DevOps for advanced manufacturing: accelerating innovation and fostering technological adoption

Teboho Sekopa[1], and *Mamodike* Sadiki[1]

[1] Optronic Sensor Systems, Council for Scientific and Industrial Research, South Africa

Abstract. This paper presents a microservices architecture platform built on open-source DevOps tools to support advanced manufacturing workflows. By using DevOps tools, this paper provides a robust, portable, and cost-effective prototype for robotics, AI/ML, Web Services, and other manufacturing applications. The presented platform accelerates innovation by enabling rapid experimentation and collaboration while promoting technological adoption through ease of use and reliability. We discuss this system's implementation, benefits, and potential applications in overcoming the disparity between industry and academia.

1 Introduction

In today's rapidly evolving technological landscape, the manufacturing sector is undergoing a significant transformation, driven by the need for increased efficiency, agility, and innovation. Advanced manufacturing integrates cutting-edge technologies and processes to deliver high-quality products with speed and cost-effectiveness [1]. Central to this transformation is the adoption of DevOps principles, which is proving to be instrumental in bridging the gap between development and operations, streamlining workflows, and accelerating the delivery of value in manufacturing. By fostering a collaborative culture, DevOps encourages empathy and cross-functional collaboration between development and Information Technology (IT) operations, ensuring resilient systems and the rapid delivery of changes [2]. The focus on automation, continuous improvement, and agile workflows enhances the development and deployment of advanced manufacturing solutions, accelerating innovation and driving technological adoption.

The integration of innovative technologies such as Artificial intelligence (AI), automation, software, and networking is at the core of advanced manufacturing. These technologies rely on software applications for data collection, analysis, and output, enabling the optimisation of manufacturing processes for increased productivity. Industry 4.0 has revolutionised manufacturing by promoting the networkability of production machines, laying the foundation for software-driven production control and real-time data collection. However, managing these software applications adds complexity, often resulting in higher costs and longer development cycles. To mitigate this, DevOps engineering is employed to automate repetitive tasks; and to consolidate development and operations into a single pipeline. This practice, while transformative, can face challenges such as inconsistent

environments, security risks, and scalability issues. These challenges can be addressed through software containerisation, which ensures consistent execution across various environments, such as embedded computers for robotics and high-end computers for AI models. Containerisation can reduce costs, enhance flexibility, and accelerate technological adoption in advanced manufacturing. This paper proposes a software engineering practice combining IT development and operations, using open-source DevOps tools, that enable scalable, portable, and cost-effective management of software applications; fosters innovation; and bridges the gap between industry and academia.

2 Background

2.1 DevOps in manufacturing

DevOps represents the integration of software development automation with IT operations, aimed at fostering a more collaborative and efficient software development lifecycle. It encompasses essential development and deployment tasks that shorten development cycles and improve deployment times, thereby enhancing the overall product lifecycle. In the context of manufacturing, DevOps principles support the planning, development, manufacturing, deployment, and serviceability of manufacturing instruments and systems. This combination of development and operations enables a seamless flow from design to production, ensuring that the software driving automation, machine learning, and data analytics is integrated smoothly into manufacturing environments.

The adoption of DevOps in manufacturing promotes greater efficiency and agility in the development and deployment of software applications that control production lines; monitor machine health; and optimise the overall manufacturing operations. As Industry 4.0 advances, manufacturing becomes increasingly reliant on networked, software-driven systems, which DevOps practices can help to streamline. By enabling faster release cycles and continuous feedback loops, DevOps ensures that manufacturers remain adaptable and capable of integrating new technologies as they emerge [3].

2.2 Containers

Containers are a lightweight, portable software component, that packages applications together with their dependencies, ensuring consistent performance across diverse environments [4, 5]. Unlike traditional virtual machines, containers share the host system's kernel while maintaining process isolation, and this significantly reduces overhead [4, 5]. Containers can run across a variety of systems, from physical servers to virtualised platforms, offering portability and flexibility that is crucial in manufacturing environments where different hardware configurations exist, such as embedded computers used in robotics or high-end performance systems required for AI processing [5].

The widespread adoption of containers has been a key driver in enhancing application lifecycle management, particularly in the context of Continuous Integration and Continuous Delivery (CI/CD) practices [5, 6]. Containers can facilitate rapid deployment, scaling, and management of applications in an efficient manner than traditional methods. In advanced manufacturing, containers can enable manufacturers to optimise resource utilisation, thus ensuring that software applications run effectively across different platforms, thereby reducing costs and enhancing system flexibility. Additionally, the use of containers supports the modular approach to system design, where applications can be independently updated, tested, and deployed [5]. Some container implementations align with open-source principles;

hence this makes them an attractive option for organisations seeking to avoid vendor lock-in.

2.3 Open-source DevOps tools

Successful DevOps implementation relies on the effective integration of the right tools to support the entire software development lifecycle. In advanced manufacturing, tools such as Jenkins, Docker, Kubernetes, Prometheus, and Terraform play an instrumental role in automating processes, improving scalability, and ensuring the reliability of software applications used in manufacturing operations. These open-source tools streamline workflows by automating routine tasks such as testing, integration, deployment, and monitoring, thus enabling faster development cycles and continuous feedback.

For example, Jenkins automates the continuous integration of code changes, and this ensures faster and more reliable software delivery. Kubernetes provides container orchestration that allows manufacturers to scale applications based on demand, while ensuring high availability and reliability [7]. Prometheus enables real-time monitoring, which is a critical feature for maintaining operational efficiency in the manufacturing setting, where downtime can lead to significant production losses [7]. Terraform allows for infrastructure as code, enabling manufacturers to automate the provisioning and configuration of infrastructure, further increasing the efficiency of software management [7]. This is not an exhaustive list of tools that can be used in CI/CD.

By leveraging these open-source DevOps tools, manufacturers can build flexible, scalable, and cost-effective systems that can adapt to the rapidly evolving landscape of advanced manufacturing. These tools reduce dependency on proprietary solutions, and they offer greater flexibility and reduce the risk of vendor lock-in. Combining and leveraging the right tools is essential to engineer a successful DevOps system that can be used in advanced manufacturing.

2.4 Advanced manufacturing applications

Advanced manufacturing is increasingly dependent on software applications to optimise production processes, enhance automation, and support innovation. The integration of cutting-edge technologies, such as robotics, AI, and Internet of Things (IoT), allows manufacturers to collect real-time data, analyse it efficiently, and use the insights to improve production outcomes [8]. The combination of DevOps practices and containerisation enhances the scalability, flexibility, and efficiency of these software applications, and thus enables manufacturers to stay ahead of industry demands and improve productivity.

By adopting DevOps methodologies and containerisation, manufacturers can overcome the challenges of deploying and managing complex software systems across various environments. Containers provide a consistent runtime environment across different hardware configurations, reducing compatibility issues and allowing for more agile deployment processes [8]. As manufacturing becomes increasingly digital, the need for seamless integration between legacy systems and modern software platforms becomes paramount [9]. Containers can facilitate this integration; and hence ensure that systems evolve without significant disruptions to existing infrastructure.

Overall, the application of DevOps principles, combined with the power of containerisation and open source tools, supports continuous innovation and accelerates the digital transformation in advanced manufacturing. These technologies allow manufacturers to optimise their workflows, improve operational efficiency, and embrace new opportunities for innovation in a competitive global market.

3 Methodology

This section outlines DevOps practices employed in the implementation the pipelines and materials used in the implementation, such as the hardware and software applications used to build containerised DevOps pipelines. The hardware selection process was heavily influenced by machinery typically used in advanced manufacturing, to simulate or mimic the infrastructure used in manufacturing, such hardware includes servers that handle data warehousing, AI development and deployment, as well as machinery controllers such as microcontrollers. In terms of software applications, the selection criteria are based on available DevOps open-source software, which have open-source communities that maintain the software applications. There are a vast number of tools that can be used in DevOps engineering, and the selection of the tools depend on the organisational needs, some organisations tend to use a set of tools chains as opposed to one tool.

3.1 Hardware and software

3.1.1 Hardware

The platform was implemented on the following computers to simulate the hardware used in manufacturing applications:

Table 1. Hardware specifics used to conduct experiment.

Computer	Dell PowerEdge R350
CPU	Intel Xenon E-2378 2.6 GHz
RAM	32GB (2X16GB)
Storage	4X8TB SATA HDD and 2X480GB SSD
Network	Broad 579 Quad port 1GbE BASE-T Adapter
Power Supply	2X Redundant Platinum 600W PSU

Table *1* presents the hardware specifications used to simulate a manufacturing environment. These systems were selected to represent a range of computing capabilities, from high-performance servers to microcontrollers, enabling the deployment and testing of DevOps pipelines across diverse platforms.

3.1.2 Software

As mentioned previously the DevOps tools used are open-source versions that have large maintenance communities. The approach to using open-source software was to ensure portability and cost-effectiveness, so as not to burden consumers with additional costs. However, these tools do have the option of paid premium versions that offer advantages, such as greater stability and secure scalability.

Building a pipeline requires a version control software tool that is commonly used in software engineering projects to foster collaboration between stakeholders and keep track of changes to software files during software development. For this experiment, Git was used as a software tool for version control, and there are other version control software tools available that can be used, that are open source. Version control software is not limited to keeping track

of changes to software files, but it can also be used for other critical files used advanced manufacturing that require collaboration between teams, like design files.

Once the version control software is in place, a software tool is needed to store, manage, and share software code with team members. Such a software tool is also commonly used by researchers to collaborate when writing research papers, to keep track of changes in the research paper, and to share and modify experimental data. It can also be used by advanced manufacturing team members to store, manage, and share files that are critical to operations. In this experiment, GitHub was used to store DevOps files.

Another tool that is critical to DevOps engineering is a software tool that handles CI/CD. This tool is for automating build, test, and deploy pipelines, and these pipelines can be shared amongst DevOps engineers, GitHub Actions was the chosen tool.

Since DevOps pipelines are automated, they require constant monitoring and logging of results. Some tools can be used for this purpose, tools such as Grafana or Prometheus, which are open-source tools. If these tools are not addressing the requirements or it is difficult to handle critical metrics for monitoring, DevOps engineers can build their monitoring tools using popular software libraries like Chart.js or Python programming language Libraries or R programming libraries. In this experiment, monitoring and logging are handled by shell scripts.

In the above paragraphs, we covered what is essential to build DevOps pipelines. The next step is to choose where these pipelines are going to be hosted. Then, different architectures can be implemented depending on the requirements. In this experiment, the microservice architecture was selected to house DevOps pipelines, because they can be independently implemented and deployed without being tied to software languages or operating systems. Therefore, Docker was chosen as a tool to host or containerize the DevOps pipelines implemented.

3.2 Implementation

The solution implemented by DevOps and microservice architecture standard practices and the system solution is implemented with open-source tools like in **Fig. 1.** Infrastructure is shown in **Table 1** and the operating system was from Linux distribution. Containers used are docker containers with DevOps pipelines from GitHub and used Git version control.

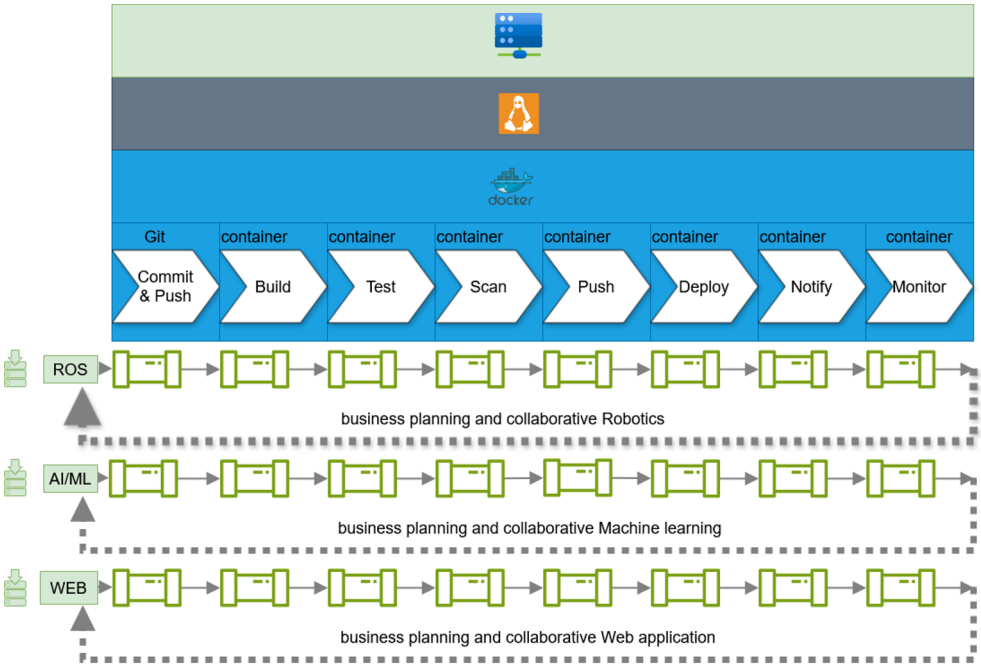

Fig. 1. DevOps architecture implemented.

4 Results

The results presented are key metrics that measure the performance of the overall system. The metrics include error rate, efficiency, and resource utilisation [10]. They can be used for comparative analysis studies or knowledge transfer for DevOps or microservices engineering. The efficiency of the pipeline is measured based on the amount of effort reduced, due to its introduction in the CI/CD process, effort in this regard is related to the time difference between the CI/CD process without a pipeline in place and the CI/CD process with a pipeline in place, i.e. if take 5 minutes to deploy manually and 3 minutes to deploy with a pipeline, then the time reduced is 2 minutes, which implies the pipeline would be 40% more efficient than the manual process of deployment [10].

4.1 Technical resource utilisation

In this study, "technical resource utilization" refers to the comparison of deployment time and error rate between manual and CI/CD automation tool deployment. The efficiency benefits brought about by automation are captured by this measurement. It might be challenging to distinguish and precisely estimate deployment time in manual deployments since the build and deployment processes are frequently entangled. CI/CD pipelines, on the other hand, distinguish between these phases, making it possible to precisely measure deployment time. Furthermore, automated deployments improve overall reliability by lowering the possibility of inherited faults from earlier stages.

According to a comparison of deployment techniques, automated CI/CD pipelines resulted in roughly 78% fewer errors than manual deployment. This decrease is explained by the automated pipelines' capacity to identify and isolate defects early in the development cycle, as opposed to manual deployments, which frequently carry over unresolved problems from earlier phases. Although monitoring tools can be used to log and detect errors during

the build phase, manual deployment typically exposes these errors only at the deployment stage. Interestingly, there was no visible decrease in the error rate for the Robotic Operating System (ROS) pipeline (**Table *2***). This could be because of the many dependencies that requires human intervention in ROS or because it was designed to run on low-power devices like microcontrollers, which could make deployment easier by nature in manual deployments and CI/CD automation deployment.

Table 2. Technical resource utilisation for ROS.

ROS	Deployment without CI/CD	CI/CD Automation Deployment	Difference (Δ)
Deployment time	3 min	2 min	1 min less
Error rate	14/19	14/19	0 no improvement

Table *3* presents the deployment metrics for AI/ML applications, comparing manual and automated CI/CD processes. The pipeline reduced deployment time by 5 minutes and significantly lowered the error rate, demonstrating the effectiveness of CI/CD automation in managing complex AI/ML workloads. These results support the adoption of DevOps practices in data-intensive environments.

Table 3: Technical resource utilisation for AI/ML.

AI/ML	Deployment without CI/CD	CI/CD Automation Deployment	Difference (Δ)
Deployment time	11 min	6 min	5 min less
Error rate	2/13	12/13	0.77 less errors

Table *4* shows the deployment performance for web services, highlighting improvements in both speed and reliability when using CI/CD pipelines. Deployment time was reduced by 24 seconds, and the error rate dropped by approximately 79% compared to manual deployment. These improvements, though modest, are meaningful in high-availability environments where even small gains in reliability and speed can have significant operational impact.

Table 4: Technical resource utilisation for Web services

Web Services	Deployment without CI/CD	CI/CD Automation Deployment	Difference (Δ)
Deployment time	3 min 24s	3 min	24s less
Error rate	2/24	21/24	0.79 less errors

4.2 Scalability and flexibility

To validate scalability, we deployed multiple container replicas for Web Services— in one machine to demonstrate scalability and across distributed environments to demonstrate flexibility. By separating application roles from underlying system dependencies, the containerisation idea exhibits inherent flexibility. As seen in Figure A, a single container image (b9f30d5389d5) was successfully repurposed across three functional contexts, functioning as a Django application, API service, and computational worker all at the same time with the same binary composition. Three main benefits are offered by this architectural approach, which is demonstrated by the shared Image ID and consistent size metrics in Figure

A: deployment consistency by ensuring identical runtime environments, operational efficiency through shared image layers, and functional abstraction where role specialisation without code duplication is made possible by logical tagging. As demonstrated by the experimental results in ***Fig. 3***. Scalability tests ***Fig. 2***, in which the same image was deployed across heterogeneous machines (ubuntu@TSEKOPA-NB3 and backup2@backup2), utilising the lightweight separation of containerisation to maintain consistent behaviour. These findings collectively demonstrate how containerisation facilitates: (1) environment-agnostic deployment through standardized packaging, (2) operational efficiency through common binary layers, and (3) horizontal scaling without image modification. The method successfully separates infrastructure dependencies from application logic, which is essential for distributed systems that need to scale elastically and reconfigure quickly.

```
ubuntu@TSEKOPA-NB3:~$ docker images
REPOSITORY           TAG       IMAGE ID       CREATED             SIZE
sekopa/worker        latest    b9f30d5389d5   7 minutes ago       537MB
sekopa/api-service   latest    b9f30d5389d5   7 minutes ago       537MB
sekopa/django-app    latest    b9f30d5389d5   7 minutes ago       537MB
```

Fig. 2. Web Services deployed TSEKOP-NB3 server to demonstrate scalability.

```
backup2@backup2:~$ docker images
REPOSITORY           TAG       IMAGE ID       CREATED          SIZE
sekopa/django-app    latest    0d78ac12634f   2 minutes ago    537MB
sekopa/worker        latest    0d78ac12634f   2 minutes ago    537MB
sekopa/api-service   latest    0d78ac12634f   2 minutes ago    537MB
backup2@backup2:~$
```

Fig. 3. Web Services deployed at backup2 computer to demonstrate flexibility.

4.3 Collaboration

Collaboration is possible with the DevOps pipeline, (***Fig. 2***) from a Collaborative containerised DevOps architecture. Docker images are sourced from different Docker Hub repositories—sekopa/ros_project for robotic operations, tryiphosa/yolo-vision for vision processing, and sekopa/django-app for the web layer. These components are orchestrated using Docker Compose to enable integrated development, testing, and deployment in a modular DevOps workflow. In collaboration, an example between the different docker repositories includes the use of the web application to visualise what is captured by the vision module, and the vision module may be used to capture the actions performed by the robotics module. The image below shows a list of docker images that were successfully deployed to a remote server using the pipeline.

```
backup2@backup2:~$ docker images
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
sekopa/django-app       latest    6912dbcc32f5   31 minutes ago   552MB
tryiphosa/yolo-vision   latest    e9cbf627dc2d   7 hours ago      6.2GB
sekopa/ros_project      latest    f024d2c3c6b0   29 hours ago     783MB
```

Fig. 4. List of the docker images deployed for collaboration.

4.4 Qualitative insights

4.4.1 Collaboration

In a containerised DevOps environment, collaboration is exemplified by the integration of Docker images from multiple repositories, each serves a distinct function within the system. These images, pulled from different DockerHub sources, collectively enable modular, scalable workflows. This coordinated use of heterogeneous containers streamlines development, testing, and deployment, while promoting interoperability and team specialisation. It reflects the essence of DevOps collaboration—autonomous components working in unison across shared infrastructure.

4.4.2 Technical resource utilisation

Containerised DevOps pipelines promote efficient technical resource utilisation by isolating services within lightweight, reproducible environments. Unlike traditional virtual machines, containers share the host kernel, this significantly reduces overhead while maintaining process-level isolation. This efficiency enables multiple containers to run concurrently on the same infrastructure, hence optimising CPU, memory, and I/O resources. Furthermore, container orchestration platforms such as Docker Compose and Kubernetes dynamically allocate resources based on real-time demand, minimising idle capacity and preventing bottlenecks. The ability to pull images from public and private Docker Hub repositories and to compose them into functional systems also encourages reuse and standardisation, reducing development effort and duplication. This resource-aware approach not only lowers operational costs but also accelerates build-test-deploy cycles, making the pipeline responsive and cost-effective across varied application domains.

4.4.3 Scalability and flexibility

To deploy a scalable and flexible a service, the architecture leverages containerisation and orchestration tools. The service is then packaged into lightweight containers, allowing consistent deployment across environments. Horizontal scaling is achieved by replicating containers across multiple nodes, with auto-scaling policies dynamically adjusting the number of replicas based on CPU utilization. Flexibility is ensured through a modular design, where each container operates independently and can be updated or reconfigured without disrupting the overall system. Environment variables and configuration files allow seamless adaptation to different deployment contexts, while CI/CD pipelines automate testing and deployment, enabling rapid iteration and reliable delivery. This approach ensures the web service remains resilient, adaptable, and performant under varying workloads.

5 Conclusion

Applying DevOps and microservices practices within a containerised environment for advanced manufacturing provides a scalable, secure, and cost-effective solution that accelerates innovation and fosters technological adoption. The modular, independent nature of containerised services allows for efficient resource utilisation, reduction of overhead and optimisation of the CPU, memory, and I/O resources, this ultimately lowers operational costs. Container orchestration platforms further enhance scalability and flexibility, enabling horizontal expansion to meet demand while allowing rapid adaptation to workload fluctuations. This resource-aware approach ensures continuous integration, streamlined

deployment, and the ability to experiment with diverse technologies such as robotics, AI/ML, and web services. By bridging the gap between academic research and industrial innovation, this platform supports resilient, adaptable, and scalable software delivery lifecycles, making it an ideal solution for contemporary advanced manufacturing.

Future work will involve developing a larger DevOps pipeline for a comprehensively practical system, to test the efficiency of DevOps pipelines on complex systems. This will allow us to look at Operational idleness of a functional system or downtime that may be caused due to introduction software or component. The implementation of larger DevOps pipelines will require more jobs to address concepts like roll back, release and in between steps of fundamental DevOps stages to make advanced pipelines. Other future work can involve deploying these pipelines in the real-world scenarios to test and get real-world results and develop framework that can address issues in advanced manufacturing.

We gratefully acknowledge the support of the Council for Scientific and Industrial Research (CSIR), particularly the Optronics Sensory Systems (OSS) cluster. We also thank our colleagues in the Digital Optronics Research Group and Research Group Leader Mr. Edwin Magidimisha for their contributions. Special gratitude to mentors Mr. Rofhiwa Seletani and Mr. Dumisani Kunene for their guidance in server development.

References

1. A. Neri, E. Cagno, S. Paredi, The mutual interdependences between safety and operations: A systematic literature review, *Saf. Sci.* **153**, 105812 (2022). https://doi.org/10.1016/j.ssci.2022.105812
2. M. Waseem, P. Liang, Microservices architecture in DevOps, *APSECW*, 13–14 (2017). https://doi.org/10.1109/APSECW.2017.18
3. M. Walker, S32K5 MCU advances zonal SDVs & NXP CoreRide platform, *Electropages* (2025). https://www.electropages.com/blog/2025/03/s32k5-mcu-advances-zonal-sdvs-nxp-coreride-platform (accessed 23 April 2025).
4. L. Chen, Microservices: Architecting for continuous delivery and DevOps, *ICSA*, 13 (2018). https://doi.org/10.1109/ICSA.2018.00013
5. R. Gandi, P. Szmrecsanyi, Using containers to address challenges around the deployment of application code, *IBM* (2019). https://www.ibm.com/think/insights/the-benefits-of-containerization-and-what-it-means-for-you (accessed 27 February 2025).
6. M. Jonas, How do technologies such as virtual machines in containers help improve operational efficiency, *TomsReviewBox* (2023). https://tomsreviewbox.com/how-do-technology-such-as-virtual-machines-in-containers-help-improve-operational-efficiency (accessed 9 May 2025)
7. L. Giamattei *et al.*, Monitoring tools for DevOps and microservices: A systematic grey literature review, *J. Syst. Softw.* **208**, 111906 (2024). https://doi.org/10.1016/j.jss.2023.111906
8. C.R. China, Smart manufacturing technology is transforming mass production, *IBM* (2023). https://www.ibm.com/think/topics/smart-manufacturing (accessed 27 February 2025).
9. Praxie, Future of manufacturing, *Praxie.com* (2025). https://praxie.com/future-of-manufacturing (accessed 9 May 2025).
10. A.P. Shanthi, Performance metrics, *Univ. Maryland* (2025). https://www.cs.umd.edu/~meesh/411/CA-online/chapter/performance-metrics/index.html (accessed 11 March 2025).