - often prohibitively - expensive time and resources required to train deep-learning models from scratch on such significant amounts of data **yosinskiHowTransferableAre2014**.

Furthermore, it has been shown that initializing a neural network with pre-trained weights (i.e., using transfer learning) can help the model converge faster and achieve better generalization than training it from scratch. Showing the model can leverage the knowledge learned from previous data to learn the new task more efficiently **yosinskiHowTransferableAre2**

Transfer Learning is useful for healthcare applications because it allows us to adapt

pre-trained models to the medical domain - something referred to as **domain adaptation**. This is particularly relevant to our problem, we can use transfer learning to adapt models trained on general object detection datasets (e.g., COCO **linMicrosoftCOCOCommon2015**, PASCAL VOC **everinghamPascalVisualObject2010**) to the task of detecting tuberculosis in microscopy images. Something that has been done before in the literature with good results **visuna_novel_2023**.

We can envision this technique as part of a continual system that automatically adapts machine learning components to new tasks using transfer learning, thus reducing the need for human intervention.

2.2.3. Active Learning

Active learning strategies selectively acquire data based on their informativeness or uncertainty to the model. Its value comes from allowing the model to guide its own data acquisition process which can potentially reduce the need for vast amounts of pre-labeled data before a model is trained or updated by ignoring those samples that are unimportant/redundant to learn from.

We can find examples in the literature where Active learning has demonstrated its potential for medical applications. Chen et al. (2015), for example, used active learning to improve the performance of a named entity recognition (NER) model to extract important entities from clinical texts (diseases, medications, procedures) **chen_study_2015**.

The authors used three different data sampling strategies to select the most informative data to label. To achieve an F-measure of 0.80, their method required 66% fewer labeled instances than a baseline model trained on randomly selected data.

Going more in depth about how active learning is implemented, we identify from the literature (**huyen_designing_2022**, **chen_study_2015**, **settlesActiveLearning2012**) three main strategies that are commonly used for choosing which instances are the most important for an ML model to learn from. These are:

1. **Uncertainty sampling**: This strategy attempts to sample the instances that the model is most uncertain about. It is usually done by selecting samples for which the model's prediction is closest to its decision boundary (i.e., where the model is

most unsure about its prediction). For example, for a probabilistic binary classification model, you would sample from instances where the model is closest to 0.5 probability.

2. **Expected model change**: This one selects the instances for which the model's parameters are most likely to change. The way this is done depends on the model. For example, for a typical DNN trained with gradient descent methods, we may priorize the samples for which the gradient of the loss function might be the largest.

3. **Query by committee** is a strategy used mostly in ensemble learning that selects the instances for which the model's predictions are most diverse. This is done by training multiple models on the same dataset and selecting the samples that the ensemble disagrees with the most.

No hago nada con query by committee, pero aún así creo que esta bien dejarlo

In the context of this project, active learning is relevant because it may enable the development of more cost-effective ML models that require significantly less annotation effort. Thus, we consider it aligns well with the objectives of our work, making it of great relevance to developing the proposed system.

2.2.4. Knowledge Distillation

One drawback of Deep Learning models is that they often require a large number of parameters just to achieve decent performance on tasks like Computer Vision or Natural Language Processing, which makes them significantly computationally expensive.

However, in situations like this, one technique from the ML literature we can take advantage of is *knowledge distillation*, in which we attempt to improve the efficiency of the ML system by transferring the knowledge of a large 'teacher' model to another 'student' model with a much lower parameter count **hinton_distilling_2015**.

This 'distillation' process is achieved by training the student to mimic the prediction of the teacher model by feedings it the same type of data and penalizing predictions that are different from the output of the bigger model in its optimization process. Figure **??** shows an example of this process.

While conceptually simple, knowledge distillation can be a very effective technique to improve the efficiency of an ML system by reducing the computational costs associated

to it. The important part of the process is to find the right balance between the size of the teacher and student models, the student needs to be small enough to be more efficient than the teacher model but also have sufficient parameters to be able to learn reliably from it **hinton_distilling_2015**.

We consider that this technique fits well in a continual adaptation setting. We envision a knowledge distillation tactic that is executed when a model is too large to be constantly deployed in a resource-constrained environment.

In such cases, the system may trigger the training of a smaller model that mimics the predictions of the larger one and use it in its place if it is deemed to be more efficient and - at least approximately - as accurate as the previous model.

2.2.5. Adversarial Training

Adversarial attacks on machine learning occur when an attacker produces samples intentionally designed to be misclassified by an specific ML model. They are created by adding small perturbations to the input data that may be imperceptible to humans but can 'confuse' the model enough to make wrong predictions.

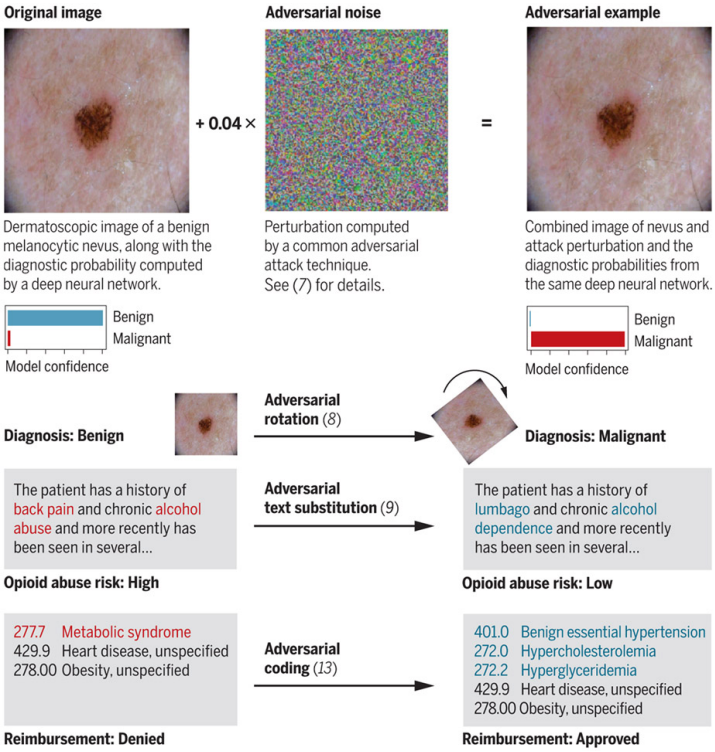

Fig. 2.4. Adversarial examples in health from Finlayson et al. (2019) **finlayson_adversarial_2019**.

Adversarial attacks can be a cause for major concern in healthcare applications. An attacker may purposely create these types of examples to fool an ML system into making incorrect decisions, which can have severe consequences like misdiagnosing a patient or prescribing them unnecessary treatment.

Furthermore, adversarial examples shine a light on the black-box nature of several types of ML models. The fact that these models can be fooled by small changes that are not perceptible by any expert raises questions about their general **reliability and trustworthiness.** This is why this phenomenon is an active area of research in the field **finlayson_adversarial_2019**.

One way to deal with adversarial attacks is by choosing models that are more **robust** to these examples. Goodfellow et al. 2015, for example, favor the use of nonlinear model families like Radial Basis Function Networks or using regularization strategies like dropout, weight decay, or gradient masking to improve their robustness [**goodfellowExplainingHarnessing**

Another popular approach to make ML models more robust to these attacks is to feed adversarial examples directly into their training data. This is known as **adversarial training**. Methods like the Fast Gradient Sign Method **goodfellowExplainingHarnessingAdversarial2015** (FGSM) or the more recent Projected Gradient Descent (PGD) [**madryDeepLearningModels2019**] can be used to generate adversarial examples and train the model on them.

For use cases where the vulnerability of healthcare systems is considered important, We consider the possibility of integrating adversarial training into a continual learning process. This can be done by continually incorporating - or generating - adversarial samples into a dataset prior to retraining/fine-tuning, in an attempt to improve the robustness of the model to adversarial attacks.

2.2.6. Dynamic Quantization and Network Pruning

We consider the idea of integrating more hardware-related optimization techniques that aim to reduce the size of a DNN and/or reduce its computational cost. Quantization and Network pruning are some interesting methods to accomplish this. These two techniques have been shown to be promising in significantly reducing the computational cost of Deep Neural Networks without significantly affecting their performance **carreira-perpinan_model_2017**,

han_deep_2016, **carreira-perpinan_compression_2018**.

The way they go about doing that is different. Network pruning attempts to remove redundant parameters of a neural network to **optimize its size and computational cost**. Quantization, on the other hand, reduces the precision of the weights and activations to **reduce its memory footprint** (e.g., converting all 32-bit floating-point numbers to 16-bit to cut memory size in half **han_deep_2016**, **carreira-perpinan_compression_2018**).

It is important to note, however, that there's generally a **tradeoff** from using these techniques as they tend to reduce the model's performance. Like in the case of knowledge distillation, one should assess a balance between task performance and computational costs to determine whether the use of these techniques is worth it.

The way we would propose such an adaptation tactic would be to continually evaluate different versions (quantized, not quantized, pruned, not pruned) of the model to determine which one is the most suitable for the given task and under what circumstances.

For example, we could have a model that is trained on a particular task and then quantized to reduce its computational cost. The system would then evaluate the performance of the quantized model on the task under all monitored aspects and determine whether the performance drop is relevant enough to warrant the use of the quantized model and reduce the computational cost of the overall system.

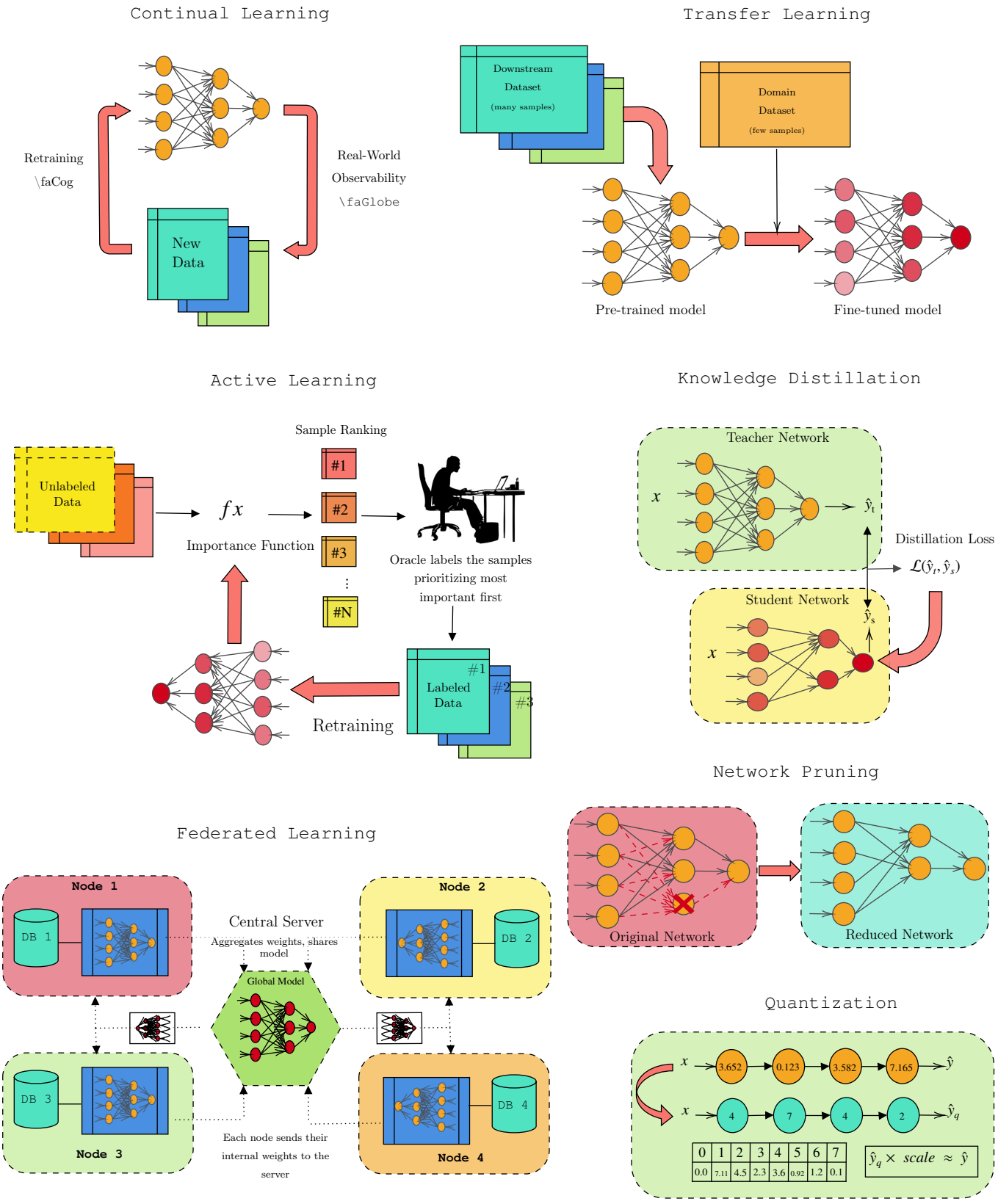

Fig. 2.5. Illustrations of the relevant Machine Learning Paradigms and Techniques

Reference	Domain	Technique(s)	Summary of Contributions
Visuña et al. (2023) **visuna_novel_2023**	Tuberculosis detection	Object detection, Image preprocessing, CNN fine-tuning, NASNetMobile	Tested novel DL-based method on 200 microscopy stain images of sputum. Reached 99.49% precision and 92.86% recall

Table 2.2. Summary of the most relevant works in the literature.

3. DESIGN OF THE SOLUTION

The work in this thesis is driven by the problem of designing an end-to-end platform that can be used to continuously develop ML models to aid in the research of TB while taking full advantage of the techniques we've discussed in the previous chapter.

The goal is that the platform can adapt the models to new data in an attempt to maximize their performance and reliability as much as possible.

The following chapter aims to present a blueprint for such a system. We discuss the relevant aspects we've considered when coming up with our solution and provide details about its design. Formalizing the different components of the system and the processes that take place in each of them.

Note is that throughout this chapter, we use the term 'model' or 'ML model' to refer to any learning-based algorithm that can be trained on data and outputs predictions. We make this distinction to avoid confusion with other uses of the term 'model' in the context of system design (e.g., database models), or other AI models that don't require any data (like some planning algorithms).

Similarly, we use the term 'platform' to refer to the system proposed, using both terms interchangeably.

The takeaway from this chapter should be a general understanding of the solution we propose and the different components that comprise it.

3.1. Aspects Considered

The following are the main aspects and characteristics that we consider that should be included in the design of a solution to the platform we propose in this work:

1. Clinical data is usually scarce and expensive to obtain, but extremely valuable for the development of ML models. Thus, the system should be able to take full advantage of the data available and the annotations made to it. The use of data-driven methods should be a priority in the design of the platform.

2. The platform should be able to handle any number of models and data samples, as well as the different types of data and models that may be used (healthcare data can be highly heterogeneous). Thus, the system should be as general and flexible as possible and follow a model-agnostic approach.

3. Once an ML model is developed by any stakeholder of the project, it may be uploaded to the platform for subsequent deployment, monitoring, and continual learning.

4. When ML models are uploaded, the data associated with them, including their corresponding labels (if the data is annotated), is accessible to the platform in a data repository, which we'll refer to as the *Knowledge Database* and denote it as $\mathcal{K}$ (in reference to the 'K' of a MAPE-K loop).

5. The system should also be capable of distinguishing between different types of data and the models that are trained on them. Mainly, $\mathcal{K}$ should differentiate data used to train a given ML model (*training data*), the one used to evaluate it (*evaluation data*), and between *annotated* and *unannotated* data regardless of whether it is used for training or evaluation, as well as support the identification of any subset of data relevant to the adaptation tactics (more on this in the next section).

6. The platform should be able to deploy any given ML model in a production and/or experimental environment, which it set up for inference tasks, and to continuously monitor and evaluate the model on any data associated with it.

7. The system should be capable of launching *adaptation tactics* when it detects that an ML model no longer performs well by any metric relevant to the stakeholders or when there's evidence that an adaptation tactic might bring significant improvement (e.g., newly annotated data becomes available, the performance of the model suddenly drops significantly after an event).

8. Finally, after evaluating the results of the adaptation process, the platform should be able to update any models that are being actively monitored with newly adapted ones that outperform their previous iteration.

9. If the adaptation process is unsuccessful, the system should continue to monitor the model and launch new adaptation tactics in the future if necessary.

Furthermore, for the purpose of this work, we consider the role of the annotation process to be a crucial aspect of the system. We refer to *annotation process* as the process of labeling the data with the correct output values, which is necessary for the training of supervised ML models.

As we have established, in the healthcare sector, labeling tasks are usually performed by expert human annotators. Thus, we consider that the solution we propose should ease annotation efforts as much as possible and focus much of our decisions to that aspect.

3.2. System Architecture

Based on the aspects described above, we propose a simplistic system architecture that is capable of addressing all the requirements of the platform. This architecture is depicted in Figure **??**. Throughout the next sections, we will attempt to describe, and even formalize, the different components of this system and how they interact together to produce the required functionality.

Note that the system depicted has been designed to be as general and flexible as possible to be applied to any ML application (as mentioned in aspect #2). However, later, we will focus on the application of the system to the problem of detecting and classifying TB from sputum smear microscopy images.

Thus, right now, we take a model-agnostic framework to the design of the system without considering any aspect of implementation or specific details about later applications. Similarly, we consider our data to be able to come in any modality (tabular, images, text), ignoring the fact that we will apply it exclusively to vision-based models.

We continue the next section with a detailed description of each of the components of the system we propose.

3.3. Components of the System

The system we propose in this work is comprised of five main components, which were explicitly designed to allow the implementation of adaptation tactics in a framework similar in fashion to MAPE-K loops (or at least inspired from it) and following the aspects described in section **??**.

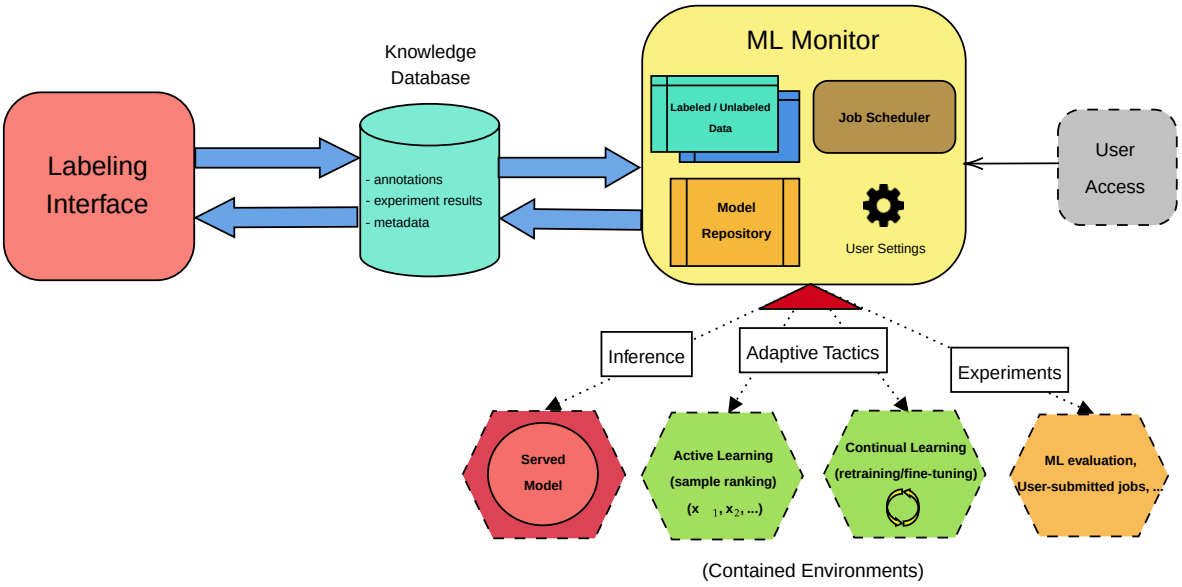

Fig. 3.1. Diagram of the Proposed System

3.3.1. Knowledge Database

The *Knowledge Database*, which we denote as $\mathcal{K}$, is the source of truth of the system. It is the component that stores all the data and information that is relevant to the platform (the *K* in MAPE-K). We have implemented it in our solution as a relational database (PostgreSQL) that stores all the data samples, annotations, and models that are part of the system.

Thus as a relational database, *K* can be characterized by the entities that comprise it and the relationships between them. Note we talk about 'entities' as types of information that map to (have a relation with) other entities in the database. We also denote 'object entities' as stored instances of these entities in $\mathcal{K}$.

Throughout this chapter, we use the notation $entity_a \Rightarrow entity_b$ to denote a relationship between two entities in *K*, where $entity_b$ can be considered as an attribute of $entity_a$. For cases where the attribute of an entity is not another entity, but a value of some datatype, we use dot notation (e.g., for "e.`name`", `name` may be a string attribute of some entity e).

With that in mind, we consider the following entities to be part of $\mathcal{K}$[2]:

[2] **??** shows the entity-relationship (ER) diagram of an implementation of the knowledge database (the one we use for our solution).

- **Entity $x \in \mathcal{K}$** is a single *data sample* that can be annotated by a human expert or by the system itself (e.g., a single clinical image, text, or a row in a table). Data samples have as attributes their corresponding annotations ($x \Rightarrow annotations$). We denote this entity x because it is the information that can be used as input to an ML model - crucial to the processes of training and inference.

- **Entity $annotation \in \mathcal{K}$** is an *annotation* of a data sample $x \in \mathcal{K}$ made by a human expert or by the system itself. Annotations have attributes related to their annotator ($annotation \Rightarrow annotator$), and the label of the annotation, which we characterize as a set of *property* entities ($annotation \Rightarrow properties$).

- **Entity $p \in \mathcal{K}$** is a property of a single *annotation* object. Properties have as attributes their name and value (p.`name`, p.`value`), which we use to describe the information carried by that annotation. For example, a 'binary label' property may have values 1 or 0 depending on the annotation.

- **Entity $annotator \in \mathcal{K}$** is an entity that creates annotations (and their corresponding properties) to any data sample x. It can be a human expert or an automated system. Annotators have a boolean *is_automatic* attribute (*annotator*.`is_automatic`) that characterizes if the annotator is a model deployed to be used by the system (e.g., as part of an adaptation tactic or to make predictions on data).

Figure **??** shows the entity-relationship (ER) diagram of the implementation of the knowledge database we designed for our solution. At the core of the schema are the same concepts that we listed above, we have only included a few more entities for data management purposes (e.g., *project*, *project_annotator*, *datastore* etc.) that are not very relevant to the design, so we won't discuss them in detail [3].

Note also that the *artifact* entity in that diagram is equivalent to the x entity we described above. For our use-case, these artifacts point to the location of the image files that we can use as input to our TB detection models, and the *annotations* of these artifacts, along with their *properties*, contain the labels of each of those images.

Such is our implementation of $\mathcal{K}$ in this project. But to keep things general, we will formalize further how we describe the relations in $\mathcal{K}$ and how to denote the act of extracting information from it.

[3] For more details about the other entities in the implementation of the database, see **??**.

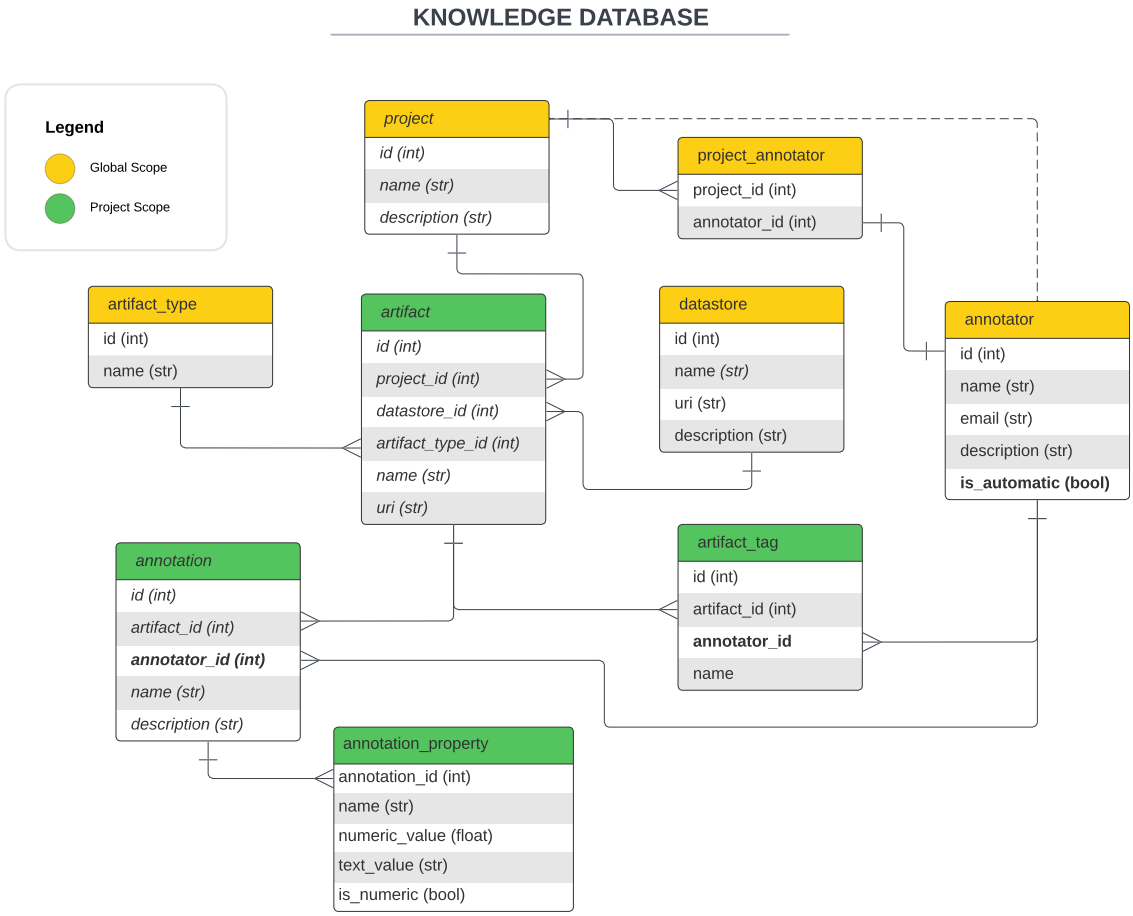

Fig. 3.2. Entity-Relationship (ER) Diagram of the Knowledge Database ($\mathcal{K}$)

First of all, we note that a single data sample can have multiple annotations. We consider relationships like $\{x \Rightarrow \mathit{annotations}\}$ to denote a one-to-many mapping that returns all *annotation* objects related to x. Similarly, we do the same with $\{\mathit{annotation} \Rightarrow \mathit{properties}\}$, and $\{\mathit{annotator} \Rightarrow \mathit{annotations}\}$.

Note we surround these types of relationships with curly brackets to emphasize that they return sets of entity objects in $\mathcal{K}$ (as opposed to single entity objects).

In an abuse of notation, we also allow the use of $\Rightarrow$ for these sets of entities. For example, the expression $\{x \Rightarrow \mathit{annotations} \Rightarrow \mathit{properties} \Rightarrow \mathit{property}.\text{name}\}$ denotes the set of all names of the properties of annotations of x. This will be useful at the time of defining the algorithms for the tactics we'll formalize later in this chapter.

Furthermore, as a database, $\mathcal{K}$ can be queried to extract any information in it (be it as entity objects or attribute values). We will use the syntax described above, along with general relational algebra syntax[4] **silberschatzDatabaseSystemConcepts2011**, to denote that action.

For example, to represent a query that gets all annotations in $\mathcal{K}$ that has a property named 'label', we use the syntax:

$$\{a \in \mathcal{K}_{annotation} \mid \text{'label'} \in \{a \Rightarrow properties \Rightarrow property.name\}\}$$

Where $\mathcal{K}_{annotation} \subset \mathcal{K}$ is the set of all *annotation* object entities in the database (we'll use the same notation to express sets of all objects of a single entity when convenient [5].

3.3.2. Model Repository

As mentioned in section **??**, we consider the *model repository* to be the 'bag' of models available to the system. We denote it as $\mathcal{M}$, and a single model as $m \in \mathcal{M}$.

$\mathcal{M}$ stores the models that are currently deployed in the system and are being actively monitored to be continuously adapted. Thus, it can be seen as a subset of all models that have been trained and evaluated (since some of them may have been discarded or replaced at some point).

The action of adding a model to $\mathcal{M}$ we denote as $m \rightarrow \mathcal{M}$, and the action of removing a model from $\mathcal{M}$ as $m \leftarrow \mathcal{M}$. Consequently, we can consider that the action of updating a model in $\mathcal{M}$ is the combination of both actions, with m_{new} being added and m removed.

Any $m \in \mathcal{M}$ can be seen as a function that takes a data sample $x \in \mathcal{K}$ as input and outputs a prediction $\hat{y}$, which can be used and tracked by the system. We denote this function as $m(x) = \hat{y}$ (for $x \in \mathcal{K}$).

One aspect of the design of the system is that we assume a one-to-one correspondence between models in $\mathcal{M}$ and the automatic annotators in the knowledge database $\mathcal{K}$. Thus, sometimes we treat them as equivalent entities in our design of the framework.

That is, we make the following equivalence:

[4] See: wikipedia.org/wiki/relational_algebra
[5] In **??**, we include an example of how this query might look like in standard SQL syntax.

$$\text{For:} \quad \mathcal{K}_{\mathcal{M}} = \{annotator \in \mathcal{K} \mid annotator \Rightarrow \text{ is automatic}\}$$

$$m \in \mathcal{M} \iff m \in \mathcal{K}_{\mathcal{M}}$$

From an implementation perspective, the mapping between the model repository and the knowledge database could be done by using the same identifier for both entities (which is what we do in our implementation), then, when you query a model from $\mathcal{M}$, you can also query its corresponding *annotator* entity from $\mathcal{K}$.

Thus, from this point on, we will use m to refer to both a model in the model repository and an automatic annotator in the knowledge database interchangeably. We also assume that any process that can pull m from $\mathcal{M}$ can also do it from $\mathcal{K}_{\mathcal{M}}$ and use it with the same function without any distinction, which will be useful for the design of the adaptation tactics that we'll describe in the next section.

Furthermore, we assume that either M or K can track the samples $x \in \mathcal{K}$ that are associated with m (the samples that can be used as input to m). We denote this set of samples as $\mathcal{K}_m$. This feature could be implemented by adding attributes or entities to the database that map ML models to data samples but to simplify things we'll refer to it simply stating it as the set $\{x \in \mathcal{K} \mid x \text{ is associated with } m\}$ (for some $m \in \mathcal{M}$).

3.3.3. ML Monitor

The ML monitor is how we refer to the component that is in charge of deploying the models in $\mathcal{M}$ in a production and/or experimental environment and continuously monitoring their performance on the data samples in $\mathcal{K}_m$.

Alongside the Knowledge database and job scheduler, the functionalities of the ML monitor complement our implementation of the MAPE-K loop that we propose in this work. We consider it to be the component that is in charge of the *monitoring*, *analysis*, and *planning* phases of the loop (MAP), while the job scheduler (which we describe in **??**), the *execution* phase.

Thus the job of the ML monitor is to keep track of all models in M, continually analyze how they perform when new data arrives and decide which strategy to take. Note that in figure **??**, we've depicted the model repository inside this component to emphasize the fact that only ML models in $\mathcal{M}$ are being monitored by the system.

We formalize the idea of monitoring and analyzing the performance of ML models by introducing the concept of *benchmarks*, functions $B(m, \mathcal{K}_m)$ that take a model $m \in \mathcal{M}$ and some data associated with it and outputs a metric of how well it performs.

The benchmark function can be any metric that is relevant to the stakeholders of the application as long as it works for the model and data (i.e., no need to assess binary classification performance on clustering tasks).

Since we consider that model and its relevant benchmarks are strictly associated, we assume that the platform can query the benchmark functions associated with any model. We denote this set B_m for some $m \in \mathcal{M}$. We implement this in our system as configuration settings that users can manually pass to the ML monitor directly (see Figure **??**).

For example, in the case of TB detection with the models we have developed for this work, we can use metrics like the *accuracy* or *F1-score* of the model on the image samples. When new microscopy images are available, the system may launch tasks that prioritize the annotation of certain samples (active learning) to then benchmark the performance of the model compared to the true annotations and launch an adaptive active if it drops below a threshold.

Because this process is fully data-driven (we only need to select which samples to pass to the already deployed model), it can be easily automated and be done without any human intervention.

3.3.4. Job Scheduler

separate ML monitor from job scheduler in the diagram

The job scheduler is the component that is in charge of actually launching the adaptation tactics that are part of the system. We denote it as $\mathcal{J}$, and a single job as $j \in \mathcal{J}$.

We consider j to be any function that requires the use of the models in $\mathcal{M}$. Which in our case can range from running a benchmark test, evaluating the performance/efficacy of models, and initiating adaptation tactics.

The operations of the job scheduler are primarily governed by the benchmark outcomes relayed from the ML monitor. If the benchmark performance of a model is flagged as underperforming (i.e., below a certain threshold), the ML monitor will interact with J to trigger an adaptation tactic on that specific model.

Furthermore, we consider j to be any function that requires the use of the models in $\mathcal{M}$. This includes any benchmark, model evaluations, and adaptation tactics.

From a practical point of view, having a job scheduler allows us to centralize the execution of all operations relating to the models in $\mathcal{M}$. This design choice reduces the overall complexity of the system and simplifies the management of the ML models (for example, only J needs access to their actual raw files).

Another point that we made sure of in the design of this component is to require each task/job to be independent of each other and containerized in its own computing environment so that they can be executed in parallel, or even put in a distributed computing setting. Which may allows to scale system like this to handle many models and data.

3.3.5. Labeling Interface

The final component we discuss in this section is the *labeling interface*. The idea of this component comes directly from the objective that a system like the one proposed should ease annotation efforts to optimize the costs associated with them.

The main aspect of this component is that it allows a human expert to annotate data samples in $\mathcal{K}$ and add them to the knowledge database, bringing them closer to the adaptation process. This human-in-the-loop approach can then be used to inform the next steps of the adaptation process.

Unlike the other components, we make no formalization about how the labeling interface should work, we consider that to fully depend on the kind of tasks that are being solved by the ML models and the data that is being annotated.

Thus, the design of such an interface should be oriented towards enabling - promoting even - these types of interactions, as to consider the information gained from them for the development of adaptation tactics that target the models monitored by the platform.

For the project in this work, we implement this interface to facilitate the identification of TB bacilli in sputum-smear microscopy images. The interface was designed to show these kinds of images (pulled dynamically from the database) along with the bounding boxes (rectangular regions in the image) where a bacillus was identified.

Figure **??** shows the interface that we developed for this project.

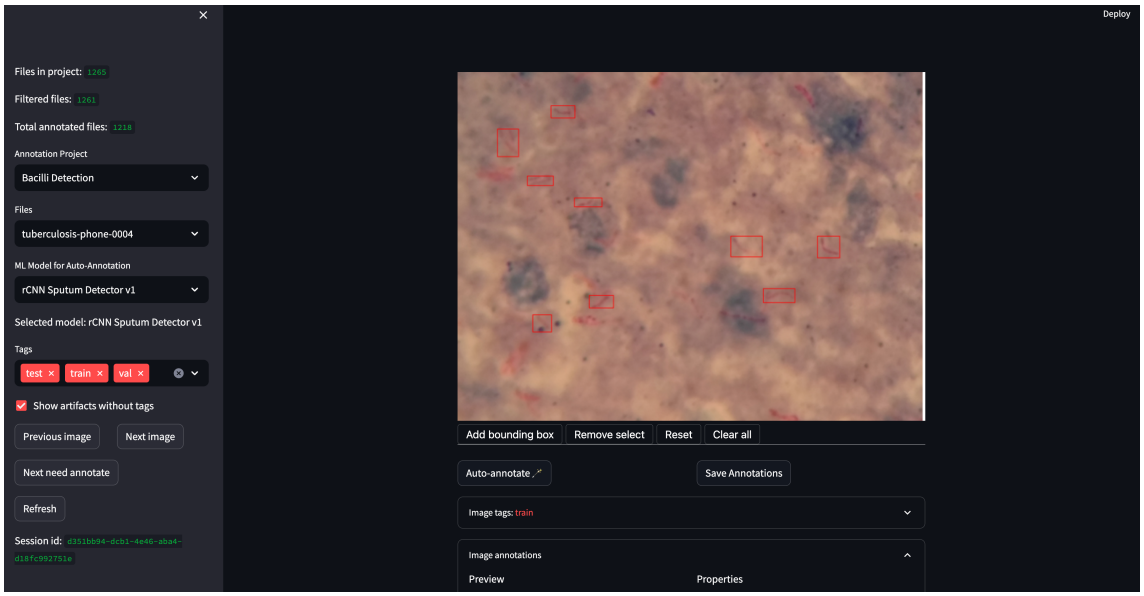

Fig. 3.3. Labeling Interface to annotate TB Bacillus:
The interface includes buttons to filter through the relevant samples in the database, and add boxes in regions of the image where the bacilli is identified. Annotations are then sent to the database to be compared with model predictions, which may trigger an adaptation tactic.

The interface also allows annotators to create their own bounding boxes in the regions of the image where they identify bacilli are located and compare that with the predictions of the model. This information is recorded and the annotations and relevant properties are saved to the knowledge database.

Adaptation tactics like *active learning* can then take the new annotations to update their prior beliefs about the samples that are most important to annotate and with that improve the productivity of the entire process.

3.4. Formalizing the Adaptive Process

We follow this section formalizing some of the adaptation tactics we consider could be applied to the system we propose. These tactics are the main drivers of the adaptation process, and thus, we consider them to be the platform's most critical components.

Thus, to provide a general understanding of how these tactics work and interact with the ideas we have described in this chapter, we will write down the algorithms that fully describe them, along with a brief explanation of how they work and their purpose within the context of this project.

3.4.1. Continual Learning

We use continual learning as a base for the adaptation tactics we propose. The main idea of a continual learning process based on the aspects of our system is to use the data available in the knowledge database to train ML models that can be used to replace the ones that are currently deployed.

The way this is approached is by training a new model, m_{new} with the data available in the knowledge database and then comparing its performance with the m that is currently deployed. If m_{new} outperforms m in the relevant metrics, then it is deployed in its place. Otherwise, the old model is kept, and the process is repeated in the future.

Furthermore, we can consider two different approaches to the continual learning process: We can either *retrain* the model, m or fine-tune it. We characterize the two approaches based simply on the data taken into account during the training process.

In the case of retraining, the model is trained with all the data available in the knowledge database that is associated with it, while in the case of fine-tuning, the model is trained with the new data only (as with transfer learning). In the latter case, the already trained model is used as a starting point, and its parameters are updated with the new data.

The following pseudocode describes the continual learning process:

Algorithm 1: ContinualLearning(m, $\mathcal{K}'_m$)

Input: model $m \in \mathcal{M}$, data $\mathcal{K}'_m \subset \mathcal{K}_m$

Result: `success` if $m_{new} \in \mathcal{M}$, `failure` otherwise

Let m_{new} = `Train`$(m, \mathcal{K}_m)$

The platform should be able to complete the training process of m_{new}

for $B_i \in \{m \Rightarrow B_m\}$ **do**

  B_m *is the set of benchmark functions associated with* m

  if $B_i(m_{new}, \mathcal{K}'_m)$ *better than* $B_i(m, \mathcal{K}'_m)$ **then**

    The new model outperforms the old one

    $m \leftarrow \mathcal{M}$

    $m_{new} \rightarrow \mathcal{M}$

    return `success`

  end

end

return `failure`

3.4.2. Active Learning

We consider the problem of active learning as a way to reduce the amount of data that needs to be annotated by a human expert. In an active learning setting, we assume that there's a way to rank the data based on their relevance to the training of a given model.

Thus, we can use this ranking to select the data that should be annotated to optimally train the model in a continual learning process. We use this idea to present a general approach to active learning as an adaptation tactic in the context of the system we propose and formalize it as follows:

Let $R(m, \mathcal{K}_s)$ be a ranking function that **ranks the samples associated with a model**. $R(m, \mathcal{K}_s)$ takes a model $m \in \mathcal{M}$ and a set of samples $\mathcal{K}_m \subset \mathcal{K}$ of data associated with m and returns an **ordered set** of the samples in $\mathcal{K}_m$, ranked according to their 'importance' (however that is defined) to the training of m.

The pseudocode below describes the process of ranking the data in $\mathcal{K}_m$ with R. The ranking is stored in $\mathcal{K}_m$ as annotations of the data made by m itself rather than a person (note that here we 'annotate' the samples with the ranking value, not the actual label).

Algorithm 2: ImportanceSampling(R, m, $\mathcal{K}_p$)

Input: ranking function R, model m in $\mathcal{M}$, pool of data samples $\mathcal{K}_p \subset \mathcal{K}$

Let $\mathcal{K}_{ranked} = (x_1, x_2, \ldots, x_n) = R(m, \mathcal{K}_s)$

Note $\mathcal{K}_{ranked}$ contains the same samples as $\mathcal{K}_p$ but ordered based on their 'importance' to the training of m

for $i \in \{1, \ldots, n\}$ **do**

- $x = \mathcal{K}'_{ranked}[i]$
- *The statements below are meant to depict direct modifications to $\mathcal{K}$*
- a_i = **new** *annotation*(*annotator* = m)
- p_i = **new** *property*(*name* = 'ranking', *value* = i)
- $a_i \Rightarrow properties = \{p_i\}$
- $x \Rightarrow annotations = \{a_i\} \cup (x \Rightarrow annotations)$
- *Note that this creates new object entities $a_i, p_i \in \mathcal{K}$*

end

Importance sampling allows the platform to prioritize the labeling of the samples that are most optimal to the learning process (i.e., the labeling interface can show the annotators the important samples so that they are labeled before any other in K_m). Then, with every new (important) sample we label, we can proceed with continual learning on the newly annotated data and repeat this process until there is no more data to annotate.

This is the concept behind active learning, which we formalize in the algorithm below:

Algorithm 3: ActiveLearning (R, m, $\mathcal{K}$)

Input: ranking function R, model $m \in \mathcal{M}$, knowledge database $\mathcal{K}$

Let $\mathcal{K}_{ranked} = \{x \in \mathcal{K} \mid (\text{'ranking'} \in (x \Rightarrow annotations \Rightarrow properties \Rightarrow property.name)) \wedge ((x \Rightarrow annotations \Rightarrow annotator) = m)\}$

Note $\mathcal{K}_{ranked}$ is the set of samples in $\mathcal{K}$ that have been ranked before for m

if $K_{ranked} = \emptyset$ **then**

We rank the samples associated with m

$\mathcal{K}_s = \{x \in \mathcal{K}_m \mid \{a \in \{x \Rightarrow annotations\} \mid (a \Rightarrow annotator) \notin \mathcal{M}\} = \emptyset\}$

Note $\mathcal{K}_s$ is the set of samples that are relevant to m and are not yet annotated by a human

if $\mathcal{K}_s = \emptyset$ **then**

There's no data to annotate

exit

end

Rank the samples in $\mathcal{K}_s$ and start over

`ImportanceSampling`(R, m, $\mathcal{K}_s$)

return `ActiveLearning`(R, m, $\mathcal{K}$)

end

$\mathcal{K}^+_{ranked} = \emptyset$

for $(x_i, \ldots x_n) \in \mathcal{K}_{ranked}$ **do**

Note we assume the samples in $\mathcal{K}'_{ranked}$ are sorted by their 'ranking' value

annotate x_i

$\mathcal{K}^+_{ranked} = \mathcal{K}^+_{ranked} \cup \{x_i\}$

if `ContinualLearning`(m, $\mathcal{K}^+_{ranked}$) = `success` **then**

start over

return `ActiveLearning`(R, m, $\mathcal{K}$)

end

end

Note we assume that with the **annotate** command, we assume that the annotation is done by a human that has domain knowledge about the data in $\mathcal{K}_m$. Thus, a delay between one iteration and the next of the **while** loop shown in the algorithm above is expected.

3.4.3. Other adaptive processes

We only implement experiments with the two adaptation tactics described above, but we can show that the system we propose can be extended to include other tactics by simply being selective on the benchmark functions and relevant data that we pass to the continual learning tactic, just as we did with active learning.

For example, to implement adversarial training as an adaptation tactic in this system, we can define a benchmark function that measures the robustness of the model to adversarial attacks (i.e. letting $B(m, \mathcal{K}_m)$ be a function that returns the accuracy of m on adversarial samples).

Then, we can pass to the continual learning process data samples that are adversarial to the model (which can be priorly tagged by annotators as such). Thus the whole process of adversarial training reduces to $\texttt{ContinualLearning}(m, \mathcal{K}_{adv})$ where $\mathcal{K}_{adv}$ is a set of adversarial samples to train m with (that are available in the knowledge database).

Furthermore, Knowledge distillation can be formalized by letting the training function $\texttt{Train}(m, \mathcal{K}_m)$ be a distillation process that uses the predictions of the current model m as a soft target for a smaller m_{new}. We can then use a benchmark function $B(m_{new}, \mathcal{K}_m)$ of how close the predictions of m_{new} are to those of the previous m.

4. METHODOLOGY

In the previous chapter, we made a thorough high-level definition of the functionalities that the platform we propose has, even formalizing each of the components that comprise the system and the adaptive tactics we have selected to implement in our solution.

This chapter aims to delve deeper into the methodology we employed to implement these functionalities and tactics for their application to TB detection. We'll discuss the specific technologies that we applied, the machine learning models used, the processing of the data used, training and evaluation processes, and the methodology we propose to evaluate the use of adaptation techniques with the data in this platform, the results from that evaluation are reported in the next chapter.

4.1. Tools Used

4.1.1. Programming Languages

The main programming language used in this work is **Python**, which is a high-level, general-purpose programming language that is widely used in the field of machine learning and data science due to its wide ecosystem of libraries and frameworks that facilitate the development of machine learning models and applications.

Additionally, we used **SQL**, a query language used to interact with relational databases, to interact with the Knowledge Database that we implemented (see section **??**). This included making queries to get data from the database and creating new objects and tables, among other database operations.

4.1.2. Frameworks and Libraries

We proceed by listing some of the most relevant frameworks and libraries we used to build the components and implement the experiments described in this work. These tools were selected based on their popularity, ease of use, and documentation and helped us speed up the development process significantly.

1. **PyTorch and TensorFlow pytorch**, **tensorflow2015** were the two frameworks used to develop, train, and/or evaluate all the deep learning models for this work. They are the most popular and widely used libraries for developing DL models. We used PyTorch for most of the experiments and models trained and TensorFlow for some of the ones that required the use of specific models that were only implemented in that framework.

2. **Numpy and Pandas numpy**, **pandas**: Numpy is a scientific library in Python that supports mathematical operations with multi-dimensional arrays and matrices. Pandas built on top of Numpy the easy data structures for easy data manipulation. We used both of these libraries extensively for data processing and writing augmentation tasks.

3. **PostgreSQL postgresql**: As we mentioned earlier, we used PostgreSQL to implement the database where we stored the data, annotations, and other information needed by the system. PostgreSQL is an open-source Object-Relational Database Management System (ORDBMS) that is very popular and widely used in the industry.

4. **Streamlit streamlit** is an open-source framework for building web applications in Python. We used it to design and implement the labeling interface that we used to annotate the data (see section **??**). The interface was designed to be simple, intuitive to use and responsive to work on mobile devices. Streamlit allowed us to easily integrate the interface with the rest of the system, making it a great choice for this task.

5. **MLFlow mlflow**, an open-source platform and Python framework for tracking ML models and experiments, was used as a means to implement our model repository. We used this platform to store our ML models and their associated metadata.

Other tools we worked with throughout the implementation of this project included **Docker docker**, a virtualization platform we used to package the different components of our system and ensure a consistent development environment, **FastAPI fastapi** for implementing the APIs that allowed communication between components, as well as a number of other tools for analysis, visualization, and image processing **matplotlib**, **scikit-learn**, **opencv**.

4.1.3. Hardware

All the methods were implemented and tested on a personal MacBook Pro (2021) with an M1 Pro chip, 16GB of RAM, and 1TB of SSD storage. Most of the ML training and evaluation experiments were also done with this device.

Additionally, we used the Google Colab platform [6] to get access to NVIDIA GPUs in the cloud, which was used to train the ML models that were too big (or too slow) to attempt to do so locally. In order to share code/data between the local and cloud environments, we used Google Drive [7] and GitHub [8].

We list some of the GPUs we had available in the Colab platform along with their VRAM capacity and performance in the table below [9]:

GPU	VRAM	FP32 Performance
T4	16GB	8 TFLOPs
V100	16GB	14 TFLOPs
A100	40GB	19.5 TFLOPs

Table 4.1. Hardware specifications of different Nvidia Tesla GPUs available in Google Colab

Note VRAM capacity is important because it limits the size of the models that can be trained on a given GPU and FP32 performance measures the number of floating-point operations that can be performed per second, which largely determines how long it takes to train a model..

For example, the DETR model we used in our experiments (see section **??**) requires at least 12GB of VRAM to be trained in its default configuration and up to 25GB of VRAM with the configurations that theoretically allow it to detect smaller objects better (DETR-DCN).

FP32 performance measures the number of floating-point operations that can be performed per second, which largely determines how long it takes to train a model. Since the cost of V100 and A100 GPUs was significantly higher than that of the T4s (which could sometimes be used for free in Colab), we only used them sparingly for the more demanding experiments.

[6] https://colab.research.google.com/
[7] https://www.google.com/intl/en-GB/drive/
[8] https://github.com
[9] See hhttps://resources.nvidia.com/l/en-us-gpu for more information

4.2. Data Source and Selection

The dataset used in this work to train the models and evaluate the performance of the proposed system was sourced from Kaggle, a popular platform for data science competitions and sharing datasets and ML notebooks **kaggle**.

The dataset [10] consists of 928 images of sputum smear microscopy along with the annotated bounding boxes at the regions where TB Bacilli is localized for a total of 3,734 ground-truth annotations. The original collection of this data was carried out through specialized software and hardware for collecting samples from sputum. A Gram stain was typically performed to identify the bacteria causing the infection **tbbacillus_kaggle_dataset**.

This data is of high relevance to our work and gives us the opportunity to train our model effectively on real-world data. Figure **??** shows some examples of the images in this dataset with their corresponding annotations.

To select the data used for training, validation, and testing of the models, we used the same methods as Visuña et al. **visuna_novel_2023** (the same dataset we used in this work), which only includes the images that were deemed more reliable and whose annotations seemed to be more accurate and consistent with domain experts.

With this, approximately only 30% of the total amount of images met the criteria for being selected. The rest of the images were discarded because they were either too blurry, had too many artifacts, or had annotations that were deemed too inconsistent (mainly due to missing annotations).

The final dataset used for training and testing the models is summarized in table **??**.

Subset	Number of Images	Number of Annotations
Training	202	1467
Validation	40	235
Test	66	351
Total	308	2053

Table 4.2. Training and Test Data

[10] https://www.kaggle.com/datasets/saife245/tuberculosis-image-datasets

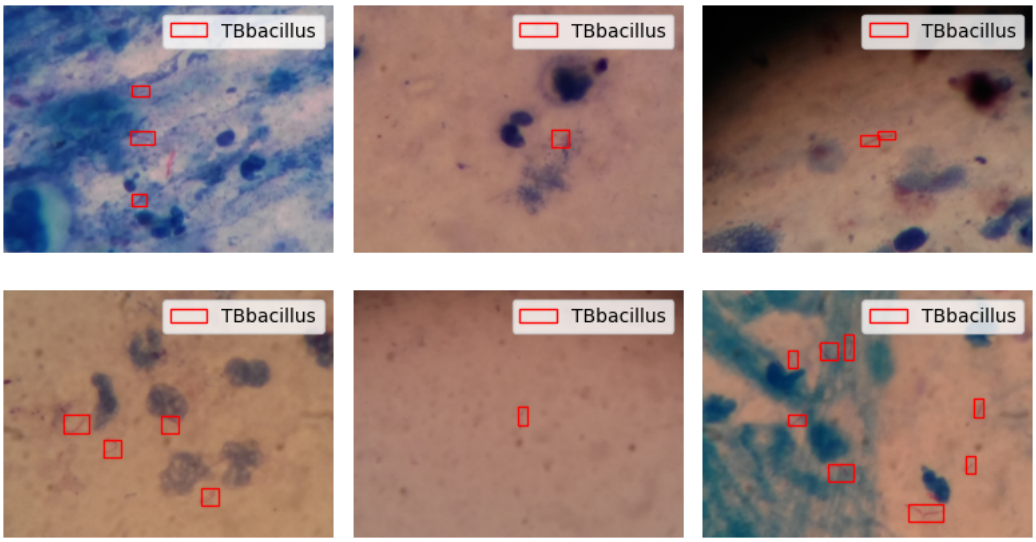

Fig. 4.1. Example of some smear-sputum microscopy images in the dataset. Boxes show the presence of TB Bacilli

4.3. ML model Training and Evaluation

The data mentioned above was used to train a baseline object detection model to detect regions in the images where TB bacilli is localized. The training and validation subsets were used during the model training process while the *test* subset was assumed to be unseen data and was used to evaluate the final performance of the model.

The idea is that this baseline model could be used to compare the performance of the system with and without the adaptive tactics we implemented. Other models were also trained using different data subsets and hyperparameter configurations depending on the technique being evaluated and compared with this one. In section **??**, we discuss the details of the experiments we conducted and the models we trained.

The deep learning model that we chose to train for this task was DETR (Detection Transformer) **carionEndtoEndObjectDetection2020**, an end-to-end computer vision model that was introduced in 2020 by researchers at Meta AI. [11]

DETR leverages the transformer architecture **vaswaniAttentionAllYou2017**, widely recognized for its prevalent use in natural language processing tasks, and extends its use to the CV domain. The model discards complex pipeline mechanisms such as non-maximum

[11] We originally considered using the same Nas-Net Mobile model used in **visuna_novel_2023**. As we've established, this model was trained on the same dataset we used. However, we found that the inference speed of this model was too slow for the workflows we wanted to evaluate, and its performance was not significantly better than that of DETR. We show the performance of this model compared to our model in chapter **??**.

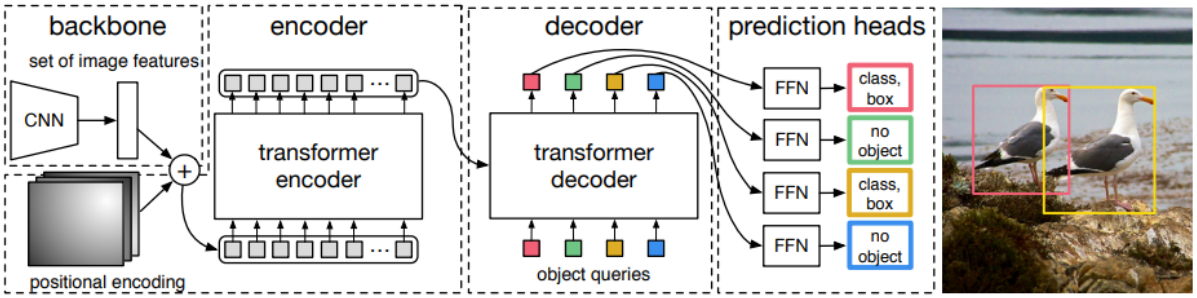

Fig. 4.2. Architecture of DETR from **carionEndtoEndObjectDetection2020**. The model first computes a set of image features from a CNN backbone and a set of positional encodings that are used to encode the spatial information of the image. Then, the model uses a transformer encoder to process the image features and positional encodings and a transformer decoder to generate a set of object queries that are used to predict the bounding boxes and class labels of the objects in the image.

suppression used in object detectors like Faster R-CNN (see section **??** in the state-of-the-art chapter) and operates end-to-end, treating object detection as a direct set prediction problem.

The specific version of DETR that we used was one open-sourced by Facebook Research through Github that is officially available in the Pytorch model Hub [12]. This model uses a ResNet-50 model as backbone **heSpatialPyramidPooling2014** and was pre-trained on the COCO dataset **linMicrosoftCOCOCommon2015**.

4.3.1. Training Process

Thus, we proceeded to fine-tune this pre-trained model on our selected dataset. Our training script was adapted from the one included in their official repository but modified to directly pull the data from our database that was specifically tagged to be used for training. We also modified the script to use the hyperparameters that we selected.

Since the original code was written to train the model with the COCO dataset, we had to adapt it to work with our data. This included creating our own data loading mechanisms that grab the relevant images from the database and apply the necessary transformations, which for DETR included resizing the images to 800x1333 pixels, normalizing them to [0, 1], and converting them to PyTorch tensors in the same output format as COCO.

While the original paper includes the training-time data augmentations of randomly cropping a part of the image and applying color jittering (randomly changing the bright-

[12] https://github.com/facebookresearch/detr

ness, contrast, saturation, and hue of the image), we realized these augmentations wouldn't be suitable to our data.

The reason for that is that - unlike the objects in COCO - the individual bacilli in our images were very small and spread apart. Because of that, the cropping technique would more often than not crop out and leave only a part of the image with no bacilli in it, making it impossible for the model to learn to detect them.

We also decided to leave out the color jittering augmentation. According to the relevant literature (**osman_tuberculosis_2011**), the color of the bacilli is a very important feature for their detection, and we didn't want to risk altering it.

Thus, the only augmentation we used from the original paper was doing a random horizontal flip to the image with a probability of 0.5 and resizing the image to different aspect ratios.

The training process was done in batches of 2 images, The model would detect 100 queries (i.e., objects) per image - the majority of which would be classified as background. The model was trained for 25 epochs with a learning rate of 0.0001 and a weight decay of 0.0001. Additionally, the learning rate was decayed by a factor of 0.1 at epochs 15 and 20 to improve the chances of convergence.

The base model was trained on a single NVIDIA Tesla T4 GPU. Each full training run would usually take around 40 minutes with the entire dataset, and convergence generally occurred at around epoch 20.

We also trained a second version of the model with the same hyperparameters but added a dilation convolutional layer to the CNN backbone to increase its receptive field, a configuration (DETR-DC5) that the authors of the paper claimed improved the performance of DETR on small objects. This modification, however, increases the computational cost of the model by a factor of 2 **carionEndtoEndObjectDetection2020**.

We trained this version on a single A100 GPU for 30 epochs, which took 15 minutes to fully train. We found that this version of the model initially achieves significantly lower validation losses but eventually converges to similar values as the regular one.

Since the performance difference was not significant enough to justify the extra cost, we decided to use the first version for the rest of the experiments. Image **??** shows the

loss and class error of the two models throughout the training process. Figure **??** shows some examples of the predictions made by the base model on the test set.

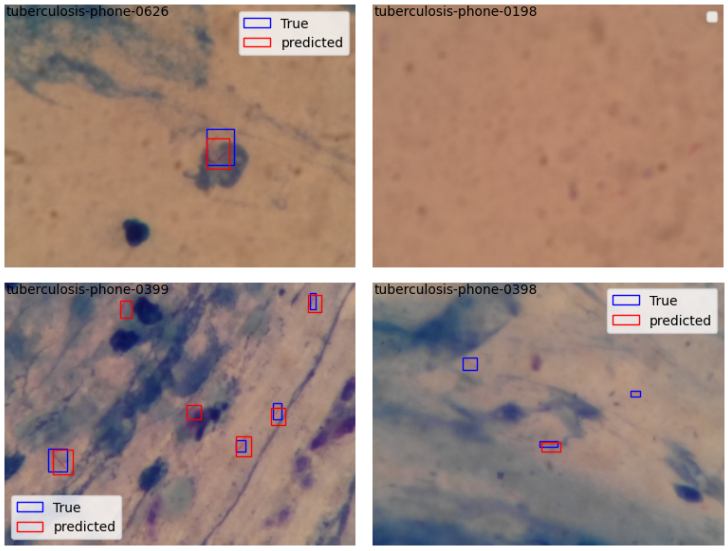

Fig. 4.3. Example of the predictions made by the DETR model on the test set. The blue boxes represent the ground truth annotations, while the red ones represent model predictions.

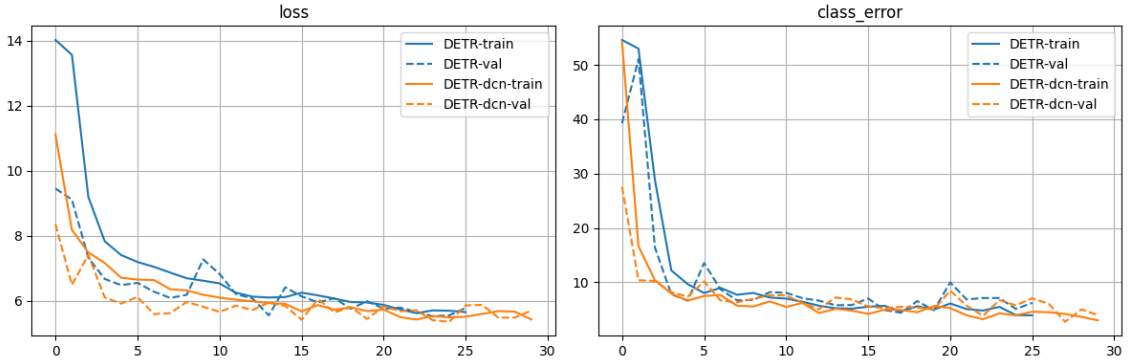

Fig. 4.4. Training and validation Loss of the DETR models and class errors after training each epoch. Class error refers to the accumulated error of classifying boxes as bacilli or background. DETR-DC5 is the model with a dilation layer.

4.3.2. Performance Metrics

The metrics we report in the evaluation process are some of the common ones to report performance for object detection tasks: Average Precision (AP) and Average Recall (AR) at different intersections over union (IoU) thresholds. Which are also the ones used for COCO and in the DETR paper.

The IoU is a measure of the overlap between bounding boxes. It is calculated as the area of overlap between the predicted bounding box and the ground truth bounding box

divided by the area of the union of the two bounding boxes. Thus, when a single detection surpasses a certain IoU threshold, we consider it to be a true positive.

Precision refers to the proportion of correct positive predictions out of the *total* predictions, while recall refers to the proportion of correct predictions out of *actual* positives.

Generally, the IoU threshold is set to 0.5, meaning that if the IoU between the predicted and ground truth bounding boxes is greater than this number, the prediction is considered correct. We call these metrics AP@50 and AR@50, respectively. But while these two are the most common metrics to evaluate object detection models, we also report the performance of the models at an IoU threshold of 0.3 (AP@30 and AR@30).

The reason behind the 0.3 threshold is that, for the purpose of TB diagnosis, we considered that the most important aspect of the task was to identify the general presence and location of the bacilli and not necessarily their exact boundaries.

Additionally, we noted some of the ground truth boxes in our data would either only cover a small portion of the bacilli or overestimate their size while our model would often predict boxes that were more accurate in size and location. Thus, a lower IoU threshold was considered to be a more accurate measure of the model's performance.

These metrics are calculated for each image in the test set and then averaged to obtain a single performance score for the model across the entire dataset. This assesses how well our model has learned to detect TB bacilli.

4.4. Experiments with Continual Adaptation

In addition to the initial model training and evaluation, we also conducted a set of experiments targeting the continual adaptation of our model to reveal how it can be improved by learning continuously from new data. These experiments were designed to test the performance of the model after being adapted using **continual and active learning techniques**.

For all experiments, we took a portion of the training dataset and reserved it for continual learning, then, each experiment used a successive fraction from that dataset to incrementally train the previous model.

The way we approached this was by first fine-tuning a new DETR model again with the same characteristics as the one we trained as the baseline, but this time only with a

random 50% of the original training data, while the other 50% we reserved as our **holdout data**. We used this model as a base to test the adaptation tactics implemented.

To simulate a continual learning (CL) process, this base model was incrementally trained on holdout data, adding a fixed number of images from this set at a time. The model was then evaluated on the test set after each incremental training step to assess its performance. This process was repeated until all the holdout data was used.

The scheduled fractions of the holdout data that we used for each incremental training step were 25%, 25%, 30%, and 20%, meaning that after our base model is trained with half of the training data, a *random* 25% of the holdout set would be used for training in the first step of the experiment, then another 25% in the second step, then 30%, and so on until all the data was included (note the values in the schedule sum to 100%).

We characterize this CL experiments as follows:

- Two continual learning tactics are tested per each step: The first is to retrain the DETR model from scratch (from the pre-trained version prior to adapting it with our data) using 50% of the original training data plus the corresponding fraction of the holdout set, the second tactic was to fine-tune the base model incrementally with the corresponding fraction of the holdout set excluding any other data in the training process.

- Active learning (AL) experiments were carried out following the same procedure as above but instead of grabbing random instances from the holdout data, we queried the most 'important' samples to train the model with as determined by the active learning strategy.

- This process was repeated three times using different random seeds and averaged the results for a more robust evaluation.

These continual learning experiments simulated a scenario whereby the model is continually exposed to new data over time, either passively receiving it or actively querying it (AL). Each experiment step is meant to simulate a training round where a portion of the data available that is associated with the model has been annotated and is ready to be used for training the model (which follows from the design of the platform we proposed in the previous chapter).

As for the active learning tactic, we used margin sampling, a technique that selects the samples with the lowest margin of confidence between positive and negative classes to train the model with.

To implement it, we wrote a function that takes an image from our tuberculosis dataset and makes a forward pass of the DETR model on it. After the forward pass, we obtain the class scores of each bounding box predicted by DETR and normalize them by applying a *softmax function*, which takes the unnormalized log scores from the model (class logits) and outputs a vector of probabilities that sum to 1. The formula for the softmax function is shown in equation **??**.

We then compute the margin of confidence between the two classes (bacilli and background) by subtracting the probability of the background class from the probability of the bacilli class and ranking the samples whose margin of confidence is the smallest.

Then, the process of obtaining the margin of confidence (MoC) is shown in equation **??**. Where we let $p_{N\times 2}$ be a vector with the class scores of each of the **N** bounding boxes predicted by DETR, with $p_{i,1}$ being the probability of the background class, and $p_{i,1}$ the probability of the background class.

$$\text{Softmax}(p_{N\times 2}) = \frac{e^{p_{N\times 2}}}{\sum_{i=1}^{N} e^{p_{i,1}}} \tag{4.1}$$

$$\text{MoC}(p_{N\times 2}) = \frac{1}{N}\sum_{i=1}^{N} |p_{i,1} - p_{i,2}| \tag{4.2}$$

After obtaining the margin of confidence for each box, we would take the values of the 20 smallest margins in the image (out of 100 bounding boxes due to the number of queries selected to train DETR) and compute the average of their values, which is what we would use to rank the images from the holdout set from most to least confidence. We would then take the top n images from that ranking (n being the number of images the holdout set was split into for that step) and use them to train the model.

Note this process fits perfectly into the algorithm for active learning we designed in the previous chapter (see algorithm **??**).

Another sampling strategy we attempted to implement was one involving expected model change sampling. To do this we built a DNN that would take the CNN backbone features of the relevant images from our DETR model and target the model's actual loss function, similarly attempted by **yooLearningLossActive2019**. We would then use it to predict the loss of the model if we were to add a new sample to the training set and rank the samples according to that. However, we were unable to get this to work properly and decided to leave it out of the experiments.

5. RESULTS

In this chapter, we present a thorough overview of the results we obtained from our evaluation, including the training and validation processes carried out throughout our study and a comparative analysis of the system's performance across different experimental conditions.

This chapter aims to provide an understanding of how well the system proposed performs for its designated task, as well as its effectiveness in learning continually from new data.

Note the focus of our evaluation was not only to see if the system could correctly identify the presence of TB Bacilli but also to evaluate how well the iterative training and variations in the active continual learning strategies could affect its performance, as per the objectives of this thesis.

5.1. Baseline Results

We start with the results of training our DETR model with the data selected data we obtained from our Tuberculosis Image Dataset. Like mentioned in the previous chapter we report the results of the training process in terms of the Average Precision (AP), Average Recall (AR) at IoU thresholds of 0.3 and 0.5 to assess how good the model is at detecting the presence of TB Bacilli in the images.

For a good comparison of our results, we also include the performance of our model with the Mobile NasNet developed by Visuña et al, which was trained to detect bacilli with the same data but using a different model architecture and training process.

As a reference, we also include the performance of the DETR model trained only with half of the training data, which we will refer to as the 'base' model. This model will be used as a reference to observe if other models can improve by applying the continual learning tactics we propose.

We consider it a success if a tactic can get close (or better) to the results of the baseline while using significantly *less* samples to train.

Model	AP@50	**AP@30**	AR@50	AR@30	Avg IoU
DETR (Full)	0.824	**0.874**	0.731	0.776	0.579
Mobile NasNet (Visuña et al.)	0.451	0.801	0.450	0.781	0.47
DETR (50%)	0.731	0.798	0.687	0.735	0.543

Table 5.1. Baseline results of different models trained with the Tuberculosis Image Dataset

Note that the approach used to train the Mobile NasNet above uses tiling and stitching to select the final bounding box predictions, which results in less precise bounding boxes, especially when multiple bacilli are together but still shows a good ability to generally localize them in the images. This explains the remarkable difference between its AP@50 and AP@50 scores compared to its @30 counterparts.

In their paper, the only criteria they used to assess a positive performance was if the center of the ground truth bounding box was within the predicted bounding box, based on that criteria, they obtained accuracy and recall scores over 0.9.

5.2. Results of the Adaptation Tactics

Besides the baseline models, we carried out the training process of 12 additional Bacilli Detection models under different data selection and training conditions to assess the effectiveness of the continual learning tactics we proposed in this thesis.

As mentioned in the previous chapter, we designed a series of experiments where half of the images in the training set were put aside as holdout data to progressively train a base model, DETR 50%, with it. At each *step* of the 'holdout schedule', we used a successive fraction from that data to incrementally train the model.

Just like with the baseline results, we evaluated the performance of these models based on the metrics of average precision and recall at IoU thresholds of 0.3 and 0.5, we also report the average Intersection over Union (Avg IoU) of the bounding boxes predicted by each model.

The results of each experiment on these metrics are shown in Table **??**, where we also include the performance of the models mentioned in **??** for comparison (DETR, DETR 50%).

Experiment	AP@50	AP@30	AR@50	AR@30	Avg IoU
DETR (Full)	**0.824**	**0.874**	**0.731**	**0.776**	**0.579**
RS - (Retr) step:1+2+3	0.766	0.873	0.690	0.774	0.549
AL - (Retr) step:1	0.794	0.869	0.574	0.617	0.438
AL - step:1	0.806	0.869	0.695	0.744	0.552
RS - step:3	0.803	0.865	0.541	0.573	0.458
AL - (Retr) step:1+2+3	0.446	0.733	0.726	0.399	0.401
AL - step:2	0.713	0.847	0.573	0.696	0.468
RS - step:4	0.761	0.842	0.704	0.773	0.540
AL - step:3	0.729	0.840	0.575	0.656	0.465
AL - step:4	0.718	0.824	0.691	0.788	0.540
RS - step:1	0.727	0.818	0.703	0.784	0.548
RS - (Retr) step:1	0.713	0.807	0.728	0.813	0.530
DETR (50%)	**0.731**	**0.798**	**0.687**	**0.735**	**0.543**
RS - step:2	0.595	0.641	0.239	0.260	0.222

Table 5.2. Evaluation metrics for the trained model. Ranked by AP@30

The experiments that employed the active learning tactic with our uncertainty sampling technique were marked with the prefix ‘AL’, while the ones that used random sampling were marked with ‘RS’. ‘Retr’ indicates that the model was *retrained* from scratch using the initial weights of DETR (prior to domain adapting it to our task) and including the other 50% of the data subset instead of fine-tuning the base model (DETR 50%).

Furthermore, experiments without the ‘Retr’ prefix were fine-tuned from the previous step of the holdout schedule - starting with the base model (DETR 50%) for those in step 1. So, for example, ‘AL - step:2’ was trained by fine-tuning the model trained previously in ‘AL - step:1’, which was itself a fine-tuned version of the base model (DETR 50%).

The numbers after the ‘step:’ suffix indicate the index of the corresponding fraction of the holdout data that was used to train the corresponding model. We use the fraction schedule that was mentioned in the methodology chapter: 25%, 25%, 30%, and 20% of the holdout data respectively for each step.

Additionally, since retraining from scratch was much more computationally expensive than fine-tuning the base model, we only retrained on the first and second-to-last steps of the holdout schedule (the last step would be omitted anyway since it implies retraining the full model).

5.3. Analysis of the results

The first insight we can draw from these results is that, as expected, the models trained with larger subsets of the data tended to perform better than the ones trained with fewer samples: with only one exception, all models performed better than the base DETR that was only trained with 50% of the available samples.

The most interesting results are the ones obtained from the models that used the active learning tactic with our uncertainty sampling technique. The models that included the first batch (step 1) of samples selected with this technique - the 25% of the holdout data that was considered the most important to the model - achieved higher AP@50 scores than the random sampling approach that was retrained with 80% of all available data.

What this insight shows is that by only *fine tuning* our base model on the 25 (25% out of 101 images in the holdout set) most important samples, we were able to achieve a better or comparative performance than a model that had to be retrained *from scratch* using seven times more data and including 55 more samples from the holdout set.

This is a very promising result that shows the potential of continual learning approaches to improve the performance of a model with a fraction of the data that would otherwise be required to manually annotate and train. Furthermore, it also shows the effectiveness of our uncertainty sampling technique to select samples that are useful for the model to learn from.

Another less intuitive observation we could draw from these results related with the models that used the active learning approach, is that those models that included the first batch of samples selected with our uncertainty sampling technique (step: 1), whether retrained from scratch or fine-tuned, achieved higher scores than the ones that fine-tuned the base model with samples that came later in the holdout schedule.

Not only that but it seems that with this strategy the more advanced we get in the holdout schedule, the more the performance degrades. This is especially evident when we compare the model that was fine-tuned in step 3 using the random sampling approach ('RS - step:3') with the one that was fine-tuned using the active learning approach in step 4 ('AL - step:4'). The former achieved a significantly better AP@30 score, despite not having seen the remaining 20% of the holdout data that the latter did.

This result can be attributed to the common issue of *catastrophic forgetting* that is inherent to continual learning approaches and that we discussed extensively in chapter **??**.

As our models adapted to the new, less 'important' data, they gradually forgot the knowledge they had acquired from the previous samples that allowed them to achieve a better performance. Incremental (fine-tuning) training processes, as opposed to those that retrain from scratch, are less robust to this issue (but are also much less computationally expensive).

This we can consider a tradeoff of sorts for active learning techniques - even though we were able to identify the most important samples for our pool of unseen data and were able to harness them effectively at the beginning of the continual learning process, subsequent queries from the same pool of data only yielded diminishing returns when trained in an incremental (fine-tuning) fashion.

The models that were specialized (fine-tuned) with the most important samples first were able to achieve a better performance than the ones that were trained with the same samples but in a different order. But, likewise, the models specialized on the *least* important samples were the ones that achieved the worst performance.

Even then, based on this information we can confirm even further the effectiveness of our uncertainty sampling technique in selecting the most important samples for the model to learn from, as well as the importance of the order in which the samples are presented to the model.

In the conclusions chapter we will discuss this issue in more detail and propose some ideas to address it in future work, as well as consider other potential research directions that we consider to be interesting in that area.

Finally, with regards to the models that used the random sampling approach, we can see that they achieved a better performance than the base model (DETR 50%) but were outperformed by the models that used the active learning approach on the same data.

6. BUDGET

Before presenting our conclusions to this work, we include an estimate of the costs associated with the realization of this work in an attempt to provide a more complete picture of the project.

6.1. Cost of Human Resources

We start with the costs of human resources associated, the student author of the work and the advisor and co-advisor of the thesis are considered to be the main contributors to the project. The student author is considered to have worked on the project in a part-time to full-time basis for the duration of the work, while the advisor and co-advisor contributed a few hours per week.

The salary per hour of each contributor was estimated to be €18, €30, and €60, respectively, to reflect the estimated salaries of a junior researcher, senior researcher, and university professor.

By analyzing the time planning of the project throughout its duration , the project is estimated to have lasted approximately eight months. During this time, the author contributed an estimated total of 530 hours, while the advisor and co-advisor an estimate of 25 and 35 hours, respectively.

include Gannt chart?

Table **??** shows a summary of the estimated costs of human resources, giving a **total budget of €13140.00** associated with the labor costs of this work.

	# of Hours	Salary per hour (€)	Total Salary (€)
Author	530	18	9540
Co-Advisor	35	30	1050
Advisor	25	60	1500
		Total Cost:	**€12090.00**

Table 6.1. Estimated Costs of Human Resources

6.2. Software/Cloud Expenses and Material Costs

In terms of non-human resource costs, the main relevant ones are those associated with the material costs utilized throughout the work. This is because we mainly used open-source software tools (Python, LaTeX, PyTorch etc.) in our work. These tools are freely available or under permissive licenses. Thus, no costs are associated with them.

Google Colab was the only tool used that is not open-source. Because of the demanding nature of some of the experiments we tried with, we bought a total of 10$ worth of Google Cloud credits to use the service. This is the only cost associated with **software and licenses** in this work.

In terms of material costs, we consider the following:

- The personal computer from which the thesis was written and experiments designed and executed. The device consisted of a 14" Apple Macbook Pro (2021) valued at €2200 at the time of writing **apple_comprar_2022**. With a 3-year depreciation of €733.33 divided by the duration of the project, this would add €489 to the material costs associated with this work.
- Other equipment, lab materials (e.g., paper, pens, etc.), and other miscellaneous costs are considered negligible and are not included in the budget.

	Software & Licenses	Personal Computer	Miscellaneous	**Total**
Cost (€)	10	489	0	**€499**

Table 6.2. Estimated Material Costs

Thus, the total material costs associated with this work are estimated to be €499.00. If we add that to the cost of human resources, we can calculate a **total budget of €12,579.00** associated with its realization.

7. CONCLUSIONS

The ideas presented in this thesis were driven by the interest and developing more efficient intelligent systems that can adapt to the changing nature of the data they are trained on and research about the problems that arise when trying to achieve this goal and how they can be addressed in a machine learning context.

We applied these concepts to the real-world problem of detecting tuberculosis from sputum smear microscopy images, a task that is challenging due to the scarcity (and reliability) of data available for training and the high cost of annotating data samples.

7.1. Summary of Results

Within this context, we can summarize the main contributions of this thesis as follows:

- We developed a web-based platform for the annotation of sputum smear microscopy images, which includes a frontend for annotators to label images and a backend for storing and managing the annotations. The platform allows its users to interact with ML models available in the system and use them to assist in the annotation process.

- Furthermore, we formalized all the components of this platform, making it easy to extend and adapt to other tasks and problems. We put special interest into understanding the design principles that we can draw from the self-adaptive systems literature and how we could apply them to the design of our platform.

- We took a transformer-based neural network model, initially trained on a large-scale dataset of images for object detection, and applied domain adaptation techniques to fine-tune it to the task of detecting tuberculosis from sputum smear microscopy images. This model achieved a high predictable performance compared to baseline models trained for the same task.

- We implemented an active learning framework that uses the model's uncertainty to select the most informative samples for annotation, which we used to train the model with a fraction of the data that would otherwise be required to train it from scratch and achieved performance comparable to one trained with 40% more data.

Furthermore, we make a thorough review of the state-of-the-art in the fields of computer vision, continual learning, active learning, and domain adaptation, among other machine-learning-related topics, to provide a solid theoretical foundation for the ideas presented in this thesis. We focus our review on how these concepts can be applied to healthcare issues and the specific challenges that arise when doing so.

7.2. Objectives Achieved

Through our experimental setup, we were able to assess how well a continual learning process can improve the performance of an ML model over time and do so while paying attention to how samples should be more efficiently collected (among other aspects). Our results directly demonstrate the adaptive capabilities of the system proposed, showing its ability to mature and refine the performance or models in response to new data.

By demonstrating the effectiveness of these tactics, we consider that we have achieved our main goal of devising a reliable and self-improving system capable of continuously enhancing the performance of ML models for real and impactful use cases, maximizing its usefulness to methods for the detection of tuberculosis.

7.3. Limitations of the System

The main obvious limitation of our system comes directly from our experimental results, which show how active learning can be a double sword when it comes to continually fine-tuning a model using a static pool of data. In each training round (after the most important samples have been identified), the model is fine-tuned with progressively less informative samples, which can lead to a decrease in performance over time (as we observed in our experiments).

With the random sampling approach, one can expect a more consistent increase in performance as the model is trained with more data since the model is likely being exposed to a more diverse set of samples that are not necessarily the most - or least - important ones for the model to learn from, however, this tactic will be less sample efficient than the active learning approach.

Thus, perhaps more of a hybrid approach is needed, where the model is trained with a combination of random and actively sampled data, which is something we did not have time to explore in this thesis.

Another limitation of the system we identified, especially within the context of this project, is the lack of explainability of the methods we used. While we can show improvements in performance metrics compare them to other models, and get an idea of why certain samples are better for training, it is very difficult to explain the model's behavior in a way that is (1) easy to understand for non-experts and (2) reliable and consistent across different models and datasets, which is a problem when trying to apply them to the healthcare industry.

Methods such as *data attribution*[13] can be promising in this area of research that can be well applied to the healthcare industry since it allows to trace the contribution of each data sample to the model's predictions, which can be very informative about the model's behavior.

7.4. Future Work

We continue this chapter by discussing some of the most immediate future work that can be done to improve the system we developed and expand on the ideas presented here.

7.4.1. Implementing new adaptation strategies to the system

By far, the most interesting future work we envision for the system we developed is to implement new adaptation strategies that can be used to test the continual improvement of ML models.

In section **??** we introduced concepts like knowledge distillation and adversarial training and formalized how they could be integrated into the system we propose in **??** but either because of time constraints, resource constraints (e.g., lack of access to a GPU), or simply because it did not fit well with the scope of this thesis, we could not implement or experiment with them in our system.

[13]Because data attribution has also proven useful for sample selection in active learning, we did some research about the state-of-the-art in this area to see which of them were available and how they could be applied to our system (but did not have time to implement them). Some promising methods are *TRAK* (**park_trak_2023**) and *influence selection* (**liu_influence_2021**).

However, with the framework we proposed, and the platform we built around it, we consider it should be relatively easy to implement new adaptation strategies and test them. This is the main reason why we put so much effort into formalizing the system and making it easy to extend and adapt to other tasks and problems.

For these reasons, we consider it to be in the most immediate future in terms of keeping the development of the ideas presented in this work further.

7.4.2. Potential to run the system as a federated learning platform

Another interesting future work direction we envision - which could be enough to be the subject of a whole thesis on its own - is the idea for the system we developed to run as a *federated learning* platform.

Federated learning is the concept of training a model using data from multiple sources without having to share the data itself. This is achieved by training the model on each source separately and combining the results to obtain a final model.

Such a system could be based on works like Joshi et al.'s (2022) **joshi_federated_2022**, which demonstrated the benefits of using a federated learning platform to train machine learning models for healthcare uses as a way to protect the privacy of patients and give more utility to the highly-valuable data that is collected in that industry.

As a federated system, the components we proposed in our design would be distributed across multiple clients (or research units), each with its own data. Each client would have its own instance of the platform, which would be connected to a central server that would coordinate the training of the models and the sharing of the trained weights.

The motivation for this comes from both suggestions made by stakeholders in this project, who expressed interest in the idea of a federated learning system, and personal experience working with healthcare data, where it is challenging to convince partners (even within the same organization) to share their data.

Figure **??** illustrates the concept of a federated learning system:

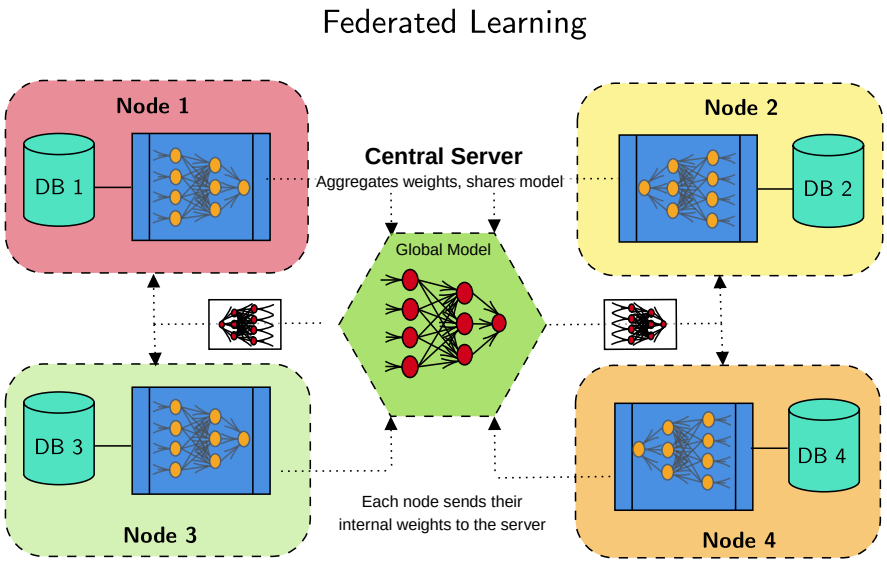

Fig. 7.1. Illustration of a Federated Learning System

7.4.3. Comments on Further Research Directions

In addition to the ideas presented in this section for future work, we also made a relatively lengthy discussion of other topics tangential to the ideas dealt with in this thesis that we consider could be potential for further research.

Unlike above, where we focused on ways to address limitations of the proposed system or ways to improve it in the immediate future, this discussion takes a broader view, focusing on ideas from emerging fields of research that we consider could be very promising in the future.

As to not distract from the content in this chapter - that is much more relevant to the actual work done - we present this discussion in the appendix of this thesis (**??**).

7.5. Final Remarks

Through this thesis, we have presented a system that can be used to continually improve the performance of ML models over time, which we applied to the real-world problem of detecting tuberculosis from sputum smear microscopy images.

The initial idea to work on the concepts that we presented comes from a personal interest in the field of machine learning and how it can be more efficiently applied to

real-world systems.

Through the development of this thesis, it also came a necessity - and deep interest - to understand the challenges that arise when trying to apply these concepts to the healthcare industry, and the importance that well-designed and reliable systems have in this field.

We consider that the ideas presented in this thesis are a step in the right direction toward achieving this goal, and we hope that they can be useful to other researchers and practitioners in the field.

REGULATORY FRAMEWORK

The material written in this thesis is published under a Creative Commons license CC BY-NC. The work here can be shared and distributed for non-commercial purposes as long as attribution is properly credited. Adaptations of the work are also permitted as long as they are shared under the same license. The sole copyright and intellectual property belongs to the author.

Similarly, the software developed for this work is openly available [14] and licensed under an Apache License 2.0, which grants permission to the use, distribution, and modification of the code for commercial and non-commercial purposes (restrictions apply, see license for details).

add url to github repo

Any desire to use this work or any material derived from it for commercial and/or monetary purposes should be communicated to the author and made with explicit written permission. Finally, note that the author is not liable for any direct or indirect consequential damages of any kind that arise from the use of any material in this work or from any derivatives of it in which the author is not directly involved.

Possible references to current or possible future legislation/regulations about the use of AI for healthcare applications, healthcare data, or other related topics (e.g., GDPR, HIPAA, EU AI Act, etc.)

Ethical Considerations

The EU published in 2019 their 'Ethic guidelines for trustworthy AI' **noauthor_ethics_2019**, which are based on seven key requirements that AI systems should meet in order to be considered trustworthy. These requirements are human agency and oversight, technical robustness and safety, privacy and data governance, transparency, diversity with regard to non-discrimination and fairness, environmental and societal well-being, and accountability.

Any AI system that is developed and deployed should meet these requirements, and

[14] All code developed in this work can be found under the following url: https://github.com/simonsanvil/..., for any questions, concerns, or comments, please contact the author at simonsviloria@gmail.com

this work is no exception . . .

Furthermore, the more recent EU AI Act **eu_aiact_2023**. . .

SOCIO-ECONOMIC ENVIRONMENT

Budget

The estimated costs of the realization of this project include those related to the human labor and material costs associated with it.

In terms of human resources, the student author of the work, and the advisor and co-advisor of the thesis are considered as the main contributors to the project. The student author is considered to have worked part-time on the project for the duration of the work, while the advisor and co-advisor contributed a few hours per week. The salary per hour of each contributor is estimated to be €18, €30, and €60, respectively, to reflect the estimated salaries of a junior researcher, senior researcher, and university professor.

include Gannt chart?

By analyzing the time planning of the project throughout its duration , the project is estimated to have lasted approximately eight months. During this time, the author contributed an estimated total of 530 hours, while the advisor and co-advisor contributed 50 and 35 hours, respectively.

Table **??** shows a summary of the estimated costs of human resources. Thus, we calculate a **total budget of €13140.00** associated with the labor costs of this work.

	# of Hours	Salary per hour (€)	Total Salary (€)
Author	530	18	9540
Co-Advisor	35	60	2100
Advisor	50	30	1500
		Total Cost:	**€13140.00**

Table 7.1. Estimated Costs of Human Resources

In terms of non-human resource costs, the only relevant ones are those associated with the material costs utilized throughout the work. This is because only open-source software tools (Python, LaTeX, Google Colab, etc.) were used. These tools are freely available or under permissive licenses, thus, no costs are associated with them.

In terms of material costs, we consider the following:

- The personal computer from which the thesis was written and experiments designed and executed. The device consisted of a 14" Apple Macbook Pro (2021) valued at €2200 at the time of writing **apple_comprar_2022**. With a 3-year depreciation of €733.33 divided by the duration of the project, this would add €489 to the material costs associated with this work.
- Other equipment, lab materials (e.g., paper, pens, etc.), and other miscellaneous costs are considered negligible and are not included in the budget.

	Software & Licenses	Personal Computer	Miscellaneous	**Total**
Cost (€)	0	489	0	**€489**

Table 7.2. Estimated Material Costs

Thus, the total material costs associated with this work are estimated to be €489.00. If we add that to the cost of human resources, we can calculate a **total budget of €13,629.00** associated with its realization.

Socio-Economic Impact

APPENDIX A

Entity Mapping Diagram of the Knowledge Database

This appendix contains the Entity Mapping (EM) Diagram of the PostgreSQL Database that we implemented in our solution system based on the design of $\mathcal{K}$ as described in chapter **??**. An EM Diagram is a visual representation of the schema of a relational database, which is composed of a set of tables and their relationships.

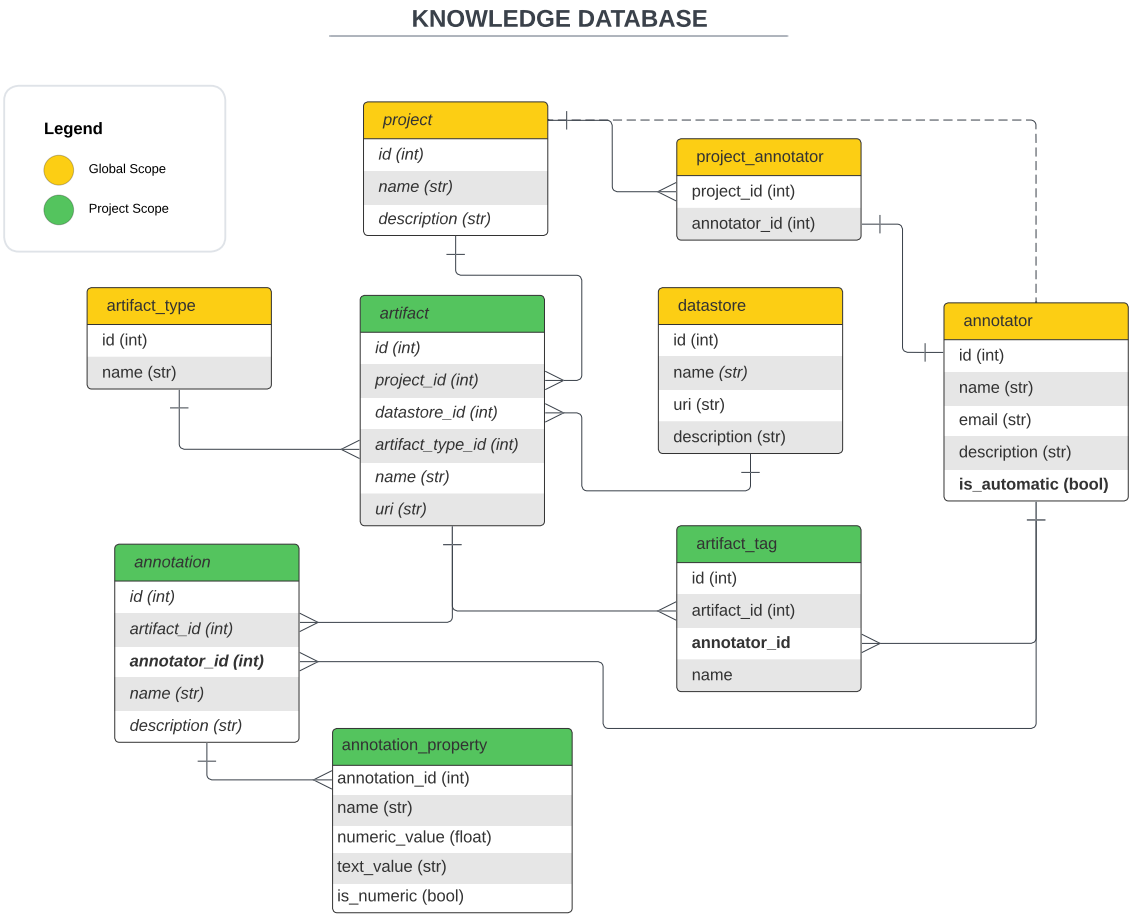

Fig. 7.2. Entity Mapping (EM) Diagram of the Knowledge Database ($\mathcal{K}$)

The tables in the 'global' scope (shown in yellow) like *project*, *annotator* and so on store information that is not exclusive to any single task, or that is shared between them. Meanwhile, the tables in the 'project' scope (shown in green) like *annotation* and *artifact*

store information that is specific to an application.

Then, assuming the schema above, an example of a query like the one shown in section **??** but using SQL syntax would be:

```
SELECT * FROM annotation
JOIN annotation_property
ON annotation.id = annotation_properties.annotation_id
WHERE annotation_properties.name = 'label';
```

In the notation used in chapter **??**, we assume this query to be equivalent to:

$$\mathcal{A}_{labels} = \{annotation \in \mathcal{K} \mid \text{‘label’} \in \{annotation \Rightarrow properties \Rightarrow property.name\}\}$$

Which returns all annotations in $\mathcal{K}$ that have a property with the name ‘label’.

APPENDIX B

The following are some of the recent and upcoming research directions in machine learning that we consider relevant to the work presented in this thesis.

The idea is that these directions could used as a starting point for future research (i.e., a Ph.D. thesis) on the topics presented, either as a continuation of some of the work in this thesis or as a completely new approach to the problem. We make no claims about the feasibility of these ideas but rather present why I consider them interesting.

7.5.1. Scalable Adaptability through Mixtures of Experts

1.5 pág

Mixture of experts (MoE) systems are a type of ensemble model that combines the predictions of multiple models to obtain a final prediction. The difference between MoEs and other ensemble models is that the predictions of the individual 'experts' are combined using a **gating/routing function** - typically a neural network - that adapts to the given data point and dynamically determines the weight of each model in the final prediction **chen_towards_2022**.

MoEs are really powerful systems. They have been shown to be able to learn complex multimodal distributions and have been used in a wide variety of applications, including object detection, language modeling, machine translation, and even multiomics **hwang_tutel_2023**, **mustafa_multimodal_2022**, **shazeer_outrageously_2017**, **minoura_scmm_2021**.

An advantage of a MoE is that each expert model can be deployed independently, allowing for a more flexible, modular system capable of being distributed and data-parallelized among different hardware resources.

Recently, MoEs have become popular due to the fact that the computational sparsity of these systems can be used to scale DNN models to outrageous amounts of parameters at a constant computational cost. Recently, researchers at Google Brain presented a MoE architecture called Switch Transformers **fedusSwitchTransformersScaling2022a** that allows language models (AI systems that can generate text) to scale to a **trillion parameters**.

Indeed, systems that take advantage of MoEs to scale LLMs (Language Models with a Large number of parameters) already exist and have been deployed to production for applications as big as OpenAI's ChatGPT (GPT-4 is thought to be a mixture of 8 experts, each with over 220 *billion* parameters **rickardMixtureExpertsGPT42023**).

In the context of the area of this work, we consider that MoEs could be used to create a more **scalable and robust adaptable system**. The idea is that it would be composed of a set of 'expert' models, each specialized in a particular aspect of the input data. The system would then adapt to new tasks by autonomously learning a new expert model or retraining an existing one when the need arises.

The main advantage of this approach is that it could potentially allow the system to **scale to a large number of tasks and data distributions**, only needing to retrain the gating function continually instead of entire models.

Another advantage of MoEs is the potential for more **failsafe systems**. By having each model deployed independently in a distributed way, one could devise a mechanism that detects when one of the expert models fails, either due to a hardware/software error or due to a significant performance drop, and automatically replaces the gating function of the current MoE with one (previously trained) that excludes the failing model.

One could even use such a mechanism to save operational costs. The system might monitor the number of instances being routed to each expert model, and if one of the models is not being used, it could be automatically shut down to save resources.

Figure **??** shows a diagram illustrating the concept of a mixture of expert system.

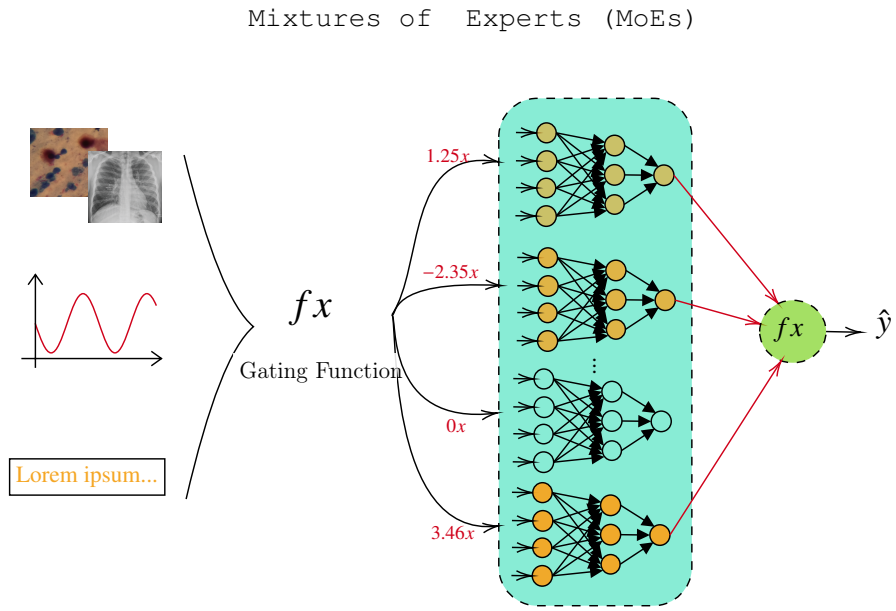

Fig. 7.3. Diagram of a mixture of experts (MoE) system.

7.5.2. Meta-Learning and L2L Systems

1 pág

Much like human learners, who, building from previous knowledge, continuously seek and filter information that could be useful to learn new concepts and skills, an area of research in machine learning concerns the design of programs/systems that can efficiently improve their learning process without the need for explicit human intervention. This area of research is known as **meta-learning**, and it is a very active area in AI .

add citation

Meta-learning is a technique that aims to improve the performance of machine-learning models by 'learning to learn' (L2L) a certain task. Such ideas have been successfully applied to a wide range of problems, including computer vision, natural language processing, robotics, video games, and more **hospedales_meta-learning_2020**.

The way meta-learning is formulated is by training a model on a variety of tasks and then using the knowledge gained from those to improve its performance on new tasks or learn it faster / more sample-efficiently than if it had been trained only on a single one **hospedales_meta-learning_2020**.

This idea is regarded to have been first introduced by Dr. Jurgen Schmidhuber in 1987 with his thesis 'Evolutionary Principles in Self-Referential Learning'. In which he proposed an algorithm that adaptively improves its learning skills by recursively applying genetic programming to itself and ensuring that only 'useful' modifications (made by the

program to itself) 'survive' in an evolutionary fashion **schmidhuber_evolutionary_1987**.

Recently, Finn et al. (2017) **finn_model-agnostic_2017** proposed a model-agnostic framework for meta-learning that can be applied to any deep-learning architecture and learning task. Which may serve as an interesting basis for applying such concepts to the kind of models we used in this thesis.

At the core of the L2L framework is the idea of building a model that can continually improve its learning process over time, which is of course at the core of the ideas we present in this work. This is the reason why we consider it an interesting topic for further research.