UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

NLPOps: A Case Study on Automation and Integration of Norwegian Medical Entity Recognition Model in Clinical Environment.

Zulfiqar Ali
INF-3990 master's thesis in Computer Science – November 2024

Supervisors

Main supervisor:	Lars Ailo Bongo	Professor, Department of Computer Science, UiT The Arctic University of Norway
Co-supervisor:	Kristian Svendsen	Associate Professor, Department of Pharmacy, UiT The Arctic University of Norway
Co-supervisor:	Mohsen Gamal Saad Askar	PhD student, Department of Pharmacy, UiT The Arctic University of Norway

Abstract

Integrating Machine Learning (ML) models into healthcare system, especially by Natural Language Processing (NLP), holds tremendous potential for improving hospital workflows and patient outcomes. This thesis presents the challenges and deployment of an NLP model, Automated Medical Entity Recognition (AMER), designed to extract medical entities from clinical text, within in cloud-based Machine Learning and Operations (MLOps) workflow. The aim of this thesis was to create a scalable, secure and efficient deployment pipeline that can meet the strict data privacy and compliance requirements of clinical environments.

The AMER model was already developed as local system, we reconfigured it for deployment on Microsoft Azure by utilizing an automated MLOps pipeline. The pipeline automates crucial stages like disaster management, model testing and deployment, minimizing the deployment from 2-3 hours to 20-30 minutes. Cost analysis showed that an initial configurations cost of $570 and on-going monthly operational expenses ranged between $262 during low workload conditions and $342 during high workload periods.

The AMER model attained an accuracy of 91.8% locally and 90.4% in the cloud with precision, recall and F1 score above 88% in both environments. Furthermore, optimization techniques were also performed that reduced latency from 450 milliseconds to 180 milliseconds, meeting the target limit of under 200 milliseconds for real-time hospital use.

Resource utilization and cost efficiency of AMER were evaluated across different workloads to test dynamic scaling of the system. At low demand (10 request per second), the Central Processing Unit (CPU) usage was 15% with the expense of $1.20 per hour, at medium demand (50 request per second) the usage of CPU increased to 35% costing $3.80 per hour and at the high demand (100 request per second) CPU was utilized at 65% with the cost up to $6.40 per hour. This adaptive scaling provided by the cloud reduces the expenses during the low workload periods and scales resources for high workloads conditions.

While the deployment using cloud provides operational advantages, this thesis discuss the challenges of integrating MLOps in clinical environments where on-premises secure data centers are often prioritized. A proposed path for further development includes using hybrid and private cloud system that allow healthcare providers to utilize MLOps based automation while maintaining compliance with local regulatory standards.

This thesis contributes to advancements in MLOps in healthcare by showing how cloud-based automation and deployment can improve the efficiency, scalability and reliability of ML models in hospital applications. However, further investigations and enhancements are needed for compliant integrations of ML-driven technologies in healthcare.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Full Form
ML	Machine Learning
MLOps	Machine Learning Operations
AMER	Automatic Medical Entity Recognizer
NLP	Natural Language Processing
AI	Artificial Intelligence
NER	Name Entity Recognition
GDPR	General Data Protection Regulation
HIPAA	Health Insurance Portability and Accountability Act
CI/CD	Continuous Integration and Continuous Delivery
UiT	University in Tromsø – The Arctic University of Norway
EHR	Electronic Health Record
RELIS	Regionalt legemiddelinformasjonssenter (Regional Medicines and Pharmacovigilance Centre)
MSA	Microsoft Azure
AMLS	Azure Machine Learning Studio
API	Application Program Interface
AAS	Azure App Service

IaaS	Infrastructure as a Service
UI	User Interface
IaC	Infrastructure as Code
SKU	Stock Keeping Unit
URL	Uniform Resource Locator
ARM	Azure Resource Manager
CLI	Command Line Interface
DevOps	Development and Operations
DRY	Don’t Repeat Yourself
AML	Azure Machine Learning
API	Application Program Interface
ACI	Azure Container Instance
CPU	Central Processing Unit
GPU	Graphics Processing Unit
JSON	JavaScript Object Notation
XML	Extensible Markup Language
SSL	Secure Sockets Layer
HTTPS	Hypertext Transfer Protocol Secure
VNet	Azure Virtual Network
DIPS	Distributed Information and Patient Data System in Hospitals

GB	Gigabytes

1 Introduction

In recent years, the integration of Machine Learning (ML) in medical care has transformed how complex data-related tasks like information retrieval and decision-making support are conducted. However, deploying ML models in real-world settings for tasks involving Natural Language Processing (NLP) brings its own challenges. NLP, which is a branch of Artificial Intelligence (AI), has gained a lot of attention due to analyzation of the human languages and have many applications in information retrieval, summarization and medical related tasks [1]. Name Entity Recognition (NER) is a subfield of NLP with the goal of identification and classification of different named entities in a text into already established categories [2]. Although NER has made great progress in the medical domain, grabbing the interest of many clinicians, who want to extract meaningful information from unstructured clinical texts, yet there remains a critical need to deploy these solutions due to technical and regulatory challenges [3].

Implementing NLP systems in healthcare faces distinct challenges, specifically when managing sensitive data that must comply with strict regulatory requirements like General Data Protection Regulation (GDPR) and Health Insurance Portability and Accountability Act (HIPAA). The issues include making the system scalable, secure, and able to integrate with existing health infrastructure. While NLP has demonstrated potential across various field, applying it in the field of medical care demands solutions that can handle data safely within regulatory limits.

Despite the growing advancement in the world of ML, there remains a gap between the model development and its successful deployment into production environments, especially when it comes to research-based projects. Many promising ML models fail to be deployed from research to real world usage due to the lack of focus on the best practices of MLOps. Research highlights a few challenges that hinder this transition, including a shortage of data science expertise, data quality issues, and the complications of managing ML models in production environments [4]. MLOps, which unites the automation and operation of ML workflows, is often left unnoticed in research-based projects, as the focus tends to be on model innovation and accuracy rather than sustainability and scalability in production [5].

While MLOPs frameworks are successfully utilized in other domains and even for NLP on health data, these solutions are not often adopted to the Norwegian healthcare context. Additionally, many of these frameworks are not fully available for research proof-of-concept projects which are typically limited in resources and scope. An important issue is that the existing solution often fail to align with the specific expectations of clinical workflows and the users, which results in limited adoption in real-world healthcare settings [6].

The lack of MLOps practices often end up to uncoordinated development process and making it hard to deploy, monitor and maintain ML model properly [7]. Many researchers lack the leadership and organizational support to apply scalable MLOps frameworks which further contributes to the gap between model creation and deployment. As research papers stress on formalizing and automating ML lifecycles – from the collection of data to the deployment of model– through MLOps can improve the success rate and efficiency of ML projects [8]. Without an organized Continuous Integration and Continuous Deployment (CI/CD) pipeline for an ML model, many projects remain confined to research settings, hence limiting their real-world impact.

This thesis focuses on challenges involved in deploying ML models, specifically NLP models like AMER, in hospital environments. The challenges addressed include:

1) Deployment of ML models in healthcare needs strict compliance to data protection standards, like GDPR. Making sure that patient sensitive data is securely maintained through the MLOps, especially in the cloud environment was the focus of this thesis.

2) Clinical settings require systems that can handle large amounts of data with low latency. This thesis finds out how to build a cloud-based system for the AMER model that can scale automatically to meet the demands while keeping response time within clinically acceptable range.

3) Classical deployment processes for ML can be more time consuming and prone to human errors. By implementing an MLOps-based workflow with CI/CD pipelines, the thesis automates operational tasks of AMER, including model testing and deployment and reduces the deployment time.

4) This thesis also analyzes the complexities of integrating AMER model with already existing hospital systems, showing the technical modifications required for interoperability with in secure on-premises environments common in hospitals.

The one and the most relevant related work to this thesis is the ongoing project that focus on integrating the AMER with hospital systems through Distributed Information and Patient Data System in Hospitals (DIPS). The project plans on deploying the AMER model within hospital settings using a microservice architecture, enabling compatibility with existing hospital systems.

However, this work does not completely address the all the operational challenges discussed in this thesis Although it focuses on data protection, regulatory standards and compatibility with hospital system, it still misses the MLOps framework required to support automated and continuous model training, testing, deployment and monitoring. This related work is primarily centered around secure integration rather than optimizing the AMER model for real-time performance, scalability and automated workflow management in hospital environments.

This thesis provides a cloud-based MLOps solution to simplify the deployment, management and scaling of AMER model in clinical settings. Utilizing the automation and flexibility of cloud provider, the solution handles the challenges including real-time performance and secure data handling. By integrating automated CI/CD pipelines, real-time monitoring and auto-scaling, this system ensure the AMER model to function efficiently under changing workloads and simplify maintenance and updates. This MLOps based approach enables that the model remains reliable, accurate and adaptable, laying the foundation for practical use of medical NER in hospital environments.

The proposed solution was evaluated across multiple aspects, including deployment efficiency, cost effectiveness, real-time processing and model performance to enable it meets the demands of hospital environments. The findings from the evaluation showed significant improvements:

1) Initially, deploying the AMER model took 2-3 hours as it was done manually. With the implementation of an automated CI/CD pipeline, the deployment time reduced to 20-30 minutes, which is more than 75% improvement. This reduction speeds up the AMER model updates, reliability and minimizes human errors.

2) The cost examination covered initial configurations, ongoing monthly maintenance and computational expenses. The initial configuration expense for deploying AMER on Azure was $570, which included setting up the CI/CD pipeline, computation resource and cloud storage. Monthly operational cost was $342 and $262 in high and low workloads respectively, showing the scalability and flexibility provided by the cloud-based autoscaling. Azure Virtual Network (VNet) and real-time monitoring costs were also added for secure and efficient model functionality.

3) The AMER model's accuracy, precision recall and F1 score were analyzed in local and cloud environments. The AMER model achieved an accuracy of 91.8% in local and 90.4% in the cloud environments, with the F1 score as 91.1% and 89.6% in the local and cloud environments respectively. These performance metrics verified the AMER model's ability to keep high performance across different environments.

4) Resource usage and cost efficiency were evaluated for AMER across different workload to examine the system's ability to scale responsively. Under low workload periods (10 requests per second), the CPU usage was minimum at 15%, resulting in low expense of $1.20 per hour. In the medium workload periods (50 requests per second), CPU utilization was 35% and the system automatically scaled, leading to the cost of $3.80 per hour. During high workload conditions (100 requests per second), CPU usage reached to 65% with further auto-scaling to maintain performance and costed $6.40 per hour. This capability of the cloud to automatically adjust computational resources based on workloads enabled cost efficiency and effective resource management, with low expense in the low-demand times and increased capacity during high demand conditions.

The results showed that deploying the AMER model on cloud-based infrastructure gives significant benefits in terms of cost-effectiveness and efficiency. The automated CI/CD pipeline simplified the AMER model deployment process, minimizing the time and reducing the error while supporting faster and reliable model updates. The cloud ability to scale automatically allowed for optimized computational resources usage, enabling the system to handle fluctuating workloads and controlling cost.

However, while the cloud providers offer these operational benefits integrating MLOps in hospital settings presents more challenges due to data protection requirements and priority of in-house data centers. A proposed way to move forward includes exploring hybrid and private cloud services,

which would allow hospitals to take advantage from MLOps automation while keeping compliance with regulatory standards. This method provides feasible pathway for deploying MLOps workflows in secure clinical settings, clearing the path for broader adoption of ML health applications.

In the following sections, this thesis is structured as follows:

Chapter 2: NLPOps Approach, describe the AMER model, its integration with Azure platforms and technical side of our implementation.

Chapter 3: Evaluation and results, covers the metrics and methods to evaluate the performance of the AMER.

Chapter 4: Explores the ongoing and the most related work: the AMER model integration in hospital.

Chapter 5: Conclusion, summarize the findings.

2 NLPOps Approach

Development and Operations (DevOps) is a set of practices that aim at improving collaboration between software developers and operations team with the goal of faster and reliable software delivery [9]. DevOps focuses on automation, continuous integration, continuous deployment which make it easier to manage and update software in production settings.

MLOps, is known as "DevOps for ML", builds on these principles that address the challenges of managing and automation of ML applications in production. MLOps is important because it handle the challenges that are unique to ML workflows, like model retraining, deployment of updated models and version control, which make sure ML models remain reliable and accurate with the passage of time [10].

NLPOps, which is a name given to this thesis, is same as MLOps but focuses on the automation and operational issues of deploying NLP models. In the NLP applications that involves medical data, NLPOps focuses on the need to a secure, scalable and compliant deployment.

MLOps plays an important role in the development and deployment of ML model, especially in scenarios like healthcare where reliability and accuracy are crucial. In this thesis, we focus on medical NLP applications as a case study, with AMER as an example. AMER is an ML model that is designed to identify medical entities such as substances, side effects and diagnosis from medical text which help clinicians in their decision-making.

For a model like AMER, which must function with high accuracy to prevent serious mistakes like incorrect diagnosis, MLOPs assures that any model updates are tested automatically and deployed safely with minimum human intervention [11]. This automation reduces the risk and time related to manual updates. Moreover, MLOps simplifies the scaling of ML models, which is important for medical applications that handle large amount of medical data. Through cloud services, MLOPs enables models like AMER to adjust to changing workloads while maintaining consistent performance [12]. This makes sure that the system can keep on helping clinicians continuously, when as demand for data processing increases.

In this section, we discuss the AMER model, the tools, technologies and the steps followed to achieve the objectives of this thesis.

2.1 The Automated Medical Entity Recognition Model

In medicine, it is very important for the professionals working in the field to take history from the patient and record it in the form of a document. These documents can then be used to identify things like side effects, drugs and to diagnose a disease in a written text. This method of identification is slow and manual which can be prone to error. The Electronic Health Record (EHR) has enabled us to electronically keep the record of patients making it more secure and instantly available to authorized people [13]. Therefore, now we can find advanced ways to solve this problem.

The EHR contains a lot of amounts of data but most of this data is unstructured which makes it difficult to examine and extract key medical insights. Medical NER is a form of NER that can

utilized on the history of a patient to identify drugs, diseases and symptoms in an EHR [14]. By identifying relevant entities in EHR, medical NER helps to turn the unstructured data into structed format, which makes data analysis more efficient.

In medical NER, an ML model is trained on a dataset, which has specific medical terms labelled. Once the model is trained, applications use Medical NER to automate tasks which not only helps clinicians but medical research. Due to language differences, specific models should be trained for each language as every language has its own grammar, sentence structure and punctuation. Then to train such a model, a big, annotated dataset is needed which has been labelled into predefined categories such as substance, disease and medicine. Moreover, annotating such dataset is a manual process which takes a lot of time.

In this regard, a AMER model for Norwegian medical text has been recently developed at the Department of Pharmacy of UiT. This model can predict the entities in the text provided to it. To the best of our knowledge a system like this which automatically annotates a dataset is new and is not implemented before for Norwegian language so far.

The development of the AMER model started by utilizing the NorMedTerm list, which consist of 77.000 unique medical entities in Norwegian together with a dataset from Regionalt legemiddelinformasjonssenter (Regional Drug Information and Pharmacovigilance Centre; RELIS), a compilation of 36 thousand question-answer pairs that are related to medicine from health professionals [15] [16].

To clean and preprocess the data, spaCy library is utilized which offers capabilities when it comes to NLP [17]. The NER module in spaCy is supported by multilayer Convolutional Neural Networks, is used to train the annotated dataset.

Figure 1 below shows detailed explanation of the steps taken to develop the AMER model. The development pipeline starts with data processing, where entities and terms are cleaned from RELIS and NorMedTerm list. The spacy Entity Ruler is then used to annotate these entities within the dataset, which creates a labelled dataset. The labelled dataset is divided into 80% for training and 20% for testing of the AMER model. Finally, after training, the AMER model can accurately recognize entities in Norwegian text. The AMER model can then be saved and implemented to identify entities in other medical texts in Norwegian.

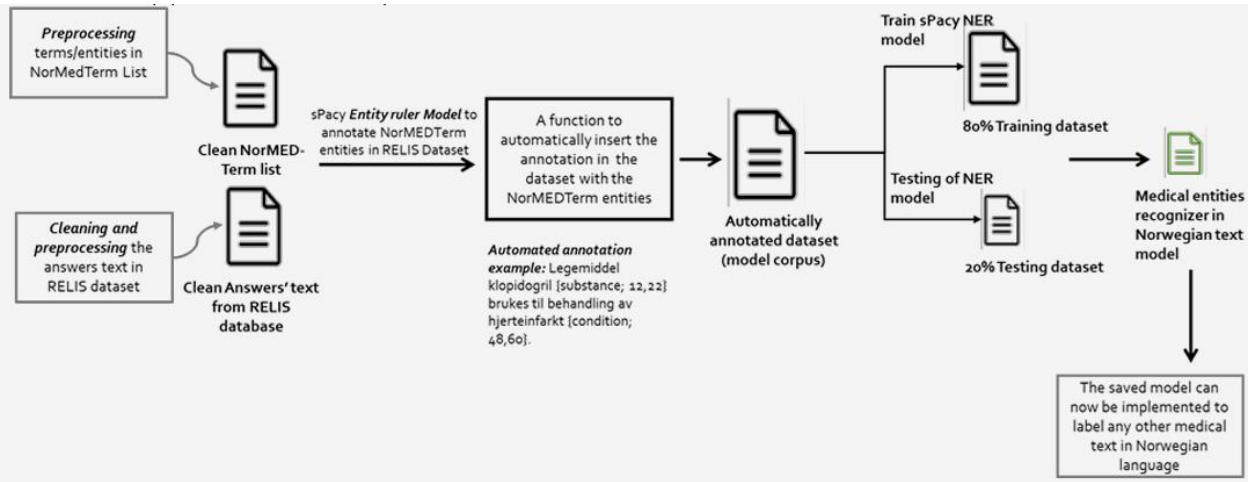

Figure 1: An illustration of the pipeline of Norwegian medical AMER model [18].

Automatic data annotation has an immense impact on the AMER model, as it streamlines the training pipeline and ensures data annotation to be more consistent and robust [19]. As a result, this trained AMER model for Norwegian text offers a range of applications that effect both medical professionals and researchers. The model can be used in real-time and batch processing scenarios depending upon the requirements.

For clinical use, AMER can be integrated into hospital infrastructure to perform queries for patient history summarization, aiding diagnosis support and clinical data extraction. For example, when a clinician is looking for a patient file, the AMER model can recognize and extract medical entities from EHR.

Applications can differ between one-off queries for specific patients or daily usage applications that run consistently across large patients' databases. For instance, the AMER model can be used for extracting entities relevant to individual patients. On the other hand, it can be applied to all patients in a clinic or hospital, automating data analysis on a large scale for quality improvement or research.

For research applications, the AMER model can help in literate search and medical text indexing, which can speed up the process of extracting terms from large amounts of unstructured data. This can be useful for researchers in clinical fields, who need to screen through large dataset to identify correlations.

2.1 System Architecture Design

We used prototyping as a design method to define the architecture for the development and deployment workflow of the AMER model using MLOPs best practices. This way of design method allowed us to quickly create and validate different working designs to find the optimal workflow of our system. For this thesis, we used Microsoft Azure (MSA) for MLOps based implementation, and its subscription was provided by the informatics department of UiT.

MSA has a range of services that are used to combine all the components of ML and start running the life cycle of ML operations. The figure 2 below shows an overview of how different platforms integrate in Azure to support MLOps workflow. The connection of each component shows the process for developing, testing, deploying and maintaining ML models.

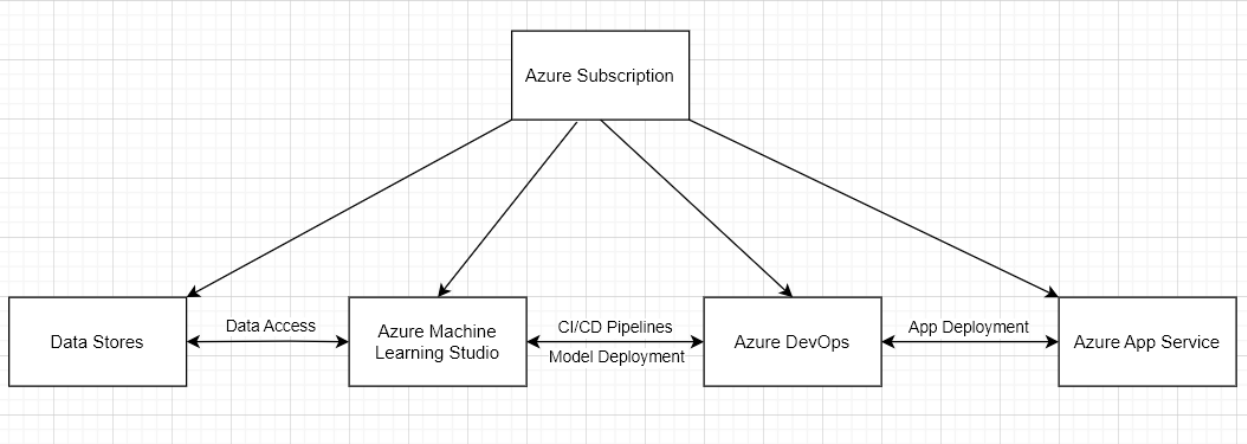

Figure 2: Overview of the Azure Platforms supporting NLPOps Workflow.

The Azure subscription, which was managed by us, serves as an entry point into the environment of Azure and allow us to use its different services [20]. Each organization using the AMER model may have its own subscription, which gives security mechanisms like role-based access control, enabling that only authorized people can have access to the model and data. In our case, we kept only one subscription which helped us control and manage costs by tracking resource usage for the AMER model deployment evaluation.

Azure Machine Learning Studio (AMLS) is a platform used in this project for building, deploying and maintaining AMER model. By AMLS we managed dataset, trained the model and deployed it as an Application Program Interface (API) [21].

In the AMER case, the preprocessed data, training dataset and output during predictions are stored inside Azure Blob Storage, which is a type of data storage in Azure's data store. This storage is used to securely manage sensitive medical data, assuring compliance and provides access to necessary data for a model lifecycle [22]. Azure DevOps acts as a main source code repository and enables CI/CD automation pipelines for each stage of AMER model workflows. This setting enables that changes are tested and deployed consistently [23]. Azure App Service (AAS) is

utilized to deploy the AMER model's front-end application and integrates well with other Azure Services [24].

To support scalability and flexibility for multiple applications of AMER, the design uses cloud platforms multi-tenant capabilities in which each instance can be managed separately, Azure DevOps pipelines for deployment and version control, AMLS for model versioning, and AAS for front-end applications deployment. As shown in the figure 3 below, Azure and GitHub are the two main services utilized here for the automated development and deployment of the AMER model.

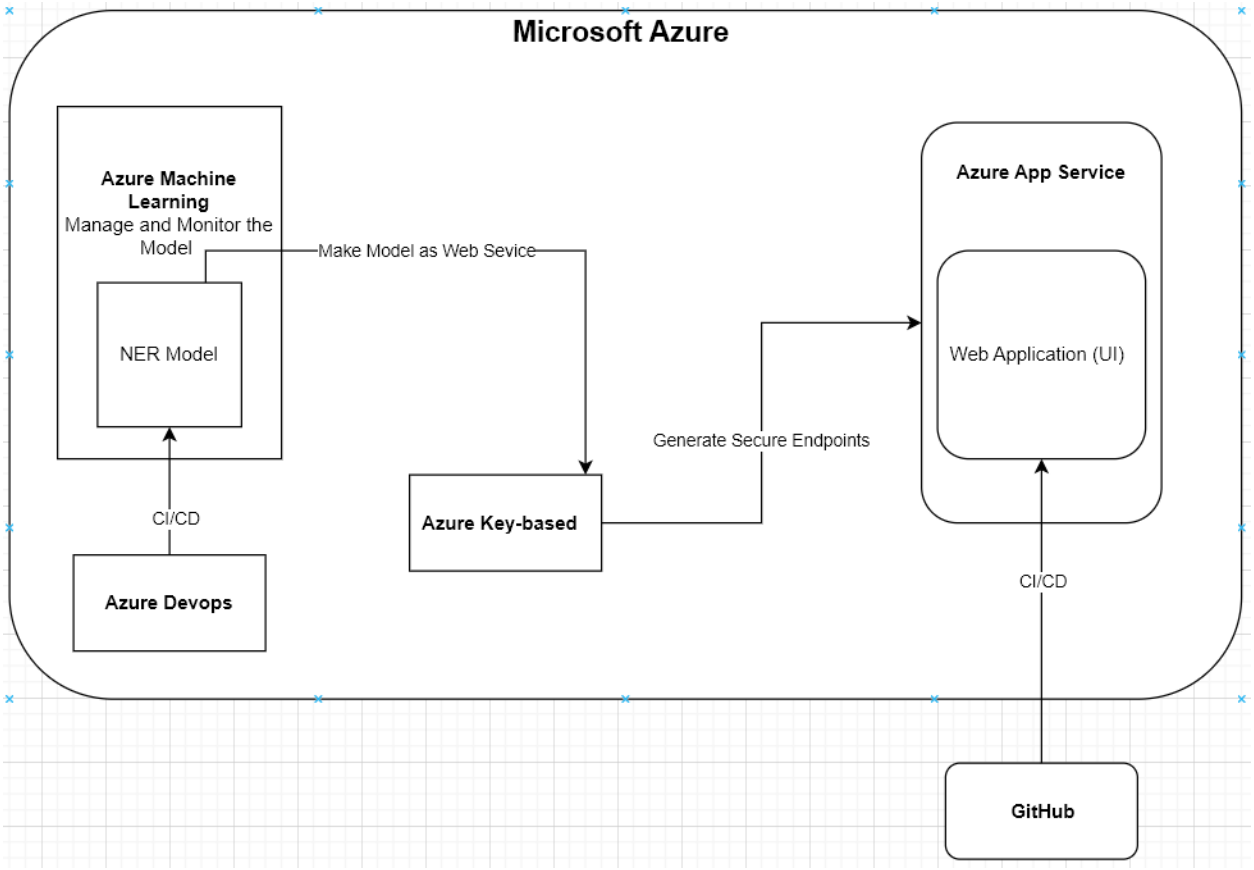

Figure 3: System Design Architecture for AMER model.

The User Interface (UI) part of the application is separated by the original AMER model to keep the system flexible and understandable. GitHub is used as a version control tool to manage the UI, and its deployment will be automated using GitHub actions. This means that every time a new version of the UI part of the code is ready to be shipped to the production, there won't be any need to manually perform any task. As the code is pushed, an automated pipeline will be initialized, and the new version will be in production.

Azure DevOps is also a version control tool which we used specifically for the AMER model part of our system. It can easily integrate with Azure Machine Learning (AML) using an automated pipeline. We will be utilizing Azure DevOps for the Infrastructure as Code (IaC) and CI/CD

pipeline of our AMER model. The IaC pipeline will create an environment in AML for the AMER model and CI pipeline will perform the tasks including running unit tests, creating computes and uploading the dataset. The source code of the AMER model will be in Azure DevOps but the model itself will reside inside AML from where it will be monitored and managed. After the model in developed in AML a release pipeline will be initiated that will deploy the model

When the model is deployed it will generate an end point. These end points will be only authorized to specific users which will add an extra security layer to the system using Azure Key-based. The secure end points can then be accessed by the UI application and the users can get real time predictions.

Once the system is fully deployed and the results from the evaluation are available, we will try to further optimize the designed solution. Potential optimizations include implementing caching solutions for frequently accessed task and refining auto-scaling to handle different workloads effectively.

2.2 Cloud vs On-premises solutions for MLOps in Healthcare

Utilizing cloud services in MLOps offers technical and operational advantages that are difficult to recreate in on-premises environments. This thesis uses cloud-based MLOps solutions to handle the needs in automation, scalability, compliance and resource management.

Cloud providers gives on-demand scalability, allowing this thesis project to smoothly manage storage and computing resources based on workload demands. This flexibility confirms that resources are available during peak time without facing the high costs of dedicated hardware. By scaling up or down as required, the thesis prevents the waste of underutilized resources during low-demand time [25].

The cloud has integrated CI/CD pipeline tools personalized for ML workflows, which contains version control, automated model retraining and deployment across different environments (development, staging and production). In this thesis, these cloud-based CI/CD abilities improves model updates and deployments, making certain that new AMER model versions are tested and deployed properly. Without the cloud, building such a pipeline would need manual integration of separate tools, adding more difficulty [26].

Cloud services provide real-time monitoring and simplify management of resources, both are important for MLOps in healthcare. This thesis gets automated infrastructure management through Infrastructure as a Service (IaaS) by the cloud, which removes the need for content maintenance and hardware upgrades. Real-time monitoring enables resource utilization, potential security threats and tracking model performance, which enables smooth and secure functionality [27].

Cloud systems come with built-in compliance and security features, including encryption and regulatory compliance frameworks like GDPR. In this thesis, the cloud security protocols assure that medical data is handled in secure and compliant manner. Reproducing these levels of security on-site would need investment and dedicated security resources, which organizations find impractical.

In the context of MLOps, many organizations choose external cloud providers due to the scalability, monitoring capabilities and convenience that they offer for ML applications. Cloud services like Microsoft Azure, Amazon Web Services and Google Cloud provide ready-to-use infrastructures and tools like model deployment, automated scaling, and real-time monitoring, which would be costly and complex to implement them on-site. These services also has global availability zones, which make sure disaster recovery and high availability.

For health organizations, data privacy and compliance regulations are important. Many hospitals, clinics and research institutions choose cloud operators with data center in their own county to comply with local regulatory standards like GDPR and HIPAA. By this way, these organizations can confirm that national legal requirements are met, and patient data is stored locally and is secure with their own control.

On the other hand, some organizations prefer open-source solutions, which gives more control over data and avoid vendor lock-in. However, this case needs more people to manage infrastructure, handle updates and scalability.

A small number of organizations, including hospitals, rely completely on in-house data centers. This approach allows them to have full control over data lifecycle without relying on an external cloud service. Although this approach improves privacy and control, yet it can create challenges in maintaining up-to-date infrastructures and scaling ML operations as data processing demands increase.

2.3 MLOps based AMER Model Integration

The below figure 4 shows MLOps based model integration design that we will utilize for the AMER Model. Each module listed in the figure belongs to one or another component of figure 3. For instance, the CI/CD pipelines and model validation belongs to Azure DevOps, Prediction Service belong to AAS and the rest of the tasks are part of AMLS. Moreover, this sub-section explains how different components are connected to create an MLOps lifecycle.

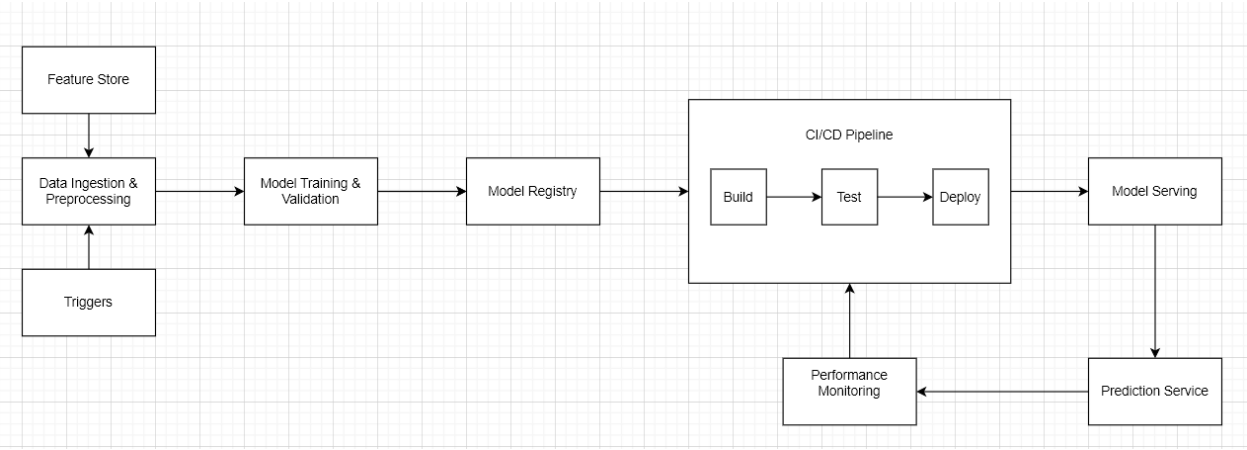

Figure 4: AMER Model MLOps based Integration Design.

The process will start by the ingestion of open access medical data, which will be cleaned and pre-processed to create an annotated dataset in the cloud. This step is important to validate the data fed to the AMER model is clean, structured and ready for analysis.

The feature store is a main repository in AMLS that stores preprocessed and features that derived from the clinical text. It provides reusable and consistent data for both model experimentations and production. As seen in the figure 4 above the Feature Store continuously provide prepared data for the data ingestion task. In this system the model updates can be planned based on approaches called manual, periodic and continuous.

In manual approach, updates are initiated by the data scientists only when required, depending upon when significant changes are needed, or new data is available. This approach gives more control when the model is updated but it can delay model improvements. Periodic updates is when the model can be set to retrain after some planned time to assure it remains current. This approach is a balance which allow the model to adapt to new trends in the data without overwhelming the system with constant retraining. Finally, with continuous updates, the model could be updated and retrained as new data comes into the feature store. By this approach the model is always using the latest data, but it comes with high computational cost. For this thesis, we choose manual approach to keep the cost low and as it was a just the beginning and starting point for the release of AMER model in production, therefore we wanted to keep it basic.

The core of this system will be Automated Pipeline, where the tasks like data extraction, data verification, model training and model evaluation will be carried out. Data extraction involves pulling relevant data to prepare it for training. Data verification is when the dataset is checked for quality and completeness. Once verified, the data is then used to train the model. After training the model, it is evaluated with unit tests to access it's working. This pipeline will automate the AMER model lifecycle and will be designed to be repeatable and scalable.

Once the AMER model is trained and tested, it will be stored in the Model Registry, which keeps track of all the versions of the model. This Model Registry will be important to enable notifications on model updates and will also help us decide to select the best version of AMER model gets in production. Once we have more than one version of AMER model, the model registry interface in AMLS provides us with an interface with the model's name, its version and different matric including accuracy, precision and response time. Then based on these metrics we can select which model version to choose but for this thesis we will train the model once.

The CI/CD will automate the deployment of AMER model in different environments. After the model is packaged, tested and validated, it will be deployed via CD pipeline. The AMER model will be available as a secure endpoint that will integrate with clinical infrastructure.

The AMER Models Serving component will allow the trained model to process the real medical data and return results. This component will be directly linked with the front-end application, simulating how AMER model would be used in actual clinical system.

Once the AMER model is deployed and start to serve requests, its performance will be continuously monitored using the performance monitoring tools of AMLS. The performance data is stored in ML Metadata Store to give feedback for future improvements.

The trigger starts the entire pipeline when there are conditions like getting new data, periodic retraining, or performance feedback showing a need for new version of the model. This mechanism helps that the system remain efficient and avoid unnecessary resource use. In our case a new pipeline will be triggers as soon as the new code is pushed into Azure DevOps repository.

This structure will support continuous AMER model improvements and scalability, enabling it to reliably process medical data. To summarize our implementation, we started by automating the deployment of the demo application of AMER, followed by creating an IaC pipeline for disaster recovery and then deploying the AMER model using CI/CD pipelines.

2.4 Creating a Resource Group

Once we have a subscription on MSA, the first step was to create a Resource Group. A Resource Group is a logical container that organizes and packs together all the assets that are required to run a project or an application [28]. It can be created on the Azure Portal by just entering a name, selecting the right subscription and choosing the geographical location of the Azure resource. After a resource group is created MSA provides you with an interface from where we can view and manage all the different Azure services inside this resource group as shown in the figure below.

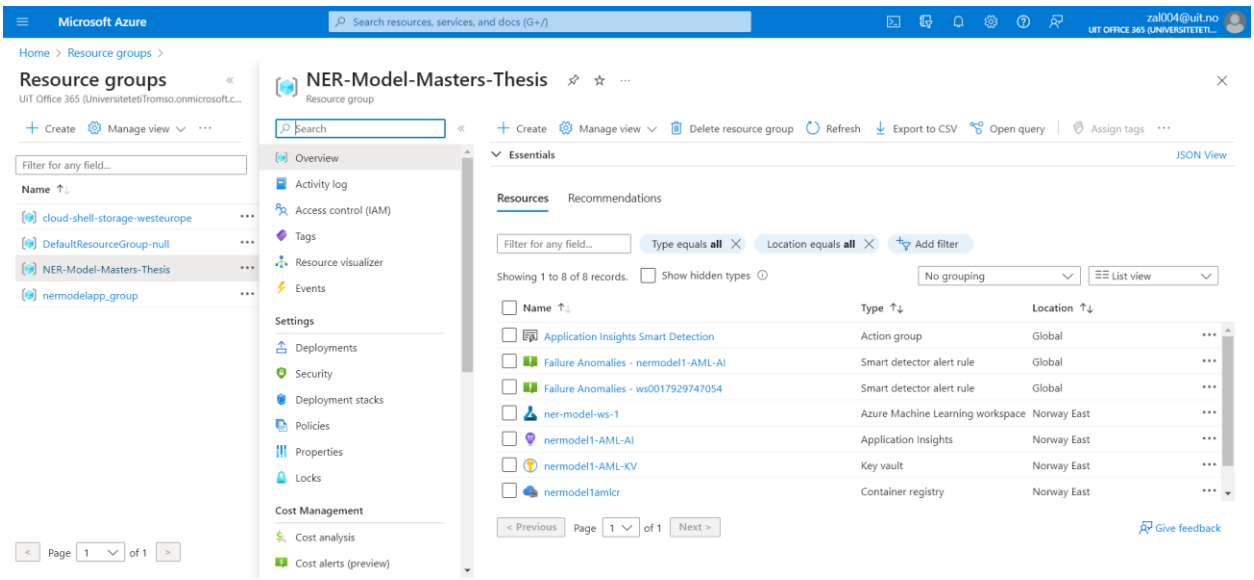

Figure 5: MSA resource group interface.

2.5 Deployment of AMER model demo application.

The AMER model already has a demo web application built on Streamlit, which is an open-source framework to build interactive web applications and is specifically designed for machine learning and data science applications [29]. Since the model and the web application are coupled together

it was important to decouple both and deploy the web application separately on AAS and the model on AML.

Before starting this thesis, the AMER model code existed locally in one machine so the first task before deployment was to push the code to a remote private repository. After cleaning the code and removing the model part of the code to decouple the system the demo application code is pused to GitHub.

The initial step of the implementation involved moving the existing local code of AMER to a remote environment. The code was initially stored in a single computer, so we decoupled the model code from the rest of demo application code and pushed it to a GitHub repository.

As part of the settings, we created a requirements file to include all the dependencies needed for the demo application to run. To make these dependencies were separated from any system-wide packages, we created a virtual environment for the application. This isolation helps in conflicts preventions and replicate the environment, which is important for our smooth implementation.

For the demo application deployment in this thesis, we utilized GitHub actions as it is simple to set up and more user friendly.

2.6 Creating Azure Application Service

As part of our system design architecture of AMER and its simplicity to use with other Azure services, we used AAS to deploy the front-end application of AMER to serve as a prediction service. Therefore, we created a new AAS instance by entering the name of our demo application, choosing the right Stock Keeping Unit (SKU), geographical region of the server, and the runtime stack. After successful deployment submission we get the dashboard created as shown in the figure 6 below. This dashboard will help us further manage and monitor the application.

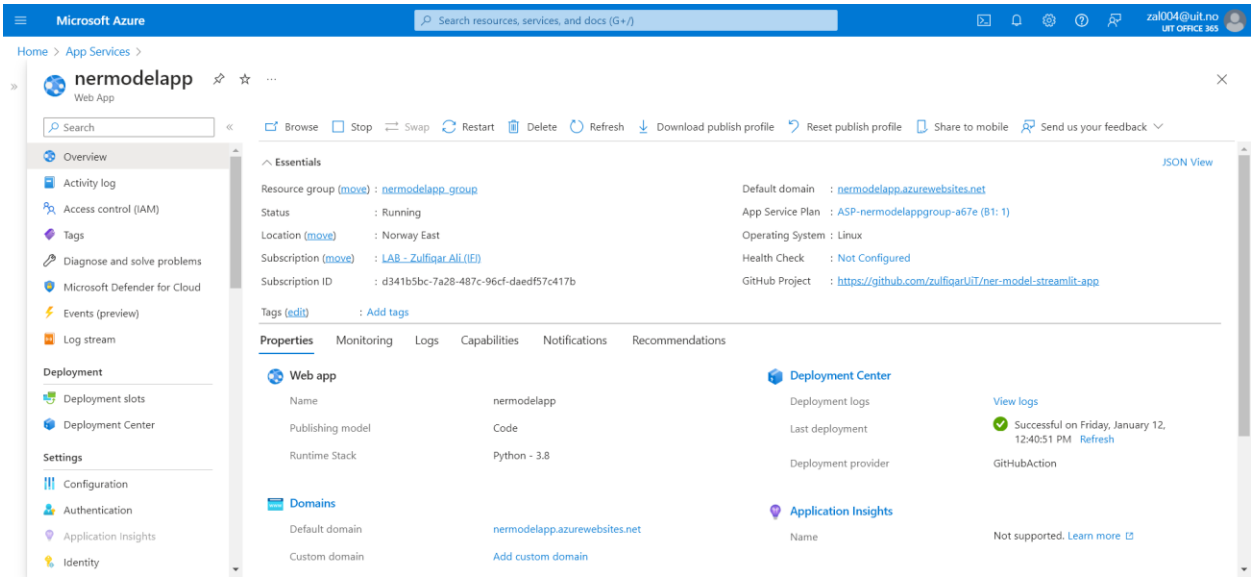

Figure 6: The dashboard of the demo application.

The name kept for the demo application was unique since it becomes our Uniform Resource Locator (URL) when the application is deployed. Initially, to keep the cost low, we decided to go for a free version of SKU but after the initial deployment it was observed that for a Streamlit application to run and to have sockets enabled we need to have pricing tier of minimum B1. Therefore, we switched the SKU from the free version of F1 to the basic paid version of B1.

Since the demo application is in Python so we selected our run time stack as python 3.8 and the operating system becomes Linux by default. We also choose nearest location as our deployment region due to GDPR compliance and data control.

The next step was to link the GitHub repository with the AAS by first authorizing the GitHub to ASS. After entering the current repository name, branch name and enabling continuous integration the GitHub and AAS are now linked together.

For this demo application, we decided to directly deploy the application from the source code as it is the most straightforward way that offers simplicity. The other ways of publishing the application include docker container or static web app, which can introduce complexities.

2.6.1 GitHub Action Initialization

After the initial submission of the deployment a GitHub workflow gets started within our repository. A YAML file was created inside the repository which defined all the steps that were required to build, test and then deploy the application on AAS. During the build process it installed all the dependencies that we wrote in our requriemnts.txt file. Then, GitHub Actions automatically initiated a build process which compiled the code from the repository, ran tests, and generated artifacts. The below figures 7 and 8 show the successful build and deploy pipelines that were triggered by GitHub actions.

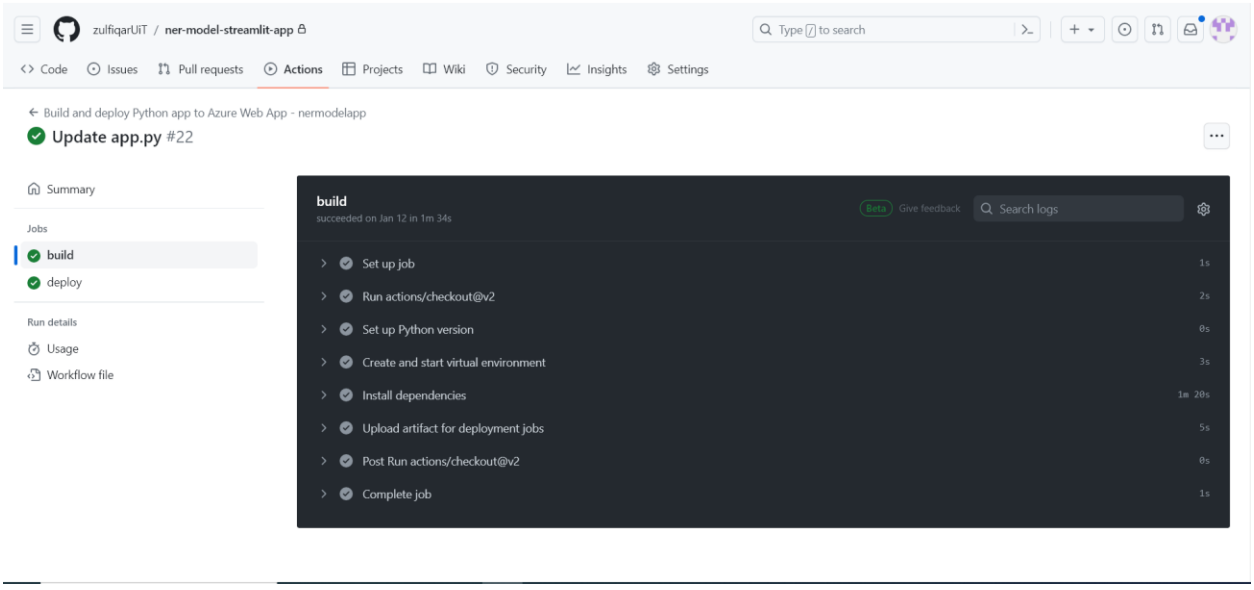

Figure 7: The GitHub Actions Build Pipeline

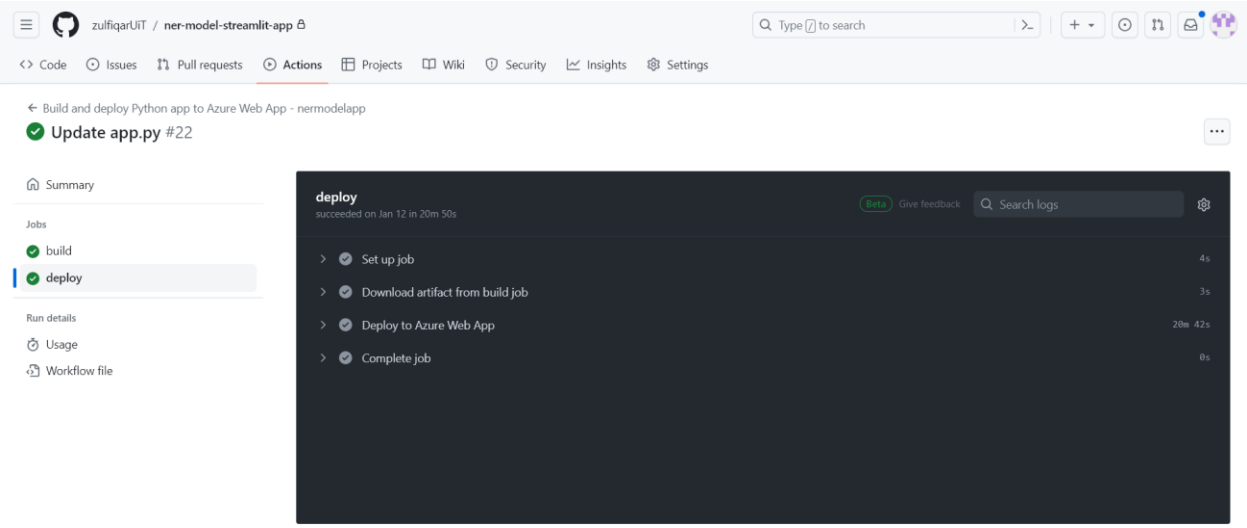

Figure 8: The GitHub Actions Deploy Pipeline

2.6.2 Running the demo application

Although the code is connected and deployed to AAS but still there is a last step we took which was to start the application every time the code is deployed. We did it by providing the startup command to our server in the configuration settings of AAS. Now we were good to view our demo application with a URL. The below figure 9 shows a live version of the demo application with CI/CD enabled. Which means that now every time a new line of code is pushed on GitHub, a pipeline will be triggered, and a new version of the application will be deployed.

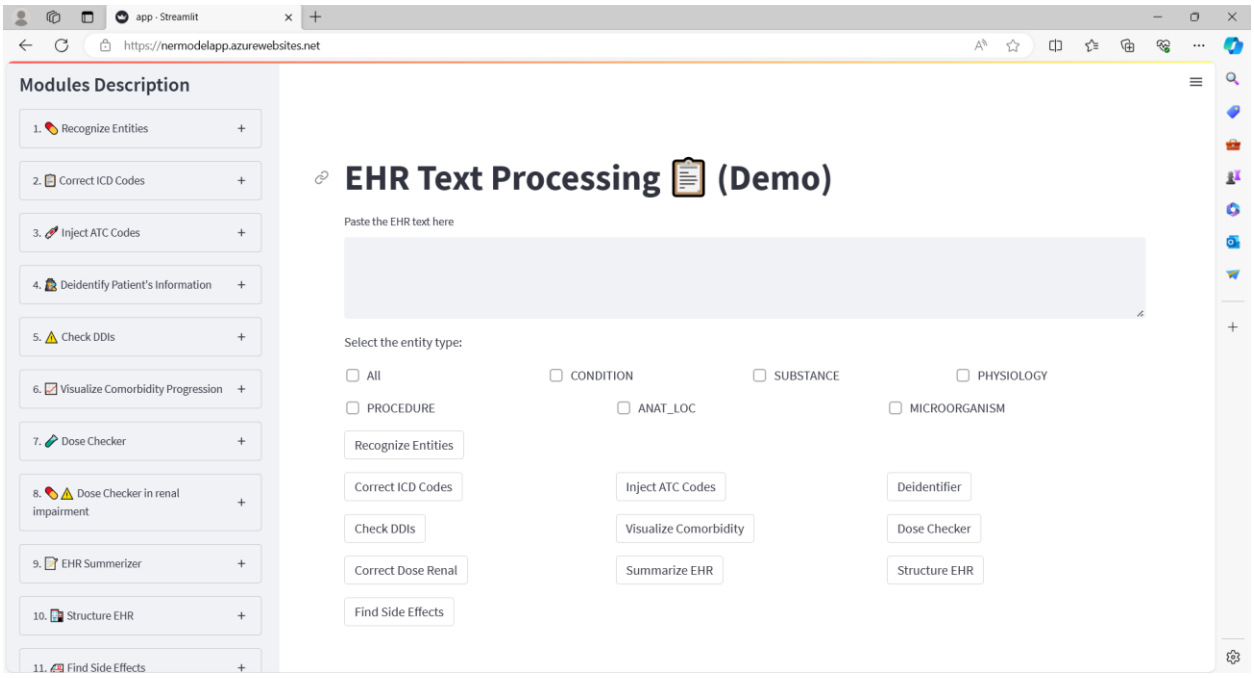

Figure 9: Live demo application deployed with CI/CD

2.7 Setting up Azure DevOps Configuration

Azure DevOps is an important component in our system design architecture that we used to create automated CI/CD pipelines and for model validation for AMER. Setting up Azure DevOps involved steps to build an environment for the development and deployment of AMER model. As we had already decoupled the AMER model from the demo web application. Therefore, it was much easier to get started with configuring Azure DevOps. Firstly, a private project was created with a unique name and a description. When the project was created successfully, a repository with the same name was created inside Azure DevOps. We chose Azure DevOps for the development and deployment of the AMER model as it preferred tool to use with Azure Cloud [30]. Therefore,

the second step was to push all the AMER model source code from the local environment on the personal computer to Azure DevOps.

2.7.1 Connecting Azure DevOps with Azure Subscription

Before moving forward, it was important to connect Azure DevOps with Azure Subscription to establish a workflow for the AMER model development and deployment. To do so, we need to create a service principle that enables us to perform secure communication between the two services. We selected Azure Resource Manager as service connection as it is used to connect to Azure DevOps and provides a consistent management layer [31]. Then after selecting the right subscription, resource group and entering the name of the service connection, a connection is created. When this was done, now our Azure DevOps is authenticated with the AML.

2.7.2 Creating an automated AML environment

Our next step was to set up an environment in AML for the development and deployment of AMER model. Although we could manually create it but after doing some study, we found out that it was highly beneficial to automatically create resources on AML. Using automated way is a best practice for disaster recovery, as it allows us to define our environment using IaC [32]. Using this way will help us recreate the environment in the same way if we lose this infrastructure and its resources.

2.7.3 Defining the variables for IaC pipeline

It was important to define the private variables in advance before using them in our IaC pipeline because not only it promotes reusability but also allows us to securely store all the sensitive data that will be utilized for building the pipeline [33]. We created the following variables in the Azure DevOps library as shown in the table below. The base name refers to the name of our project in AML and the location is the geographical position of the servers. The service connection name and the resource group are the same as we had already created. Finally, the workspace is the main environment that we will use for the AMER model, and the workspace service connection is the authorization of workspace with Azure resources.

Table 1: Variables for automated AML environment pipeline

Name	**Value**
BASE_NAME	nermodel1
LOCATION	germanynorth
AZURE_RM_SVC_CONNECTION	azure-resource-connection
RESOURCE_GROUP	NER-Model-Masters-Thesis
WORKSPACE_NAME	ner-model-ws-1

WORKSPACE_SVC_CONNECTION	ner-model-ws-connection

2.8 Infrastructure as Code pipeline

Before starting to build the IaC pipeline, we pushed the decoupled code of AMER model on Azure DevOps to keep it in a remote environment.

After configurations and then linking the pipeline with the script we were able to initiate and successfully run the execution. The pipeline execution can be seen in the figure 10 below of Azure DevOps along with each step and other related details.

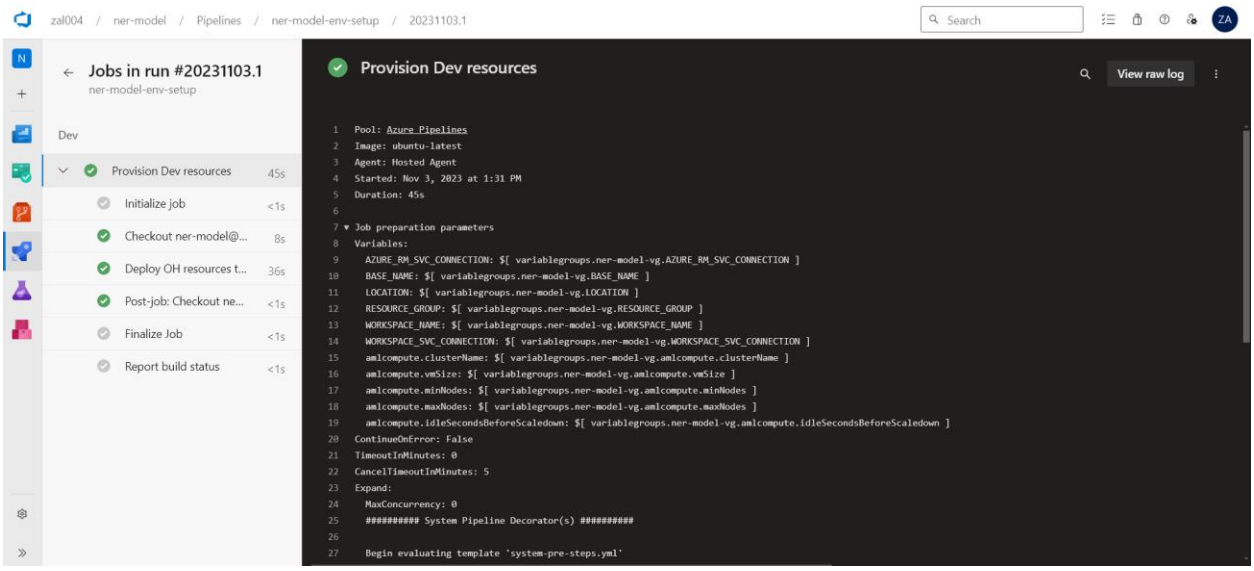

Figure 10: Environment setup pipeline on Azure DevOps

During this process, we explored and found different ways to create our first automated pipeline and to build an environment in AML by Azure DevOps, one way was by using an ARM template and other was Azure Pipelines YAML.

The Azure Resource Manager (ARM) template can be used to deploy environments in the Azure as it helps us in the codification of infrastructure using IaC. We can provide different ARM templates to the ARM based on the task we want to perform, and the ARM deploys our environment on the main level of subscription. The ARM template also supports repeatable deployments because of IaC [34]. The below figure 11 shows the flow of ARM templates to create an automated pipeline that we used to build an environment in AML.

For our IaC implementation we choose Azure Pipelines YAML, which works in the same way as ARM Template, but we write our own custom script instead using a template. The reason for choosing this code first approach in this case was because of its simplicity and flexibility to create an environment in AML [35].

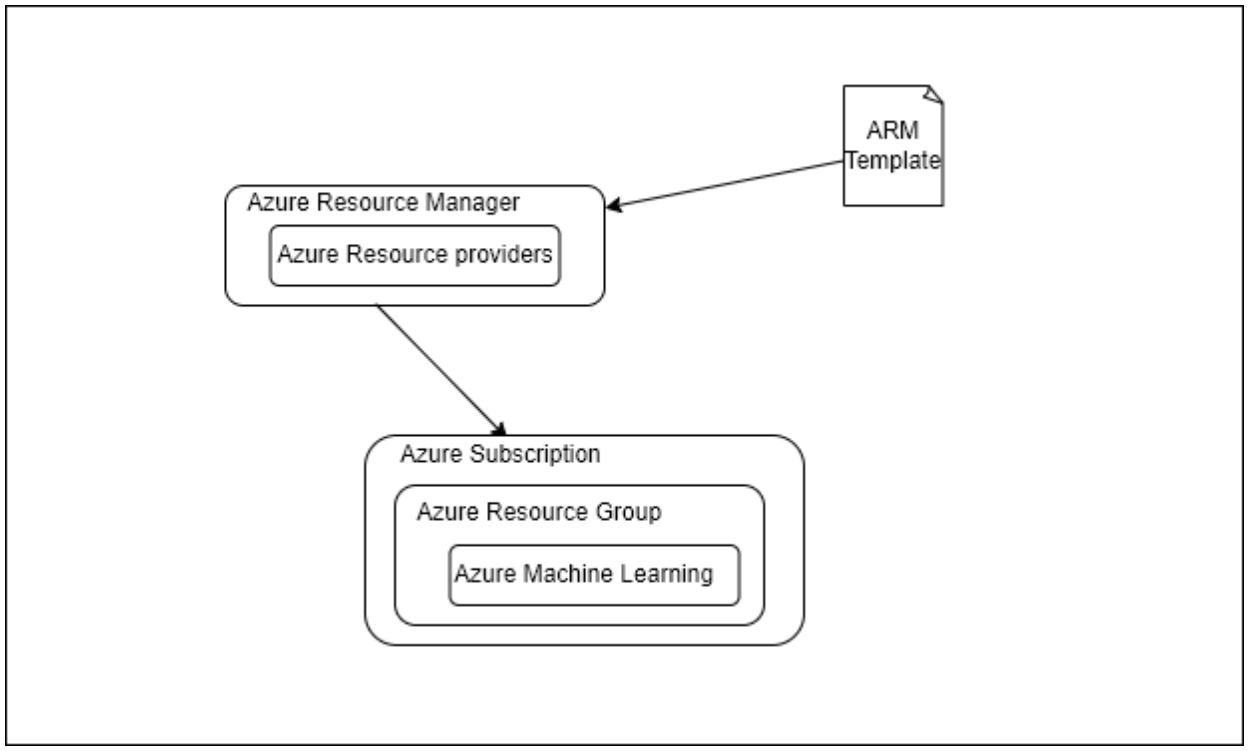

Figure 11: ARM flow for AML pipeline.

Now, we had the environment in which we could start our MLOps journey by building CI/CD pipelines. The good thing to notice here is that if our system runs into any disaster, since the infrastructure creation is automated, we can easily run this pipeline script that will configure Azure DevOps pipeline and re-create a new AML environment.

2.9 Continuous Integration Pipeline

After this, our next step was to focus on building the CI/CD pipelines for the automated integration and deployment of AMER. As per our MLOPs based AMER model integration design in figure 4, we started by building continuous integration pipeline for AMER followed by continuous deployment pipeline.

A CI pipeline automated the process of our AMER model development. Every time, before a new piece of code will be integrated into the main system, it will be automatically tested using this pipeline. The new changes to the AMER model can only be made once all the requirements and steps in the CI pipeline are successfully completed. Hence CI pipeline will ensure that our AMER model development workflow is efficient and consistent [36]. All our CI pipeline tasks ran successfully as shown in the figure 12 below.

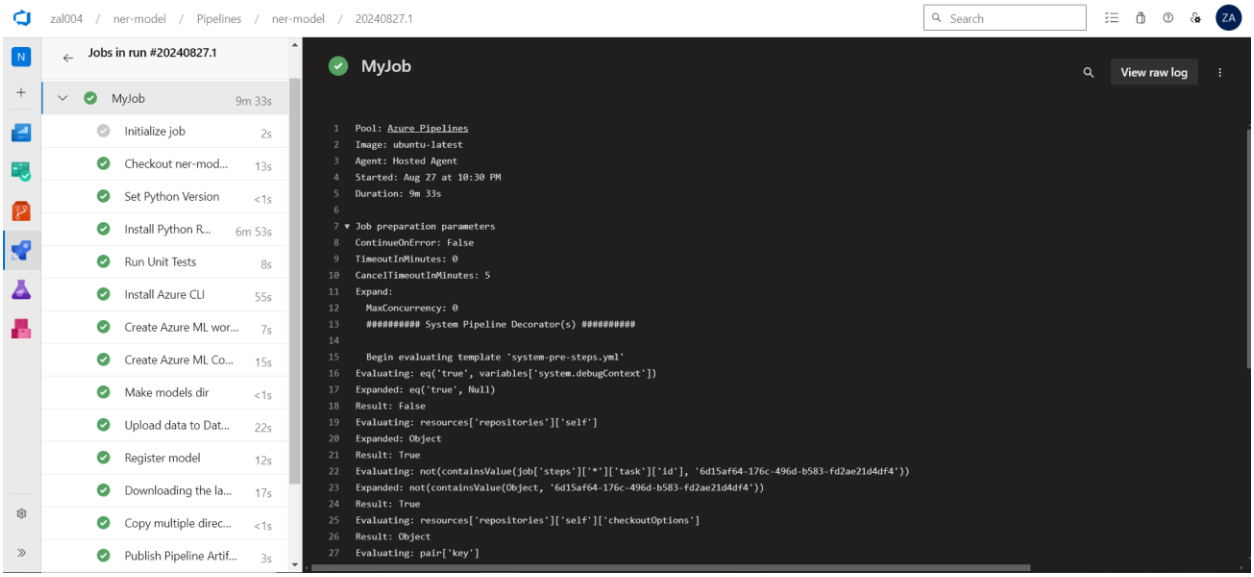

Figure 12: Compete CI pipeline of AMER model on Azure DevOps

To get started we utilized classical editor, which is a graphical interface in Azure DevOps for building CI pipeline for AMER model. Previously, we used Azure Pipeline YAML for building the IaC pipeline but this time we utilized the classical editor as it is the simpler to integrate unit tests and all the other steps into the pipeline by just using the interface [37]. The execution of this pipeline will take place as soon as new code will be pushed to the AMER model repository.

We added the following tasks in the CI pipeline in the classical interface by simply defining the type of the task and then executing it using a command to run that task. All the tasks that we will define in the interface will be converted to YAML file and saved into our repository. Azure DevOps will use this YAML file to execute the pipeline.

2.9.1 CI pipeline tasks

The first task we added was to set up a python environment. While installing the environment we also specified the version of python to ensure consistency across our system. Therefore, a virtual environment was created in which we installed all the dependency packages required by the AMER model. Since we have already described all dependency packages for our demo application in the requirements file, so we created a new requirements file and just pointed out the file name to be executed by the pipeline.

While building a CI pipeline it is very important to consider writing unit tests as they verify the correctness of every small component that are part of the big ML model code [38]. Verifying each part separately of the AMER model will ensure the smooth execution of the whole model code. So, to get started we also wrote a basic unit test to verify the creation of the AMER model with a small set of predefined data. The result of this unit test will help us to move forward in the execution of the pipeline.

Our next step was to authenticate and connect Azure DevOps, which is the agent that is running our AMER model pipeline, to our AML by creating a similar service connection we have already created to link Azure DevOps with our subscription. Therefore, we recreated a service connection on Azure DevOps but this time with the scope of AML workspace.

2.9.2 Installing Azure Command Line Interface

Now when we have connected Azure DevOps with AML, we needed a coding way to write down the steps that we need to perform in our pipeline for the development of AMER model. For instance, we want DevOps to create a compute on AML, but it is not a human that can create it by interacting with AML studio. Therefore, we need a way of coding that will help us do it for the development of the AMER model. We can do it by creating an Azure Command Line Interface (CLI), that is a terminal-based tool in which we can write down the instructions in Azure DevOps and it will create a compute on AML for us [39].

The AMER model CI pipeline that we are building is running on an Azure Pipeline agent, which are responsible to execute the tasks that we have defined. This agent needs to have Azure CLI installed beforehand to make sure that we can successfully execute the further steps of the pipeline. So before moving forward, we integrated the Azure CLI in the classical editor of our AMER model CI pipeline by writing the inline installation script.

2.9.3 Creating AML Workspace

The next step was to make sure we have AML workspace already setup. AML workspace is a centralized hub which is utilized for maintaining and managing our AMER model artifacts including the AMER model itself and its dataset. Although we have already created an AML workspace when we ran our IaC pipeline but to follow the best practice, we also integrated it into the CI pipeline just in case it does not exist or is destroyed. We wrote an inline script to create an AML workspace but since it already existed it was successful by default.

2.9.4 Defining AML compute

A compute is a computational resource that will be utilized to execute workflows of our AMER model, especially its development and deployment. It will also ensure that these workflows are managed and executed properly. Therefore, we wrote another inline script for the creation of a compute resource. In this script we also defined the minimum and maximum number of the nodes that we want to utilize for horizontal scaling. Finally, to keep the cost low in the beginning, we defined the type of compute according to the basic standard and the number of idle seconds of compute before it scales down.

2.9.5 Defining the variables for CI pipeline

Since the inline scripts that we had defined contained variables therefore we needed to define them in the pipeline variables section separately. The table below shows the variables that we defined for our pipeline. The first three variables defined were utilized in the inline script to create AML workspace and the rest of the variables defined were used to create a computational resource.

Table 2: AMER model CI pipeline variables

Name	**Value**
azureml.resourceGroup	NER-Model-Masters-Thesis
azureml.workspaceName	ner-model-ws-1
azureml.location	germanynorth
amlcompute.clusterName	nermodelcompute
amlcompute.minNodes	0
amlcompute.maxNodes	2
amlcompute.vmSize	STANDARD_E96ADS_V5
amlcompute.idleSecondsBeforeScaledown	300

2.9.6 Registering the dataset

The data required for the development of the AMER model is stored in our Azure DevOps repository, but we need to upload it to the AML where it can be stored and registered. This data can be uploaded manually using an AML interface, but we need to automate this task and add it as a step in our AMER model CI pipeline. Automating this step will save us a lot of extra workloads every time a developer or a data scientist have modified the data and won't have to manually register the data to AML.

Since we already had Azure CLI installed we wrote an inline script in Azure DevOps that uploaded a registered data on the AML. We also specified in the script that if the data already exists in AML than create a new version and overwrite that previous AMER model data. As traceability is one of the best practices of MLOPs offered by cloud-based solutions therefore, we can see all the versions of the data that we have uploaded to AML.

2.9.7 Running the CI pipeline

Finally, we were able to successfully execute our AMER model pipeline and each step that we had defined was running in a sequential manner. In this pipeline each step was dependable on the next

step so if one step failed all the CI pipeline execution is failed. The below figure 13 shows all the steps defined and successful execution of our AMER model CI pipeline. After this, we navigated to AML Studio where we were able to not only cross verify the creation of the workspace and a compute but also that our data is also registered on AML. Finally, this automated CI pipeline helped us save a lot of time and avoid errors for the first-time execution and will continue to help us whenever we want to have modification of the AMER model.

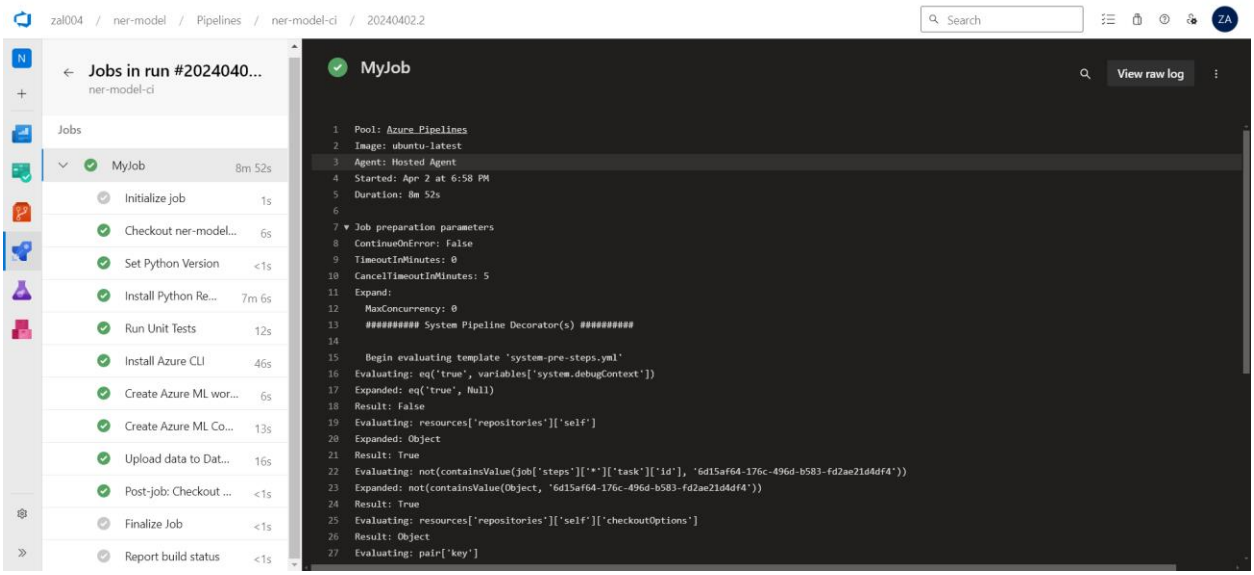

Figure 13: CI pipeline of AMER model on Azure DevOps

2.9.8 Make a model directory

Our last step before the training of model was to make a dumping directory in which we can store our trained model while the pipeline is executing. The directory can be used to grab our model, register it in AML and then deploy it. So, we added one more step in our continuous integration pipeline and wrote an inline bash command that creates a directory for us.

2.10 Training the AMER model

Our most important step in the continuous integration was training the model using an automated pipeline. We used an inline command to train our AMER model using Azure CLI, which we have already installed in our pipeline. Before running the command, we needed to make few changes in our AMER model training script to make it more optimized and run in Azure environment. For example, we made some changes in the script that were not following Don't Repeat Yourself (DRY) principle and were prone to errors for azure environments. In this way we were able to make the script more clean, maintainable and follow best practices. Finally, we added two variables in our pipeline which defined the name of experiment and the name of the model that will be trained in AML.

2.10.1 Running the training script

Upon running the training script, we faced our first challenge in the execution of CI pipeline. The default time limit of each pipeline that runs in Azure DevOps is 60 minutes but the AMER model that we were training took longer than 60 minutes. We have been using a very minimum low costing compute, which costed us $0.30 per hour, for our pipeline job, so initially we believed it was due to low levels of compute. To fix the problem, we tried to increase the compute level by changing it with a high-cost compute of $0.80 per hour. So, when we tried to execute the pipeline with more powerful compute, we still faced the same problem. When we further investigated the issue, we found that we can increase the execution time of the pipeline by utilizing Microsoft-hosted agents, but it comes with the cost of $40 per month [40]. We knew that the recommended best practice of MLOps was to automate the training of the model, but as our goal in this thesis was to keep the cost minimum so we decided to train the AMER model manually using AML notebook. AML provided an integrated notebook environment which we utilized for development of AMER model.

To start training the model manually we started by linking our AML environment with Visual Studio Code, which is an integrated development environment. This method helped in keeping the training process very smooth. As the data required by the model was already in the same repository, so we were able to successfully train the model in AML.

2.10.2 AMER model performance monitoring

We had initiated the process of performance monitoring when we were training the AMER model in the cloud and included scripts to record few important metrics. Therefore, to evaluate the effectiveness of the AMER model in the cloud, we recorded the following metrics:

- **Accuracy:** It measures the proportion of accurately identified entities compared to the total number of entities in a dataset.

- **Precision:** It can be calculated by dividing the total number accurate anticipated positive results by the total number of positive results predicted by the model. In medical systems, high precision is important to avoid incorrect medications, side effects or diagnosis.

- **Recall:** It is calculated by dividing the total number of true positive results in a dataset by the number of actual positive findings. High recall is important in medical contexts to prevent important diagnoses are not missed.

- **F1 score:** It provide us with a balance between precision and recall by taking the harmonic mean of both. F1 score is helpful tool when these two criteria are at odds.

These metrics not only helped us in getting insights about the AMER performance in the cloud, but also helped us in the evaluation part of this thesis

2.10.3 Registering the model

Model registration is an important step in machine learning workflows as it enables the trained model to be versioned and tracked in a centralized environment. We automated this part by writing a script that registered the AMER model in AML. Before running the pipeline, we defined the name of the model in a variable storage to enable a secure CI process. The script registered the AMER model using Azure CLI register command where it is named and versioned.

2.10.4 Downloading the latest version of AMER model

Once the AMER model is registered in AML it can be easily accessed for different operations such as testing and deployment. We wrote an inline script in our main pipeline file using Azure ml download command that allowed us to fetch the registered AMER and store it in the model directory that we have created earlier. One of the important things that we wrote in this script was to download only the latest version of the model that is registered in AML. This helped us by making sure that only the latest version of AMER are available for various operations.

2.10.5 Copy multiple directories

We also automated the process of coping multiple directories using an inline script. This step was important, as these multiple directories, which contained all the files of the AMER model and its configuration, made sure are copied to a temporary location. This temporary location was then used in our next step to fetch the copied files. This task copied all files, which were of 6 Megabyte, that we wanted to utilize in our staging pipeline. The files what we copied were the AMER model file, python scripts like score.py, and some important configuration files. Automating this task enabled us with flexible and centrally available file system for the testing and deployment stage of AMER model.

2.10.6 Publish pipeline artifacts

Finally, we utilized the publish pipeline task in Azure DevOps to automate the publishing of our multiple copied directories' artifacts by fetching them from the temporary location as done in the previous step. This final step was important as it made sure all our relevant folders and artifacts, which we prepared earlier, were made available for the CD pipeline

Overall, this whole process of CI pipeline provided us with consistency, reduced manual intervention and ensure smooth implementation of all the pipeline steps.

2.11 Continuous Deployment Pipeline

After successful creation of the CI pipeline, our next step was to create a CD pipeline in Azure DevOps to test and deploy AMER model as an endpoint in a staging environment. As part of our MLOps based integration design shown in figure 4, the goal of this CD pipeline was that every change that we will make in the codebase is tested automatically and then deployed into an environment where we can review it. Like our CI pipeline, in our CD pipeline we had to define certain steps which will lead of the deployment of the AMER.

The figure 14 below illustrates a complete staging pipeline in Azure DevOps, showing each step involved in the deployment process.

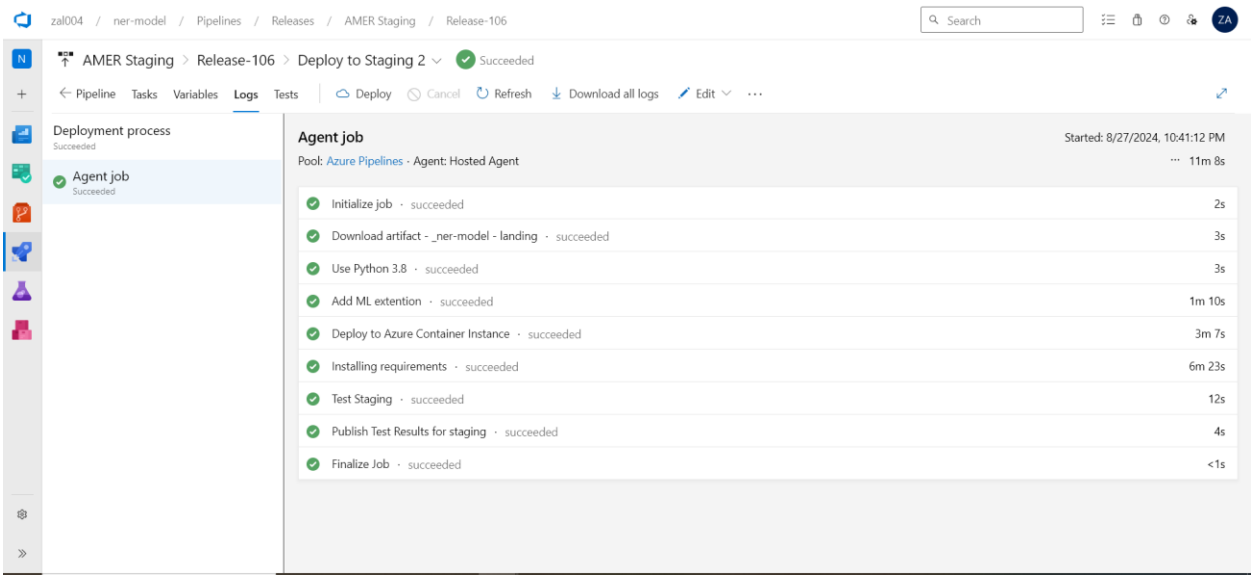

Figure 14: Compete CD staging pipeline of AMER model on Azure DevOps.

The first thing that we did when we started to create the release pipeline was to point out the artifacts that we build in our CI pipeline. By doing so, our release pipeline would know which script to take as input when it will execute. Next, we gave a name to our release pipeline and specified the agent on which our release pipeline would run. We selected Ubuntu operating system as the agent due to its stability and reliability.

2.11.1 Define Python Version

In the first step of our CD pipeline, we specified the python version to make sure there is consistency and compatibility across all the tasks that were to be carried out. This step was also important because Python is the main programming language that is used for running ml scripts and different version of Python can have different level of support of libraries. We specified the Python version as 3.8 to make sure that the environment that will be used in staging is the same as the one used during the CI pipeline, preventing issues related to version compatibility.

2.11.2 Add AML extension

In the second step, we added AML extension into our pipeline. This extension is important cause it provides all the commands and tools for interacting with AML directly thorough Azure DevOps. By adding this extension, we made sure that our pipeline can perform various tasks such as testing and deployment of AMER model. Without this extension, these tasks would require a lot of manual work and additional scripts, which could make the task more complex and prone to errors.

2.11.3 Deploy to Azure Container Instance

In the next step, we added the task to deploy the AMER model to Azure Container Instance (ACI). In this task we build a temporary environment for hosting the AMER model and ran requests on it. This process used Azure CLI to automate this task and make sure that the AMER model and its required configuration files are deployed correctly across environments.

The inline script we wrote deployed the specific version of the model into a specific workspace of Azure. We also configured the deployment by defining two scripts, one for container resource requirements and other for specifying the runtime environment. The first script creates a container for AMER, allocating with 1 Central Processing Unit (CPU) and 1 Graphics Processing Unit (GPU) with memory limit of 2 Gigabytes (GB) and 8 GB respectively. The second script describes the python as a runtime environment and points to a scoring script, which contains the logic for running prediction requests and to another file that contains configurations for that scoring file.

This process was important as it allowed us to validate the functioning of the model before we move it to the production. Additionally, the ACI provided us with a scalable, on-demand environment, which is cheap compared to Azure Kubernetes Service.

2.11.4 Installing requirements

In our fourth step, we installed all the dependencies required by the AMER model by executing a shell script. The script first validated the Python version, installed various packages related to Azure and then installed all the essential libraries of the AMER, including spacy, and pandas. This task was important for setting up an environment with the required packages and to support the proper functioning of the model and its supporting scripts.

2.11.5 Running Unit Test for staging

Moving to the next part we focused on running unit tests in our pipeline. The reasons for writing these unit test was because it allowed us to catch the bugs and issues in the last stage of deployment. This task ensured that the AMER deployed in the staging is validated by checking its response to a predefined test case.

We wrote two scripts to support the execution of our unit test. Our first script provided custom pytest fixtures and made sure the correct scoring URL is passed to the test case [41]. Our second script described the actual unit test by sending a POST request to the AMER scoring URL using a sample input data. This test made sure that the response returned by the request is successful and is in the correct JavaScript Object Notation (JSON) format.

2.11.6 Publish Test Results for staging

In our last step, we included a task in our pipeline that published the results of our unit test in a form of an Extensible Markup Language (XML) report. By posting these results in a format we made sure that they can be reviewed properly. The test results included information about the number of tests, their statuses and error messages if the test failed. This step played a vital role in our pipeline by enforcing accountability and transparency, allowing us to easily identify bottlenecks in our deployment cycle.

2.11.7 Manual Testing

Finally, our staging pipeline for AMER was successfully executed. Every step, from defining the python version to deploying the model in an ACI, running test cases and publishing the test results, was carefully arranged to ensure the stability and readiness for production.

After our pipeline ran successfully, we had to manually confirm few things before we could move forward. We checked the unit test cases results to confirm everything ran as expected. The test case results as shown in the figure 15 below, showed the unit tests were passed without errors. Moving ahead, we logged into AML portal to check whether an endpoint had been created when the pipeline executed. We navigated to the endpoint section, and saw an endpoint was created by the service connection by the name that we had defined in the variable section of our staging pipeline.

We used the scoring URL provided in Endpoint section of AML for further manual testing of our model. We created an API call function that connects to the AMER's endpoint and merged this function into our local front-end application, which we had decoupled before. In our local application, we replaced the code where AMER was used from local repository with the new API call function. After that, we sent requests using dummy data to the deployed model from our front-end application. By utilizing different prediction functions of the application, we verified that the model was working as expected. This step ensured that the deployed model was integrated correctly and performing accurately in staging.

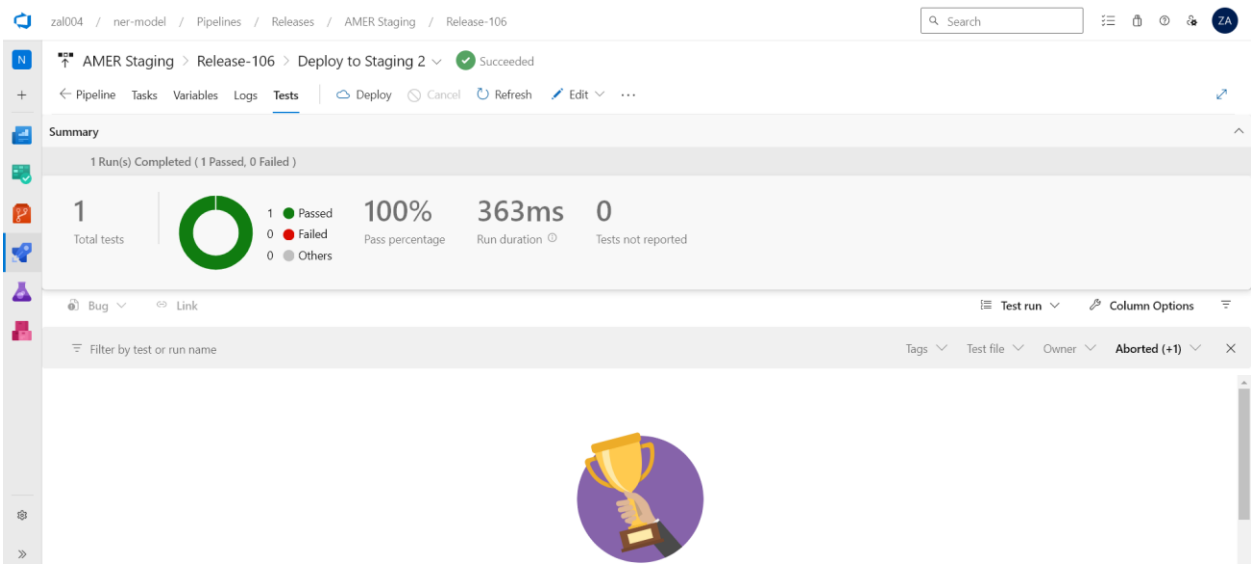

Figure 15: Unit Test Results of AMER Model on Azure DevOps.

2.12 Deployment to production

The deployment to production is a critical phase in the development lifecycle of a machine learning model. This is the stage, where the model, which has been thoroughly tested in a staging environment is transitioned to where it is ready to serve live requests. This section outlines the methodology used to deploy the AMER model in production and details the steps involved for a reliable production environment.

2.12.1 Creating environment in AML

In the process of our deployment to production, the first step was to create an environment in AML. This step was important for defining the configurations and dependencies that AMER model would require for proper execution.

To start, we navigated to the Environments section in AML and choose to create a new Docker image. Although we had other option available such as using an existing Docker context, but we went for our own docker context to create a fully customizable environment that matches the specific dependencies and versions required by AMER. This also allows us to have full control over the underlying libraries and configuration, which would not have been accomplished with an existing image.

We customized the Docker context by editing the image file provided by AML. We wrote the scripts to install key dependencies like azureml-core, azure-storage-blob and spacy. We also created a custom conda environment with Python and specified that the AMER model would be served using Gunicorn library through the AML inference server. We included azureml-core for integration with AML services and azure-storage-blob to get the data from Azure Blob storage during deployment. Similarly, Spacy was added for the AMER and Gunicorn to efficiently handle API requests in production.

When the environment creation was completed, AML initiated a job to build a docker image with the configuration that we had specified. The job ran successfully, confirmed that our image was built and ready for production deployment.

2.12.2 Creating a Production Endpoint

To deploy the trained AMER model, we navigated to the endpoint section of AML and selected the model and its required version.

We selected “Managed” as the compute type, but we also had another option which was Kubernetes. We chose Managed as it provides a simple way to deployment, including scaling and handling infrastructure. This allowed us to focus on our AMER model rather than managing the computed resources. For the selection of authentication type, we selected azure key-based because only the people with the required keys can access the AMER model’s endpoint. We also set the scoring Timeout to 60 seconds to make sure that the AMER model has enough time to process a request and return a prediction. This setting was important to prevent timeouts if we had to deal with more complex and larger input request. Then we selected the same scoring script that we used in our deployment pipeline which included the way in which AMER model should be loaded and utilized. We also set the custom environment that we created in the last step and set the selected the lowest compute to keep the cost as minimum as possible. Lastly, we adjusted the Live Traffic to 100%, which meant that all the incoming requests would be directed to this new endpoint that we had created. Doing this whole process allowed us to successfully create a production endpoint for AMER.

Once the AMER was in production, we had to manually make sure that everything was working fine. We utilized the Test section in AML by sending a JSON request to the AMER through an

interface and were able to simulate real-time input. This JSON contained the necessary input required by the model and as the input was submitted the model returned the expected results.

2.13 Simulation for integration of AMER model

We used front-end application of AMER as a simulation for its integration with AMER model. To make the simulation secure like in medical settings we performed the following steps.

Establishing Secure Transfer

When the AMER model was deployed in production it generated a secure endpoint in AML. This endpoint API facilitated secure communication between the front-end medical application and the AMER model. Azure Key based authorization and Secure Sockets Layer (SSL) were utilized to restrict the authorized users and keep the data encrypted during transfer respectively.

Private Endpoints and VNet were configured to confirm that all the communication between the AMER model and the local application occurred security and directly, bypassing other public networks. This solution minimized the risk of data exposure and ensured compliance with health regulatory standards.

Real-Time Processing

In clinical scenarios, where fast data extraction is required, the integration focused to simulate near-real time communication between the AMER model and the front-end application. To access the response time, we conducted few tests by sending request to the AMER model from our front-end application with different workloads. These tests calculated the time taken from when the request was sent to AMER model until the response was received. The focus in this was on achieving a response time of less than 2000 milliseconds per request. This limit was set to copy real-time hospital use, in which delays in data retrieval could affect timely decision-making process. Tools like Azure Monitor and Application Insights were utilized to record and track the response times which assured performance and identifying any latency.

2.13.1 Challenges faced

During this process, we faced few challenges including:

Latency in Real-Time Processing

The initial tests that we performed showed higher latency than expected, which could affect the ability of the system in the real-world medical settings. To solve this, the autoscaling functionality and additional computations of Azure were used to meet the demand.

Compatibility with Local Systems

The local application which acted as a medical infrastructure, needed to sync with the output of AMER model. This required to implement a communication function which correctly parsed the JSON output from the AMER model.

2.13.2 Solutions Implementation

To address the challenges, the following solution were implemented.

Latency Optimization

To resolve the latency challenges, the AMER model was deployed to a local region closer to the user base. Other ways included utilizing Azure caching mechanism to speed up the frequent operations.

Interoperability

A flexible API function was designed to address this problem. This solution made it possible that any data input to the local application could be processed by the AMER model efficiently and the response was returned in a proper structured format.

3 Evaluation and Results

In this section, we discuss the evaluation of the cloud-based implementation and deployment of the AMER model using MLOps driven CI/CD pipeline. The evaluation covered the main elements of the AMER model integration and focused on the following questions:

- **AMER Performance and System Latency:** How well does cloud-based AMER model operate in terms of performance and system latency?
- **Operational Scalability:** How does the AMER model CI/CD pipelines implementation scales with different workloads?
- **Deployment Time and Cost Effectiveness:** What is the deployment time and cost-effectiveness of automation on a cloud-based system for AMER model?

3.1 Model Performance Evaluation

For the evaluation, AMER was trained and tested locally on a computer to set up a baseline performance. This baseline provided us a point of comparison to analyze how well the AMER model performed in different environments. Furthermore, the evaluation metrics that we selected helped us access to find if the cloud deployment had affected the performance of AMER model. The table below shows the acceptable threshold for each of the metric.

Table 3: Acceptable threshold limit for different metrics.

Matric	**Threshold Limit**
Accuracy	Above 90%
Precision and Recall	Above 85%
F1-score	Above 87%

After noting the metrics from local environment, we compared the local and cloud environment results to find out differences in model performance. It focused on if the cloud deployment affected the model's capabilities to recognize medical entities with the same accuracy, precision, recall and F1 score that we noticed in the local environment.

3.1.1 Results

The table below compares the performance of AMER model in both, local and cloud environments across different metrics. Each metric was recorded with multiple executions and then averaged to minimize the impact of random variations. Confidence intervals are also shown to point out variance across executions and any performance instability. The variance was important for

understanding the reliability of AMER model in each environment as it indicates how much results change due to factors like network latency and data processing.

Table 4: Performance Metrics of AMER in local and cloud environments.

Metric	**Local Environment**	**Cloud-Based Environment**
Accuracy	91.8% ± 0.3%	90.4%± 0.5%
Precision	92.5% ± 0.2%	91.1%± 0.4%
Recall	89.7% ± 0.4%	88.2%± 0.6%
F1 Score	91.1% ± 0.3%	89.6%± 0.5%

After integrating the AMER model with a front-end application and performing optimizations, we were able to successfully simulate a secure medical setting. As shown in the table below, these optimizations led to reducing latency by over 50% to bring it within acceptable range for real-time medical data processing.

Table 5: Latency Comparison of AMER model (Pre vs Post Optimization)

Integration	**Before Optimization**	**After Optimization**
Average Latency (milliseconds)	450	180

3.1.2 Discussion

In both environments, AMER had a high performance across the metrics that we included. Compared to the local environment, the lower performance of cloud was expected due to latency of remote data processing. However, this difference was small and do not impact the performance and effectiveness of AMER.

Accuracy in the local environment was a bit higher than in the cloud, showing the low latency and immediate data access when working locally. In the cloud environment, a delay was observed which resulted in lower accuracy score. However, the difference of 1.4% between both environments is small and unlikely to have a major impact in real-world applications.

Precision and Recall were high in both the environment but a small decrease in the cloud was observed. This shows that while the AMER remained strong, there was a bit performance tradeoff when deploying it to the cloud. These tradeoffs arose due to the delay in data transmission or increased computation overhead. Despite these reductions, the cloud deployment of AMER still maintained high precision and recall.

F1 Score showed the difference seen in precision and recall but with a low score in the cloud environment. This shows that while the cloud environment has challenges, the overall balance between false positives and false negatives remain strong in both environments.

The minor reduction seen in the cloud environment could be resolved in future by optimizing the AMER for cloud-based processing or by improving communication methods between the cloud and the clinical environment. These fixes could help in reducing latency and further align performance with local environment.

Finally, high latency presented a crucial challenge to achieve real-time data processing, which is important for decision making in hospitals. The reductions achieved after optimizations showed the effectiveness of cloud infrastructure in providing low latency solutions for ML in medical systems. Meeting the target for latency threshold shows that with proper deployment strategies, the models hosted on cloud can offer the results needed in hospital settings without compromising performance.

3.1.3 Implications and Recommendations

The operational benefits of deploying the AMER model in a cloud-based environment like scalability, ease of integration with MLOps best practices and the ability to handle large data outperforms the minor performance tradeoffs.

Improving the ability of AMER in handling data communication and processing in the cloud help minimize latency and increase the performance metrics like F1 score and recall. This could involve optimizing the architecture of cloud or utilizing edge computing techniques to reduce impact of remote data processing.

In a real medical setting, the language used in medical documentation may change over time. Therefore, using model drift detection tools in the cloud deployment will be important for identifying any performance degradation over time.

3.2 Operational Efficiency and Scalability

This subsection evaluates the AMER model's implementation operational efficiency and scalability by observing aspects including cloud scalability, resource management and deployment time. During the process we assessed, how well the CI/CD pipelines improved operational processes and supported scalable deployments.

3.2.1 Results

The results that we got showed several improvements that were brought by using cloud environments and implementing CI/CD pipeline for AMER model.

Each scenario was tested based on CPU Utilization, cost per hour and if auto-scaling was required or not. The table below shows how different workload affected these metrics for AMER. As the number of requests per second increases, the utilization rate of CPU increases, demanding additional computation resources. This increases the cost per hour as the cloud services charges

on the resource usage. For medium and high workloads, the system automatically scaled resources to keep performance and enabled that it can handle high demand.

Table 6: Operational Efficiency and Scalability of AMER with different workloads.

Workload	**CPU Utilization**	**Cost per Hour (USD)**	**Resource Scaling Time**
Low (10 requests/second)	15%	1.20	Not required
Medium (50 requests/second)	35%	3.80	Scaled automatically
Medium (100 requests/second)	65%	6.40	Scaled automatically

3.2.2 Evaluation Methodology

This evaluation was carried out in several steps, which are relevant to the deployment process and cloud infrastructure's ability to scale.

Computational Resources and Scalability of AMER

The deployment of AMER model in AML was analyzed to check its scalability under different workloads. We simulated increased workloads by testing the system with inference rate of 10, 50, and 100 per second. To measure the scalability of AMER we recorded the following metrics:

- **Resource Allocation** is the amount of CPU and other resources allocated by the cloud provider under different workloads.
- **Cost per hour** is estimated hourly cost of operating the AMER model under different workloads.
- **Resource scaling Time** is the time taken by the cloud service provider to allocate more resources to meet the high demand when workload increases.

3.2.3 Discussion

The integration of CI/CD pipeline and deployment using cloud has enhanced the operational efficiency and scalability of AMER model. These improvements are important for maintaining performance in real-time applications.

The automatic scaling of AMER not only enabled that the system can meet performance needs during high demand but also minimized unnecessary costs during low interest periods by avoiding over provisioning. This difference with static infrastructure, in which resources are allocated based on high interest results in inefficiencies and higher costs when there is less demand.

The ability to balance performance and costs is important in cloud deployments specifically in computationally intensive applications like ML in healthcare. The AMER model's capacity to

scale in response to peak demand while maintaining low cost is an indicator of usefulness of cloud-based deployments for these applications.

Finally, for future implementations, further optimization of auto-scaling, resource management and automating AMER model training process can improve cost savings and efficiency.

3.3 Deployment Time and Cost Effectiveness

To evaluate the deployment time and cost of maintaining and deploying the AMER model on the cloud, we reviewed initial costs, computational prices associated with CI/CD and ongoing maintenance expenses.

In the beginning, the manual deployment time of AMER took 2-3 hours per deployment. After automation using the CI/CD pipeline, the deployment time was reduced to 2-3 hours per deployment. This reduction not only speeds up the release time but also reduces the risk of human errors.

3.3.1 Cost Analysis

The cost analysis that we performed included three components:

1) **Initial cost for setup:** This was the cost that came from configuring the cloud environment, creating the IaC infrastructure and implementing the CI/CD pipelines.

2) **Ongoing Maintenance costs:** This is the monthly expense which is associated with storage, compute resources and the services that are important to maintain the AMER model in the hospital environment.

3) **Operational costs for CI/CD and computation:** This is the costs for data processing, model inference, executing automated pipelines and real-time monitoring.

Table 7: Initial and Monthly Cost of cloud-based AMER model.

Component	Initial Cost (USD)	Monthly Cost (USD)	Description
AML	100	56	Used for AMER model training, managing and versioning.
AAS	30	19	Hosts AMER model API and front-end application.
Azure Blob Storage	20	15	Used to store data related to AMER model.
Azure DevOps	70	32	Used for automated testing and deployment of AMER
Computes	300	90 - 170 (low – high demand)	Used for data processing, model inference and workloads.
VNet	25	10	Secures Data communication of AMER by adding a secure layer.
Real-Time Monitoring	20	26	Provides real-time tracking of AMER model and computes.
Network Costs	5	14	Cost of data transfer between Azure services.

The details of the cost by each component of Azure are shown in the table above. The total of the initial cost was $570 while the ongoing monthly operational cost was $342 in high workload period and $262 in low workload conditions.

3.3.2 Discussion

After the implementation of CI/CD pipeline of AMER the deployment time reduced by more than 75%. This timesaving was due to the automation of the tasks that were previously performed manually, AMER model's configuration, validation and deployment. The use of automated pipelines enabled that the new version of AMER can be deployed more easily by allowing fast iterations and quick response times for model improvements. The reduction for manual interference also reduces the risk of human error and makes the deployment more reliable.

Cost analysis showed high initial setup cost but was followed by flexible monthly costs that changes with demand. When the demand is high, the auto scaling adjusts resources as required, which increased the costs but keep consistent performance. Although the use of VNet and real-time monitoring added monthly costs yet it improved security, system reliability and regulatory

compliance, which are important for health applications. On the other hand, in the low demand periods expenses reduced and made the system cost effective as compared to in the house servers, where auto scaling is complex to achieve and leads to more cost.

Overall, these results shows that using cloud infrastructure can effectively balance costs, deployment times, scalability and security, which makes it can ideal choice for medical ML applications.

4 Integration of AMER Model in Hospital Infrastructures

In related work, an important project is the integration of AMER core system into a hospital environment. This project is currently under development, and it aims to facilitate the deployment of NER models in hospitals by focusing on secure in-house data processing rather than the cloud. This system is designed with a microservice architecture that focuses on compatibility with existing medical infrastructure by integrating with DIPS in Norwegian Hospitals. DIPS runs on secure, in-house servers within hospital servers and uses message brokers for data processing. This configuration aligns well with the hospital's requirements for data privacy and compliance with strict health data regulations as the data remains secure in an internal network.

While this architecture supports secure on-premises deployments and enable interoperability with DIPS and similar infrastructures, it does not cover the full range of MLOps abilities. The current focus of this work is on integration rather than the complete lifecycle of ML models. Although this system includes retraining of AMER, the system lacks continuous integration, automated testing and deployment and real-time monitoring. Without these aspects, scaling the AMER model, maintaining its high availability and establishing reliability becomes challenging over time when retraining and updates are needed in hospital settings.

This thesis uses a cloud-based approach to fully operationalize the AMER model using MLOps best practices. This approach offers advantages, like flexible storage, automated deployment pipelines, scalable compute resources and monitoring tools. These benefits contribute to more efficient, faster and reliable model deployments and updates. By using the cloud service, this method simplifies the automation of model management processes, enable real-time monitoring and reduces the deployment of time.

To bridge the gap between the in-house AMER system in hospital and automation and scalability offered by the MLOPs in the cloud, a hybrid approach is needed in medical environments. Here we outlined a path forward in which hospital-based information systems could implement MLOps principles in a secure and compliant manner.

One solution for this is a hybrid infrastructure in which data remain secure in the hospital in-house data center and model training and experimentations are conducted in a secure cloud environment. In this way, patients' data will be anonymized locally before any interaction with the cloud. This will allow hospitals to have control over sensitive data and computationally intensive task offloaded to the cloud. The trained AMER model can then be securely transferred back to the hospital servers for integration and deployment with the local system.

For the hospitals that are reluctant about public cloud usage, private cloud could also be used. Public cloud can be deployed in a hospital infrastructure that would help hospital to use MLOps pipelines and services and maintain data privacy. Edge computing, where processing happens close to the source of data, can offer real-time processing benefits for AMER without fully relying on the cloud. These edge devices would run containerized versions of AMER and would process data at the level of hospital while synching with centralized MLOps systems for maintenance and updates.

To clone the benefits of automated pipelines in the cloud MLOps in a secure setting, in-house CI/CD tools can be installed that are compatible with hospital infrastructure. In this case using tools like Jenkins and Azure DevOps Server (in-house version of Azure DevOps) would allow hospitals to automate different deployment stages in a secure network. This method will enable that AMER model updates can be automatically tested before deployment and any performance issues can be identified and addressed quickly.

For hospitals implementing local monitoring for ML applications solutions like Prometheus and Grafana can be used to track model performance, resource usage, data flows and latency in real-time. Alerts can also be configured for any errors in the system health or AMER model output. Finally, logging tools like Elasticsearch, Logstash and Kibana can help hospitals examine long-term trends in AMER model performance and support improvements.

By adopting MLOps best practices for hospital environments and following compliance regulations, we can move ahead to a secure, scalable and efficient solution for deploying ML models in healthcare.

5 Conclusion

This thesis analyzed the development, deployment and operationalization of AMER model, a system designed for medical entity recognition in hospital environments. By using MLOps best practices, this thesis aimed to build an automated, efficient and scalable deployment system on a cloud infrastructure and provided insights into operational and technical challenges of implementing ML in medicine.

The AMER model, which was initially build as a local solution was revamped for deployment on Azure by using an MLOps based pipeline to automate ML model lifecycle. The implementation showed substantial improvements in deployment time, reliability and cost effectiveness by reducing the deployment time from hours to minutes through CI/CD automation. Moreover, the cloud system allowed for dynamic scaling of AMER and made the model adaptable to changing workloads, which is common in healthcare settings.

A cost analysis showcased the financial impact of this deployment and balancing initial cost with ongoing operational expenditure. This evaluation highlighted the value of cloud resources especially in high computes demand, in which auto scaling and resource management allowed efficient cost allocation. However, the thesis also acknowledged the tradeoff linked with cloud deployments in terms of cost and scalability.

Alternatively, to previous approach, which is limited to in-house systems, this thesis also provides a future pathway for using MLOps techniques into secure hospital environments. By using hybrid architecture and private clouds, advantages of MLOps can be gained while enabling data protection and compliance with regulations. This approach bridges the gap between ML best practices and the security needs of health data.

The findings in this thesis shows the potential of MLOps workflows to simplify and secure the deployment of NER models like AMER in hospital settings. By reducing deployment time, improving scalability and maintaining model accuracy and performance, this thesis contributes to developing ML in healthcare, preparing the way for secure and more reliable AI-driven systems in hospital settings. Future study should focus on making further improvements in secure MLOps architectures designed for hospitals by allowing integration of ML technology and maintaining data integrity and patient privacy

6 Appendices

6.1 Declaration of the Usage of Artificial Intelligence

For this master's thesis, I acknowledge the use of generative artificial intelligence from the below tools and for the following purposes in the report and implementation.

1) (https://app.grammarly.com)
2) (https://gemini.google.com/app)
3) (https://chatgpt.com)
4) (https://copilot.cloud.microsoft)

Purposes include:

1) To improve the text and correct the grammatical mistakes.
2) To maintain the academic style and tone in the text.
3) To enhance the clarity of explanations in the text.
4) To get suggestions for structuring content.
5) To fix the bugs in the code.
6) To get suggestions on Azure specific code. For example, Azure CLI Commands.

7 Reference List

[1] D. Khurana, A. Koli, K. Khatter and S. Singh, "Natural language processing: state of the art, current trends and challenges," *Multimedia Tools and Applications,* vol. 82, no. 2023.

[2] A. Pathak, "Named Entity Recognition (NER) Explained in Layman's Terms," 2023. [Online]. Available: https://geekflare.com/named-entity-recognition/.

[3] P. Bose, S. Srinivasan, W. C. S. IV, J. Palta, R. Kapoor and P. Ghosh, "A Survey on Recent Named Entity Recognition and Relationship Extraction Techniques on Clinical Texts," *Applied Sciences,* vol. 11, 2021.

[4] T. Nguyen, "9 Significant MLOps Challenges and Lessons Learned," [Online]. Available: https://www.neurond.com/blog/mlops-challenges-solutions.

[5] Datacamp, "The Past, Present, and Future of MLOps," 2021. [Online]. Available: https://www.datacamp.com/blog/the-past-present-and-future-of-mlops.

[6] R. M. Ratwani, J. Reider and H. Singh, "A Decade of Health Information Technology Usability Challenges and the Path Forward," 2019. [Online]. Available: https://sci-hub.se/downloads/2019-02-05//d0/10.1001@jama.2019.0161.pdf.

[7] K. Salama, J. Kazmierczak and D. Schut, "Practitioners guide to MLOps: A framework for continuous delivery and automation of machine learning.," 2021. [Online]. Available: https://services.google.com/fh/files/misc/practitioners_guide_to_mlops_whitepaper.pdf.

[8] A. Bodor, M. Hnida and D. Najima, "MLOps: Overview of Current State and Future Directions," *Lecture Notes in Networks and Systems,* vol. 629, 2023.

[9] M. Senapathi, J. Buchan and H. Osman, "DevOps Capabilities, Practices, and Challenges: Insights from a Case Study," *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering,* vol. Part F137700, pp. 57-67, 2018 .

[10] D. Kreuzberger, N. Kühl and S. Hirschl, "Machine Learning Operations (MLOps): Overview, Definition, and Architecture," 2022. [Online]. Available: https://arxiv.org/abs/2205.02302.

[11] G. Symeonidis, E. Nerantzis, A. Kazakis and G. A. Papakostas, "MLOps -- Definitions, Tools and Challenges," in *In Proceedings of the 12th IEEE Annual Computing and Communication Workshop and Conference (CCWC 2022) (pp. 1-6). IEEE.,* 2022.

[12] V. Moskalenko and V. Kharchenko, "Resilience-aware MLOps for AI-based medical diagnostic system," 2024.

[13] H. IT, "What is an electronic health record," [Online]. Available: https://www.healthit.gov/faq/what-electronic-health-record-ehr.

[14] C. Wen, T. Chen, X. Jia and J. Zhu, "Medical Named Entity Recognition from Un-labelled Medical Records based on Pre-trained Language Models and Domain Dictionary," 2021.

[15] I. Pilán, P. H. Brekke and L. Øvrelid, "Building a Norwegian Lexical Resource for Medical Entity Recognition. To appear in Proceedings of the 2nd workshop on Multilingual Biomedical Text Processing (MultilingualBIO)," 2020.

[16] J. Schjøtt, L. Reppe and P. Roland, "A question–answer pair (QAP) database integrated with websites to answer complex questions submitted to the Regional Medicines Information and Pharmacovigilance Centres in Norway (RELIS): a descriptive study," 2012.

[17] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.," 2017.

[18] M. Askar, "Automated medical entity recognizer for Norwegian language text". A Disclosure Of Invention (DOFI) application submitted to The Arctic University of Norway - UiT. Unpublished.," 2023.

[19] J. Heddes, P. Meerdink, M. Pieters and M. Marx, "The Automatic Detection of Dataset Names in Scientific Articles," 2021.

[20] Microsoft, "Subscriptions, licenses, accounts, and tenants for Microsoft's cloud offerings," 2023. [Online]. Available: https://learn.microsoft.com/en-us/microsoft-365/enterprise/subscriptions-licenses-accounts-and-tenants-for-microsoft-cloud-offerings?view=o365-worldwide.

[21] S. Singh, K. R. Ramkumar and A. Kukkar, Analysis and Implementation of Microsoft Azure Machine Learning Studio Services with Respect to Machine Learning Algorithms., Springer, Singapore, 2023.

[22] M. Learn, "Medical data storage solutions," [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/solution-ideas/articles/medical-data-storage.

[23] Microsoft, "What is Azure DevOps?," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?toc=%2Fazure%2Fdevops%2Fget-started%2Ftoc.json&view=azure-devops.

[24] marcmercier and mauro-msft, "Azure App Service Patterns and Features for the Azure Well-Architected Framework," 2022. [Online]. Available: https://techcommunity.microsoft.com/t5/fasttrack-for-azure/azure-app-service-patterns-and-features-for-the-azure-well/ba-p/3696156.

[25] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah and P. Merle, "Elasticity in Cloud Computing: State of the Art and Research Challenges," 2018.

[26] B. Eken, S. Pallewatta, N. K. Tran, A. Tosun and M. A. Babar, "A Multivocal Review of MLOps Practices, Challenges and Open Issues," 2024.

[27] S. Lehrig, H. Eikerling and S. Becker, "Scalability, Elasticity, and Efficiency in Cloud Computing:a Systematic Literature Review of Definitions and Metrics," 2015.

[28] Microsoft, "Manage Azure resource groups by using the Azure portal," 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/manage-resource-groups-portal.

[29] "Streamlit • a Faster Way to Build and Share Data Apps," [Online]. Available: https://streamlit.io/.

[30] P. D. Tender, "Azure DevOps vs GitHub: Which DevOps Tool Should You Choose?," 2023. [Online]. Available: https://www.sitepoint.com/azure-devops-vs-github/.

[31] Micorsoft, "Connect to Azure by using an Azure Resource Manager service connection," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/devops/pipelines/library/connect-to-azure?view=azure-devops.

[32] Microsoft, "Deploy and manage resources in Azure by using JSON ARM templates," [Online]. Available: https://learn.microsoft.com/en-us/training/paths/deploy-manage-resource-manager-templates/.

[33] D. Rendón, Building Your Environment with Azure DevOps and ARM Templates, Apress, Berkeley, CA, 2022.

[34] D. Bundor, "Using ARM Template to Deploy an Environment in Azure," 2023. [Online]. Available: https://medium.com/@danielbundor91/using-arm-template-to-deploy-an-environment-in-azure-c3bf101bbb03.

[35] Datascientest, "Azure DevOps Pipeline YAML: why configure CI/CD pipelines with YAML?," 2023. [Online]. Available: https://datascientest.com/en/azure-devops-pipeline-yaml-why-configure-ci-cd-pipelines-with-yaml.

[36] Microsoft, "Use Azure Pipelines with Azure Machine Learning," 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/machine-learning/how-to-devops-machine-learning?view=azureml-api-2&tabs=arm.

[37] K. L. Curve, "CONTINUOUS INTEGRATION AND DEPLOYMENT WITH AZURE DEVOPS PIPELINES," 2023. [Online]. Available: https://kenslearningcurve.com/tutorials/continuous-integration-and-deployment-with-azure-devops-pipelines/.

[38] S. Garg, P. Pundir, G. Rathee, P. Gupta, S. Garg and S. Ahlawat, "On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps," 2022.

[39] Microsoft, "Set up MLOps with Azure DevOps," 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/machine-learning/how-to-setup-mlops-azureml?view=azureml-api-2&tabs=azure-shell.

[40] Microsoft, "Microsoft-hosted agents," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/devops/pipelines/agents/hosted?view=azure-devops&tabs=yaml.

[41] P. Documentation, "pytest fixtures: explicit, modular, scalable," [Online]. Available: https://docs.pytest.org/en/6.2.x/fixture.html.