Self Adaptation of Machine Learning Enabled Systems Through QoS-Aware Model Switching

Thesis submitted in partial fulfillment
of the requirements for the degree of

*Master of Science in **Computer Science and Engineering** by Research*

by

Shubham Shantanu Kulkarni
2022701003
shubham.kulkarni@research.iiit.ac.in

International Institute of Information Technology
(Deemed to be University)
Hyderabad - 500 032, INDIA
May 2024

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled **"Self Adaptation of Machine Learning Enabled Systems Through QoS-Aware Model Switching"** by Shubham Shantanu Kulkarni, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Advisor: Prof. Karthik Vaidhyanathan

|| श्री ||

|| श्रीपाद राजं शरणं प्रपद्ये ||

|| अर्पणपत्रिका ||

जयांचें केलिया स्मरण । होय सकळ विद्यांचें अधिकरण । तेचि वंदूं श्रीचरण । श्रीगुरूंचे ।। ज्ञाने.१३.०.१ ||

|| भगवान श्री श्रीपाद श्रीवल्लभ स्वामी महाराजांचे चरणकमली समर्पित ||

To

The most revered and holy feet of Bhagwan Shreepad ShreeVallabh Swami Maharaj

whose mere remembrance grants knowledge of all kinds

This thesis is humbly dedicated

Acknowledgments

Starting my MS journey at IIIT Hyderabad, I was lucky to have Dr. Karthik Vaidhyanathan as my advisor. He showed me how amazing software and the area of self-adaptation could be. From teaching me the basics like how to use Linux, nurturing my curiosity, to guiding me through the complexities of my research, his push and help has been incredible. He always had time for our discussions, allowing me to explore my interests while making sure I had all the support I needed. He meticulously reviewed my work, offered ideas and constructive feedback, and spent countless hours discussing and brainstorming ideas with me, which significantly contributed to my papers and research confidence. His support went beyond academics; he was always there to offer timely advice on both professional and personal matters, helping me navigate the complexities of graduate life with ease. He built a supportive team around me, creating an enriching research environment that was both challenging and rewarding. His belief in my potential was a constant source of motivation to do better and explore new things.

Arya, a colleague in my lab, played a key role in my thesis work. His coding skills and proactive approach were essential. We made sure that our discussions turned into well-built research projects. Tackling challenging projects with Arya, we developed a strong personal bond, becoming best friends who could talk about anything and be there for each other.

Working with Meghana was a pleasure. She brought energy and commitment to our projects, making our work together both successful and fun. Her eagerness to explore new concepts and dedication to our research significantly advanced the sustainability aspects of my study.

Hiya's support, care and friendship were invaluable during my MS journey. We explored new ideas for my thesis together, leading to significant results. Her positivity kept me focused and well. Our shared adventures and personal bond enriched my college experience, becoming a highlight of my time here. She is intern here at our SERC Lab from MU Jaipur, her involvement added greatly to this journey.

I also want to thank my lab mates Prakhar, Adyansh, Shrikara, Raghav Donakanti, Rudra, Chandrashekhar, VJS Pranav, Dhiraj, Saianirudh, Raghav Mittal and Akhila for their friendship and the collaborative lab environment. Their company and spirit made my research journey more enriching. Also I would like thank my colleague Adithyan CP for amazing work with edge self adaptation. We had really nice time together during his summer internship program.

The generous travel grant from Microsoft Research India, ACM SIG, ACM India and advisor (SERC LAB) enabled me to attend ASE'23 conference in Luxembourg, SEAMS/ICSE'24 Conference in Lisbon, Portugal and ISEC'24 Conference in Bangalore respectively, further broadening my horizons and providing invaluable exposure to the global research community. This experience along with the support and insights of many at the conference, enriched my academic journey significantly.

I would like to extend my gratitude to all my professors, our center head Prof. Dr Raghu Reddy and IIIT academics that gave me exposure to various evolving fields in technology, primarily artificial intelligence and software engineering.
I also thank the Software Engineering Research Center (SERC) lab and the institute for providing the computational resources and infrastructure required for my research.
I thank the staff and administration, for promptly resolving any issues. I also thank them for providing me with the opportunity to travel and present my work at the conference.
To my hostel mates, classmates, and the wonderful friends I made along the way, your support and companionship made my time at IIIT Hyderabad unforgettable. The shared meals in Yuktahar and VC, birthday celebrations, college events, treats, late-night discussions, and collaborative projects enriched my college life experience beyond measure.
Finally, my deepest gratitude goes to my family. My mother's unwavering support and belief have been the bedrock of my journey. Her encouragement and sacrifices have empowered me to chase my dreams and achieve what I have today. She has always inspired me to give my best, offering endless motivation and love. I am equally grateful to my brother, Shardul, for his support and for maintaining our home as a haven of peace, enabling me to focus on my work. I also hold deep appreciation and love for my father, who, although no longer with us, left a lasting impact on my life. The education he emphasized, the sacrifices he made, and the values he instilled in us have significantly contributed to where I stand today. His legacy continues to guide and inspire me, shining as a beacon of strength and perseverance. Together, their love and support have been the guiding lights of my journey.
This journey has been about growth, discovery, and the power of collaborative effort. I stand grateful and humbled by the support and inspiration I've received from each individual who has been part of my MS journey. Thank you all for believing in me, guiding me, and helping me grow not just as a researcher, but as a person.
In humble acknowledgment of the divine, I recognize the Krupa and blessings of Bhagwan Shreepad ShreeVallabh Swami Maharaj, whose presence has been the guiding light behind all my endeavors. With a heart full of devotion, I see my work not as my own effort but as a flow of divine grace through me. The accomplishments and insights gained are manifestations of His divine will, a reminder that wisdom and strength are bestowed in His remembrance. I am merely a medium for His grace, a conduit for His blessings to be channeled into this world. The success and achievements of this thesis are not mine to claim but are offerings at His lotus feet. It is through His divine intervention and guidance that my path has been illuminated, allowing me to act as an instrument of His will. May this work serve as a humble tribute to His infinite wisdom and love, reflecting the devotion He embodies. In remembrance of Him, I find the true essence of success and fulfillment.

Abstract

Machine Learning (ML), particularly deep learning, has seen vast advancements, leading to the rise of Machine Learning-Enabled Systems (MLS). However, numerous software engineering challenges persist in propelling these MLS into production, largely due to various run-time uncertainties that impact the overall Quality of Service (QoS). These uncertainties emanate from ML models, software components, and environmental factors. Self-adaptation techniques present potential in managing run-time uncertainties, but their application in MLS remains largely unexplored. As a solution, this thesis proposes Machine Learning Model Balancer, a novel concept focusing on managing uncertainties related to ML models by using multiple models in runtime.

Subsequently, the thesis introduces AdaMLS, an novel approach that leverages the Machine Learning Model Balancer concept for continuous adaptation. AdaMLS extends the traditional MAPE-K loop, employing lightweight unsupervised learning for dynamic model switching, thereby ensuring consistent QoS in dynamic environments. The effectiveness of AdaMLS is demonstrated through an object detection use case, showcasing its ability to effectively mitigate run-time uncertainties and surpass both naive approaches and standalone models in terms of QoS.

We further developed SWITCH, an exemplar to demonstrate the practical application of our research in self-adaptation of MLS. The discussion on SWITCH highlights its role as a tool designed to enhance self-adaptive capabilities in MLS through dynamic model switching in runtime. SWITCH is developed as to cater to a broad range of ML scenarios. It features advanced input handling, real-time data processing, and logging for adaptation metrics, supplemented with an interactive real-time dashboard for system observability. Through its architecture and user-friendly interface, SWITCH not only demonstrates adaptability and performance but also serves as a valuable platform for researchers, practitioners, and students to explore self-adaptation in MLS.

Beyond the primary focus on ensuring optimal QoS, we also explore the application of the Machine Learning Model Balancer concept in two main areas. Firstly, the EcoMLS approach leverages this concept to enhance the sustainability of MLS. By optimally balancing energy consumption with model confidence through runtime ML model switching, EcoMLS marks a significant step towards sustainable, energy-efficient ML solutions. Secondly, RelMLS approach to systems, where machine learning is deployed in streaming mode. Through experiments and implementation, this approach has shown promising results in object detection use case, adopting a software architecture-based solution to dynamically switch between models based on contextual reliability.

This thesis encapsulates a comprehensive exploration of the Machine Learning Model Balancer concept, from its theoretical introduction to practical applications in diverse MLS scenarios. Through AdaMLS, SWITCH, EcoMLS, and RelMLS implementations, we demonstrate the potential of our approaches in enhancing both the QoS and sustainability of MLS. Our findings suggest that the judicious application of model switching and self-adaptation techniques can significantly mitigate run-time uncertainties, paving the way for more resilient, efficient, and adaptable MLS.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software technology has advanced, integrating machine learning (ML), deep learning (DL), and artificial intelligence (AI) into a wide array of applications. These technologies have enabled the development of machine learning-enabled systems (MLS), such as ChatGPT, Amazon Rekognition, DALLE-2 etc, capable of performing tasks that require human-like intelligence, including decision-making and outcome prediction. MLS leverage ML models to interpret data, adapt to new inputs, and make informed decisions based on identified patterns. These systems, ranging from recommendation engines and spam filters to more complex applications like autonomous vehicles and advanced healthcare diagnostics, showcase the potential of MLS to learn from data and adapt to new inputs, thus making them integral to technological advancement [3]. As represented in Figure 1.1 referred from a case study [3], a typical MLS workflow begins with incoming requests, processed through APIs, leading to data preprocessing, model inference, and post-processing stages before delivering the output. The core of an MLS lies in its ML models, stored in a repository and loaded as needed to ensure accurate and timely responses. This process highlights the system's ability to adapt and learn from data, a fundamental attribute of MLS that drives their application across diverse domains.

Deploying and maintaining such machine learning-enabled systems presents challenges [49, 3], including engineering tasks like data management, ML model integration, and versioning. These tasks are important for utilizing the more accurate and efficient models. Additionally, operational challenges such as scalability and resource management are essential for managing variable loads and ensuring a high Quality of Service (QoS) [64]. Nearly half of MLS projects face hurdles in reaching production stages, primarily due to inconsistent model performance and software component instability, as noted by a Gartner report [1]. Beyond the engineering and operational challenges, MLS are particularly vulnerable to runtime uncertainties that influence their Quality of Service (QoS) [9, 13]. QoS encompasses the system's ability to deliver timely, accurate, and reliable outputs, ensuring user satisfaction and operational efficiency. These uncertainties that arises during the system's runtime, directly affects system performance. As highlighted by Casimiro et al [15] these mainly include:

Data-Driven Uncertainties: Where the quality and relevance of input data vary over time, leading to potential inaccuracies in the system's outputs.

Model Management Uncertainties: Stemming from the need for continuous model evaluation and updates to accommodate new data or changes in the operational context.
Resource Allocation Uncertainties: The dynamic demand for computational resources can lead to bottlenecks, affecting the system's ability to process data efficiently and meet response time requirements.
System Stability: Frequent changes in the operational environment or data patterns can disrupt the balance between system adaptability and stability, resulting in unpredictable behavior.
These runtime uncertainties challenge the system's capability to maintain a high QoS, showing the necessity for a dynamic approach to adapt the behavior and structure of the system in run-time. The focus on runtime is important because it is during the system's operation that these uncertainties have the most significant impact, directly influencing the effectiveness, efficiency, and reliability of MLS outputs. Addressing these uncertainties requires a way that allows the system to autonomously adjust its operations, ensuring that the QoS is not compromised despite the changing conditions. This shows why systems need to adapt by themselves, introducing us to self-adaptation. Self-adaptation refers to

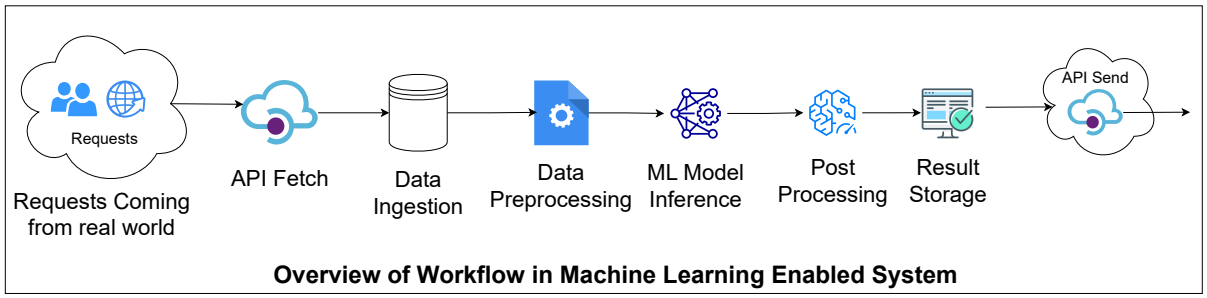

Figure 1.1: Typical Workflow in Machine Learning-Enabled Systems

the capability of a system to autonomously modify its structure and/or behaviour in response to changes within its operating environment or internal state [17]. The prefix "self" in self-adaptation implies that the system will have the capabilities to autonomously (with minimal human intervention) decide the action to be performed in the event of any uncertainty [79]. This concept has been extensively explored within traditional software systems, leading to advancements in system resilience, efficiency, and user satisfaction by automatically managing uncertainties and maintaining system objectives under changing conditions [42]. Self-adaptive systems are designed to manage a variety of uncertainties, ensuring that the system continues to meet its objectives amidst changing conditions [87]. Although self-adaptation has proven effective in domains like IoT, healthcare, traffic management, and finance [11, 22, 84], its application in MLS is largely unexplored [15]. Thus, by enabling MLS to autonomously adjust their behavior, we can significantly enhance their capability to maintain service quality, model accuracy, and overall system performance tackling data, system and operational uncertainties [87, 15].
Hence in this thesis, we scope our research area on self-adaptation within machine learning-enabled systems. Further details on our research questions and the proposed solutions are discussed in the following sections.

1.1 Research Questions

Recognizing the impact of runtime uncertainties on MLS and noting that self-adaptation within MLS remains largely unexplored, this study aims to explore self-adaptation for these systems. It will focus on developing strategies that allow MLS to effectively respond to changing conditions and sustain high QoS. The research is structured around three research questions:

RQ1 In the context of machine learning-enabled systems, how can self-adaptive mechanisms be developed and applied to mitigate runtime uncertainties, thereby enhancing their Quality of Service (QoS)?

RQ2 What tools can be devised to facilitate the implementation and exploration of self-adaptation within machine learning-enabled systems for researchers, students, and practitioners?

RQ3 How can self-adaptation in MLS be applied and generalized across MLS deployment modes and aspects in computer vision domain, broadening their applicability and impact?

RQ1 seeks to develop self-adaptive strategies for MLS to mitigate runtime uncertainties, with a focus on maintaining or improving the systems' QoS. RQ2 aims at creating tools that facilitate the practical application and experimentation making self-adaptation accessible and actionable to students, researchers and practitioner. RQ3 explores the potential for broader application of self-adaptation strategies, examining their adaptability and effectiveness across other aspects like sustainability and deployment modes of MLS like streaming deployment mode of MLS.

1.2 Self-Adaptation of MLS: Solution Overview

To address the identified challenges and answer the research questions outlined above, this thesis presents a solution enabling self-adaptation in Machine Learning-Enabled Systems (MLS). This is aimed at equipping MLS with the ability to self adapt to the dynamic run-time uncertainties that affect their Quality of Service (QoS), ensuring consistent performance under varying operational conditions.

The main contribution of our solution is the development of the *Machine Learning Model Balancer* concept. This novel concept suggests that dynamically switching between multiple models[1] during runtime can significantly enhance the adaptability and performance of MLS. It is based on the understanding that no single ML model is universally optimal across all operational scenarios. By leveraging the strengths of different models according to the current context, MLS can achieve a balance between model accuracy and computational efficiency, thereby maintaining or even improving their QoS.

Building upon the Machine Learning Model Balancer concept, we introduce AdaMLS, a novel self-adaptation approach that leverages this concept. AdaMLS provides a structured approach to self-

[1] unless specified otherwise by model we imply ML model in this thesis

adaptation in MLS. Through a self-adaptive object detection system prototype, we demonstrate AdaMLS's effectiveness in balancing system and model performance achieving better QoS.

To complement the contributions of our research and provide a practical tool for exploration and experimentation, we developed SWITCH. This exemplar serves as a platform for real-world MLS environments, enabling the academic and professional communities to engage with self-adaptive strategies without the constraints of live deployment. SWITCH provides a versatile environment for testing and refining self-adaptive mechanisms, facilitating understanding of their potential benefits and limitations.

Additionally, this thesis explores the application of the Machine Learning Model Balancer concept in two specific areas: sustainability and contextual self-adaptation in streaming modes of MLS. These explorations aim to extend the generalizability of the concept beyond performance improvement, addressing the broader implications of self-adaptation in MLS, including energy efficiency and contextual reliability in data streaming contexts.

Sustainability in Machine Learning Systems: This aspect of our research emphasizes the importance of energy efficiency in MLS operations. Through EcoMLS approach, by integrating sustainability considerations into the self-adaptive process, we seek to reduce the environmental impact of MLS while ensuring they remain effective and reliable through run-time model switching.

Contextual Self-Adaptation in Streaming Modes: Recognizing the increasing relevance of MLS in applications that require continuous data processing, we propose RelMLS approach which aim to ensure these systems can maintain context accuracy and reliability. This involves extending the Machine Learning Model Balancer concept to adaptively manage model selection in streaming environments, where the context and data are continuously changing.

In summary, this thesis contributes to the field by proposing a novel research work for self-adaptation in MLS, demonstrated through the Machine Learning Model Balancer concept, AdaMLS approach, and SWITCH exemplar. Furthermore, by exploring the application of this concept in sustainability and streaming data contexts, we highlight the broader potential of self-adaptation to enhance the adaptability, efficiency, and environmental responsibility of MLS. Through these efforts, we aim to provide valuable insights into the capabilities and future directions of self-adaptation of machine learning-enabled systems.

1.3 Research Activities

To address the challenges in self-adaptation of Machine Learning-Enabled Systems (MLS) and respond to the outlined research questions, the following stages were systematically undertaken in this thesis as illustrated in Figure 1.2:

Stage 1: State-of-the-Art Analysis

A comprehensive review of current literature in both self-adaptation and MLS was conducted. This revealed a notable gap in real-time adaptation strategies for MLS. Identifying this gap was important, as it directed our research focus towards innovating real-time adaptive solutions for MLS, laying the groundwork for our subsequent contributions and deciding the research goal.

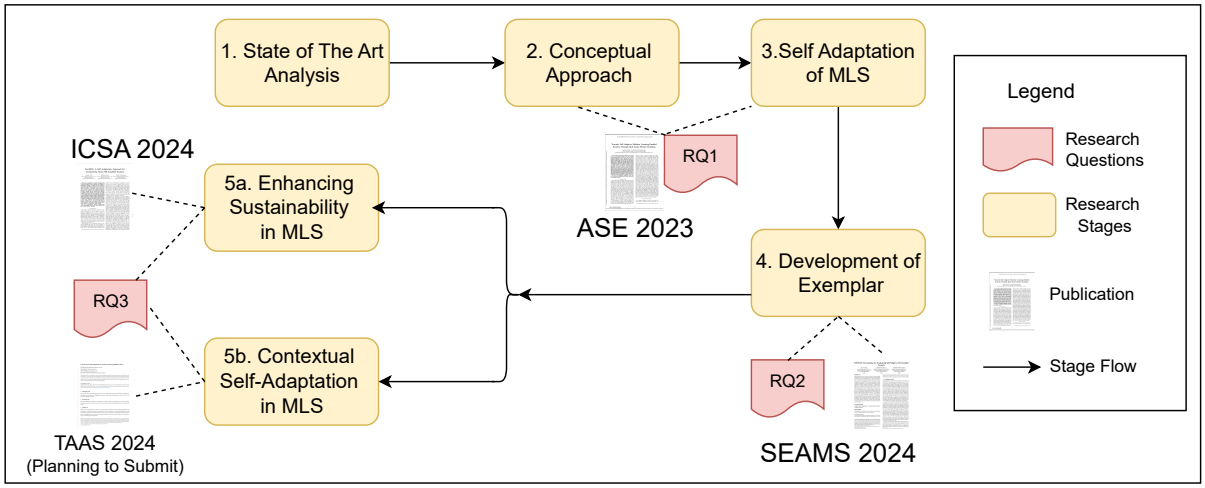

Figure 1.2: Stage by stage development of overall research

Stage 2: Conceptual Approach

The research goal, the research challenges, and our domain knowledge were used in the second stage and we proposed the Machine Learning Model Balancer as a mechanism for enabling real-time model switching in MLS. This novel concept aims to dynamically select the better model during runtime to enhance system adaptability and performance, directly addressing the challenge posed in Research Question 1 (RQ1). The development of this concept was detailed in a paper presented at the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE) 2023 [47].

Stage 3: Self Adaptation of MLS

With the conceptual foundation established, we introduced the AdaMLS approach, leveraging the Machine Learning Model Balancer concept. AdaMLS showcases the practical application of dynamic model switching to improve MLS adaptability, aligning with the goals of RQ1. Details of the AdaMLS approach were shared in the same ASE 2023 publication [47].

Stage 4: Development of Exemplar

Recognizing the need for a practical tool to test and validate self-adaptive strategies in MLS, we developed SWITCH. This exemplar, published in the artifact track of 19th International Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS) 2024, co-located with the 46th International Conference on Software Engineering (ICSE 2024), facilitates experimentation with MLS under various uncertainties. SWITCH's development aligns with Research Question 2 (RQ2), focusing on providing a platform for empirical evaluation of self-adaptive strategies in MLS.

Stage 5a: Enhancing Sustainability in MLS

Further exploring the applications of the Machine Learning Model Balancer concept, we focused on sustainability within MLS through the EcoMLS approach. Detailed in our submission to the 8th International Workshop on Green and Sustainable Software (GREENS'24) part of 21st IEEE International Conference on Software Architecture (ICSA 2024), EcoMLS aims to integrate energy efficiency into

MLS, addressing the objectives of Research Question 3 (RQ3). This work aims at reducing the ecological impact of MLS operations without sacrificing performance, highlighting our commitment to sustainable technological solutions.

Stage 5b: Contextual Self-Adaptation in Streaming MLS

This section details our ongoing work on self-adaptation of MLS deployed as streaming service, intended for submission to the ACM Transactions on Autonomic and Adaptive Systems (TAAS). Focused on RQ3, this research aims to enable MLS to autonomously adjust their processing to maintain context reliability in real-time data streams, enhancing adaptability and QoS in dynamic conditions. This work is conducted in collaboration with Hiya Bhatt, an intern at our Software Engineering Research Center (SERC) from Manipal University Jaipur; Arya Marda, an undergraduate student; and guided by my supervisor. Their collective expertise and contributions are invaluable to the advancement of this research.

Also, our contributions have been recognized within the academic and professional communities as follows:

1. The work was honored with the Best Student Poster Award at the Innovation in Software Engineering Conference (ISEC) 2024 held in Bangalore. This accolade serves as a testament to the relevance and impact of our research in addressing the outlined research questions.

2. The practical implementation of our proposed machine learning model balancer concept on the Qualcomm's QIDK HDK8550 Development Kit as an edge device has yielded promising preliminary results. The author of this thesis presented these findings at Qualcomm's University Platforms Developer Conference 2024, highlighting the practical applicability of our research.

1.4 Thesis Structure

The remainder of this thesis is organized as follows:

Chapter 2: Background

In Chapter 2, we provide background details of various concepts underlying this thesis. It covers the basics of MLS, the fundamentals of self-adaptation, the significance of our object detection use case within MLS, and the use of exemplars in research. This background information is necessary for understanding the context and motivation behind our research.

Chapter 3: Literature Review

Here, we present a thorough review of existing literature related to self-adaptation in MLS, the application of self-adapting strategies, the use of exemplars, challenges in object detection as our use case, and considerations for environmental sustainability and streaming data processing in MLS. The aim is to highlight the current state of research and identify areas where further work is needed.

Chapter 4: Machine Learning Model Balancer Concept

This chapter addresses Research Question 1 (RQ1) by introducing and detailing the Machine Learning

Model Balancer concept. It discusses the rationale behind this concept and how it aims to improve the adaptability and Quality of Service (QoS) of MLS by addressing runtime uncertainties.

Chapter 5: AdaMLS : Novel Approach For Self-Adapting MLS
Continuing with the exploration of RQ1, Chapter 5 focuses on AdaMLS, a novel approach that applies the Machine Learning Model Balancer concept. The chapter elaborates on the design, functionality, results and challenges addressed by AdaMLS within the context of self-adaptive MLS.

Chapter 6: SWITCH : An Exemplar for Evaluating Self-Adaptive ML-Enabled Systems
Dedicated to Research Question 2 (RQ2), this chapter discusses SWITCH, an exemplar tool created to test and evaluate self-adaptive strategies in MLS. It highlights the tool's role in supporting research, experimentation, and educational efforts in the field of self-adaptation of MLS.

Chapter 7: Applications of Model Balancer Concept
Focusing on Research Question 3 (RQ3), this chapter explores the application of the Machine Learning Model Balancer concept in enhancing sustainability through EcoMLS and in contextual self-adaptation for streaming data environments through RelMLS. It outlines the motivation, methodology, results and contributions of these applications to the broader field of self-adaptive MLS.

Chapter 8: Conclusion and Future Work
The concluding chapter summarizes the main findings and contributions of this thesis. It reflects on the significance of our work and suggests future research directions that could further advance the domain of self-adaptive MLS.

1.5 Research Publications

The research presented in this (as depicted in Figure 1.2) has resulted in the following peer-reviewed publications :

1. S. Kulkarni, A. Marda and K. Vaidhyanathan, "Towards Self-Adaptive Machine Learning-Enabled Systems Through QoS-Aware Model Switching," 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), Luxembourg, Luxembourg, 2023, pp. 1721-1725.
(*Thesis author's contributions:* Overall idea, methodology, approach, evaluation and writing of the publication under the guidance of the supervisor. Implementation was collaboratively carried out with co-author A. Marda).

The following publications are accepted for publication in respective conferences and are available as technical reports:

2. A. Marda, S. Kulkarni and K. Vaidhyanathan, "SWITCH: An Exemplar for Evaluating Self-Adaptive ML-Enabled Systems" International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '24), April 15–16, 2024, Lisbon, AA, Portugal.

Available at: https://arxiv.org/abs/2402.06351
(*Thesis author contribution:* Conceived and designed the research framework for the SWITCH exemplar, outlining the overall idea, methodology, and approach with co-authors. Led the writing of the publication, ensuring a coherent presentation of the research contributions with co-authors. The implementation of the SWITCH exemplar, including coding and tool development, was done by the first author A. Marda in collaboration with other authors)

3. M. Tedla, S. Kulkarni and K. Vaidhyanathan, "EcoMLS: A Self-Adaptation Approach for Architecting Green ML-Enabled Systems" 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C), Hyderabad, India, 2024
Available at: https://doi.org/10.36227/techrxiv.171177355.53837507/v1
(*Thesis author contribution:* Overall idea, conceptual methodology, evaluation setup and writing along with the other authors.)

Chapter 2

Background

*In this chapter, we explore the concept of **Self-Adaptive Systems (SAS)**, detailing their development, applications, and mechanisms for self-improvement in response to environmental or internal changes along with object detection use case.*

2.1 Self Adaptive Systems

Self-adaptive systems (SAS) constitute a significant and evolving research domain within computer science, particularly in the context of increasing software and environmental complexity as shown in Figure 2.1. The principle driving SAS is the capability to autonomously modify behavior in response to changes within their environment, internal states, or in response to observed faults, thereby enhancing their performance, dependability, and utility. This section provides a comprehensive overview of self-adaptive systems, elucidating their theoretical foundations, methodologies, application domains, challenges, and future directions. The genesis of self-adaptivity can be traced back to the challenges associated with

Figure 2.1: What is Self-Adaptation?

managing increasingly complex software systems. An IBM's vision of autonomic computing highlighted the impending software complexity crisis, underlining the need for systems that could manage themselves based on high-level administrator objectives, heralding the era of autonomic computing. This initiative laid the groundwork for the development of self-adaptive systems, aiming to mitigate the complexity and manageability challenges inherent in contemporary computing systems [42].

At the core of SAS is the ability to perform self-adaptation, which encompasses autonomy in adaptation, driven by overarching objectives and facilitated by feedback loops. These systems are distinguished by their capacity to autonomously decide and act upon changes in their context or environment with minimal or no human intervention. Definitions offered in literature tells that the system's capability to adjust its operations in response to environmental changes, pursuing goals through adaptive actions. This adaptivity is realized through various approaches, including external control mechanisms, component-based software engineering (CBSE), model-driven development, nature-inspired strategies, and multi-agent systems, each contributing uniquely to the adaptivity process [17, 52].

The evolution of self-adaptive systems has seen a transition from theoretical formulations towards practical implementations and holistic frameworks. This progression is evidenced by the broad spectrum of application domains, from networking and web services to robotics, and more recently, the Internet of Things (IoT) and Infrastructure as a Service (IaaS). Methodologies employed in the design and development of SAS incorporate a blend of traditional software engineering approaches with insights from cybernetics, artificial intelligence, and control theory, aiming to furnish systems with the capabilities to self-manage and adapt in real-time to changes [87].

Self-adaptive systems find applicability across a diverse array of domains, including but not limited to, embedded systems, decision-making processes, healthcare, and the software industry itself. Noteworthy among the challenges faced in these applications is the user's understanding and trust in the systems' adaptive behaviors. Efforts to address these concerns include the development of human-readable explanations of system actions and the incorporation of user feedback into the adaptation process. Furthermore, the quest for creating systems that can learn from new experiences and adapt to evolving conditions remains a central research direction [52].

As the domain of self-adaptive systems matures, the emphasis shifts towards addressing the scalability, reliability, and security concerns inherent in these systems, particularly in the context of application areas such as machine learning systems, cyber-physical systems and cloud computing. The future trajectory of SAS research is poised to explore bio-inspired approaches, enhance decision-making algorithms, and refine feedback control mechanisms to better manage the complexity and dynamism of modern computing environments. As explained in Figure 2.2 the continuous evolution of self-adaptive systems promises to significantly impact how future computing systems are designed, deployed, and managed, steering towards more autonomous, efficient, and resilient computing paradigms.

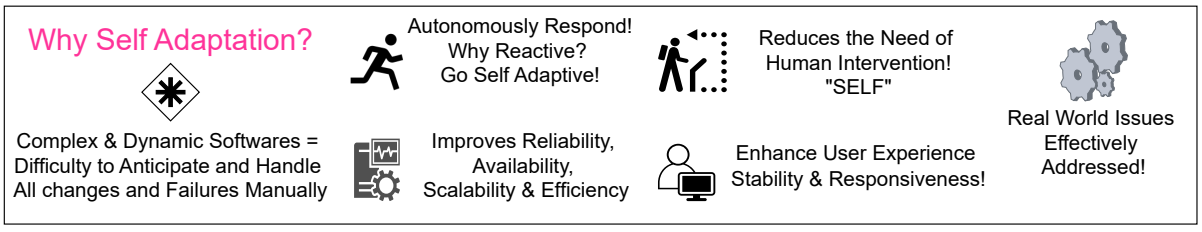

Figure 2.2: Why Self-Adaptation?

2.1.1 Conceptual Model of a Self-Adaptive System

The conceptual framework[82, 79] of self-adaptive systems (SAS) delineates their structure through four primary components: the environment, the managed system, adaptation goals, and the managing system, as illustrated in Figure 2.3. This model provides a vocabulary foundational to the self-adaptation discipline and guides the organization and focus of knowledge within this field.

Environment encompasses both physical and virtual entities that interact with the SAS. This interaction is mediated through sensors and effectors, allowing the system to perceive and act upon its surroundings. The environment is distinguished from the SAS based on control limitations, introducing uncertainty in sensed information and effected actions.

Managed System refers to the application layer responsible for the system's domain functionality, interacting directly with the environment to fulfill its designated tasks. Adaptation support within the managed system necessitates sensing capabilities for monitoring and actuators for executing adaptations, ensuring minimal disruption to regular activities.

Adaptation Goals represent the objectives guiding the managing system's oversight of the managed system, often pertaining to the software qualities of the latter. These goals include self-configuration, self-optimisation, self-healing, and self-protection, each addressing distinct aspects of system adaptability and resilience.

Managing System embodies the adaptation logic that manages the managed system towards achieving adaptation goals. It monitors and adapts the managed system based on real-time analysis, potentially across multiple levels of adaptation strategies.

This conceptual model abstracts from specifics such as software distribution and adaptation decision coordination, providing a generalized framework for understanding and developing SAS.

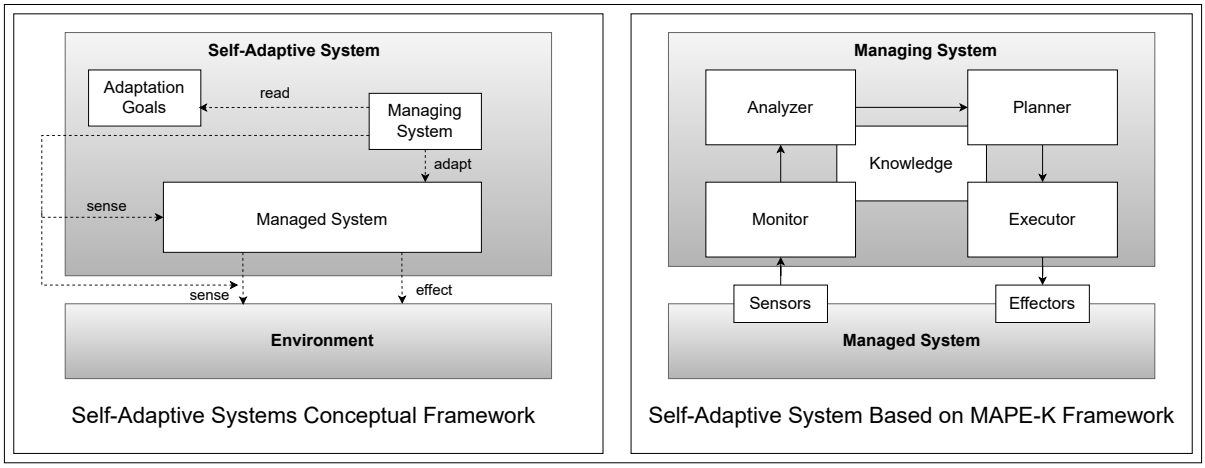

Figure 2.3: Conceptual Model and MAPE-K Framework of a Self-Adaptive System

2.1.2 The MAPE-K Framework

The MAPE-K (Monitor, Analyze, Plan, Execute over a shared Knowledge base) framework represents a cornerstone in the architecture of managing systems within SAS, orchestrating the adaptive cycle through its constituent activities (see Figure 2.3). Introduced by Kephart and Chess, and further elaborated by IBM and others, MAPE-K encapsulates the essence of self-adaptation mechanisms[42].

- **Monitor:** This phase involves continuous surveillance of the managed system, collecting data on its operation, metrics, and logs, which are then relayed to the Analyze component.
- **Analyze:** Here, the system assesses the need for adaptation based on the monitored data and predefined adaptation goals, triggering the planning phase upon identifying such needs.
- **Plan:** This stage devises an adaptation strategy to meet the goals, utilizing a variety of techniques from model-checking to machine learning for plan formulation.
- **Execute:** The execution phase implements the adaptation plan through the managed system, altering its behavior or structure via effectors that facilitate dynamic adjustment.
- **Knowledge:** Acting as the framework's central repository, the Knowledge base accumulates and disseminates data among the MAPE components, supporting informed decision-making and adaptation.

The MAPE-K framework effectively embodies the adaptive loop, enabling SAS to dynamically respond to changes and maintain or enhance their operational objectives. Now that we have established a foundational understanding of self-adaptive systems and their underlying mechanisms, let's explore the object detection as a use case in the context of these systems in the following section.

2.2 Object Detection

Object detection is an important area of study in artificial intelligence (AI) and machine learning (ML), offering the capability to identify and locate objects within an image or a video. This capability extends across various applications impacting sectors like autonomous driving, surveillance, augmented reality, and beyond. Object detection algorithms analyze visual inputs, breaking down images into elements or objects, and identifying instances of particular categories such as humans, vehicles, or animals. In object detection, an "image" refers to digital representations captured by cameras or sensors, which are processed by AI models. These models are trained on vast datasets to recognize patterns and features corresponding to different objects. The process involves the model scanning the image, predicting the presence of objects, and drawing bounding boxes around them. Each detection is accompanied by a "confidence score," which quantifies the model's certainty regarding the object's presence and classification. This score filters out less likely detection, enhancing the reliability of the results.

Key Concepts of Object Detection:

- **Confidence Score:**: A metric that measures the probability that a detected object belongs to a certain category. Higher confidence scores indicate greater assurance in the detection's accuracy.
- **Bounding Boxes**: Rectangular borders drawn around detected objects to specify their location within the image. These boxes are defined by coordinates that pinpoint the object's position.
- **Response Time**: The duration the object detection model takes to analyze an image and identify objects. This metric is critical for applications requiring real-time detection, such as autonomous vehicles and video surveillance systems.

Object detection's significance in AI and ML is multifaceted, impacting both technological advancement and practical applications. It enhances machines' understanding of visual contexts, enabling them to interact with their surroundings in a more human-like manner. This capability is pivotal for creating intelligent systems that can autonomously navigate environments, recognize and track objects in real-time, and make informed decisions based on visual inputs.

YOLO Algorithm and Its Importance: The YOLO (You Only Look Once) algorithm stands as as notable advancement in the object detection domain, epitomizing efficiency and accuracy in processing visual data. Originating from the foundational work of Redmon et al., YOLO revolutionized the approach to detecting objects by analyzing an entire image in a single evaluation, rather than processing multiple segments of the image separately [69]. This innovation not only accelerates the detection process but also enhances the algorithm's ability to understand contextual information, improving the accuracy of object identification.

YOLO's significance is further highlighted by its iterations and improvements, with Ultralytics' YOLOv5 being among the latest and most efficient versions [41]. YOLOv5 benefits from an optimized architecture and pre-trained models provided by Ultralytics, facilitating rapid deployment and integration into a wide array of applications, from real-time video analysis to autonomous navigation systems. The availability of these pre-trained models democratizes access to high-quality object detection capabilities, allowing developers and researchers to focus on application-specific challenges rather than the intricacies of model training.

The decision to incorporate the YOLO algorithm, particularly YOLOv5, into our study as use-case is motivated by its balance between speed and accuracy, its adaptability to diverse scenarios, and the support provided by Ultralytics. In summary, the YOLO algorithm, with its emphasis on speed, accuracy, and ease of use, represents a cornerstone in the development of efficient object detection systems. Its evolution, marked by significant contributions like YOLOv5, continues to shape the landscape of artificial intelligence, driving forward the capabilities of machines to perceive and interpret the visual world with remarkable precision.

Despite remarkable progress, object detection faces challenges, especially in processing speed, accuracy, and the detection of small or partially obscured objects. Future research is directed towards developing lightweight models for faster inference on edge devices, enhancing end-to-end detection pipelines for improved accuracy and efficiency, and advancing 3D object detection for comprehensive spatial understanding[93]. Moreover, innovations in cross-modality detection and open-world detection aim to create systems capable of learning from new, unlabeled data, mimicking human learning efficiency and adaptability. In conclusion, object detection represents a dynamic and evolving field within AI and ML, driven by the pursuit of creating more intelligent, responsive, and autonomous systems. As technologies advance, object detection will undoubtedly play a key role in bridging the gap between artificial perception and human-like understanding of the visual world, catalyzing the development of next-generation AI applications [93].

2.3 Discussion

This chapter provided an overview of Self-Adaptive Systems (SAS) and their role in modern computing, alongside an introduction to object detection as a key application area in AI and ML. Understanding SAS and object detection lays the groundwork for addressing the complexities of dynamic and unpredictable environments through adaptive strategies.

- We highlighted the evolution of SAS from foundational concepts to their application in complex computing systems, emphasizing their capacity for self-management and adaptation.
- The challenges and future directions of SAS were discussed, particularly focusing on the need for enhanced adaptability, security, and scalability.
- Object detection was introduced as a vital AI/ML domain, showcasing its significance in various applications and the ongoing challenges and innovations in this area.

In the following chapter 3, we will delve into a detailed literature review pertinent to our research, exploring how SAS and object detection intersect and identifying gaps where our work can contribute to advancing these fields.

Chapter 3

Literature Review

In this chapter, we discuss the literature on Self-Adaptive Systems (SAS), focusing on their development, challenges, and the role of machine learning, ***paving the way to our exploration of self-adaptation in Machine Learning-Enabled Systems.***

3.1 Self Adaptive Systems

Exploring the foundational concepts and evolution of Self-Adaptive Systems (SAS) lays the groundwork for understanding their role in the development of modern computing infrastructures. This section delves into the journey from their inception to their integration with machine learning (ML), highlighting the importance of adaptability and resilience in software systems.

The inception of self-adaptive systems (SAS) research was marked by the introduction of autonomic computing by IBM in 2003 [42], establishing the MAPE-K loop as an architecture for systems to autonomously monitor, analyze, plan, and execute adaptations. This foundational work underscored the necessity for systems to self-manage in the face of environmental changes and internal challenges. Building upon this Cheng et al provided a research road-map that detailed the active field of SAS, emphasizing their capacity to adapt and improve in response to changes, thus setting the stage for future research directions in the field [17].

Building upon the foundation of autonomic computing, the integration of machine learning has marked an enhancement in the adaptability and potential applications of SAS. This evolution shows a natural progression from foundational principles to advanced applications in modern computing.

The role of machine learning in enhancing the adaptability of SAS was explored by Wenys et al, highlighting its integration and pointing to the potential for greater application, especially in unsupervised learning contexts [83]. Further expanding the discussion, De Lemos et al presented a second research road-map, focusing on the design space for adaptive solutions, the shift from centralized to decentralized control, and the importance of run-time verification and validation for SAS [25]. The further research work by de Lemos et al addressed the complex issue of providing assurances in self-adaptive systems, considering their evolving nature and need for reliability and trustworthiness [24]. An empirical study on

the resilience of architecture-based SAS compared to traditional code-embedded adaptation methods demonstrated the benefits of an architectural approach to adaptation [10]. Cámara Javier et al explains a method for elucidating architectural design trade-offs, using machine learning and quantitative verification to aid architects in making informed decisions under uncertainty [12].

Mendonça Nabor C et al discussed the balance between the generality and reusability of SAS frameworks, advocating for solutions that meet the adaptation needs of contemporary software systems [56]. Also, a research work from Cheng et al focused on the progression of SAS from theoretical models to practical, engineered systems, emphasizing the importance of self-adaptation in the software engineering landscape [18]. These contributions collectively highlight the evolution of self-adaptive systems from theoretical constructs to essential components of modern computing infrastructure, emphasizing the ongoing importance of adaptability, resilience, and autonomy in software systems.

As we navigate through the advancements and challenges in SAS, it becomes evident that integrating self-adaptive mechanisms into machine learning-enabled systems represents an uncharted but promising frontier in enhancing system adaptability and efficiency. The next subsection further explore the depth of literature surrounding SAS and the emerging discussions on their role in ML systems.

3.1.1 Literature Reviews in Self-Adaptive Systems

The exploration of self-adaptive systems (SAS) has been enriched by several comprehensive literature reviews, each contributing to a broader understanding of the field's state of the art and guiding future research directions.

Self-adaptive systems: A systematic literature review across categories and domains in 2022 offer extensive overviews of SAS research, tracing the evolution from theoretical underpinnings to practical implementations and more holistic approaches [87] as shown in Figure 3.1. These reviews underscore the field's dynamic progression, noting a shift toward integrating SAS in areas such as networking, web services, and notably, the Internet of Things (IoT) and Infrastructure as a Service (IaaS). They highlight the maturation of SAS research, from initial concepts to frameworks and exemplars that have practical applications in diverse domains, including bio-inspired approaches, security, and cyber-physical systems.

Applying Machine Learning in Self-adaptive Systems: A Systematic Literature Review by Gheibi Omid et al in 2021 specifically addresses the intersection of machine learning (ML) and self-adaptation, noting a rapid increase in leveraging ML within SAS for tasks such as environmental modeling and configuration management [32]. Despite the growth of literature in this niche, the review points out a lack of systematic overviews, which it then provides, focusing on the MAPE feedback loop. This paper highlights on the use of ML for updating adaptation rules and managing resources, emphasizing the dominance of supervised and interactive learning methods while pointing out the underutilized potential of unsupervised learning.

The industrial application and relevance of SAS are scrutinized in [85] in 2023, which contrasts academic research with practical needs. This survey reveals insights into the adoption of self-adaptation for enhancing robustness, performance, and user experience in industrial settings, while also highlighting

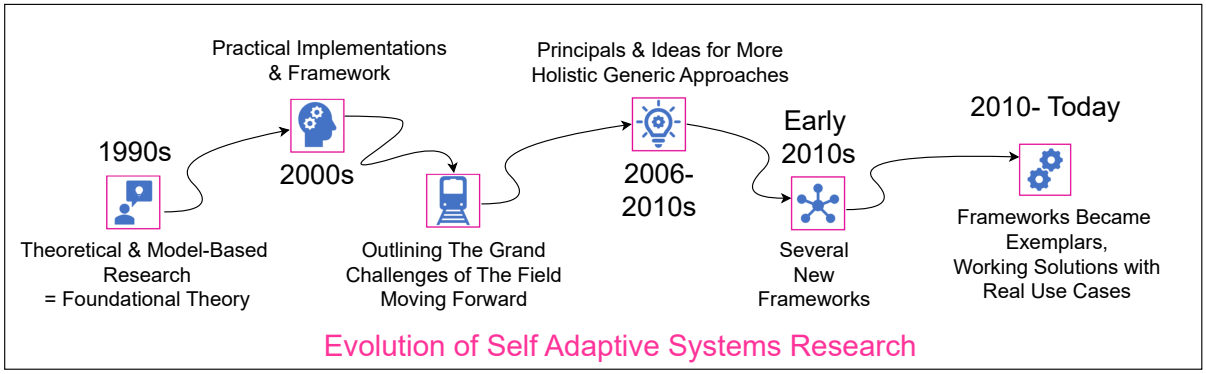

Figure 3.1: Self Adaptive Systems: Research Evolution

the challenges faced when engineering SAS in practice. It suggests a significant alignment between academic pursuits and industrial implementations, with opportunities for future collaborations to foster self-adaptation in real-world applications.

Further, work entitled 'Self-adaptive systems: A survey of current approaches, research challenges and applications' shows current approaches, challenges, and applications of SAS, emphasizing the necessity of autonomy in adaptation and the critical role of feedback loops [52]. It outlines the broad application of SAS across embedded systems and software engineering, supported by various European research initiatives. Lastly, work by Gheibi et al in 2021 delves into the impact of applying ML on the decision-making processes within SAS, urging a consideration of ML's implications on system strategy and performance [33].

Despite the depth of research and systematic reviews covering SAS and the integration of ML, there remains a notable gap in literature specifically addressing the adaptation of ML systems themselves. These comprehensive reviews highlight the dynamic progression of SAS research and shows a significant gap: the underexplored integration of SAS principles within machine learning-enabled systems. Addressing this gap presents a promising avenue for future research.

3.1.2 Addressing Uncertainty in Self-Adaptive Systems

Understanding and managing uncertainty is a fundamental aspect of self-adaptive systems (SAS), as it significantly influences their ability to meet objectives under varying conditions. Research work entitled as 'The Uncertainty Interaction Problem in Self-Adaptive Systems' by Cámara, Javier introduces the Uncertainty Interaction Problem, highlighting the complexities in mitigating uncertainty due to the inter-dependencies among different types, sources, and dimensions of uncertainty [13]. This work emphasizes the need for integrated approaches to uncertainty modeling, aiming to construct more resilient SAS by understanding how uncertainties interact and affect system properties. Further exploring this theme, Calinescu Radu et al surveys the research community's perception of uncertainty within SAS [9]. Findings suggest that systems can be engineered to cope with unanticipated changes through evolving

actions, synthesizing new actions, or employing default actions. The study advocates for the use of confidence intervals, probabilities, and multiple models to manage uncertainties effectively, pointing out the ongoing challenge of ensuring safety-critical SAS.

The potential of decentralized approaches in handling uncertainty is examined in [66], which discusses the advantages of decentralizing adaptation functions for better coordination and management of uncertainties. Moreno Gabriel et al presents an approach that significantly reduces adaptation decision time, demonstrating the effectiveness of proactive self-adaptation under uncertainty [62]. Additionally, Kinneer Cody et al and Moreno Gabriel A et al contribute to the discourse by suggesting methodologies like plan reuse and stochastic search to manage and reduce uncertainty in SAS, underlining the importance of innovative strategies in tackling the inherent uncertainties of these systems [43, 61].

As current literature largely focuses on SAS without a specific emphasis on ML systems, bridging this gap to understand uncertainty management in ML-enabled SAS could lead to the development of more adaptable, reliable, and efficient systems. The exploration of uncertainty in SAS shows the necessity for innovative strategies that could be pivotal when applied to ML systems. The advancements discussed offer a glimpse into how addressing uncertainty could enhance the adaptability and reliability of ML-enabled systems, an area we aim to explore further in next subsections.

3.1.3 Advancements and Techniques in Self-Adaptation

Recent advancements in self-adaptive systems (SAS) shows the field's progression towards integrating sophisticated architectures, machine learning (ML), and human-machine teaming to address complex adaptation challenges as explained in Figure 3.2.

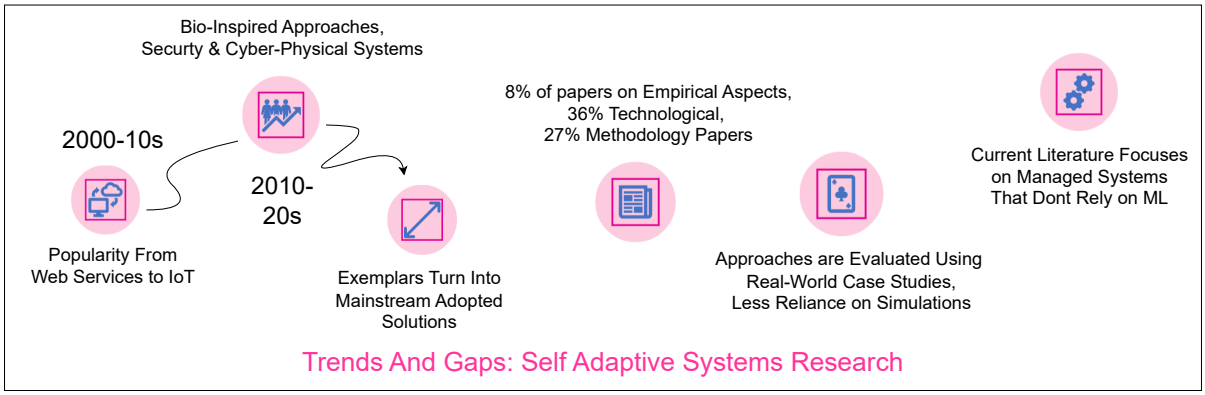

Figure 3.2: Trends and Gaps: Self Adaptive Systems

Moreno Gabriel A et al delves into software architecture and task plan co-adaptation for mobile service robots, highlighting the need for SAS to adapt both structural and behavioral aspects dynamically [11]. This work illustrates the complexity of making runtime decisions based on the mutual dependencies between software architecture and task planning under uncertainty. The project by Weyns Danny et al introduces a self-adaptive service-based system exemplar, offering a reference implementation to evaluate

and compare new self-adaptation methods, techniques, and tools. This exemplar serves as a foundational platform for further SAS research, particularly in service-based systems [84].

Cámara Javier et al presents a tandem approach that combines quantitative verification and machine learning to architect self-adaptive IoT systems capable of maintaining Quality of Service (QoS) levels amidst inherent uncertainties [22]. This innovative methodology signifies a shift towards proactive self-adaptation, enhancing decision accuracy and system resilience. Bureš, Tomáš proposes a vision where self-adaptation stands in equal relationship to AI, suggesting a synergistic approach that benefits from and enables AI, marking a pivotal direction for future SAS development [8].

Furthering the discourse, C. Kinneer et al [44], Schmerl Bradleyet al [73], and Moreno Gabriel A et al [60] introduce techniques such as stochastic self-* planners, architecture-based self-protection, and proactive latency-aware decision-making. These techniques demonstrate the evolving toolbox for SAS, aiming to improve re-usability, security, and performance. Works by Cámara Javier et al and Cleland-Huang et al reflect on real-world applications and extensions of SAS [21, 20]. The former shares insights from integrating the Rainbow platform for architecture-based self-adaptation into industrial middleware, while the latter augments the MAPE-K loop with support for Human-Machine Teaming (HMT), emphasizing the importance of human interaction in autonomous systems.

Li Nianyu et al and Cheng ShangWen et al address the human aspects and resource management challenges in SAS, proposing frameworks to prepare human operators for tasks in self-adaptive environments and improve architecture-based self-adaptation through resource prediction [50, 19]. Lastly, Simon Reichhuber et al and Gheibi Omid et al explore the frontiers of lifelong self-adaptation and the application of active reinforcement learning [70]. These works by Gheibi Omid et al introduces a novel approach that enhances SAS with a lifelong machine learning layer to adapt to concept drift [31], while [70] provides a road-map towards curious classifier systems for self-adaptation.

These contributions collectively represent the current state of self-adaptation, highlighting the integration of advanced architectures, machine learning, and human-centric approaches to tackle the multifaceted challenges of SAS. As we have observed significant work in self-adaptation, the next section will explore the challenges associated with software systems that incorporate machine learning and further investigate the potential for self-adaptation within these ML-enabled systems, where managing uncertainty will play a important role.

3.1.4 Challenges in Machine Learning-Enabled Systems

As the integration of machine learning (ML) into software systems becomes increasingly common, new classes of challenges have emerged, demanding attention and novel solutions. These challenges, spanning a wide array of applications and technical issues, highlight the complexity of deploying and maintaining ML-enabled systems. Branco Bernardo et al and Erickson Bradley J et al illustrate the application-specific challenges of ML in fraud detection and medical imaging, respectively, pointing out the reliance on ML for critical classification tasks further emphasizes the ubiquity of ML in systems

requiring dynamic rule management for fraud detection, underlining the operational challenges faced by these systems[7, 27, 4].

The issue of dataset shift, as explored in [2] and further investigated in [67], presents a significant hurdle for ML systems. These studies explore the detection of dataset shift and its impact, emphasizing the silent failure modes of ML systems when confronted with unexpected inputs. Miller Brad et al addresses the integration of ML with expert reviewers in malware detection, showcasing the critical role of human expertise in complementing ML, especially in evolving threat landscapes [57].

Microsoft's case study sheds light on the software engineering challenges encountered when developing AI-based applications, revealing the intricacies of data management, model customization, and the modular integration of AI components within software projects [3]. This is complemented by Gartner's report, which starkly notes that half of ML models fail to transition into production, highlighting the gap between development and operationalization [1].

Technical challenges such as detecting concept drift in data streams [65], avoiding adversarial attacks [38], implementing machine unlearning [14], and rapid retraining of ML models [88] shows the dynamic nature of ML-enabled systems and the continuous need for adaptation and vigilance.

These studies collectively outline the multifaceted challenges faced by ML-enabled systems, from application-specific hurdles to broader issues of data management, model robustness, and integration within traditional software engineering practices. As we delve into these challenges, the potential for self-adaptation in addressing the complexities of ML systems becomes evident, opening up intriguing possibilities for enhancing the reliability, efficiency, and security of these systems. The exploration of self-adaptation strategies tailored for ML-enabled systems promises not only to mitigate these challenges but also to advance the state of the art in software engineering for AI.

3.1.5 Self-Adaptation in Machine Learning-Enabled Systems

As machine learning (ML) systems become increasingly integral to a wide array of applications, the unique challenges they present become more apparent. The literature reveals a breadth of issues, from architectural considerations to the dynamic nature of data and operational environments these systems must navigate. Lewis Grace A et al and Henry Muccini et al outline the software architecture challenges specific to ML systems, emphasizing the necessity of addressing data-dependent behavior, drift over time, and the critical need for continuous retraining informed by timely ground truth capture [49, 64]. These challenges necessitate architectural practices and patterns that cater specifically to the nuances of ML systems.

The integration of ML into traditional software architectures not only introduces new complexities but also magnifies existing challenges around data management, model customization, and component integration, as detailed in the Microsoft case study [3]. Issues such as dataset shift [2] and silent failures in response to unexpected inputs [67] further underscore the unpredictable nature of ML systems' operational environments. Detecting concept drift using human feedback [65], avoiding adversarial attacks [38], and tactics for unlearning and retraining [14, 88] represent specific technical challenges that

highlight the need for adaptable and responsive ML systems. Zhang, Jeff (Jun) et al introduces a novel idea to managing fluctuating workloads by switching between models of varying complexity by external system in cloud computing, hinting at the potential for self-adaptation strategies in ML systems [92].

Despite the growing body of knowledge on engineering challenges and techniques for ML-enabled systems, there remains a significant gap in the literature concerning the self-adaptation of these systems. Casimiro Maria et al take the first steps toward addressing this gap by proposing frameworks for self-adapting systems reliant on ML components [15, 16]. These works highlight the necessity of detecting misbehavior in ML components and adapting to maintain desired system qualities, such as accuracy and efficiency, amidst changing operational conditions and emerging patterns. Krupitzer Christian et al, Ervasti Kim et al and Wohlrab Rebekka et al further the discussion by exploring engineering approaches for SAS that incorporate ML models, emphasizing the complexities of managing these systems and the importance of considering multiple objectives and stakeholder preferences in adaptation decisions [46, 28, 86]. The advancements in self-adaptation techniques, especially those incorporating machine learning and human-centric approaches, open new pathways for addressing the complex challenges of ML-enabled systems. As we conclude this exploration of SAS, the potential for these techniques to revolutionize the adaptability and efficiency of ML systems is both exciting and largely uncharted, guiding the direction of our subsequent investigation.

Moreover, to the best of our knowledge, we are not aware of any comprehensive work that thoroughly analyzes and discusses the nuanced interplay between self-adaptive systems (SAS) and machine learning-enabled systems (MLS) in depth. This oversight in the literature reveals a critical gap in our collective understanding of how self-adaptive mechanisms can be effectively integrated into MLS to address the unique challenges they face, particularly in managing uncertainty and adapting to dynamic operational environments. As we conclude this section, it's evident that despite substantial progress in identifying and tackling the challenges within ML-enabled systems, the realm of integrating self-adaptation into these systems remains largely unexplored. This thesis seeks to bridge this gap, proposing a novel exploration of self-adaptation through the mechanism of runtime model switching within ML systems, showing promising avenues for enhancing their performance and reliability. Our research not only aims to enrich the field of self-adaptive systems research but also aims to equip ML systems with the robustness, adaptability, and resilience needed to thrive amidst the ever-changing demands of their operational contexts.

3.2 Object Detection: A Use Case for Self-Adaptation in ML Systems

In exploring the integration of self-adaptive systems within machine learning, object detection serves as a practical use case. This section delves into how self-adaptation can address inherent challenges in object detection, showcasing the potential for enhanced system performance.

As our research showcases, the novel approach of self-adaptation through runtime ML model switching, particularly at the system and inference level, is exemplified through the domain of object detection.

Object detection, a main field within machine learning/deep learning, has seen remarkable advancements and plays a key role in various applications, from autonomous driving to smart surveillance systems. Object detection in 20 years: A survey in 2023 [93] provides a comprehensive survey of the achievements and challenges in object detection over the past two decades, highlighting the evolution of techniques and the emergence of lightweight, end-to-end, and small object detection methods. Despite considerable progress, there remains a performance gap, especially in real-time applications and detecting objects with multi-source information, pointing towards the necessity for innovative approaches in object detection.

J. Huang et al delves into the trade-offs between speed, accuracy, and memory usage in modern convolutional object detection systems, emphasizing the importance of selecting the right detection architecture for specific application needs [37]. This work guides through various configurations within ”meta-architectures” to achieve desired balances, which is important for deploying object detection in varied contexts, from mobile devices to high-accuracy requirements. Recent advancements, such as Interactron presented in CVPR 2022, propose methods for adaptive object detection by continuing model training during inference, without explicit supervision. This approach hints at the potential for systems to adapt in real-time to environmental changes, improving detection accuracy significantly [45].

Hou Jie-Bo et al and Zhang Chong et al introduce techniques aimed at enhancing object detection through self-adaptive aspect ratios and progressive representation alignment, respectively [36, 91]. These methods address specific technical challenges within object detection, such as the variation in aspect ratios and domain shift, demonstrating the ML community's efforts towards more adaptable detection models. Despite these advancements, challenges persist, particularly when addressing complex scenes with dynamic backgrounds or illumination variations, as noted in [35]. The Global-Local Self-Adaptive Network proposed in [26] for drone-view object detection tackles tiny-scale objects and unevenly distributed object challenges, showcasing the potential of self-adaptive strategies in specialized detection scenarios. While advancements in object detection have led to significant improvements, the application of self-adaptive strategies at a system level remains underexplored. This oversight presents a critical gap where self-adaptation could offer substantial benefits. The persistent challenges and the lack of comprehensive exploration into self-adaptive object detection systems shows the need for innovative research approaches.

Moreover, we're not aware of any detailed work that fully explores how self-adaptation can be applied in the context of object detection within Machine Learning-Enabled Systems. Even though object detection is a key area with lots of innovations, where self-adaptation could greatly benefit system performance and reliability, the focus has mostly been on improving models themselves, not on how these models can be dynamically switched or adapted at a system level based on changing conditions. There's a clear gap when it comes to looking at object detection from a broader, system-level perspective, where self-adaptation could help select the best model to use based on the situation, constraints, and requirements at hand. Previous studies like those by Casimiro Maria et al have started to look into the self-adaptation of ML-based systems focusing on strategies like retraining models [15, 16]. However,

they haven't delved deeply into the idea of switching between models as a way to adapt. Despite the progress in object detection, these efforts mainly aim at making small, model-specific improvements.

This leaves room for a new research: applying self-adaptation on a system-wide level to tackle the dynamic challenges found in object detection. Our research aims to fill this gap, presenting a way that not only improves object detection systems' performance and reliability but also demonstrates how self-adaptation can be practically implemented. By doing so, we offer a valuable contribution to the fields of vision, AI, and ML, potentially guiding future research and applications in ML-enabled systems towards adopting more flexible and responsive adaptation strategies.

3.3 Exemplars for Self-Adaptive Systems

Exemplars play a key role in the research and development of self-adaptive systems, serving as testbeds for novel strategies and frameworks. This section discuss the importance of such exemplars in advancing our understanding and capabilities in self-adaptation.

As we've navigated through the intricacies of self-adaptive systems, literature reviews on such systems, the importance of addressing uncertainty within these frameworks, and the advances in self-adaptive machine learning-enabled systems, we've established a strong foundation on the necessity of self-adaptation. We've also discussed the challenges within machine learning-enabled systems and our contribution which is an innovative approach of runtime model switching to tackle these challenges, as showcased through object detection use cases.

However, venturing directly into real-world applications of such systems can be tricky and financially risky. This necessitates a sandbox environment or a tool where strategies can be tested and refined safely. This is where the concept of an exemplar comes into play—an exemplar is essentially a software tool or framework designed to facilitate experimentation and learning in a controlled environment.

The SEAMS community [1], a symposium focused on Software Engineering for Adaptive and Self-Managing Systems, has been at the forefront of promoting such exemplars. Since its inception, SEAMS has been a repository and breeding ground for exemplars across various domains, underscoring the community's commitment to advancing research and application in self-adaptive systems. Despite this rich repository, there remains a noticeable gap: the absence of exemplars tailored for machine learning-enabled systems.

A review of the current exemplars listed on the SEAMS website[1], such as SWIM[63] for web/cloud/service applications, ATRP[89] for autonomous vehicles/robotics, and DeltaIoT[39] for cyber-physical systems/IoT, reveals a diverse range of domains being explored. Yet, there is a distinct lack of tools specifically designed for experimenting with self-adaptive strategies in ML systems.

Moreover, we haven't seen much work dedicated to creating exemplars for machine learning-enabled systems that need self-adaptive strategies. This gap is significant because testing and refining these strategies in a safe, controlled environment could lead to breakthroughs in how we apply self-adaptation

[1] `https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/`

in ML systems. In response to this identified need, our work contributes SWITCH, an exemplar developed to explore self-adaptive strategies within ML systems. This tool represents a step toward enriching the exemplar repository with resources that directly address the challenges of self-adaptation in the context of ML.

For further details on the tradition of exemplars within the SEAMS community and the wide array of existing exemplars, please refer to their repository[1]. This initiative not only acknowledges the rich heritage and contribution of SEAMS to the advancement of self-adaptive systems but also highlights the need for expanding this repository to include tools that address the emerging challenges of ML-enabled systems.

3.4 Sustainability and Self Adaptation

In this discussion, we focus on how self-adaptive systems, especially within machine learning, plays an important role in addressing sustainability challenges. The growing concern over the environmental impact of ML technologies calls for innovative solutions. The burgeoning field of Green AI addresses the environmental impact of ML, highlighting a need for adaptive strategies that ensure sustainability without compromising system performance. Recent advancements in ML, especially in natural language processing (NLP) and deep learning, have significantly increased model accuracies. However, this progress comes at a cost, notably in terms of energy consumption and associated carbon emissions. Strubell et al [75] quantified the financial and environmental costs of training state-of-the-art NLP models, revealing substantial energy requirements and urging the research community towards more cost-effective and equitable practices.

The energy efficiency of ML accelerators is another area of focus. Reuther et al [71] surveyed and benchmarked a range of processors and accelerators, identifying trends in power consumption and performance that are crucial for designing sustainable ML systems. These insights are particularly relevant for applications in domains with strict Size, Weight, and Power (SWaP) constraints, emphasizing the need for low-power solutions that do not sacrifice computational capabilities. As we consider the integration of ML into broader AI services, the call for Green AI technologies becomes louder. The review by Mehta Yukta et al not only champions sustainable solutions but also highlights the challenges at the community level, pointing out the necessity for accurate forecasting in energy consumption and the selection of appropriate ML algorithms to minimize environmental impact [55].

The estimation of energy consumption in ML, as discussed by Garcia Martin et al [29], presents a comprehensive overview of methodologies to assess energy use in data mining and neural networks. This review stresses the importance of incorporating energy metrics into the design and operation of ML systems, advocating for a shift towards energy-aware ML development. Martinez et al [54] contribute to this dialogue by surveying software engineering approaches for AI-based systems, underscoring the predominance of software testing and quality in research, while highlighting the overlooked area of

software maintenance. Data-related issues emerge as a recurrent challenge, emphasizing the integral role of sustainable practices in the life-cycle of AI-based systems.

The quantification of carbon emissions by Lacoste et al [48] provides a practical tool for the ML community to better understand the environmental impact of training models. Their Machine Learning Emissions Calculator serves as a call to action, encouraging both individual practitioners and organizations to adopt measures that mitigate carbon emissions. However, the pursuit of larger and more complex language models raises concerns about their environmental and financial viability. Bender et al [6] critically examine the trend towards ever-larger models, advocating for a balanced approach that considers the environmental, financial, and societal costs of such developments.

Despite the growing awareness and efforts to address the environmental implications of ML, there remains a significant gap in strategies that enhance the energy efficiency of model inference in real-world applications. The discussions surrounding Green AI have predominantly focused on optimizing the training phase, with less emphasis on the energy demands of inference [90, 5, 40, 80]. This oversight shows the need for self-adaptation techniques that can dynamically balance energy efficiency with Quality of Service (QoS), an area opens up for exploration [58].

The field of Green AI has attracted substantial research interest, focusing on energy efficiency in Machine Learning (ML) systems. Verdecchia et al [81] reviewed 98 studies, highlighting a dominant emphasis on energy efficiency mechanisms. Despite this, the practical application of these findings, particularly in making ML-enabled systems sustainable, has been limited. Studies like those by Jarvenpaa et al [40] propose tactics for sustainability, mainly at the design stage, covering data-centric methods [80] and optimization of algorithms and models. However, the aspect of runtime energy efficiency has been relatively unexplored. The concept of self-adaptation in software, originating from IBM's autonomic computing [87], has evolved to include Machine Learning Systems (MLS). Research in this area [15, 64, 59] categorizes self-adaptation into software design approaches (SDA) and system engineering approaches (SEA), focusing on design-time solutions. Recent studies by Gerostathopoulos et al explore enhancing adaptability at runtime, including unsupervised learning and model switching [30]. Yet, these approaches often limit systems to pre-set configurations, as noted by Tundo et al [77].

Moreover, to the best of our knowledge, there is a noticeable lack of research focusing on directly applying self-adaptation principles to improve sustainability in machine learning systems. While there's a growing conversation around Green AI and its importance, actual strategies that dynamically adjust model operations to balance performance with energy efficiency are sparse. As we explore sustainability as our application domain, it becomes clear that self-adaptation could play a pivotal role in bridging this gap, offering a pathway to more environmentally friendly ML applications without compromising on the quality of service or performance.

Our research directly addresses this gap by demonstrating the potential of self-adaptation mechanisms in ML systems to enhance sustainability. By integrating energy efficiency considerations into the adaptive decision-making process, we present a novel approach that not only mitigates the environmental impact

of ML technologies but also advances the field towards more responsible and sustainable computational practices.

3.5 Discussion

In this chapter, we've looked at Self-Adaptive Systems (SAS) and found areas that need more attention:

SAS and ML Integration: We've seen that SAS have come a long way, but such systems with machine learning (ML) inside is something not much talked about in research.

Self-Adaptation for ML Systems: The idea of making ML systems self-adaptive based on uncertainties hasn't been explored much, especially across different ways of using ML like in streaming data or online-deployment mode for real-time analysis.

Uncertainty in ML Systems: While SAS are known for handling uncertainty, how this applies to ML systems specifically hasn't been covered enough.

Tools for Testing Self-Adaptive Systems: We noted a lack of special tools (exemplars) for testing how self-adaptive strategies work in ML systems, showing a gap in resources for researchers.

Sustainability Through Self-Adaptation: Our look into sustainability showed it's possible to make ML systems be more energy efficient through self-adaptation, a topic has been unexplored.

Moving forward, we aim to fill these gaps, focusing on how self-adaptation can be specifically applied to and developed for machine learning-enabled systems, across various deployment modes, and how they can help make these systems more energy-efficient.

Chapter 4

Machine Learning Model Balancer

In this chapter, we introduce ***Machine Learning Model Balancer concept*** as a way to self-adapt machine learning-enabled systems through dynamic model switching to address operational uncertainties and challenges for enhancing Quality of Service.

4.1 Introduction

Building upon the foundations laid in Chapters 2 and 3, we have established a comprehensive understanding of self-adaptive systems and identified a significant gap in the literature concerning the self-adaptation of machine learning-enabled systems. Particularly, while the need for adaptive actions in such systems is well-recognized, concrete implementations of these actions remain largely unexplored. Notably, the works by Maria et al [15, 16] and take first steps toward bridging this gap. Their studies shows the potential of various adaptation strategies, including retraining, transfer learning, and adversarial actions, yet they stop short of detailing their practical application within self-adaptive systems.

The discussions this far have highlighted the challenges faced by machine learning systems at the inference level. Uncertainties such as data drift and model drift not only compromise the accuracy and reliability of these systems but also call into question their ability to meet predefined system goals under evolving operational conditions. It is within this context that our research introduces a novel contribution: the concept of runtime model switching as a dynamic and sensible action for maintaining and enhancing service quality in machine learning-enabled systems named as *Machine Learning Model Balancer* concept.

This chapter explains this concept, positing that among the various adaptive actions proposed in the literature, switching between multiple machine learning models at runtime presents a feasible and impactful solution. Unlike the approach suggested by [92] , which introduces model switching as a mechanism to manage fluctuating workloads externally, our concept centers on an inherently self-adaptive methodology. This distinction emphasizes not just the novelty but the necessity of our approach, particularly in scenarios where real-time data processing and decision-making are paramount.

Our model switching idea is based on the understanding that no single ML model can optimally address all operational scenarios. By enabling a system to autonomously switch between an array of models at the inference level, we aim to circumvent the limitations posed by static model deployments. This approach aligns with the overarching goal of self-adaptive systems: to achieve optimal performance and reliability in the face of dynamic operational environments and evolving system objectives.

In the forthcoming sections, we will explore the challenges of integrating machine learning into self-adaptive systems in greater depth, laying the groundwork for a comprehensive discussion on the Machine Learning Model Balancer concept. In this exploration, we aim to explain the idea behind our concept and show its practical use. We want to show how it can change the way self-adaptation works in systems that use machine learning.

4.2 Challenges in Self-Adaptation of ML-Enabled Systems

Addressing operational challenges in ML-integrated systems, especially during inference, is crucial for maintaining performance and reliability. Our focus is on recognizing these uncertainties and their impacts, laying the groundwork for adaptive strategies to improve adaptability and efficiency. We highlight some key challenges underscoring the need for our Machine Learning Model Balancer concept.

1. Environmental Uncertainty (Varying Load): The unpredictability of user demand and system load, which can fluctuate significantly, impacting the system's ability to process and respond in real-time.

2. Data and Model Drift: Changes in the operational environment that lead to data drift and model drift, compromising the accuracy and relevance of the ML model over time.

3. Resource Uncertainty: Fluctuations in the availability and allocation of computational resources, influenced by competing processes or inherent limitations of the deployment environment.

4. Network Uncertainty: Variability in network conditions affecting the system's ability to access external data sources or services, impacting the timeliness and accuracy of ML predictions.

5. Unstable Software Components: The presence of other software components within the system that may exhibit unpredictable behavior, affecting overall system stability and ML model performance.

6. Quality of Service Uncertainty: The challenge of ensuring a consistent level of service quality, given the probabilistic nature of ML predictions and the dynamic operational environment.

7. Performance Uncertainty: The variability in the effectiveness and efficiency of the ML model under different conditions, complicating the prediction of system performance.

As explained in Table 4.1 the limitations of existing self-adaptive strategies, highlighting the need for a sophisticated solution capable of navigating the demands of ML-enabled systems. The Machine Learning Model Balancer, our novel contribution, is specifically designed to address these challenges.

These challenges underline the limitations of standard adaptive strategies for ML systems. We present the Machine Learning Model Balancer as a solution, enabling flexible run-time model switching to improve adaptability and maintain performance across different situations. This approach aims to offer a practical way to address these challenges, which we will detail further in the upcoming section.

Table 4.1: Challenges, impact on ML-Enabled Systems and necessity for adaptive mechanisms

Challenge	Impact on ML-Enabled Systems	Necessity for Adaptive Strategies
Environmental Uncertainty (Varying Load)	Leads to potential system overloads or underutilization during fluctuating user demands.	Adaptive strategies optimize resource utilization and model deployment to maintain performance.
Data and Model Drift	Decreases model accuracy over time, necessitating continual adaptation to preserve system effectiveness.	Real-time adaptation mechanisms ensure models remain relevant and accurate.
Resource Uncertainty	Limits the system's capacity for efficient data processing and model execution.	Dynamic resource management enables the system to operate within available constraints.
Network Uncertainty	Introduces delays and inconsistencies in data access, affecting prediction accuracy.	Adaptation strategies compensate for network variability, ensuring reliable ML predictions.
Unstable Software Components	Affects overall system reliability and complicates model performance consistency.	Implementing robust adaptation approaches enhances system stability despite component variability.
Quality of Service Uncertainty	Challenges the system's ability to deliver a consistent level of performance and reliability.	Adaptable models and strategies improve service quality under dynamic operational scenarios.
Performance Uncertainty	Complicates the predictability of system efficiency and effectiveness.	Adaptive model selection and deployment strategies optimize performance under varying conditions.

4.3 Machine Learning Model Balancer

As we navigate through the evolving landscape of self-adaptive systems (SAS) enhanced by ML as a part of underlying managed system, the introduction of the Machine Learning Model Balancer concept addresses the intrinsic limitations and challenges faced by ML-enabled systems. This concept emerges as a necessity for ensuring optimal performance and reliability across diverse operational scenarios. The Machine Learning Model Balancer is designed to autonomously switch between a ”squad” of ML models

in real-time, each tailored to specific aspects of the operational environment. This dynamic selection ensures that the system can adapt to environmental changes, varying loads, and shifting objectives without human intervention. It represents a solution that enhances system adaptability, efficiency, and effectiveness, leveraging the strengths of different models to achieve the best possible outcome under any given conditions. The core of the Model Balancer lies in its ability to seamlessly navigate the complexities of runtime operation, making informed decisions about which ML model is best suited for the current context. This process involves:

- **Dynamic Selection**: Analyzing the current operational conditions and selecting the most appropriate model from the "squad" based on predefined criteria.
- **Seamless Switching**: Transitioning between models with minimal latency, ensuring continuous system operation.
- **Performance Optimization**: Continuously monitoring system performance to optimize the selection and switching process, thereby enhancing the overall Quality of Service (QoS).

Consider a user engaging with ChatGPT. For rapid queries, ChatGPT-3.5 might be preferred for its speed, whereas deeper inquiries might benefit from ChatGPT-4's comprehensive analysis. Traditionally, toggling between these versions requires user discretion. Imagine, however, a system so advanced that it intuitively selects the appropriate version based on the user's query type, optimizing for either speed or depth as needed. This scenario shows how the idea of model switching can be used in practical scenarios. The Machine Learning Model Balancer directly addresses the challenges identified in ML-enabled systems as explained in Table 4.2:

Implementing the Machine Learning Model Balancer requires careful consideration of several key factors, including:

- **Criteria for Model Switching**: Establishing clear guidelines for when and why to switch between models based on performance metrics and environmental conditions.
- **Managing Transition Latency**: Developing strategies to minimize the impact of model switching on system operation, ensuring seamless performance.
- **Efficient Model Transition Mechanisms**: Creating robust mechanisms for model switching that maintain system state and dependencies.
- **Strategic Model Placement**: Identifying optimal scenarios and system components for model switching to maximize impact on performance and reliability.

By addressing these considerations, the Machine Learning Model Balancer concept offers a structured approach to enhancing the adaptability, resilience, and performance of ML-enabled SAS. This approach addresses the challenges posed by integrating ML into self-adaptive systems.

Table 4.2: Addressing Challenges with the ML Model Balancer

Challenge	Solution Offered by the ML Model Balancer
Environmental Uncertainty (Varying Load)	Optimizes resource utilization and model deployment dynamically to maintain performance during fluctuating user demands.
Data and Model Drift	Ensures models remain relevant and accurate through real-time adaptation, addressing the decrease in model accuracy over time.
Resource Uncertainty	Implements dynamic resource management to efficiently operate within the available constraints, adapting to fluctuations in computational resources.
Network Uncertainty	Compensates for network variability through adaptable strategies, maintaining reliable ML predictions even under adverse conditions.
Unstable Software Components	Enhances system stability by adapting to component variability, ensuring consistent model performance and system reliability.
Quality of Service Uncertainty	Improves service quality under dynamic operational scenarios through adaptable models and strategies, addressing the probabilistic nature of ML predictions.
Performance Uncertainty	Optimizes performance under varying conditions through adaptive model selection and deployment strategies.

4.4 Discussion

RQ1: In the context of Machine Learning-Enabled Systems (MLS), how can self-adaptive strategies be developed and applied to manage and mitigate runtime uncertainties, thereby maintaining or enhancing their Quality of Service (QoS)?

Responding to RQ1 as discussed in chapter 1, this chapter introduces a significant step forward in the development and application of self-adaptive strategies for managing and mitigating runtime uncertainties in Machine Learning-Enabled Systems (MLS), thus aiming to maintain or enhance their Quality of Service (QoS). In this discussion the Machine Learning Model Balancer concept emerges as a pivotal solution to the highlighted challenges.

Addressing Identified Gaps: The discussion in this chapter has not only pinpointed but also elaborated on the complex challenges that MLS face during their operational phase, particularly at

the inference level. By clearly outlining these challenges—from environmental uncertainties to the limitations imposed by ML models—we have identified the pressing need for dynamic strategy capable of navigating through the fluctuating landscapes of operational demands and system performances.

Practical Implementation of Adaptive Strategies: The Machine Learning Model Balancer concept, introduced here, represents an innovative approach that moves beyond theoretical discussions, providing a practical solution that enables runtime model switching. This concept surpasses traditional adaptation methods by introducing a real-time decision-making layer, allowing systems to autonomously choose the most suitable ML model based on the prevailing operational conditions. This strategy addresses the outlined challenges directly, opening pathways to improve system resilience, flexibility, and efficiency.

Shifting Paradigms: This innovative approach encourages a shift in how we perceive and implement self-adaptation within MLS. By granting systems the ability to dynamically switch among a set of models, we're addressing uncertainties head-on and actively enhancing the system's capacity to sustain high service quality under diverse conditions. This marks a notable advancement toward realizing the potential of self-adaptive systems, bridging the gap between static ML deployments and the dynamic nature of operational environments.

Looking Ahead: The Machine Learning Model Balancer concept sets the foundation for future exploration and research. The forthcoming examination of AdaMLS will demonstrate a concrete application of this concept, illustrating its practical benefits and feasibility of runtime model switching. This exploration reinforces our response to RQ1 while paving the way for addressing further research questions, contributing to the evolution of self-adaptive strategies within MLS.

Toward a Comprehensive Understanding of Adaptive Strategies: The conversation around the Machine Learning Model Balancer concept, its implications for self-adaptive MLS, and its significance in the broader landscape of computational environments culminates in a broader call to action. There's an evident need to expand our collective knowledge, tools, and methodologies to fully embrace the dynamic and uncertain nature of today's computational challenges. The work from conceptualization to practical application, as will be demonstrated by AdaMLS in upcoming chapter 4, highlights the iterative and collaborative effort required to mitigate the uncertainties of self-adaptation in MLS.

In summary, our discussion not only answers RQ1 by showcasing the development and application of self-adaptive strategies within MLS but also emphasizes the Machine Learning Model Balancer as a important concept for enhancing QoS amidst runtime uncertainties. Through the principles of dynamic model switching, this novel approach aims to envision a future where MLS can seamlessly adapt to ensure optimal performance and reliability, irrespective of operational environment challenges.

Chapter 5

AdaMLS: Approach For Self-Adaptation of ML-Enabled Systems

In this chapter, we explore the advancement and integration of machine learning model balancer concept within self-adaptive systems, establishing a *foundation in adaptive machine learning* for our exploration into Machine Learning-Enabled Systems (MLS). Here, we introduce ***AdaMLS: a novel approach for enhancing system's QoS through dynamic model switching and uncertainty mitigation***, laying the groundwork for addressing the complexities of runtime uncertainties in MLS.

5.1 Introduction

In the preceding chapters of this thesis, we delved into the Machine Learning Model Balancer concept, illustrating how integrating machine learning into modern software systems presents numerous challenges. Self-adaptation emerged as a promising solution for addressing these complexities. Specifically, in Chapter 4, we discussed the potential of dynamically switching between models at runtime to improve the Quality of Service (QoS), presenting a significant stride towards mitigating the inherent uncertainties of Machine Learning-Enabled Systems (MLS). This raises a crucial question: how can we operationalize this concept? Is there an approach that allows us to implement this dynamic model switching effectively?

To address this, we leverage the Machine Learning Model Balancer concept, proposing a novel approach, AdaMLS, which is software architecture-based self-adaptation approach using the MAPE-K (Monitor, Analyze, Planner, Executor, Knowledge) framework. This framework facilitates a structured method for adapting system behavior in real-time, ensuring optimal performance and service quality. Machine learning-enables systems operates in various operational modes, such as online, batch, and streaming. In this chapter, we focus on the online mode—a setup where a request triggers the machine learning model within a software system, processing it to deliver an output. This setup shows the operation of many current ML-enabled software systems, providing services in real-time to meet user demands. In such a scenario, the Machine Learning Model Balancer concept holds the potential to significantly enhance system adaptability and performance.

This chapter discusses AdaMLS, which operationalizes the concept of the ML Model Balancer. AdaMLS consistently excels in navigating the intricacies of online ML deployments, ensuring superior

QoS. This includes: i) monitoring model and system parameters; ii) analyzing model and system quality for QoS violations; iii) using knowledge from lightweight unsupervised learning to dynamically switch models, ensuring QoS; and iv) executing system adaptation. Prioritizing ML model adaptability, AdaMLS shifts from conventional load balancing to QoS-aware dynamic ML model switching. By continuously tuning model selections in response to environmental uncertainties and system demands, AdaMLS guarantees MLS QoS, promoting consistent MLS operation in live settings. This represents a self-adaptive MLS, designed to maintain an optimal performance equilibrium amidst changing data and user demands. We evaluate AdaMLS using an object detection use case through utility showcasing a self-adaptive prototype. Our preliminary findings indicate that the runtime model switching, facilitated by lightweight unsupervised learning, effectively manages both system and model performance. This enables AdaMLS to surpass both naive strategies and individual models in terms of Quality of Service (QoS). Our work adapts the MAPE-K loop to address the uncertainties inherent in MLS, emphasizing dynamic model-switching approach. Through AdaMLS's real-world application, we highlight our move towards self-adaptive MLS that can quickly switch between models based on data shifts and user demands, always maintaining optimal QoS.

The structure of the chapter is as follows: Section 5.2 discusses the running example. Section 5.3 details the AdaMLS methodology and section 5.4. Section 5.5 presents the results from its application. Section 5.6 provides the conclusion.

5.2 Running Example

Our approach, AdaMLS, is demonstrated using an object detection system. Object detection system signifies progress in the evolution of machine learning technologies over recent decades [93]. The architecture of AdaMLS follows a design typical to current machine learning systems (MLS). It begins with a *web_service* featuring a REST API, the primary method for user interaction and request submission. This component is important as it conceals the system's complexity and ensures easy integration with various applications. Next to the web service is the *model_repo* i.e. model repository which serves as a central place for storing and managing different machine learning models. This repository, together with the with *model_loader* for dynamic model selection and versioning, guarantees that the most suitable and up-to-date models are available for inference tasks. The system also includes a *message_broker*, a key element for orchestrating data flow—such as image data for object detection—through the system. Acting as an intermediary, the message broker ensures efficient and reliable message transfer between the web service, the model repository, and the object detection models. This configuration facilitates system compartmentalization and scalability. Finally, at the core of our system is the ML model, pretrained for object detection tasks, utilizing the YOLO algorithm [69], known for its effectiveness and precision in real-time object detection challenges. Our architecture reflects the structures of leading solutions like Google Cloud Vision and Amazon Rekognition, highlighting its pertinence and viability for real-world MLS applications.

In our example, we define a set M of available machine learning models, each denoted by m_j where j ranges from 1 to n, and n represents the total number of models in the set. This ensemble includes variants of the YOLOv5 models—YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x—each provided by Ultralytics and pre-trained on the COCO 2017 training dataset[41]. The models in M are evaluated based on their mean Average Precision (mAP), along with other performance metrics such as processing time per image (τ'), model's processing time (τ), and the overall system response time (r), which includes network, queuing, and processing delays.

To illustrate, we present a table summarizing key characteristics and performance metrics for two YOLOv5 model variants as examples:

Table 5.1: Summary of YOLOv5 Model Variants' Characteristics and Performance Metrics

Model	**Parameters**	**mAP**	**System Processing Time (τ')**	**Model Processing Time (τ)**	**Overall System Response Time (r)**
YOLOv5n	1.9M	28	$\tau' =$ $\tau +$ other processing times	45 ms	$r = \tau' +$ queuing time
YOLOv5x	86M	50.7	$\tau' =$ $\tau +$ other processing times	766 ms	$r = \tau' +$ queuing time

The diverse models offer varying response times and mAP scores, yet none present a perfect balance between these two factors. To manage this, we establish a set of thresholds for operational parameters:

- mAP scores: $C_{\max}$ and $C_{\min}$ define the upper and lower limits for detection accuracy, ensuring the system maintains high accuracy while accommodating performance variability.
- Response times: $R_{\max}$ and $R_{\min}$ delineate the acceptable range for response times, balancing promptness against the computational demands of precision models.

This setup allows for the prioritization of accuracy and efficiency within our system, ensuring adaptability and high performance across varying operational conditions.

Given the context of our object detection system, our primary goal is to maximize the utility function U. This utility function evaluates the performance of each processed image in terms of mean Average Precision (mAP) score c_i and response time r_i, applying penalties p_{ev} and p_{dv} for exceeding predefined thresholds. The total utility U for all k uniquely identified images processed by the system is calculated as per Equation 5.1:

$$U = \sum_{i=1}^{k} U_i \tag{5.1}$$

Where the utility U_i for each image i is defined as per Equation 5.2

$$U_i = w_e E_{\tau_i} + w_d T_{\tau_i} \tag{5.2}$$

In this formula, w_e and w_d represent weights, while E_{τ_i} and T_{τ_i} are piece-wise functions that evaluate the mAP score c_i and response time r_i respectively defined as per Equation **??**

$$E_{\tau_i} = \begin{cases} c_i & \text{if } C_{\min} \leq c_i \leq C_{\max} \\ (c_i - C_{\max}) \cdot p_{ev} & \text{if } c_i > C_{\max} \\ (C_{\min} - c_i) \cdot p_{ev} & \text{if } c_i < C_{\min} \end{cases} \tag{5.3}$$

$$T_{\tau_i} = \begin{cases} r_i & \text{if } R_{\min} \leq r_i \leq R_{\max} \\ (R_{\max} - r_i) \cdot p_{dv} & \text{if } r_i > R_{\max} \\ (r_i - R_{\min}) \cdot p_{dv} & \text{if } r_i < R_{\min} \end{cases} \tag{5.4}$$

To extend our AdaMLS framework's applicability beyond object detection to other ML systems, we introduce Equation 5.5 as a generalized utility function template:

$$U = \sum_{i=1}^{k} \sum_{l=1}^{q} w_l \cdot f_{l_i} \tag{5.5}$$

In this generalized form, f_{l_i} symbolizes the l^{th} key performance metric for the i^{th} request, q is the total number of key performance metrics considered, and w_l represents the weight or importance assigned to each metric within the system's optimization goals. This flexible approach allows AdaMLS to dynamically adjust to the specific requirements and critical performance metrics of various ML applications, ensuring optimal performance across different scenarios.

Additionally, we introduce an evaluation dataset d, comprising x rows, each row corresponding to a unique request or image processed. The count x represents the total number of requests or images evaluated in this dataset, providing a practical basis for assessing the performance and effectiveness of the AdaMLS approach across diverse machine learning scenarios.

5.2.1 Conclusion of the Running Example

Key features of the utility function include:

- The integration of weights and piece-wise evaluations, reflecting the dynamic and varied nature of ML system environments.
- The capability to adapt to different utility formulations, illustrating the system's versatility in representing Quality of Service (QoS) across diverse operational contexts.

AdaMLS is inherently flexible and capable of enhancing alternative utility formulations, which can represent the system's QoS in various operational contexts. This flexibility signifies that AdaMLS is not rigidly utility-driven; rather, it is designed to explore and capitalize on the potential of model switching strategies to enhance system performance under varying conditions, demonstrating the effectiveness of the ML Model Balancer concept, with the current utility function serving as one of many possible representations of system objectives.

5.3 AdaMLS Approach

AdaMLS approach as shown in Figure 5.1 is explained ahead.

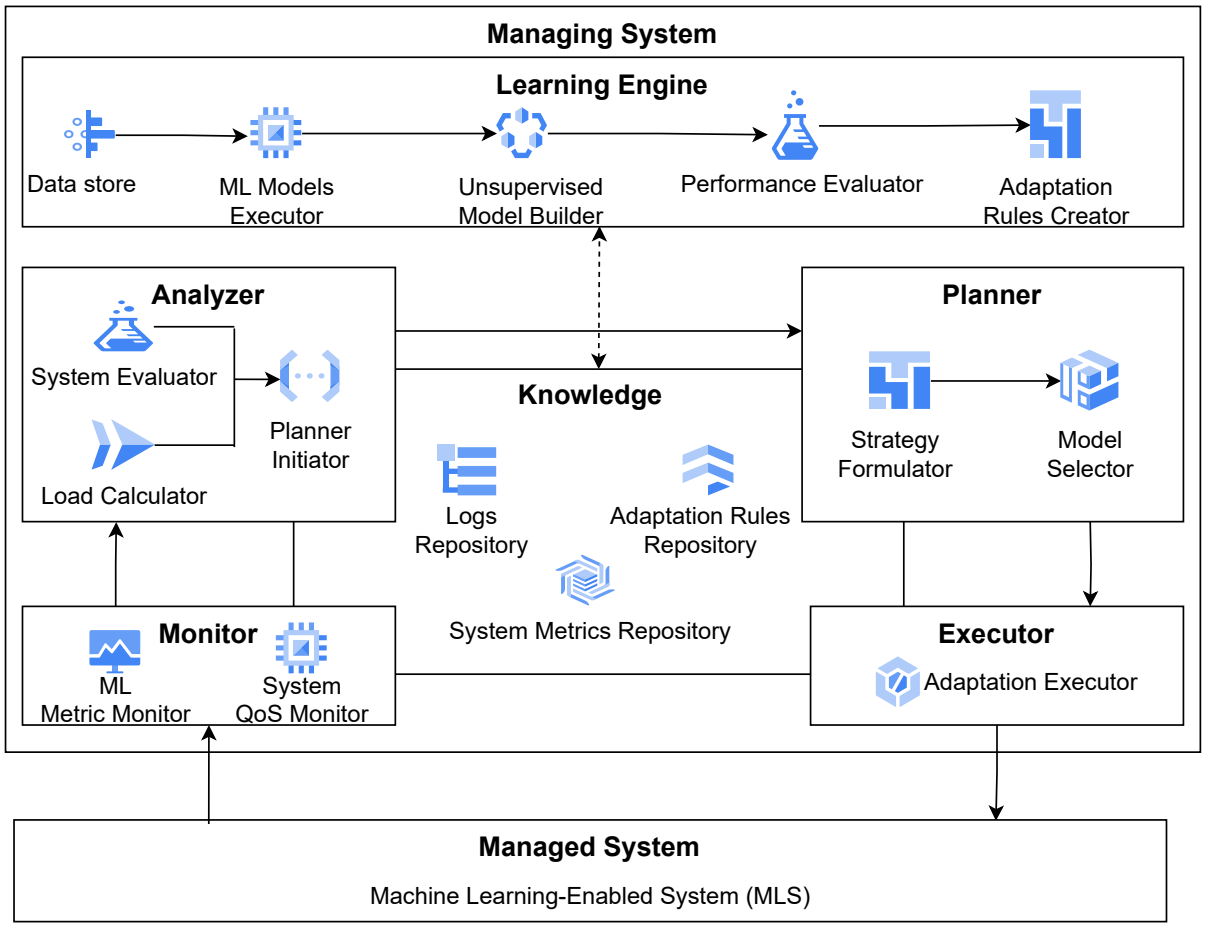

Figure 5.1: AdaMLS Approach Diagram

5.3.1 Learning Engine

In the AdaMLS approach, the Learning Engine (LE) is essential for analyzing and optimizing the performance of machine learning models within the system's operational environment. Its primary goal is to utilize real-world performance data to facilitate dynamic model selection, thereby improving the Quality of Service (QoS).

5.3.1.1 Data Store and ML Model Executor

The operation of the Learning Engine (LE) within the AdaMLS framework is initiated by the *ML Models Executor*. This component is tasked with running models, $m_j \in M$, in a managed system's environment that mirrors the final, production-ready setup, encompassing all necessary software components. This configuration ensures that key performance metrics—such as confidence score (c), processing time (τ), response time (r), and CPU consumption (s)—accurately reflect the model's interaction within a realistic operational context. The *ML Models Executor*'s primary function includes the collection of output result data for evaluation datasets d_j stored from the *Data Store* for each model m_j inside the managed system. This comprehensive data collection is used to the effectiveness of each model in synergy with the system's other software components. Choosing the right evaluation dataset is important to the LE's effectiveness. The evaluation dataset is selected to mirror the real-world challenges and scenarios the system is expected to navigate, making it an indispensable component of the LE's operational framework. A carefully selected evaluation dataset serves as a proxy for the real world, enabling the LE to make well-informed decisions regarding model selection. The congruence between the evaluation dataset and actual real-world conditions plays a crucial role in the LE's ability to accurately predict and enhance real-world system performance.

5.3.1.2 Unsupervised Model Builder and Performance Evaluator

The *Unsupervised Model Builder* is a key component of the AdaMLS approach, leveraging unsupervised learning techniques, notably K-Means clustering, to systematically analyze datasets d_j. This analysis employs methods such as the Elbow method and the Silhouette coefficient to ascertain the optimal clustering of models based on performance metrics, facilitating an impartial evaluation process. Emphasis is placed on τ', the processing time per image by the system, exclusive of network and queuing delays, due to its significance in assessing system efficiency and responsiveness. Following the analysis, the results from the models are organized into g distinct clusters, with each image i in the evaluation dataset d assigned a cluster label l, categorizing it into one of the performance-based groups. This classification aids the *Performance Evaluator* in constructing a performance matrix P_j for each model m_j, crucial for a comprehensive cross-model performance comparison. The matrix P_j, and its extended form P'_j, provide a detailed comparative analysis of model performances:

$$P_j = \begin{bmatrix} f_{1_i} & f_{2_i} & \dots & f_{f_i} & l_i \\ f_{1_{i+1}} & f_{2_{i+1}} & \dots & f_{f_{i+1}} & l_{i+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ f_{1_x} & f_{2_x} & \dots & f_{f_x} & l_x \end{bmatrix}_{x \times (f+1)}$$

In P_j, f_{l_i} denotes the l^{th} key performance metric for the i^{th} request or image, where f_{1_i} could represent the average confidence score and f_{2_i} the response time for image i, with l_i providing the cluster label based on clustering outcomes. The enriched matrix P'_j further incorporates additional metrics from other models for the same requests or images, facilitating a more expansive performance analysis across all models. For illustrative purposes, below is a simplified example in Table 5.2 showcasing aggregated performance metrics from different YOLOv5 model variants, focusing on 'Large' and 'Nano' models for a subset of images and metrics.

Table 5.2: Simplified Example of Aggregated Performance Metrics for 'Large' and 'Nano' Models

Image ID	**'Large' Model**			**'Nano' Model**		
	Conf.	**Time (s)**	**CPU (%)**	**Conf.**	**Time (s)**	**CPU (%)**
000000020806	0.885	0.391	45.2	0.628	0.041	15.3
000000503824	0.933	0.363	50.6	0.825	0.037	24.6

This meticulous way, culminating in P'_j, equips AdaMLS with a robust dataset for evaluating each model's performance, forming the empirical foundation for informed model selection and system optimization. By juxtaposing metrics across models, AdaMLS can dynamically adapt model selection to optimize system performance and efficiency in any operational context.

5.3.1.3 Adaptation Rule Creator

The *Adaptation Rule Creator* in AdaMLS approach by calculates the 90% confidence intervals (CI) for each performance cluster l across every model m_j. These confidence intervals define the statistical boundaries for Key Performance Indicators (KPIs), thereby minimizing uncertainty in the system's adaptive processes. This statistical method enhances the predictability of system adaptations by providing a defined range for expected KPI values, based on empirical data.

For illustration, let's consider the implementation of this approach for a 'nano' model variant represented by m_j within the ensemble of models M. For each performance cluster l pertinent to 'nano', the CIs for performance metrics are computed using data points exclusively from that cluster. In parallel, for a consistent set of images, CIs for performance metrics of all other models ($m_k \neq m_j$) are also determined, offering a holistic view of potential performance variability across different models.

This analytical process results in a CI matrix specific to the 'nano' model, encapsulating the expected performance range when evaluating the collective model ensemble. Replicating this methodology for each model variant in M yields a series of CI matrices, each reflecting the performance variability within the clusters of the respective model m_j. Through periodic analyses conducted by the Learning Engine (LE), AdaMLS accumulates in-depth insights into potential variations in model performance, facilitating statistical projections of their impact on the system's KPIs during model transitions.

Consider the following table as an example of the CI matrices produced through this process:

Table 5.3: Sample Confidence Intervals for Model Performance Metrics

Cluster	**Metric**	**Medium CI Lower**	**Medium CI Upper**	**Example Comparison CI**
1	Avg. Confidence	0.675	0.676	Nano: 0.544-0.544
1	Response Time (s)	0.121	0.122	Nano: 0.032-0.032
1	CPU (%)	43.48	43.49	Nano: 31.49-31.50
0	Avg. Confidence	0.578	0.579	Small: 0.559-0.560
0	Response Time (s)	0.120	0.121	Small: 0.064-0.065
0	CPU (%)	43.35	43.36	Small: 40.03-40.05

This CI matrix exemplifies how the *Adaptation Rule Creator* quantifies model performance variability, allowing AdaMLS to make more informed decisions regarding model adaptations. By establishing clear statistical limits for KPIs, AdaMLS enhances its capacity to optimize the system's performance and Quality of Service (QoS), ensuring efficient adaptation to changing operational conditions.

5.3.2 MAPE-K Loop

5.3.2.1 Knowledge

As per the structure depicted in Figure 5.1, the Knowledge (K) base in our system is primarily a repository divided into three sections: the *Log Repository*, the *Adaptation Rule Repository*, and the *System Metrics Repository*. The *Log Repository* stores system logs, including vital KPIs. For instance, in our running example, this would mean processing time per image by the system τ, the system response time r, model confidence c, and CPU consumption s for all processed requests k, as recorded by the MLS. The *Adaptation Rule Repository* houses the CI matrix generated by the LE, acting as a set of

adaptation rules for the Planning phase. Lastly, the *System Metrics Repository* keeps track of various system metrics such as real-time incoming request rate per second denoted as v and system logs if any.

5.3.2.2 Monitor

The monitoring phase of the AdaMLS approach is divided into two main components: the *ML Metric Monitor* and the *System QoS Monitor*, which together ensure comprehensive tracking of system performance and environmental conditions. The *ML Metric Monitor* is responsible for continuously tracking key performance indicators (KPIs) of the machine learning models and system QoS. In the context of our object detection use case, this includes:
1. The average number of detection boxes per processed image, denoted by b.
2. The processing time per image by the model, excluding network or queuing delays, denoted by τ'.
3. The overall system response time, including network, queuing, and processing delays, denoted by r.
4. The confidence score of the detection, denoted by c.
5. The CPU consumption for the model processing, denoted by s

These metrics are measured over the last i processed requests and are collectively represented as k' to provide a real-time overview of the system's operational performance. Simultaneously, the *System QoS Monitor* focuses on the operational aspects of the system, particularly:
1. The current model in use, represented by m', which is critical for understanding the immediate context of system performance.
2. The version of the model, denoted by v, to track updates or changes in the model repository that might affect performance.

This component also monitors the number of pending requests, i_w, which offers insights into the current workload and potential system stress, indicating when the system may need to adapt to maintain optimal QoS. Both sets of monitored data are forwarded to the Analyze function for assessment and determination of necessary adaptations. This ensures the system's self-awareness and adaptability in response to internal performance metrics and external environmental conditions. All monitored data are also logged in the Knowledge component, maintaining a comprehensive record for future analysis and learning.

5.3.2.3 Analyzer

The Analyzer phase is pivotal in the AdaMLS approach, featuring two key components: the *Planner Initiator* and the *System Evaluator*, which together assess the need for system adaptation based on the monitored data.

The *Planner Initiator* leverages the *System Evaluator* to analyze data collected by the Monitor components. Its primary objective is to ascertain whether the current operational conditions necessitate a system adaptation to maintain or enhance QoS. The *System Evaluator* determines the necessity for adaptation. It achieves this by identifying the closest performance cluster, denoted by l, within the confidence interval (CI) matrix pertinent to the current model in use, represented as m'. This identification

process relies on analyzing the mean values of the most recent k' results (e.g., the last 50 processed requests) for m', with a particular focus on the two key performance indicators (KPIs) demonstrating the highest variance. This methodological approach is instrumental in mitigating uncertainties related to data drift. Upon pinpointing the relevant cluster l, the *System Evaluator* then establishes a feasible request rate range, represented as $[v_{\min}, v_{\max}]$. This range is derived from the CI matrix for m' by utilizing the inverse of the upper and lower confidence interval bounds for τ' for m'. This process facilitates the determination of an adjusted operational threshold that aligns with the current system's performance capabilities and environmental conditions.

Following the identification of the feasible request rate range, the *Load Calculator* computes the adjusted request rate, v_{adj}, by incorporating the count of pending requests, i_w, that exceed $v_{\max}$. This calculation ensures that the system's operational capacity is optimized in real-time, balancing workload demands with the available computational resources and the performance characteristics of the currently deployed ML model (m'). This adjustment in the request rate, v_{adj}, is crucial for maintaining system resilience and QoS, especially under varying workload conditions. By dynamically adapting to the current operational state and the identified performance cluster (l), the AdaMLS approach ensures that the system remains responsive and efficient, effectively addressing the challenges posed by data drift and other forms of runtime uncertainty.

If v_{adj} is not within the range $[v_{\min}, v_{\max}]$, the *Planner Initiator* implements a waiting period, denoted by t_{wait} (e.g., 0.25 seconds), to mitigate the potential for unnecessary system adaptations. This pause allows for a reassessment of the system's state before proceeding. Following this brief interlude, the *Planner Initiator* activates the Planning phase, equipped with the adjusted request rate v_{adj}, the current model in use m', and the identified performance cluster l, to ensure that any system adaptations are both necessary and optimally timed for the current operational conditions.

5.3.2.4 Planner

The Planner phase is a critical component of the AdaMLS approach, with the *Strategy Formulator* and *Model Selector* working in tandem to develop a targeted adaptation strategy based on insights from the Analyzer and the Knowledge base.

The *Strategy Formulator* utilizes the outputs from the Analyzer—specifically the adjusted request rate v_{adj} and the identified performance cluster l—along with the accumulated knowledge base to draft a preliminary adaptation strategy. This strategy involves identifying potential models within the set M that are capable of accommodating v_{adj} while belonging to the specified cluster l. The compatibility of a model for selection is assessed by comparing v_{adj} against the inverse of the lower value of the confidence interval (CI) bound for τ' for each model. For the current model m', this comparison utilizes the most recent n results to ensure relevance and accuracy. Conversely, for other models within M, the CI matrix associated with m' is referenced to maintain consistency in evaluation criteria.

Following the Strategy Formulator's assessment, the *Model Selector* undertakes the task of choosing m_{best}, defined as the model that exhibits the highest lower CI value for c, thereby effectively addressing

potential goal and model drift uncertainties. This decision-making process prioritizes models that not only meet the operational demands but also align with the system's QoS objectives. If the current model m' is identified as m_{best}, indicating that it remains the optimal choice under the present conditions, the Planner abstains from initiating further actions to avoid unnecessary system adjustments. Conversely, should a more suitable model be identified—signaling a divergence from $m' = m_{\text{best}}$—the Planner proceeds to signal a model switch to the Execution phase. This transition is activated only when a clear advantage in performance or efficiency is discerned, ensuring that system adaptations are both purposeful and beneficial. In instances where no model within the current evaluation is deemed sufficiently advantageous to perform a switch, the system continues its operations with m', prioritizing stability and avoiding the potential for counterproductive changes.

5.3.2.5 Executor

The Executor phase implements the adaptation strategy devised during the Planning phase. This process is managed by the *Adaptation Executor*, which actualizes the system's response to the identified need for adaptation. The *Adaptation Executor* is responsible for enacting the planned adaptations to the system. This involves a critical decision point based on the outcome of the Planning phase:
1. If the Planner has identified an optimal model, m_{best}, that differs from the currently deployed model m', indicating that a model switch would enhance the system's performance or efficiency, the system will transition to utilizing m_{best}.
2. Conversely, if the Planning phase concludes that continuing with the current model m' remains the best course of action, no switch is initiated. This decision is based on the assessment that m' is still the most suitable model given the current operational conditions and performance metrics.
In both scenarios, the execution of the adaptation strategy is designed to ensure the system's autonomy and its capacity for self-adaptation in response to changing conditions. This dynamic adaptability is fundamental to maintaining or enhancing the Quality of Service (QoS) while addressing potential challenges such as goal drift or environmental uncertainties. Furthermore, the implementation of any adaptation, whether it involves a model switch or the continuation with the current model, is accompanied by an update to the Knowledge base. This ensures that all adaptations are logged, providing a rich historical dataset that can inform future decision-making processes and enhance the system's learning capabilities. By systematically recording the outcomes and impacts of adaptations, the Knowledge base becomes an invaluable resource for continuous improvement and learning within the AdaMLS approach.

5.4 Uncertainty Analysis in AdaMLS

As seen in the previous section, AdaMLS approach operates within the dynamic and often unpredictable domain of Machine Learning-Enabled Systems (MLS), necessitating a robust approach to handling uncertainties. This section delves into the various sources of uncertainty AdaMLS encoun-

ters and outlines the strategies implemented across its components—Monitor, Analyzer, Planner, and Executor—to mitigate these uncertainties. Self-adaptive systems are distinguished by their capability to continuously deliver service despite changes in the system, environment, or goals. Main aspect of such systems is a self-adaptive layer that executes adaptation actions based on monitored information, adhering to a MAPE-K loop —Monitor, Analyzer, Planner, Executor, with a Knowledge component. AdaMLS manage uncertainties stemming from multiple sources, influencing the satisfaction of system goals and properties. Uncertainty in self-adaptive systems can emerge from various locations, including the adaptation functions, environment, goals, and resources, each contributing to the system's dynamic nature.

5.4.1 Uncertainty Sources and Mitigation Strategies

The following table outlines the specific uncertainties AdaMLS encounters and the corresponding mitigation strategies employed:

Table 5.4: AdaMLS: Mitigating Uncertainty in Self-adaptive Systems

Uncertainty Source	**Uncertainty Explanation**	**Mitigation Strategy**	**Implemented In**
ML Models in MLS	Abstraction of ML model	Evaluate all models on test set; abstract performance with statistics	Learning Engine
Resources	Dynamicity in the system	Preload ML Models; reduce latency and adaptation action costs	MLS
Environment	Incoming requests unpredictability	Monitor request rate and produced results continuously	Monitor
Adaptation Function	Impact of tactic	Utilize confidence intervals for adaptation rules, ensuring assurance	Learning Engine
Goal	Conflicting or dependent goals	Prioritize most accurate model among eligible models for requests	Planner
Data Drift	Untrained or rare data	Map performance for past 'n' requests to nearest cluster for insight	Analyzer
Model Drift	Model performance decay	Rank models per goal; auto-remove degraded models	Planner

AdaMLS's design and operation are informed by a understanding of the sources and nature of uncertainty within self-adaptive systems. By embedding mechanisms for continuous monitoring, analysis, planning, and execution, AdaMLS effectively reduces uncertainty's impact on system performance and goal satisfaction. The approach's reliance on statistical metrics, confidence intervals, and dynamic model evaluation ensures that AdaMLS not only adapts to present conditions but also anticipates and mitigates future uncertainties.

Through its comprehensive approach to uncertainty management, AdaMLS demonstrates a sophisticated application of self-adaptive system principles, ensuring that machine learning-enabled systems can operate reliably and efficiently in the face of inherent and emerging challenges.

5.5 Results

5.5.1 Implementation Setup

We implemented AdaMLS on an object detection system, utilizing YOLOv5 variants in conjunction with FastAPI, as detailed in Section 5.1. For testing, we utilized a situation derived from FIFA98 [72], capable of handling up to 28 parallel requests per second, totaling 25,000 requests. The testing framework employed the COCO 2017 unlabelled dataset [51] for testing purposes, with the COCO 2017 test dataset serving as our evaluation dataset. Our data clustering process, facilitated through Python and PySpark's MLlib[1], identified optimal clusters using the elbow method [78]. Complete specifics of our implementation, parameters, and the associated results are detailed in repository[2].

5.5.2 Results Analysis

In response to RQ1 as explained in chapter 1, which queries how self-adaptive strategies can be developed and applied in MLS to manage runtime uncertainties while maintaining or enhancing QoS, this chapter explains our approach through AdaMLS.

RQ1: In the context of Machine Learning-Enabled Systems (MLS), how can self-adaptive strategies be developed and applied to manage and mitigate runtime uncertainties, thereby maintaining or enhancing their Quality of Service (QoS)?

Building on the foundational concepts introduced in Chapter 4 regarding the Machine Learning Model Balancer, AdaMLS operationalizes dynamic model switching, effectively addressing runtime uncertainties inherent in MLS. By integrating self-adaptation mechanisms within the MAPE-K framework, AdaMLS not only exemplifies a novel application of these principles but also validates the efficacy of dynamic model switching in enhancing system QoS.

[1] `https://spark.apache.org/`
[2] `https://github.com/sa4s-serc/AdaMLS`

In our evaluation and analysis, AdaMLS is assessed against the naive approach and individual YOLOv5 models. The key parameters set for our utility calculations include $p_{ev} = 1$, $p_{dv} = 1$, delineating the penalty multiplication coefficients for exceeding operational thresholds. These coefficients are integral to our utility function, a piecewise linear construct designed to quantify the effectiveness of model switching strategies within various operational contexts. Specifically, the utility function applies a penalty when performance metrics, such as confidence scores or response times, deviate from the desired range set by $C_{\max} = 1$, $C_{\min} = 0.5$, $R_{\max} = 1s$, and $R_{\min} = 0.1s$, representing the max and min thresholds for optimal performance.

The performance comparison, as shown in Table 5.5, highlights AdaMLS's strategic advantage in model switching, executing 308 instances compared to the naive approach's 49 also shown in 5.2.

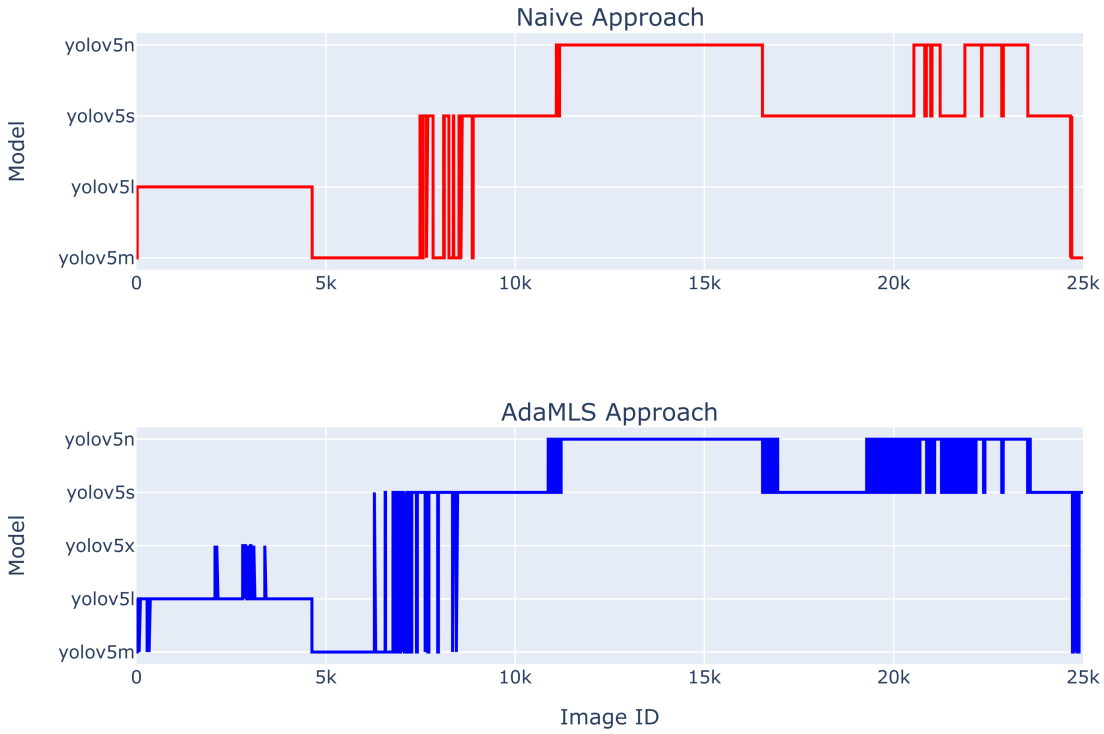

Figure 5.2: AdaMLS: Model Switching: Naive Vs. AdaMLS

This increased frequency in model switching underlines AdaMLS's dynamic adaptability. Despite YOLOv5xl's superior average confidence score (c) of 0.689, its practical applicability is hindered by nearly 25,000 penalty instances for response time (r), underscoring the inherent trade-offs between model complexity and operational efficiency. Conversely, YOLOv5n, with the best average response time of 0.28 seconds, suffers in terms of confidence score and penalty instances for c, showcasing the

challenges in balancing speed with accuracy. AdaMLS, with an average confidence of 0.6, demonstrates a commendable balance, indicating its ability to maintain a high quality of service without leading in every individual metric.

Utility, as detailed in Table 5.6, is a pivotal measure, with AdaMLS demonstrating its efficacy across varying weight scenarios (w_e and w_d), reflecting the system's adaptability to different operational needs. Despite not always achieving the highest scores in individual metrics, AdaMLS excels in overall utility, achieving up to a 39% improvement over YOLOv5n as shown in 5.3. This utility measure is a testament

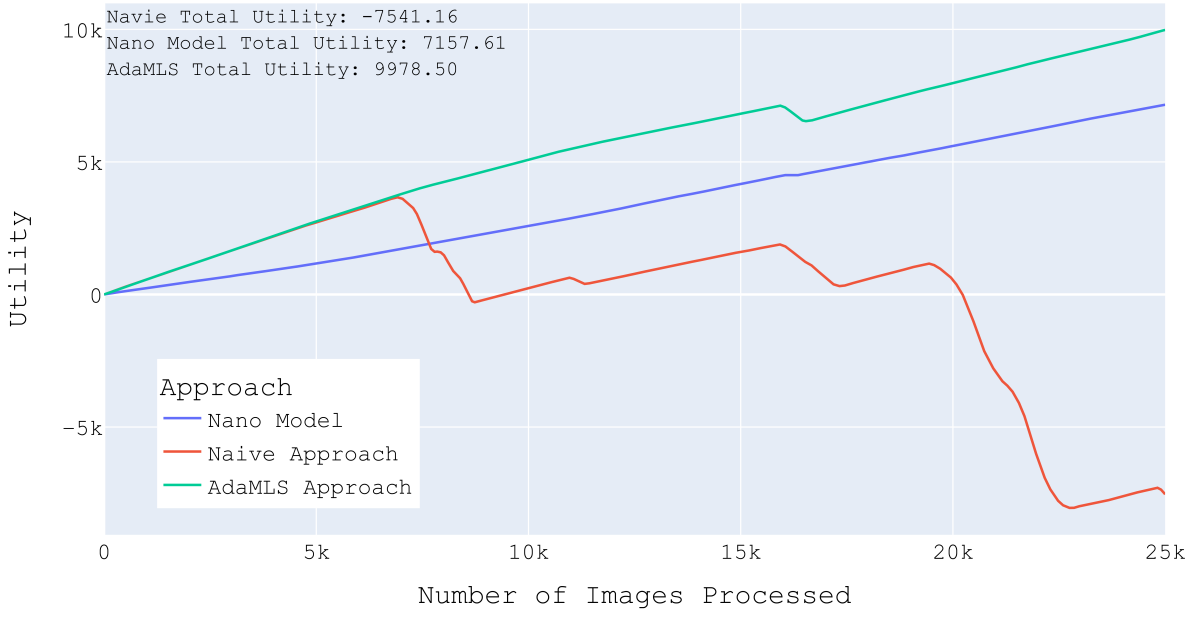

Figure 5.3: AdaMLS: Utility Function Over Requests processed

to AdaMLS's proficiency in intelligently navigating the trade-offs between different performance metrics for optimal outcomes. Moreover, the efficiency of our approach is highlighted by the model transition time, which is maintained below the crucial threshold of 0.01 seconds. This efficiency, coupled with the system's adaptive capabilities, underscores AdaMLS's effectiveness in enhancing the Quality of Service (QoS) for ML-driven systems. In conclusion, AdaMLS exemplifies a model switching strategy that significantly enhances system performance and QoS. Through analysis of performance metrics and utility scores, AdaMLS has proven its capability to dynamically adapt and optimize operational efficiency and accuracy. Its strategic application of model switching, grounded in empirical analysis and utility optimization, sets a new benchmark in the adaptive management of ML models within self adaptation of machine learning-enabled systems.

The AdaMLS approach, detailed in Section 3 and its subsequent empirical validation in Section 4, showcases significant advancements in managing and mitigating the uncertainties associated with MLS. The results shows AdaMLS's capability to dynamically adapt to operational conditions, thereby

Table 5.5: AdaMLS : Performance Comparison

Parameter	Yolov5n	Yolov5s	Yolov5m	Yolov5l	Yolov5xl	Naive	AdaMLS
Model Switch Instances	N/A	N/A	N/A	N/A	N/A	**49**	308
Avg. Confidence Score (c)	0.535	0.61	0.651	0.6756	**0.689**	0.606	0.6
Avg. Response time (r)	**0.28**	32.5	418.29	1589.7	3273.7	1.94	0.47
Avg. CPU Consumption	**42.94**	52.92	70.92	80.65	81.69	66.42	65.27
Penalty Instances for r	**365**	13991	17843	20361	24998	7731	927
Penalty Instances for c	9770	5382	3366	2342	**1941**	5843	6137

Table 5.6: AdaMLS: Utility Comparison

w_e	w_d	Yolov5n	Yolov5s	Yolov5m	Yolov5l	Yolov5x	Naive	AdaMLS
0	**1**	5817	-7.9×10^5	-10×10^6	-39×10^6	-81×10^6	-27313	**9978**
0.25	**0.75**	6487	-5.9×10^5	-7.8×10^6	-29×10^5	-61×10^6	-17427	**8992**
0.5	**0.5**	7157	-3.9×10^5	-5.2×10^6	-19×10^6	-40×10^6	-7541	**9978**
0.75	**0.25**	7827	-1.8×10^5	-2.5×10^6	-9.9×10^6	-20×10^6	2345	**10964**
1	**0**	8498	12566	14612	15270	**16264**	12231	11949

optimizing QoS. Through its implementation using an object detection system and comparative analysis against both naive strategies and individual model performances, AdaMLS has demonstrated a profound ability to balance accuracy with efficiency, resulting in a superior utility score. This balance is pivotal in enhancing the QoS, highlighting AdaMLS as a pivotal step forward in applying self-adaptive strategies within MLS.

5.6 Discussion

In this section, we first provide details on the different lessons learned from the experiments and evaluations. Following this, we list down the possible threats to validity.

5.6.1 AdaMLS: Lessons Learned

Extending Applicability and Encouraging Innovation: AdaMLS, through its implementation and validation in an object detection use case, showcases not just an application but a methodology that invites further exploration and customization. By providing open APIs and a generic design framework, AdaMLS encourages researchers, practitioners, and students to not only use the approach as-is but also to customize and extend it to fit a wide range of MLS applications beyond object detection. This

flexibility paves the way for innovative self-adaptive strategies across various domains of machine learning, including but not limited to vision tasks, natural language processing, and predictive analytics.

Bridging Theory and Practice The AdaMLS approach bridges the theoretical aspects of self-adaptive systems with practical applications in MLS, offering a novel approach that can dynamically respond to changing environmental conditions and operational demands. The strategy of utilizing unsupervised learning for dynamic model switching presents a novel pathway for MLS systems to maintain optimal performance metrics, emphasizing AdaMLS's role as a pioneer in the self-adaptive MLS landscape.

Future Directions Looking forward, the adaptability of AdaMLS invites exploration into a broader spectrum of machine learning tasks and challenges. The approach's inherent flexibility and the success demonstrated in the object detection use case lay a foundation for extending its application to other domains. Future research could focus on refining the AdaMLS approach to enhance its efficiency, exploring additional learning techniques, model-switching strategies, and adapting it to varied operational contexts and machine learning tasks.

5.6.2 Threats to Validity

In acknowledging the contributions of AdaMLS, it's important to consider potential threats to validity that could influence the interpretation and generalization of the results.

External Validity: While the AdaMLS approach has been validated in the context of an object detection system, its effectiveness and generalizability to other machine learning tasks and domains remain to be fully explored. Future research should aim to test AdaMLS across diverse MLS applications to evaluate its adaptability and performance in varied contexts.

Internal Validity: The performance of AdaMLS is contingent upon the specific experimental setup, including the choice of datasets, machine learning models, and operational parameters. Variations in these aspects could impact the results, necessitating a comprehensive examination of AdaMLS's performance under different conditions.

Construct Validity: The evaluation metrics and operational thresholds defined to assess the performance and effectiveness of AdaMLS play a critical role in determining its success. Ensuring that these metrics accurately reflect the Quality of Service and system performance is essential for validating the approach's utility.

Conclusion Validity: The conclusions drawn from the empirical evaluation of AdaMLS are based on specific experimental conditions and performance comparisons. Expanding the scope of empirical evaluations and incorporating a broader range of datasets, models, and operational scenarios will enhance the robustness of the conclusions and support the approach's scalability and generalizability.

Addressing these threats through continued research and development is essential for advancing the AdaMLS approach and confirming its role as a valuable asset in the evolution of self-adaptive Machine Learning-Enabled Systems.

Chapter 6

SWITCH: An Exemplar for Evaluating Self-Adaptive ML-Enabled Systems

In this chapter, we introduce ***SWITCH***, a novel exemplar designed to facilitate the exploration and evaluation of self-adaptive strategies within Machine Learning-Enabled Systems.

6.1 Introduction

Throughout our exploration of self-adaptation of machine learning-enabled systems, especially highlighted in Chapters 2 and 3, we saw the role of self-adaptation in managing the uncertainties inherent to Machine Learning-Enabled Systems (MLS). As seen in chapter 3, our literature review pinpointed a significant gap: the field of self-adaptation within MLS is largely unexplored, with a notable absence of tools or software designed for this purpose. Progressing into Chapter 4, we introduced the Machine Learning Model Balancer as a new concept aimed at enhancing the Quality of Service (QoS) through dynamic run-time ML model switching in response to runtime uncertainties. This led to the development of AdaMLS, an innovative approach that successfully demonstrated the benefits of runtime model switching in MLS, particularly in the context of object detection. AdaMLS not only showed potential in managing uncertainties but also in significantly improving system utility and QoS. However, to advance our research and develop more sophisticated self-adaptive strategies for MLS, we identified the need for a specialized environment or tool. This is where the concept of an exemplar comes into play, serving as a tool for simulating real-world scenarios to allow researchers, students, and practitioners to experiment with and evaluate self-adaptive strategies in MLS. Despite the existence of various exemplars within the SEAMS community[1], our investigation revealed a clear gap: the lack of exemplars specifically designed for MLS.

Addressing this research gap, we present SWITCH in this chapter, an exemplar of real-world ML-enabled systems. SWITCH is an exemplar specifically developed for MLS, showcased through its application in the object detection domain. This choice of application is strategic, aligning with our

[1] `https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/`

AdaMLS case study and underscoring the significance of object detection within MLS research. SWITCH provides a comprehensive platform for runtime ML model switching, designed to cope with varying operational demands, data drifts, and the complex uncertainties typical of MLS. Its key features, including advanced input handling, real-time data processing, and a user-friendly dashboard, make it an effective tool for monitoring and experimenting self-adaptation with MLS in realistic scenarios.

Utilizing SWITCH to validate the AdaMLS approach highlights its potential to enhance QoS in MLS, making it an invaluable resource for the self-adaptation and MLS research community. SWITCH not only facilitates thorough testing, analysis, and refinement of self-adaptive strategies but also ensures their practicality and effectiveness for real-world MLS applications. Accessible through its Git Repository[2] and an official website[3], SWITCH is further complemented by a You-Tube video[4] demonstration, providing a detailed insight into its functionality and application.

The subsequent sections of this chapter are organized as follows: Section II introduces SWITCH, offering an overview of its capabilities. Section III delves into the architecture and design of SWITCH, detailing its operational framework. Section IV discusses system usage and adaptation strategies enabled by SWITCH. Section V presents an empirical validation case study, focusing on the encountered technical challenges and the devised solutions. Section VI reviews related work, comparing SWITCH with existing self-adaptive systems and exemplars. Finally, Section VII outlines future research directions and concludes the chapter, setting the stage for further advancements in self-adaptive MLS.

6.2 Overview

SWITCH is designed as a practical web service for ML, functioning in an online deployment mode. It stands out as an exemplar in the field of MLS, offering a unique simulation platform for dynamic model switching through software architecture-based self-adaptation through MAPE-K framework [23]. This approach effectively addresses the challenge of maintaining Quality of Service (QoS) in the face of operational uncertainties. The system's architecture is tailored for handling complex ML scenarios, particularly demonstrated through object detection use cases. SWITCH integrates input handling via FastAPI[2], facilitating seamless and efficient user interactions.

It employs state-of-the-art YOLOv5u object detection models [41] for real-time data processing, ensuring high accuracy and responsiveness. The system also features systematic logging of observability metrics and system logs in Elasticsearch[2], providing a robust framework for data management & analysis. A standout feature of SWITCH is its interactive, real-time dashboard, implemented using Kibana[2]. This user-friendly interface is designed for effective experiment management and system performance monitoring. It allows researchers to visualize the model switching process in action and evaluate its

[2] https://github.com/sa4s-serc/switch
[3] https://tool-switch.github.io
[4] https://www.youtube.com/@tool-switch

impact on the system's behavior and overall QoS. This dashboard plays a vital role in offering insights into the system's adaptive mechanisms and their outcomes.

6.3 Architecture and Design

SWITCH as shown in Figure 6.1 comprises core components like Managed System, Front-end, Environment Manager, and Managing System, each integral to the system's adaptability and user interaction. Unlike traditional SAS techniques that modify software architecture, SWITCH addresses specific ML challenges like handling data variability and ensuring model accuracy. This approach reflects a shift from architectural adjustments to dynamic ML model management, showcasing a novel aspect of adaptability in real-world MLS systems.

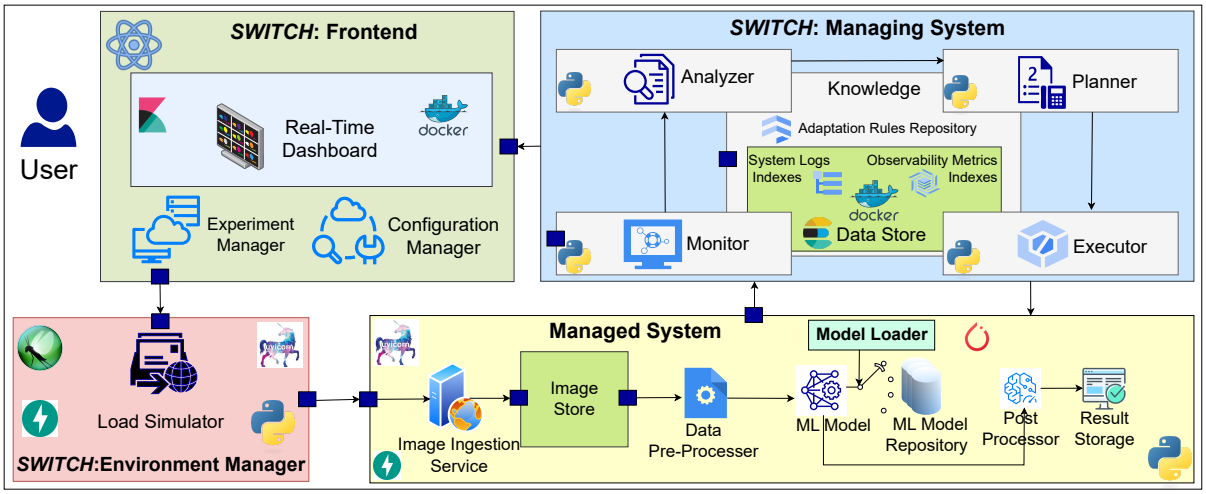

Figure 6.1: SWITCH : Architecture Diagram

6.3.1 Managed System

In real-world scenarios, especially for online-deployed Machine Learning Systems, user requests are processed asynchronously, i.e. continuously accepting them regardless of the processing time by model - a key feature of dynamic and responsive ML services. SWITCH emulates this real-world behavior in its *Image Ingestion Service*. This service, powered by FastAPI, Uvicorn[2] and python, receives image data from users (simulated by the *Load Simulator*) and stores it in the *Image Store* in a Base-64 Encoded format. This service efficiently handles concurrent, real-time data at variable rates, mirroring asynchronous user interaction for seamless subsequent processing. *Image Store* in SWITCH functions as a dynamic queue within the local storage. It stores the incoming image data from the *Image Ingestion Service*. Images are queued here and are picked for processing based on their arrival order and then removed from the queue, implementing a first-in-first-out (FIFO) mechanism. At any given time, the

number of images in the queue reflects pending images to be processed, providing a real-time view of the workload similar to operational queues in MLS deployments.

Data Preprocessor picks the oldest unprocessed image from the *Image Store* for preprocessing. It prepares the image data for object detection. It opens and loads image data from a byte array into memory for model processing, ensuring data readiness for model inference. *Model Loader* in SWITCH is a dynamic component that manages the *ML Model* in use as shown in Figure 6.1. It continuously monitors a specific file ('model.csv' in SWITCH) for indications of which model to load and process. This approach allows for real-time model switching based on external inputs, reflecting a key aspect of adaptability in real-world MLS systems. When SWITCH starts, it preloads all the models and stores them in the Model Repository as a dictionary, ready for switching. *Model Loader* ensures that the system is always ready to respond with the appropriate model as required. *Model Repository* of SWITCH houses YOLOv5nu, YOLOv5su, YOLOv5mu, YOLOv5lu & YOLOv5xu - all preloaded models of the YOLOv5u algorithm provided by Ultralytics [41], a state-of-the-art object detection system renowned for its accuracy and efficiency. YOLOv5u models, developed using the PyTorch framework and trained on the COCO dataset [51], are ready for deployment in the repository. The concept of 'preloading' models here means that each model is initialized and kept ready for immediate use, ensuring the system's adaptability and responsiveness to different object detection requirements. In addition to this, SWITCH's Model Repository allows users to integrate different types of object detection models.

In SWITCH, the *'ML Model'* refers to the currently active YOLOv5 model processing the image data. This model, selected by the *Model Loader*, is the primary driver of the object detection task within the system. It receives preprocessed images, applies the detection algorithms, and generates results, embodying the core functionality of an ML system in operation. *Post Processor* refines detection outcomes with a confidence score threshold (e.g., 0.35), focusing on desired classes (e.g., Humans, Cars). It computes total detections and average confidence. System metrics include the processing timestamp, count of processed requests (e.g., Request No. 370), current model name, model processing time, total time from image receipt to output (total_time), duration since project start (absolute time), and utility based on response time and confidence. System logs in JSON format are also generated for detailed performance insights. *Result Storage* is a temporary storage which manages processed data flow into the Elasticsearch-based *Data Store*, inside *Knowledge* in the managing system. REST API ensures seamless data transfer from the *Result Storage* to *Data Store*, critical for real-time performance understanding and adaptive decision-making. This integration enables SWITCH to maintain efficient storage, runtime observability and adaptability.

6.3.2 Switch: Front-end

SWITCH's front-end, comprising the Experiment Manager, Configuration Manager, and Real-Time Dashboard, is the hub for user interaction, offering an intuitive and user-friendly experience. Built with React, these components form the primary interface for user interaction. *Experiment Manager* facilitates the uploading of image data and interarrival rate files. Users can either upload .zip or can directly give

the path of the image data folder stored locally. *Configuration Manager* is where users select or upload MAPE-K files for the managing system, determining the adaptation strategy. This includes choosing from predefined strategies like AdaMLs [47] or Naive [47], or uploading custom MAPE-K files. The backend, developed using FastAPI and hosted with Uvicorn as an ASGI server, manages interaction with the frontend UI through a series of API endpoints defined in FastAPI as shown in table 6.1. It enables operations such as starting and stopping processes, uploading data, and retrieving metrics, ensuring a seamless interaction between the user and the system.

Real-Time Dashboard, developed using Kibana, is a key component of SWITCH's user interface. Integrated as an iframe within the React application and runs in a docker container, it provides real-time interactive visualizations of the latest system performance metrics, sourced directly from the Elasticsearch-based *Data Store* through REST API endpoints. Kibana's data visualization capabilities allow users to explore a wide array of metrics interactively. The *Real-Time Dashboard* enables users to analyze and learn about self-adaptation strategies in MLS through runtime model switching, thereby playing a crucial role in the adaptive process of the MLS. In conclusion, the front-end components of SWITCH —from the interactive React-based UI to the insightful Kibana *Real-Time Dashboard*—significantly enhance user engagement, providing critical insights into the system's performance and adaptive strategies in run-time.

6.3.3 SWITCH: Environment Manager

SWITCH's Environment Manager features a *Load Simulator*, essential for replicating real-world API traffic showing load uncertainty in the environment and testing the system's adaptability. It receives image data and interarrival rate files from the *Experiment Manager* of the SWITCH frontend through FastAPI endpoints. This component uses Locust, an open-source load testing tool, to emulate user behavior and manage incoming image data distribution. Locust's Python script simulates user interactions, which is crucial for evaluating SWITCH's performance under various operational loads.

The simulator leverages interarrival rate data from user inputs, like the FIFA98 World Cup logs [72], to create realistic traffic patterns. These patterns, characterized by time gaps between image uploads, test the system's responsiveness to fluctuating and peak load conditions. These scripts direct traffic to *Image Ingestion Service* through FastAPI within the Managed System, ensuring realistic and varied testing scenarios. By mirroring real-world user behaviors and traffic scenarios, this setup is essential for practitioners to analyze and enhance MLS systems' adaptive capabilities by assessing the impact of their adaptation strategies in practical environments.

6.3.4 SWITCH: Managing System

Knowledge component within Switch's Managing System plays a central role in adaptive decision-making of model switching. It contains *Adaptation Rules Repository*, storing adaptation rules by user for MAPE loop. Its Elasticsearch-based *Data Store*, which is adept at handling large-scale data processing and analytics, runs in a docker container. It continually receives data from the Managed System via REST

Table 6.1: SWITCH: API endpoints and their descriptions.

API Endpoint	Description
`/api/stopProcess`	Stops the current process
`/api/downloadData`	Downloads logs and metrics
`/api/latest_metrics_data`	Retrieves the latest metrics
`/api/latest_logs`	Retrieves the latest system logs
`/api/changeKnowledge`	Changes adaptation knowledge
`/api/upload`	Uploads input to the server

API and stores in two indexes: - *new_logs* for troubleshooting and performance analysis in *System Logs Indexes* as JSON documents. - *final_metrics* for monitoring system performance and guiding adaptations in - *Observability Metrics Indexes* as JSON documents. Elasticsearch's JSON-based REST API facilitates CRUD operations and data searches, enhancing the system's adaptive capabilities. *Knowledge* provides these metrics to *Real-Time Dashboard* for visualization and to *Monitor* for monitoring the system's realtime performance through REST API.

6.3.4.1 Self-Adaptation Through MAPE-K Framework

Switch's self-adaptive capabilities are primarily demonstrated using the MAPE-K framework, which is a standard approach for implementing self-adaptive systems.

i) ***Monitor*** retrieves metrics from ***Knowledge*** using API calls for real-time system monitoring; ii) ***Analyzer*** analyzes the monitored data to assess whether adaptation is necessary; iii) ***Planner*** develops strategies based on the analysis for potential adaptations; iv) ***Executor*** executes the adaptation strategies, influencing the managed system as needed. SWITCH offers flexibility by allowing users to use, customize, or create adaptation strategies within the MAPE-K framework. This versatility makes it an ideal tool for exploring various model-switching approaches in self-adaptation for MLS. SWITCH's API, facilitates interaction with the system for custom strategy implementation. Table 6.1 lists the key API endpoints that users can leverage to programmatically interact with SWITCH. These endpoints allow real-time adaptation and monitoring.

6.4 System Usage & Adaptation

As discussed in chapter 1, RQ2 is :

RQ2: What tools or frameworks can be devised to facilitate the implementation and exploration of dynamic model switching strategies within Machine Learning-Enabled Systems for researchers, students, and practitioners?

In response to RQ2, SWITCH exemplifies a pioneering tool designed to bridge the gap in self-adaptation within MLS, notably with the action of dynamic model switching. As a comprehensive exemplar, it incorporates the crucial elements of self-adaptation through the MAPE-K framework, facilitating the implementation, testing, and refinement of various model switching strategies. Its development was motivated by the identified need for a specialized environment that allows for hands-on exploration of self-adaptive strategies, making it an invaluable asset for the research community and educational sectors alike. In the following sections, SWITCH not only answers the call for such a tool but also extends its utility by enabling detailed experimentation and analysis, fostering a deeper understanding of self-adaptation's potential benefits and challenges in MLS.

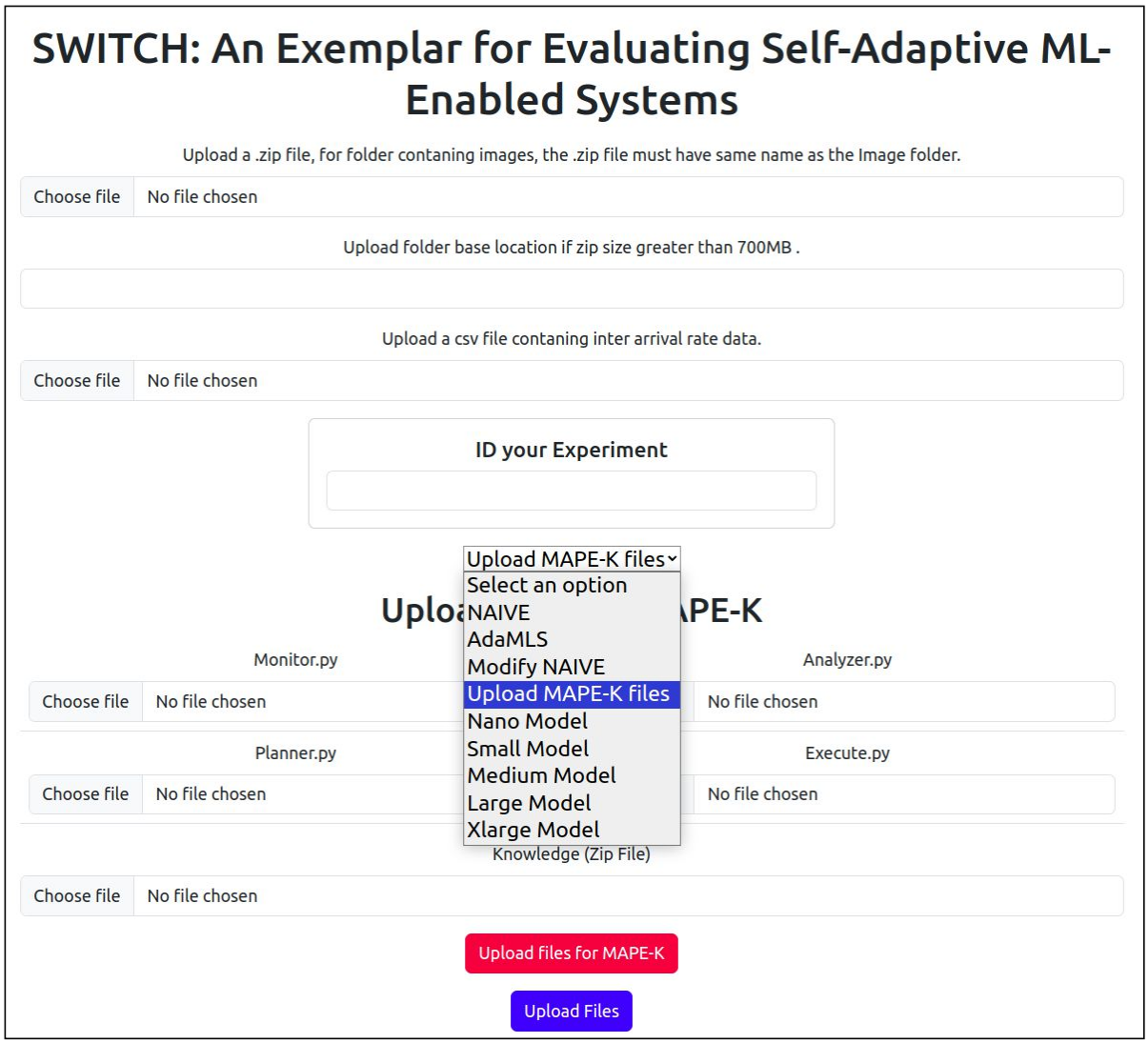

Figure 6.2: SWITCH User Interface: Home Page

6.4.1 System Usage:

SWITCH, an exemplar for Machine Learning-Enabled Systems (MLS), offers an intuitive and straightforward user experience. It reflects real-world scenarios of MLS in its design and operation. Upon initiating SWITCH, the user's actions set in motion a series of automated processes. Docker Compose uses containers to launch Elasticsearch and Kibana services to establish the backend for data visualization and storage. Concurrently, the backend services of SWITCH are brought online through the execution of the Node.py script, creating essential links between the system's front and back ends. The React application, comprising the Experiment Manager and Configuration Manager, becomes operational and presents the user with SWITCH's home page, an interactive web interface as shown in figure 6.2 and preloads all ML models in the repository through *Model Loader*. Once experiments commence, the integrated Kibana dashboard within the React application becomes accessible, offering real-time insights into system performance.

In SWITCH, users engage in the following activities on the home page to start the experiment: **i) Upload Image Data:** Users can upload images either as a .zip file or directly from a local folder. This versatility enables SWITCH to handle data directly from the user's environment; **ii) Inter-arrival Rate File:** Users upload a .csv file for inter-arrival rates, allowing them to test the system's performance under varied real-world conditions; **iii) Experiment ID:** Users assign an ID to their experiment, under which all related logs and metrics are organized and stored; **iv) Select Self-Adaptation Strategy:** Through a drop-down menu, users can choose from a range of self-adaptation strategies detailed in the subsequent subsection, tailoring the system's adaptation approach to their needs.

6.4.2 Adaptation Strategies

The switch incorporates a range of self-adaptation strategies, notably the NAIVE and AdaMLS approaches, each uniquely enhancing the system's adaptability as explained in [47].

Single Model Strategies: These strategies (no switching) involve running a single YOLOv5u model variant throughout the experiment. Users can choose from five YOLOv5u models, each catering to specific performance requirements.

NAIVE and Modified NAIVE: The NAIVE approach, based on the incoming rate of images, switches between models to balance speed and accuracy. For instance, it uses YOLOv5nu (nano) for higher rates (15-30 images/sec) and YOLOv5xu for lower rates (below 2 images/sec). The Modified NAIVE approach allows users to customize these threshold values and adapt the strategy to their specific needs.

AdaMLS: This novel approach, based on unsupervised learning, assesses the capabilities of different models in real-time and selects the one offering the highest confidence score while meeting the target response time. AdaMLS's detailed methodology and its impact on enhancing Quality of Service are discussed in [47]. SWITCH provides AdaMLS implementation, enabling users to experiment with this advanced adaptation strategy.

Custom MAPE-K Strategies: SWITCH is designed to be flexible, allowing users to develop and deploy their own MAPE-K strategies with ease. The system's user-friendly interface and accessible APIs make monitoring and executing custom strategies straightforward. Below is an example showing how users can set up both monitoring and model-switching functionalities.

Listing 6.1: SWITCH: Custom MAPE-K Strategy Implementation

```
# Monitoring Metrics from Elasticsearch
def fetch_metrics(index_name, fields, num_docs_to_fetch):
    query = {"size": num_docs_to_fetch, "sort": [{"log_id": {"
        order": "desc"}}]}
    response = es.search(index=index_name, body=query)
    # Example field processing for 'model_processing_time'
    processed_metrics = process_response(response, fields)
    return processed_metrics

def process_response(response, fields):
    # Logic to process fields like 'model_processing_time' from
        response
    return averaged_metrics

# Switching Model based on a condition
def switch_model(model_name):
    with open("model.csv", "w") as file:
        file.write(model_name)
    # SWITCH checks 'model.csv' and updates the model accordingly
```

In this implementation, the 'fetch_metrics' function retrieves desired metrics from Elasticsearch, and the 'switch_model' function updates the 'model.csv' file to switch models. SWITCH continuously monitors this file and adapt the active model as specified, demonstrating its capability for real-time, dynamic self-adaptation.

6.5 Empirical Evaluation

Switch employs YOLOv5u models [41] for a wide range of object detection scenarios. These models are renowned for their accuracy and efficiency and are trained on the COCO dataset [51], which includes 80 object categories.

Case Study: *1. General Detection:* Utilizing 10,000 images from the COCO 2017 Unlabelled dataset (1.6 GB) [51], Switch handles a wide variety of 80 categories, showcasing its capacity to deal with diverse general object detection tasks. *Note:* Evaluations for *2. Crowd Detection* and *3. Traffic Detection*

was also conducted, focusing on different scenarios and datasets. Results and analyses for these cases are available on our GitHub repository[5].

Customizing Object Detection: SWITCH enables easy customization for detection tasks. Users can modify the detection process by altering the process.py file. For instance, filtering results based on a confidence threshold (e.g., 0.35) and desired class IDs allows targeted detection for classes like 'crowd' or 'vehicle'.*Example: For 'crowd' class: if confidences[i] ¿= 0.35 && class_list[i] == 0* {...}

System Requirements: SWITCH can be deployed on any laptop or PC capable of running Docker and supporting Linux. For detailed technical requirements and setup instructions, please refer to our GitHub repository.

6.5.1 Evaluation using AdaMLS Approach

The AdaMLS approach from [47] was directly applied in SWITCH, with tests on a 12th Gen Intel(R) Core(TM) i5-12500H -12 CORE system. We primarily focused on General Object Detection, using load conditions simulated with scaled FIFA98 logs [72]. AdaMLS's rules, which can be recalibrated based on different setups, were applied in SWITCH. For a comprehensive interpretation of the AdaMLS results, refer to the AdaMLS paper [47].

Comparison of Model Switching Approaches: To illustrate the effectiveness of model switching strategies in SWITCH, we conducted a comparison between the AdaMLS and a baseline 'Nano Model' approach. The latter represents a non-switching scenario for contrast. The following table 6.2 presents the comparison in terms of various performance metrics for General Object Detection:

Table 6.2: SWITCH: Comparison of General Object Detection using AdaMLS Approach and Nano Model-(No Switching)

Metric	AdaMLS	Nano Model
Total Images Processed	10000	10000
Average Confidence Score	0.7	0.65
Average CPU Consumption	20	20.14
Total Objects Detected	47026	37829
Average Model Processing Time (s)	0.033	0.015
Average Image Processing Time (s)	0.25	0.35

The results from the AdaMLS application, as shown in the Table 6.2 and Figure 6.3, illustrate the dynamic model switching capabilities of SWITCH. The table indicates that in 10,000 processed images, each containing multiple detectable objects, a total of 47,026 objects were detected. 'Average Model Processing Time' refers to the duration for which an image is processed within the model itself, while

[5] https://github.com/sa4s-serc/switch

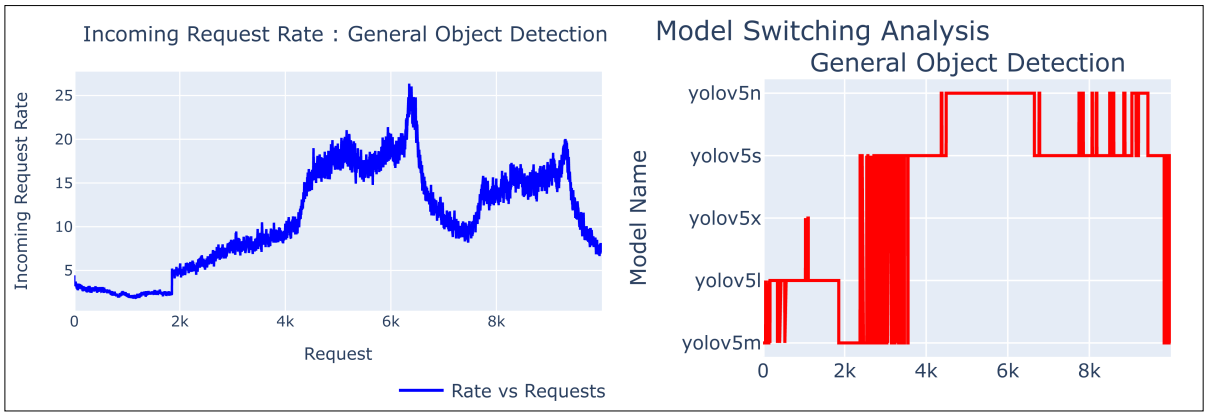

Figure 6.3: SWITCH : Request Rate and Model Switching

'Average Image Processing Time' represents the end-to-end life-cycle of an image — from queueing to processing completion.

Figure 6.3 provides insights into the varying request rates over time and how AdaMLS smartly switches between models in response. When the load is high, SWITCH quickly transitions to the Nano Model for faster processing. Conversely, during periods of lower request rates, it opts for models with higher accuracy, achieving better confidence scores. In this way, SWITCH is used to test approaches. This adaptability demonstrates the critical need for smart model switching strategies in MLS, as it allows for a balance between speed and accuracy based on real-time demands.

Effectiveness of SWITCH: SWITCH preloades 5 YOLOv5u models, consuming 21.3 to 34.8 CPU, with load time of around 0.25 seconds and energy usage of about 10.56 joules. Model switching is executed in approximately 100 microseconds on average.

Real-time Dashboard: The SWITCH dashboard, illustrated through Figures 6.4 to 6.7, offers an in-depth view of the system's real-time metrics, including processing times versus the number of requests processed. It features a rich array of key performance indicators and dynamic visual elements such as pie charts, bar plots, filters, and histograms for a comprehensive, real-time overview.

Figure 6.4 shows a glimpse of the dashboard's capability to present runtime metrics, providing users with critical data on system performance at a glance. This feature is key for making informed decisions on-the-fly. Figure 6.5 reveals how SWITCH users can apply runtime filters to the dashboard. This functionality allows for the segregation and deeper analysis of data based on specified criteria, enhancing the user's ability to monitor and adjust strategies in real-time.

In Figure 6.6, the dashboard displays histograms that offer insights into the distribution of system performance metrics over time. Such visualizations are crucial for identifying trends, spikes, or anomalies in system behavior, facilitating a understanding of system dynamics.

Lastly, Figure 6.7 encapsulates the dashboard's analytics capabilities, presenting processed data in a way that highlights operational insights and performance benchmarks. This aspect of the dashboard

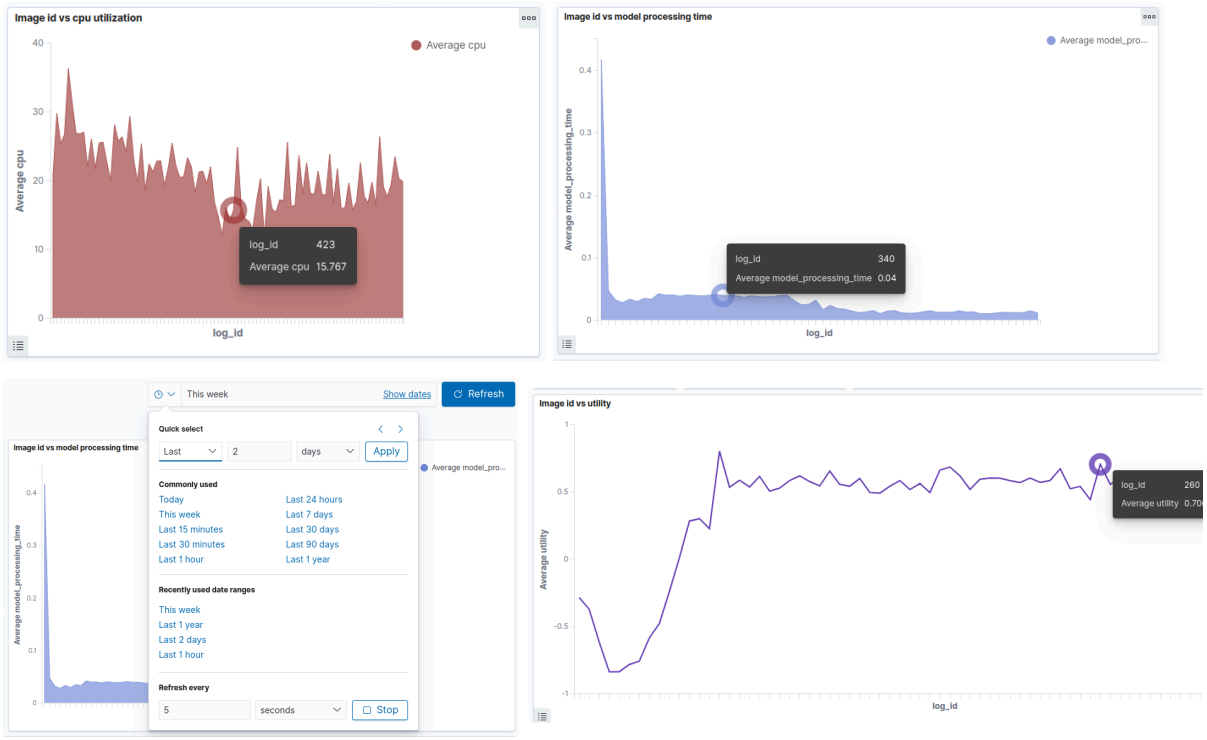

Figure 6.4: SWITCH Dashboard: Runtime Metrics Excerpts

supports the continuous refinement of machine learning systems, encouraging the development of new adaptive strategies tailored to the evolving landscape of MLS.

This user-friendly dashboard empowers users to effectively analyze and adapt their custom strategies for machine learning systems, bolstering the development and refinement of innovative approaches to MLS. Through its comprehensive visualization and analysis tools, SWITCH stands as a pivotal platform for exploring the potential of self-adaptation in the realm of machine learning-enabled systems.

6.5.2 Technical Challenges & Solutions

In SWITCH, we've preloaded various YOLOv5u models to enable quick switching based on real-time performance, addressing the challenge of working with different ML models. For efficient handling of large volumes of image data, images are stored locally for swift processing, and Elasticsearch is used for dashboard updates, ensuring smooth data management. The dashboard, developed with Kibana and Elasticsearch, offers an intuitive and interactive experience, making real-time data understandable for users. Although SWITCH is designed for Linux environments, it can also be used on non-Linux systems through a Linux virtual environment, enhancing its compatibility across different computer types.

By tackling these challenges, we've made SWITCH a more flexible tool. It's now better suited for research and practical applications in the field of MLS that adapt themselves based on changing conditions.

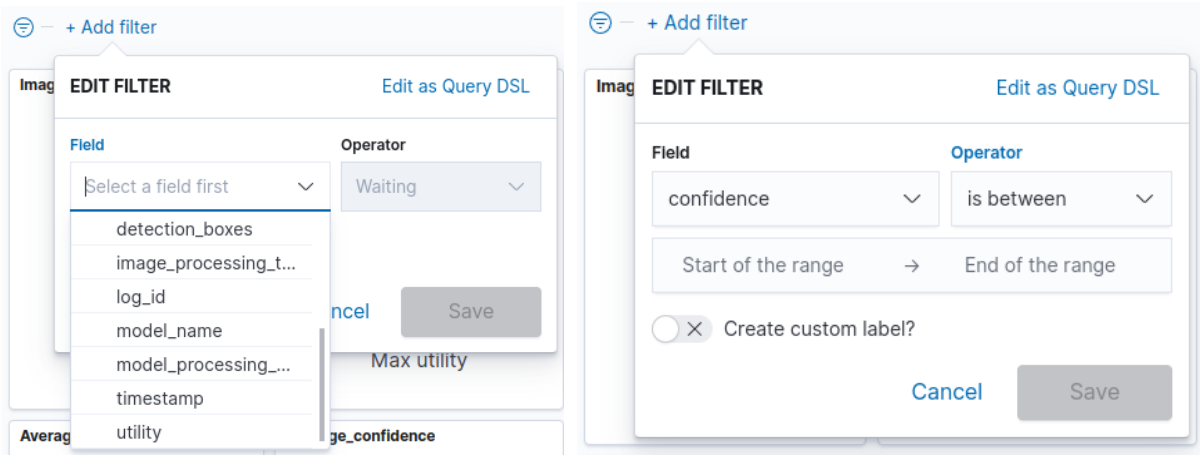

Figure 6.5: SWITCH Dashboard: Runtime Filters Excerpts

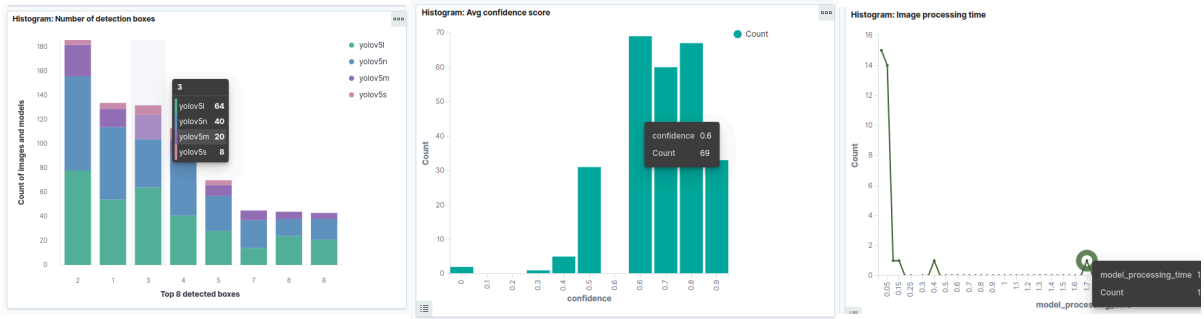

Figure 6.6: SWITCH Dashboard: Runtime Histograms Excerpts

6.6 Discussion

This chapter has highlighted SWITCH's response to RQ2, offering a comprehensive tool for exploring dynamic model switching strategies within Machine Learning-Enabled Systems (MLS). The development of SWITCH is a significant stride towards filling the existing gap in self-adaptive MLS tools, providing a robust platform for both the research community and practitioners to engage deeply with self-adaptive mechanisms in MLS.

6.6.1 Lessons Learned from SWITCH Deployment

Enhanced User Experience and Accessibility: SWITCH's architecture is meticulously designed to be user-friendly and accessible, allowing users to not just use the platform but also customize it according to their specific requirements. The provision of detailed API endpoints enables users to create their own model-switching strategies, facilitating a broader exploration of self-adaptation scenarios. This

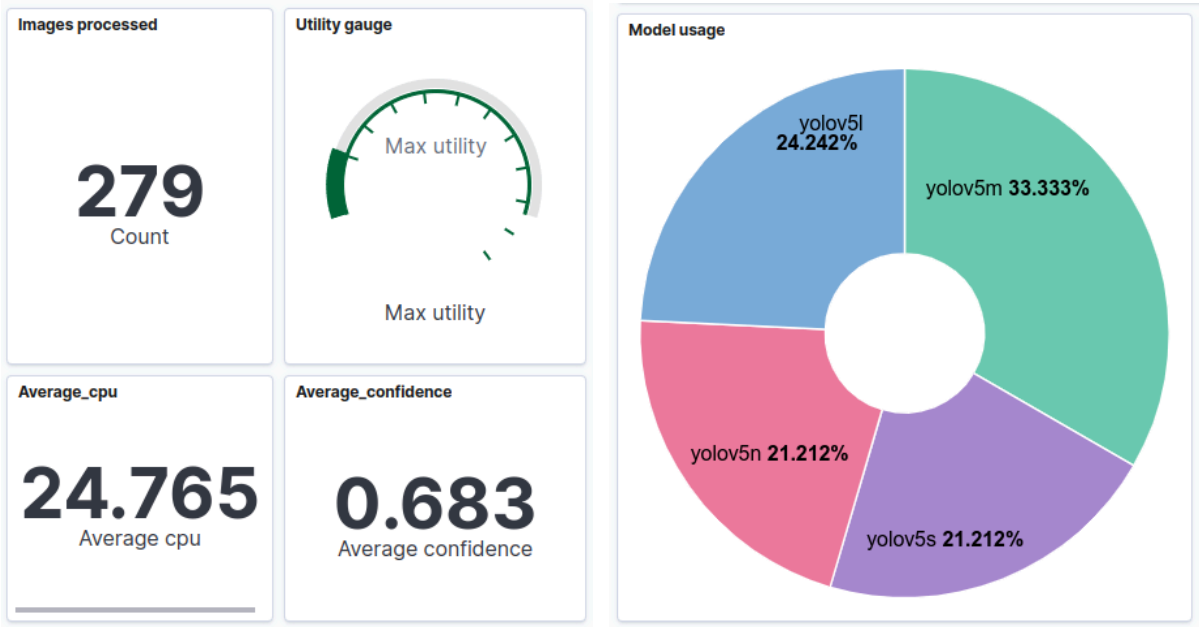

Figure 6.7: SWITCH Dashboard: Runtime Analytics Excerpts

level of customization underscores SWITCH's versatility and adaptability, making it a powerful tool for educational and research purposes.

Validation and Empirical Support: The insights gained from the application of SWITCH, especially in validating the machine learning model balancer concept introduced in Chapter 4 and the AdaMLS approach detailed in Chapter 5, provide empirical support for the effectiveness of dynamic model switching. SWITCH's ability to adaptively manage runtime uncertainties, thereby enhancing the quality of service (QoS), serves as a practical demonstration of these concepts. These validations not only prove the utility of SWITCH but also contribute to the broader understanding of self-adaptive systems within the machine learning domain.

Design Versatility and Applicability: The generic design of SWITCH not only serves the object detection domain but also sets a precedent for its applicability across various MLS domains. This versatility opens avenues for its application in other vision tasks and beyond, highlighting the adaptability of the SWITCH framework to accommodate diverse MLS use cases. Such design flexibility ensures that SWITCH remains a relevant and invaluable tool as MLS continues to evolve and expand into new domains.

Flexibility and Adaptability: SWITCH's architecture and design principles underscore the critical importance of flexibility in MLS. By enabling dynamic model switching based on real-time data, SWITCH addresses one of the key challenges in MLS—maintaining high performance in the face of fluctuating operational conditions.

Educational Value: As a research and educational tool, SWITCH provides a unique platform for students, researchers, and practitioners to gain hands-on experience with self-adaptive systems. It

demystifies the complexities of implementing dynamic model switching strategies, giving a deeper understanding of their benefits and challenges.

Moving Forward: Future versions of SWITCH will aim to extend its functionality and application scope. This includes integrating mechanisms for model retraining, exploring transfer learning opportunities, and facilitating user-driven customization. By pushing the boundaries of what is currently achievable with SWITCH, we anticipate uncovering new insights and strategies that will further advance the field of self-adaptive MLS.

6.6.2 Threats to Validity

In presenting SWITCH and its applications, we acknowledge several threats to validity that could influence the interpretation and generalizability of our findings:

External Validity: While SWITCH demonstrates the practical application of dynamic model switching strategies in object detection, its applicability to other MLS domains remains to be explored. Future research should aim to test SWITCH across various MLS tasks to validate its effectiveness more broadly.

Internal Validity: The performance metrics and results obtained from SWITCH depend significantly on the specific configurations, models, and datasets used in our experiments. Variations in these parameters could lead to different outcomes, suggesting the need for extensive testing under diverse conditions to ensure the robustness of our findings.

Construct Validity: Our choice of metrics and evaluation criteria in assessing the performance of SWITCH could affect our conclusions. Ensuring that these metrics accurately reflect the objectives of self-adaptive MLS is crucial for the validity of our results.

Conclusion Validity: The statistical significance of the improvements observed with SWITCH could be impacted by the scale and design of our experiments. Future studies should consider larger datasets and more complex scenarios to strengthen the conclusion validity of the research.

Addressing these threats to validity will be critical in future work as we continue to refine SWITCH and expand its application domain. By doing so, we aim to solidify its role as a key tool in the development and exploration of self-adaptive MLS strategies.

Chapter 7

Applications of ML Model Balancer

In this chapter, we expand the application of the Machine Learning Model Balancer concept into two areas: ***sustainability and streaming mode operations in MLS, exploring novel strategies like EcoMLS and RelMLS for enhancing system efficiency and reliability.***

7.1 Introduction

In this chapter, we build upon the foundation laid out in the previous chapters, particularly Chapter 4, which introduced the Machine Learning Model Balancer concept, and Chapter 5, where we discussed the AdaMLS approach. AdaMLS exemplified the application of the model balancer concept, specifically targeting the speed-accuracy trade-off prevalent in machine learning-enabled systems (MLS). Further, Chapter 6 introduced the SWITCH exemplar, enabling users with a platform to perform self-adaptation within MLS, providing a hands-on platform for experimentation and validation of self-adaptive strategies. With the utility and versatility of the Machine Learning Model Balancer concept established, this chapter aims to explore its application across two areas: sustainability aspect and streaming mode in MLS.

The imperative for sustainability in MLS is highlighted by the environmental footprint of AI technologies, which, while driving advancements in fields such as autonomous vehicles and smart cities, also contribute significantly to energy consumption and carbon emissions. As discussed in literature review in chapter 3, research, including works by Emma Strubell et al [75] and Lacoste Alexandre et al [48], has illuminated the substantial energy requirements and resultant carbon footprint associated with training and deploying ML models, likening the energy consumption of training a single AI model to that of five cars over their lifetimes. This comparison highlights the necessity of integrating sustainable AI practices, aiming to balance technological progression with environmental conservation. Efforts have largely focused on optimizing the training phase for energy efficiency; however, the inference phase remains relatively unexplored, characterized by significant energy utilization yet essential for maintaining system responsiveness and accuracy. In response to this need, our research introduces EcoMLS, an approach leveraging the machine learning model balancer concept, devised to dynamically modulate energy consumption alongside model confidence in response to the operational conditions and request

variability. EcoMLS uses novel self-adaptive model switching strategy that not only aims to optimize energy usage across MLS applications but also seeks to uphold, if not elevate, the systems' performance metrics. Through object detection use case, we validate EcoMLS, showcasing its potential to minimize energy consumption while maintaining high levels of model accuracy, thus contributing a self-adaptive strategy towards the sustainability of MLS. Parallelly, the significance of deploying MLS in streaming

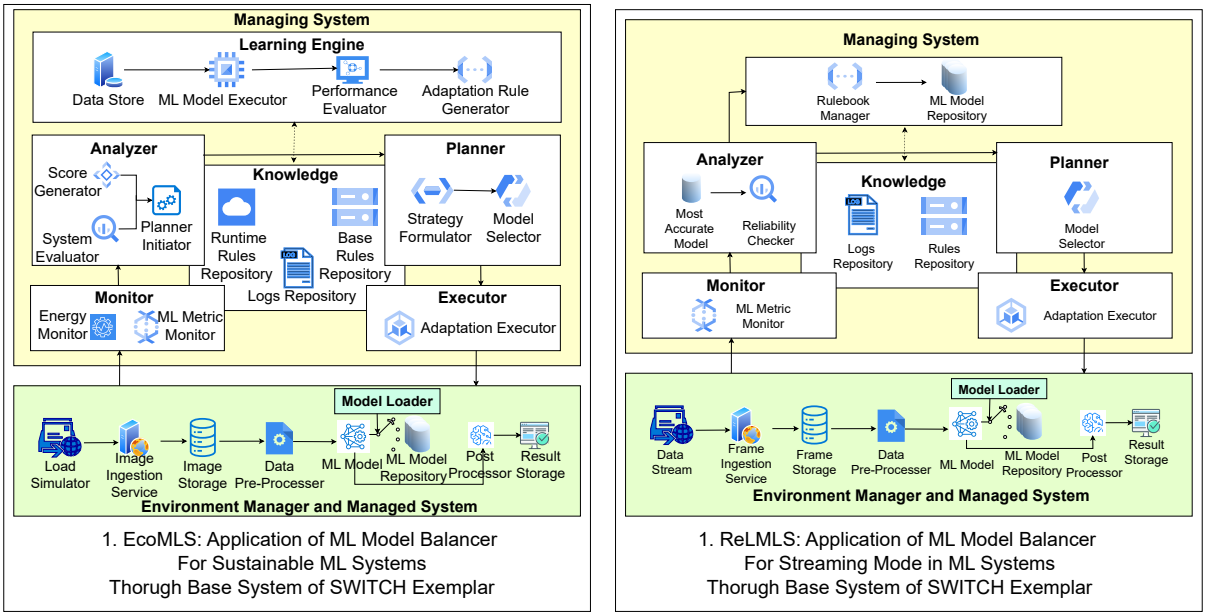

Figure 7.1: Architecture of the EcoMLS and RelMLS Approach

modes, especially for tasks such as object detection in video streams, cannot be overstated. Streaming mode poses unique challenges, including the imperative of maintaining contextual reliability—ensuring the accuracy and relevance of detected objects over continuous video feeds [34]. This mode extends the concept of online deployments, where instead of singular request-response transactions, there is a continuous influx of data requiring real-time processing. In the object detection use case, this translates to video object detection, where not just speed and accuracy, but the reliability of contextual information—such as the number and accuracy of detected objects—becomes important. To navigate these challenges, we introduce RelMLS, another novel application of the Machine Learning Model Balancer concept, tailored to enhance MLS reliability in streaming contexts in terms of reliability detection outcomes. RelMLS is designed to evaluate the contextual reliability of current models against predetermined benchmarks, initiating model switches, aiming to maintain detection results within an acceptable range, thereby ensuring decision-making processes are informed by accurate and relevant data.

Both EcoMLS and RelMLS approached are depicted in Figure 7.1 represent our efforts toward advancing the self-adaptability and efficiency of MLS, each targeting distinct yet equally important aspects of modern machine learning applications. As we proceed, the ensuing sections will offer a comprehensive overview of these applications. We will explore the methodology, experimentation, and results that showcase the practical implications and benefits of applying the Machine Learning Model

Balancer concept in addressing the challenges of sustainability and streaming mode deployments in MLS. Through this exploration, we aim to provide valuable insights and strategies that propel MLS toward greater adaptability, energy efficiency, and operational effectiveness.

7.2 EcoMLS: Enhancing sustainability in MLS

This EcoMLS work and approach in collaboration with M. Tedla, illustrates the implementation of the model switching concept to enhance sustainability in machine learning systems. The primary focus of this thesis section is to demonstrate the practicality and effectiveness of model switching as a strategy for balancing energy efficiency and system performance. Readers interested in a comprehensive understanding of the EcoMLS approach, its detailed architecture, and operational dynamics are encouraged to read the publication "EcoMLS: A Self-Adaptation Approach for Architecting Green ML-Enabled Systems." This reference provides complete details into the EcoMLS approach and its foundational principles[1].

7.2.1 EcoMLS: Running Example

Our implementation of the Machine Learning Model Balancer concept is exemplified through the integration with the SWITCH [53] exemplar's managed system and environment manager, tailored for an object detection web service. This service employs a comprehensive architecture designed to efficiently process image-based requests. It includes an *Image Ingestion Service* for simulating real-world asynchronous request handling; an *Image Store* functioning as a dynamic first-in-first-out (FIFO) queue for incoming data; a *Data Preprocessor* that prepares images for detection; a *Model Loader* responsible for dynamically selecting machine learning models; a *Model Repository* containing a variety of preloaded models for rapid deployment; an *ML Model* that performs the core detection processing; a *Post Processor* for refining detection outcomes; and a *Result Storage* for the temporary holding of processed data before its final transfer. This arrangement reflects the operational design of services like Google Cloud Vision or Amazon Rekognition, showcasing its practicality and adaptability in handling object detection tasks.

Within this system, a set of machine learning models, denoted as M, is defined. Each model m_j within M is a variant of the YOLO algorithm [69], distinguished by its size and computational demands. These models are evaluated based on two primary metrics: energy consumption, which measures the electricity used during the inference process, and the confidence score, which indicates the model's accuracy in object identification and classification. The set M encompasses the YOLOv5 models: YOLOv5n (nano), YOLOv5s (small), YOLOv5m (medium), and YOLOv5l (large), all of which are provided by Ultralytics [41] and pretrained on the COCO 2017 training dataset [51]. The models vary in their parameter count, affecting both their energy consumption (E_j) and detection confidence score (c_j), where j serves as the model index within M.

[1]The EcoMLS approach complete details are not covered here; its specifics are beyond this thesis's scope as the concept's application, not its development, represents the thesis author's main contribution

The model variants from YOLOv5n to YOLOv5l demonstrate a spectrum of trade-offs between energy consumption (E_j) and confidence score (c_j). For instance, YOLOv5n, which is optimized for low energy consumption, utilizes 2 mJ of energy (E_1) and achieves a mean Average Precision (mAP) of 45.7 (c_1). Conversely, YOLOv5l, designed for higher accuracy, consumes 16 mJ (E_n) to reach an mAP of 68.9 (c_n). This variation underscores the necessity to balance computational demand with detection accuracy in real-world applications.

The operational framework processes a continuum of image requests (i), selecting an appropriate model (m_j) for each based on a strategy that optimizes the balance between energy efficiency (E_i) and detection confidence (c_i). Through the EcoMLS approach, we aim to dynamically navigate between these models, striving to strike an optimal balance that enhances both the sustainability and the efficacy of the detection service.

7.2.2 EcoMLS: Approach

EcoMLS applies the Machine Learning Model Balancer concept to improve sustainability in Machine Learning-Enabled Systems (MLS) by dynamically adjusting model selection to balance energy efficiency and performance. The approach integrates a Learning Engine and utilizes the MAPE-K loop (Monitor, Analyze, Plan, Execute, Knowledge) to continuously assess and adapt machine learning models based on their energy consumption and performance metrics. A key feature of EcoMLS is the implementation of an ϵ-greedy algorithm that makes decisions based on a performance score calculated as Energy * (1 - Confidence). This strategy allows for a dynamic switch between models, prioritizing energy efficiency when possible while maintaining confidence in the system's outputs. The algorithm optimizes this balance by either reducing energy consumption or enhancing output confidence, depending on real-time requirements. Complete details of the approach available in our publication 'EcoMLS: A Self-Adaptation Approach for Architecting Green ML-Enabled Systems' [76].

7.2.3 EcoMLS: Experimentation and Results

Focusing on the application of machine learning model balancer concept to enhance sustainability within Machine Learning-Enabled Systems (MLS), this section is dedicated to illustrating the efficacy of model switching as derived from the broader context of self-adaptation, directly addressing the overarching research question (RQ3) outlined in Chapter 1. This discussion centers on the adaptability of MLS, specifically within the aspect of sustainability in the computer vision domain, and examines the practical implications of applying model switching strategies to balance energy efficiency and system performance. The discussion concentrate on two key aspects outlined in following research questions:

RQ3.1 How does the implementation of model switching idea compare to other static and non-adaptive methodologies in terms of enhancing the sustainability of MLS?

RQ3.2 In what ways does model switching contribute to the balance between energy efficiency and performance within ML-Enabled Systems?

To comprehensively address these questions, we initially outline our experimental framework, subsequently stating the outcomes of our empirical evaluations. This approach aims to explain the practical benefits and the broader applicability of model switching in promoting sustainability and efficiency in machine learning deployments.

7.2.4 EcoMLS: Experimental Setup

To evaluate the EcoMLS approach, our experimental setup is adopted from the SWITCH [53] exemplar. It employs an object detection system, as detailed in Section 7.2, utilizing YOLOv5 models and FastAPI simulating requests using the FIFA98 World Cup log dataset, processing 25,000 image requests. For evaluation, as detailed in our approach, the COCO 2017 test dataset is utilized. For the testing phase, which includes performance results, the COCO 2017 unlabelled dataset is employed. The YOLOv5 models (YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l) used are pre-trained by Ultralytics on the COCO 2017 training dataset. The experiments were conducted on a system equipped with an Intel Core i7-11370H processor, NVIDIA GeForce RTX 3050 Ti 4GB Graphics, 16GB DDR4 3200MHz SDRAM, and developed using Python 3.11. To measure energy consumption, we utilized pyRAPL[2], a Python package specifically designed for assessing the energy consumption and power usage of software applications running on Intel processors. The EcoMLS approach's evaluation included varying the ϵ value to analyze its impact on balancing exploration and exploitation. We compared EcoMLS's adaptive model selection with individual YOLOv5 model performances and three naive strategies: (1) using fixed knowledge for model switching (naive 1), (2) updating knowledge based on average confidence (naive 2), and (3) incorporating dynamic updates of both confidence and energy metrics in knowledge (naive 3). Our approach builds on naive 3 by adding an ϵ-greedy mechanism, enhancing exploration through probability. The complete specifics of our implementation, parameters and results can be found here[3].

7.2.5 EcoMLS: Results

In exploring the efficacy of the EcoMLS framework, particularly the role of model switching for sustainability within Machine Learning-Enabled Systems (MLS), we undertook a comparative analysis against baselines and static model approaches, focusing on their energy consumption and accuracy metrics. The examination is following research questions RQ3.1 and RQ3.2, with an emphasis on understanding how model switching, a core aspect of the EcoMLS approach, contributes to the sustainability and performance of MLS.

[2]https://pypi.org/project/pyRAPL/. Latest version released on Dec 19, 2019
[3]https://github.com/sa4s-serc/EcoMLS

RQ3.1 How does the implementation of model switching idea compare to other static and non-adaptive methodologies in terms of enhancing the sustainability of MLS?

Our assessment uses detailed metrics from Table 7.1, showing that EcoMLS, with a strategic $\epsilon = 0.1$ setting, processed 13,040 images within a low score range, demonstrating its ability to maintain a balanced trade-off between energy efficiency and model accuracy. This contrasts with the 'nano' model, which, while processing a higher number of images (20,265), did so at the expense of lower average confidence (0.536), highlighting a preference for energy savings over accuracy. In stark comparison, the 'large' model, with the highest accuracy (C_avg = 0.675) and energy demand (17.705), positions EcoMLS as an effective intermediary, achieving an average confidence of 0.61 with considerably reduced energy usage (2.762). This not only signifies a balanced approach but also demonstrates the efficacy of model switching in enhancing the system's sustainability and accuracy.

Table 7.1: EcoMLS: Model score frequency table

Name	0-1	1-2	2-3	3-4	4-5	5-6	6-7
nano	20265	4480	237	16	1	0	0
small	5301	11870	6139	1258	290	102	31
medium	2394	4447	6018	5319	3670	1870	706
large	645	2398	2304	2691	3204	3148	2693
EcoMLS	13040	10740	841	180	61	37	31
NAIVE1	14301	9996	562	96	27	9	4
NAIVE2	14860	9560	491	62	12	4	3
NAIVE3	9188	12408	2698	492	156	43	9

Figure 7.2 details the frequency of model switches within EcoMLS, underscoring its adaptability and the strategic use of the ϵ-greedy mechanism for model selection. The frequent switching, particularly evident with 160 switches at $\epsilon = 0.1$, reflects a understanding of operational conditions, allowing EcoMLS to outperform traditional and static models by dynamically balancing energy consumption against model confidence.

RQ3.2 In what ways does model switching contribute to the balance between energy efficiency and performance within ML-Enabled Systems?

The core of our evaluation of EcoMLS centers on its ability to navigate the intricate balance between energy use and accuracy. Employing a performance score metric ($Score_m = E_j \times (1 - C_j)$), we found that EcoMLS, especially at $\epsilon = 0.1$, exhibits an optimal blend of low energy consumption (2.762) and high model confidence (0.61). This balance, as detailed in Table 7.2, is further highlighted by a

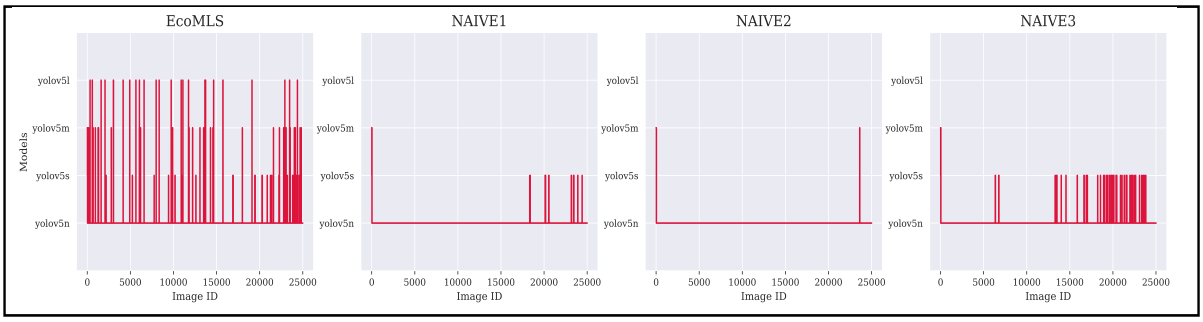

Figure 7.2: Model switching: Naive baselines Vs. EcoMLS

comparative analysis of energy consumption and confidence scores, visualized in Figure 7.3.
The graph shows EcoMLS's performance predominantly aligning below a delineated trade-off line, especially marked at $\epsilon = 0.1$, indicating a successful maintenance of balance despite the inclination for more detections to exceed this line at higher ϵ values.

The dynamic model switching inherent to EcoMLS proves instrumental in this balance, making the system a flexible approach to sustainably manage MLS operations. This balance is not merely about achieving lower energy consumption or higher accuracy in isolation but about harmonizing these objectives within the operational dynamics of MLS, thereby ensuring sustainable and efficient system performance. Through these evaluations, the value of model switching within EcoMLS is demonstrated, aligning with the aim of sustainability in MLS. The detailed results from this exploration affirm the significance of self-adaptive model switching, as facilitated by EcoMLS, in advancing the field of sustainable MLS by efficiently managing the dual objectives of energy conservation and accuracy enhancement.

7.3 RelMLS: Self-Adaptation of Streaming Mode MLS

The RelMLS approach, applying the Machine Learning Model Balancer concept within MLS deployed in streaming mode, is demonstrated through an adapted version of the SWITCH exemplar architecture, specifically designed for video object detection for this evaluations.

7.3.1 RelMLS: Running Example and Implementation Details

RelMLS running example implementation is tailored to handle the continuous stream of data inherent to video feeds, making it well-suited for dynamic environments where contextual reliability is paramount.

In this system, a *Data Stream Component* simulates the reception of video streams from real-world scenarios, directing the data to the *Frame Ingestion Service*. This service meticulously extracts individual frames from the stream, storing them in a *Frame Storage* queue for sequential processing. The *Data Preprocessor* then retrieves each frame, preparing it for analysis by the ML model.

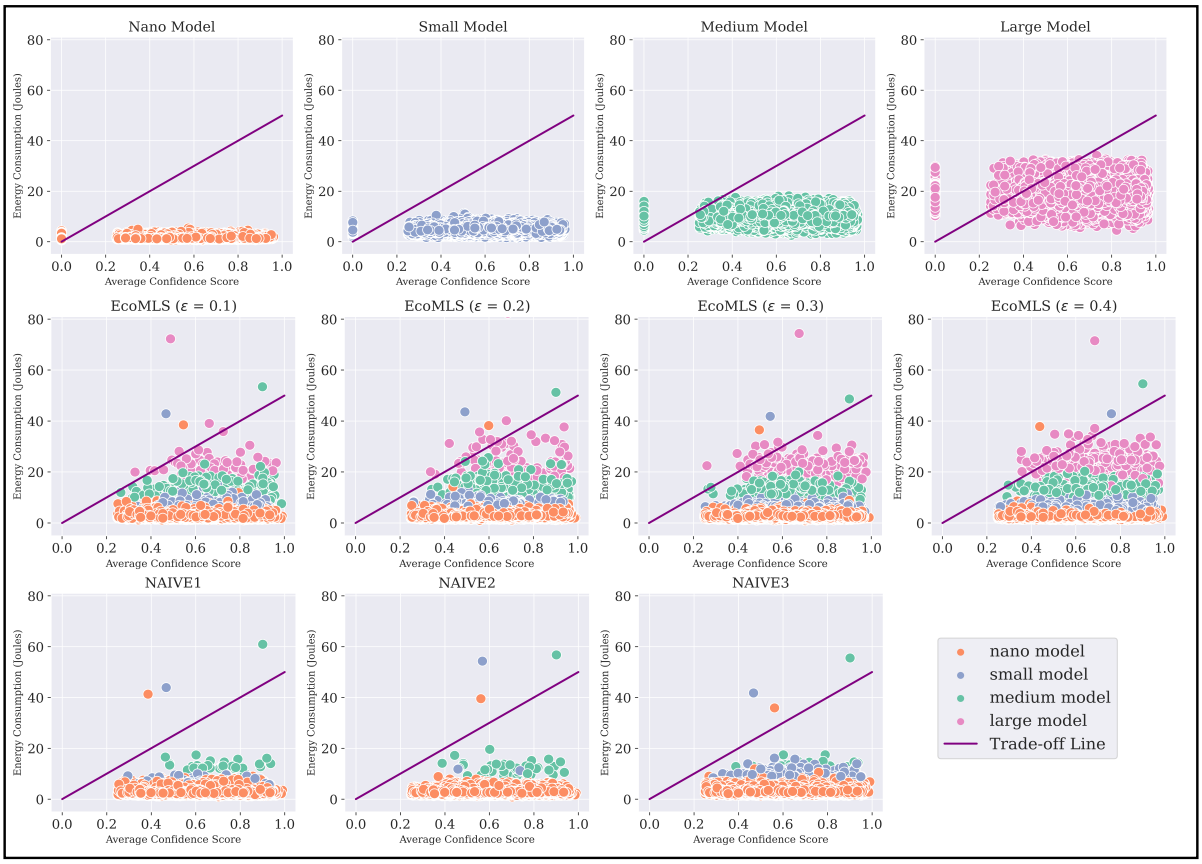

Figure 7.3: Trade-off between energy consumption and the average confidence score of individual models (first row), EcoMLS with varying ϵ (second row), and naive baselines (third row).

Given the system's foundation on the SWITCH exemplar, we've evolved its architecture to cater specifically to video inputs. While SWITCH primarily processes static images, RelMLS advances this capability to address the dynamic nature of video streams. This adaptation involves a *Model Loader* that dynamically selects from a range of ML models housed within the *Model Repository*, optimizing for contextual reliability—defined here as the accuracy of detected objects within frames, crucial for applications like crowd counting.

For this running example, we focus on crowd counting within video object detection, selecting four models to span the spectrum from speed to accuracy:

1. Nanodet: A lightweight model optimized for speed, ideal for mobile applications where rapid frame processing is essential. [68] 2. P2PNet: The state-of-the-art in crowd counting, offering unparalleled accuracy in dense object environments. [74] 3. YOLOv5m and YOLOv5l: These models from Ultralytics strike a balance between speed and detection precision, making them versatile choices for varied operational scenarios. [41]

Each model is chosen for its unique attributes: Nanodet for its swift processing capabilities, P2P for its exceptional accuracy in counting, and the YOLOv5 variants for their balanced performance. This

Table 7.2: EcoMLS: Comparison of energy metrics and confidence scores across different approaches

Approach name	C_{avg}	E_{avg}	$E_{monitor}$	$E_{analyzer}$	$E_{planner}$	$E_{executor}$	E_{mape-k}	$E_{avg} + E_{mape-k}$	No. of Switches
nano	0.536	1.61	-	-	-	-	-	1.61	0
small	0.611	4.327	-	-	-	-	-	4.327	0
medium	0.652	8.918	-	-	-	-	-	8.918	0
large	0.675	17.705	-	-	-	-	-	17.705	0
EcoMLS ($\epsilon = 0.1$)	0.61	2.762	1.284	0.001	0.001	0.0	1.285	4.047	160
EcoMLS ($\epsilon = 0.2$)	0.612	3.044	1.166	0.001	0.001	0.0	1.168	4.212	324
EcoMLS ($\epsilon = 0.3$)	0.613	2.912	1.035	0.001	0.001	0.0	1.037	3.949	313
EcoMLS ($\epsilon = 0.4$)	0.616	3.564	0.959	0.001	0.001	0.001	0.961	4.525	676
NAIVE1	0.609	2.47	1.226	0.001	0.001	0.0	1.228	3.697	20
NAIVE2	0.609	2.399	1.21	0.001	0.001	0.0	1.213	3.612	5
NAIVE3	0.609	3.319	1.658	0.001	0.002	0.0	1.661	4.98	106

diverse model lineup ensures that RelMLS can adaptively switch between models based on the current video frame's contextual requirements, maintaining a reliable count of objects (e.g., people) detected in each frame. For testing purpose we have used used Japan walk crowd videos from Japan Explorer YT channel[4]. Particularly video named as 'Japan's Thousands Gate a paradise that Mesmerizes the travelers'[5] from Japan Explorer YouTube channel[6]. It has 39150 frames, with 30 fps, total duration of 21 min 45 seconds, dimensions 1280 x 720, size 391 Mb and in MPEG-4 video format.

The system processes video streams frame by frame, employing the *ML Model* selected by the *Model Loader* for object detection. Post-detection, the *Post Processor* refines the results, which are then stored temporarily in the *Result Storage*. This process repeats for each incoming frame, ensuring a seamless and adaptive pipeline that prioritizes contextual reliability in the detection results.

By focusing on crowd counting, RelMLS addresses a critical application of video object detection, demonstrating the capability of the Machine Learning Model Balancer concept to enhance the reliability and effectiveness of ML systems in streaming mode. This approach not only underscores the importance of adaptability in ML systems but also showcases the potential of dynamic model selection in maintaining high-quality, contextually relevant detection outcomes in real-time video streams.

[4]Proper permission to use the video for academic research purposes has been obtained from Japan Explorer YT

[5]`https://youtu.be/ndJ3X2uiHM4?si=e5txlFIraVdc7kG8`

[6]`https://www.youtube.com/@Japanexplorer1`

7.3.2 RelMLS: Approach

The exploration of the Machine Learning Model Balancer concept extends into the realm of RelMLS, an approach designed specifically for enhancing contextual reliability in streaming Machine Learning-Enabled Systems (MLS). Emphasizing the real-time analysis and adaptation of video stream processing for tasks such as object detection, RelMLS aims for advancing MLS towards greater operational integrity and reliability.

7.3.2.1 RelMLS Approach: System Architecture

The architecture of RelMLS integrates a series of components structured to manage and process continuous video streams effectively as shown in 7.1:

In running example managed system it has *Frame Ingestion Service* Manages the intake of video streams, partitioning continuous footage into discrete frames for processing. *Data Preprocessor* Prepares frames for analysis, ensuring they meet the necessary criteria for accurate object detection. *Model Loader* Dynamically selects the optimal model for the current context, balancing processing demands with detection accuracy. And in managing system, RelMLS has *ML Model Repository* which houses a diverse range of ML models, including a designated model serving as the benchmark for reliability (the "most reliable model"). *Rulebook Manager* and MAPE-K Knowledge Components dynamically adapt operational strategies based on real-time data analysis, continually refining the system's approach to model selection and adaptation.

7.3.2.2 The Core of RELMLS: Contextual Reliability Index (CRI)

At the heart of RELMLS is the Contextual Reliability Index (CRI), a novel metric designed to assess the trustworthiness of the model's predictions against a dynamically chosen benchmark model within the ML Model Repository. The CRI is calculated as per Equation 7.1 :

$$CRI = \frac{\text{Contextual result from model-in-use}}{\text{Result from most reliable model}} \tag{7.1}$$

This ratio, aimed to hover between 0.5 and 0.9 for optimal reliability, informs the system's adaptive strategies, ensuring that the contextual information remains both accurate and trustworthy.

7.3.2.3 Operational Dynamics: MAPE-K Framework Implementation

RelMLS harnesses the MAPE-K framework for its adaptive loop, outlined as follows:
Monitor observes system performance and contextual reliability through metrics such as CPU usage, response times, confidence scores, and the number of detected objects (contextual information). This data is stored in the *Logs Repository*. *Analyzer* This component periodically receives the latest processed frame and evaluates it against the most reliable model to determine the CRI. Based on the CRI value and predefined thresholds, the Analyzer decides whether adaptation is necessary. *Rulebook Manager* Operates

asynchronously, receiving frames from the Analyzer to compare across all models in the repository, updating the *Rules Repository* with new adaptation rules based on the outcomes. *Planner* Informed by the CRI and the *Rules Repository*, the Planner devises strategies for model switching. It selects models that balance speed and accuracy within the desired CRI range.

The decision-making process by *Planner* utilizes a set $\mathcal{M} = \{m_1, m_2, ..., m_n\}$, where each m_i represents a model, ordered by processing speed. Given the current Contextual Reliability Index (CRI) and a target reliability range $CRI_{target} = [0.5, 0.9]$, the selection of an optimal model m^* is formalized as follows:

1. If $CRI < 0.5$, the search begins from the slowest model to prioritize accuracy.
2. If $CRI > 0.9$, the search starts with the fastest model to prioritize speed.
3. For each model $m_i \in \mathcal{M}$, evaluate if $CRI_{m_i} \in CRI_{target}$.
4. Select $m^* = \min\{m_i : CRI_{m_i} \in CRI_{target}\}$, prioritizing the fastest model that meets the CRI criteria.

This ensures RelMLS dynamically adapts to the contextual requirements, maintaining an optimal balance between detection speed and accuracy.This highlights how the Planner assesses and selects models based on the CRI, incorporating both proactive and reactive measures to adjust the model in use. It delineates the process of selecting the fastest model within an acceptable CRI range or defaulting to the most reliable model when necessary, thereby ensuring that the system dynamically maintains its contextual reliability within the specified operational parameters.

Executor Implements the Planner's decisions by switching to the selected model, thus ensuring the system's continuous adaptation to maintain contextual reliability.

Through the detailed exposition of the RelMLS approach, encapsulating its system architecture, the pivotal role of CRI, and the intricacies of its adaptive cycle within the MAPE-K framework, we demonstrate a structured methodology for enhancing contextual reliability in MLS. This approach underscores the adaptability and precision of RelMLS in handling real-time video stream processing for critical ML tasks, ensuring operational efficiency and reliability.

7.3.2.4 RelMLS: Empirical Validation and Results

RQ3.4: How does RelMLS maintain operational efficiency while ensuring high contextual reliability in real-time streaming video object detection tasks?

The empirical results demonstrate that RelMLS effectively maintains a balanced trade-off between operational efficiency and high contextual reliability. By dynamically adapting the model selection based on the Contextual Reliability Index (CRI), RelMLS ensures that the system remains responsive to the streaming video's demands without compromising the accuracy of object detection. This is evident in the comparative analysis (Table 7.3), where RelMLS optimizes both the average persons detected and frames per second (FPS), outperforming individual baseline models in terms of detection accuracy while maintaining a reasonable FPS rate. This section explores the empirical validation of the RelMLS

approach, focusing on its efficacy in the context of video object detection and, more specifically, crowd counting accuracy. The validation is conducted through a comparative analysis involving baseline models (Nanodet, YOLOv5m, YOLOv5l, and P2P) against the adaptive RelMLS framework. Results emphasize the adaptive capabilities of RelMLS in balancing detection accuracy and operational efficiency within a streaming context.

Experimental Setup and Metrics : The "Japan walk crowd" video, characterized by dynamic and densely populated scenes, served as the testbed for this evaluation. Key metrics for comparison included the average number of persons detected per second (averaged over 10-second intervals to normalize differences in frame processing rates across models) and the frames per second (FPS) achieved by each model during processing.

Table 7.3: RelMLS: Comparative Analysis of Detection Performance and Efficiency

Model	**Nanodet**	**Yolov5m**	**Yolov5l**	**P2P**	**RelMLS**
Avg Persons Detected (10 sec avg)	8.62	7.95	8.36	25.49	15.23
FPS (10 sec avg)	24.89	8.72	4.28	0.56	5.83
Total Persons Detected	1129	1041	1087	3340	1997

Table 7.3 presents a comparative analysis showcasing the balance between detection performance and operational efficiency across different models, including the adaptive RelMLS approach. The averaging of persons detected and FPS over 10-second intervals accounts for the variability in output rates due to differing processing speeds of each model.

1. Nanodet and Yolov5 variants demonstrated high FPS, indicative of their suitability for scenarios requiring rapid frame processing. However, their detection accuracy, as reflected by the average number of persons detected, was lower compared to the more computationally intensive P2P model.

2. P2P, while achieving the highest detection accuracy, suffered significantly in terms of FPS, highlighting the trade-off between accuracy and operational efficiency inherent to existing models.

3. RelMLS emerged as a balanced solution, optimizing both detection accuracy and FPS. By dynamically switching between models based on the Contextual Reliability Index (CRI), RelMLS adapts to the operational context, ensuring reliable crowd counting without substantial compromises on processing speed.

The RelMLS approach, through its adaptive mechanism, effectively bridges the gap between the need for high detection accuracy and the demand for operational efficiency in real-time video processing tasks. By averaging over 10-second intervals, the results underscore RelMLS's capability to maintain a higher average detection accuracy compared to individual baseline models (except P2P) while achieving

a more favorable FPS rate than the most accurate model (P2P). This balance underscores the effectiveness of RelMLS in managing the intrinsic trade-offs of MLS applications, particularly in scenarios where contextual reliability is crucial.

Therefore, in response to RQ3.4, RelMLS demonstrates significant effectiveness in managing the intrinsic challenges of streaming MLS applications, showcasing its ability to provide reliable and efficient real-time video object detection.

7.4 Discussion

In this section, we provide the insights, lessons learned and threats to validity from the comprehensive evaluations of the EcoMLS and RelMLS approaches. Both were meticulously developed to apply the Machine Learning Model Balancer concept to two distinct yet important aspects of machine learning-enabled systems: sustainability and streaming mode adaptability.

7.4.1 Lessons Learned: EcoMLS

The EcoMLS approach provided us with valuable lessons on the intricate balance between sustainability and system performance in MLS. Through its implementation and testing, several key insights emerged:

1. Adaptability is Key to Sustainability: EcoMLS demonstrated that dynamic model switching, driven by real-time performance and energy consumption metrics, significantly enhances the sustainability of MLS. By adapting to varying operational conditions, EcoMLS minimized energy consumption without compromising on the accuracy or reliability of the system.

2. Balancing Trade-offs: One of the principal challenges encountered was the trade-off between energy efficiency and model confidence. EcoMLS showed the importance of a nuanced approach to model selection, where not only are energy consumption and accuracy balanced, but the operational context is also considered to make informed decisions.

3. Impact of Exploration Strategies: The use of an ϵ-greedy mechanism in EcoMLS highlighted the impact of exploration strategies on system performance. It was observed that a carefully chosen ϵ value could effectively balance the exploration of new models with the exploitation of known efficient models, optimizing the system's overall performance.

4. Sustainability Beyond Energy Consumption: The EcoMLS evaluation extended the conversation on sustainability beyond mere energy consumption to include aspects such as computational resources and system longevity. It illuminated the broader implications of adaptive systems in reducing the environmental footprint of MLS.

7.4.2 Lessons Learned: RelMLS

Similarly, the RelMLS approach provided profound insights into the adaptation of MLS in streaming modes, emphasizing the importance of contextual reliability:

1. Contextual Reliability as a Cornerstone: The concept of Contextual Reliability Index (CRI) introduced by RelMLS proved to be pivotal in ensuring the accuracy and reliability of streaming video object detection. It showcased the necessity of maintaining high contextual reliability to inform adaptive decisions effectively.

2. Dynamic Benchmarking for Adaptation: The implementation of RelMLS illustrated the value of dynamic benchmarking, where models within the system could be re-evaluated against changing operational scenarios. This dynamic benchmarking facilitated informed model switching, enhancing the system's adaptability and performance.

3. Challenges in Streaming Adaptation: Addressing the unique challenges of streaming MLS, such as maintaining high throughput and low latency, shows the need for optimized data processing pipelines and efficient model management strategies. RelMLS's architecture offered a blueprint for addressing these challenges, emphasizing the role of efficient frame processing and model selection.

4. Balance Between Speed and Accuracy: RelMLS's evaluations reaffirmed the critical balance between processing speed and detection accuracy in streaming MLS. The approach highlighted the necessity of choosing models that not only provide high accuracy but also maintain a viable frame rate to ensure the system's responsiveness.

Both EcoMLS and RelMLS contribute valuable perspectives on implementing the Machine Learning Model Balancer concept, each addressing distinct aspects of adaptability in MLS. Their evaluations offer novel insights into the challenges and opportunities of enhancing MLS with self-adaptive capabilities, setting the stage for future research and development in this promising field.

7.4.3 Threats to Validity

In this subsection, we scrutinize the potential limitations and biases in our research methodologies, addressing how these might influence the outcomes and interpretations of our studies on EcoMLS and RelMLS.

7.4.3.1 EcoMLS: Threats to Validity

Threats to *external validity* concern the focus on a single type of task and a select group of machine learning models, and by limiting our examination to the inference phase. To address the first challenge, we chose a range of YOLOv5 models, varying in complexity (from YOLOv5n to YOLOv5l), and used diverse datasets, including COCO 2017. This strategy aimed to cover different visual data types and model sizes. However, our decision to focus only on the inference phase, without considering the full lifecycle of machine learning models like training and tuning, was intentional. This choice was made to

study energy consumption during inference specifically, acknowledging its narrow scope in reflecting the entire machine learning process.

A threat to the *internal validity* could be the impact of varying hardware conditions like temperature changes on the results of the evaluations. To tackle this, we implemented a 24-hour sleep period before each test to stabilize hardware conditions and performed a warm-up run to maintain consistency throughout our experiments. The threats to *construct validity* could be constituted by the accuracy of our energy consumption measurements. To mitigate this, we utilized the pyRAPL library in Python (pyRAPL makes use of the RAPL library provided by Intel), which is a well-known library for measuring energy consumption. Concerning *conclusion validity*, the main threat is the potential low statistical power of our tests, which we addressed by conducting multiple experiments across different settings and conditions. Additionally, we took precautions to minimize the impact of extraneous variables, such as background tasks, on our energy consumption measurements by ensuring a clean experimental environment.

7.4.3.2 RelMLS: Threat to Validity

Threats to *external validity* for RelMLS also revolve around the choice of application domain (video object detection, specifically crowd counting) and the selected models for the evaluation. While we aimed to represent a realistic scenario by choosing a real-world video and a mix of models including Nanodet, P2PNet, and YOLOv5 variants, this choice may not encompass all streaming MLS applications. Future work could expand the scope to include a broader range of tasks and models, particularly those requiring different forms of contextual analysis.

For *internal validity*, similar to EcoMLS, the consistency of hardware performance over time could influence the results. The streaming nature of RelMLS, involving continuous data processing, might accentuate such effects. To mitigate this, consistent environmental conditions were maintained, and evaluations were structured to allow for system normalization before each testing session.

The accuracy of contextual reliability measurements and model switching decisions represents a potential threat to *construct validity*. Ensuring the precision of these measures was critical, and we relied on established metrics and methodologies to evaluate model performance and contextual accuracy. However, the dynamic nature of streaming applications and the potential variability in video content complexity could affect these evaluations.

Lastly, regarding *conclusion validity*, the statistical significance of RelMLS's performance improvements was carefully considered. The experimental design included repetitions and varied conditions to ensure robust conclusions could be drawn. Nevertheless, the complexity of real-world streaming environments means that further validation in diverse operational contexts would be beneficial to confirm these findings.

Overall, while both EcoMLS and RelMLS present promising approaches to their respective challenges, acknowledging these threats to validity is important for accurately interpreting their contributions and identifying areas for future research and improvement.

Chapter 8

Conclusion and Future Work

In this chapter, we conclude our exploration into self-adaptive mechanisms within Machine Learning-Enabled Systems (MLS), ***summarizing key findings and proposing directions for future research to continue advancing the field.***

8.1 Conclusion

In this thesis, we have started with a exploration of the runtime uncertainties faced by Machine Learning-Enabled Systems (MLS) and their impact on the Quality of Service (QoS). Our investigation has highlighted the challenges presented by such uncertainties, from environmental variability to data and model drifts, which collectively pose significant hurdles to maintaining consistent QoS. In response to these challenges, we proposed self-adaptive mechanisms, mainly Machine Learning Model Balancer concept and the AdaMLS approach, aimed at dynamically managing these uncertainties to uphold or enhance the QoS.

8.1.1 Addressing Research Questions

RQ1: *In the context of machine learning-enabled systems, how can self-adaptive mechanisms be developed and applied to mitigate runtime uncertainties, thereby enhancing their Quality of Service (QoS)?*

The Machine Learning Model Balancer concept and the AdaMLS approach have demonstrated that self-adaptive strategies can be effectively developed and applied within MLS. By allowing real-time model switching based on operational conditions, they provide a novel solution for mitigating runtime uncertainties and ensuring an optimal balance between model accuracy and system performance. This not only answers RQ1 but also highlights the potential of engineering self-adaptive capabilities into MLS.

RQ2: *What tools can be devised to facilitate the implementation and exploration of self-adaptation within machine learning-enabled systems for researchers, students, and practitioners?*

The development of SWITCH as an exemplar for facilitating the implementation and exploration of dynamic model switching strategies stand as a novel contribution to the field. SWITCH, designed for both theoretical exploration and practical application, offers a versatile platform for researchers, students, and practitioners to delve into the complexities of self-adaptive MLS, directly addressing RQ2 by providing a platform to engage with and refine self-adaptive model switching approaches.

RQ3: *How can self-adaptation in MLS be applied and generalized across MLS deployment modes and aspects in computer vision domain, broadening their applicability and impact?*

Through the EcoMLS and RelMLS approaches, we expanded the application of self-adaptation principles across different aspects, demonstrating their generalizability and impact. EcoMLS showcases how these strategies can be leveraged for environmental sustainability, optimizing energy efficiency without highly compromising model confidence. Meanwhile, RelMLS shows promising improvement in QoS in MLS for streaming data, highlighting the versatility and broad applicability of self-adaptive strategies, thereby offering a answer to RQ3.

8.1.2 Summary of Contributions

Our research contributions include:

- The conceptualization and validation of the Machine Learning Model Balancer as a novel concept for enhancing the adaptability of MLS.
- The development of AdaMLS, a novel approach that leverages unsupervised learning for dynamic model switching, demonstrating significant improvements in QoS.
- The creation of SWITCH, an novel exemplar that enables hands-on experimentation and investigation into dynamic model switching strategies as self-adaptation mechanism for MLS.
- The application and generalization of self-adaptation principles through EcoMLS and RelMLS, showcasing their efficacy in diverse operational contexts and aspects.

Therefore, we advance the state-of-the-art in the field of self-adaptive machine learning-enabled systems by introducing dynamic model switching strategies that enhance system adaptability and performance. Our work achieves an improvement in the quality of service across various operational contexts, highlighting the potential of our approaches for broader application and impact. In conclusion, we believe our contributions not only address the challenges posed by runtime uncertainties but also pave the way for future research and development in self-adaptive MLS.

8.2 Future Work

We believe that the research presented in this thesis marks only the beginning of a comprehensive exploration into the domain of runtime uncertainties in Machine Learning-Enabled Systems (MLS) and their management through self-adaptive strategies. The evolving nature of software systems, coupled with rapid advancements in machine learning technologies, necessitates ongoing innovation and research. Here, we outline the future directions that stem from our work, highlighting areas for further exploration and development.

Future Directions for AdaMLS: The AdaMLS approach, with its foundation on dynamic model switching to manage runtime uncertainties, opens up several avenues for future research. Firstly, there is a significant opportunity to explore a diverse range of learning techniques and model-switching strategies to further enhance the adaptability and effectiveness of AdaMLS across various operational contexts. Secondly, extending AdaMLS's applicability to diverse domains beyond object detection, such as Natural Language Processing (NLP) and autonomous systems, could explore its potential in a broader spectrum of ML applications. Lastly, an important future direction involves exploring opportunities for improving the environmental and economic sustainability of MLS through AdaMLS, focusing on strategies that minimize resource consumption while maximizing system performance.

Advancements in SWITCH: SWITCH has demonstrated its potential as a versatile tool for facilitating the implementation and exploration of dynamic model switching strategies. Future enhancements to SWITCH could include further development to support a broader range of self-adaptive MLS tasks, including retraining, transfer learning, and user-driven customization, beyond the MAPE-K framework. Additionally, conducting empirical studies and adaptations of SWITCH to address emerging challenges and uncertainties in MLS will ensure that it remains at the forefront of research and practical applications. Enhancing SWITCH as an educational tool by integrating additional features that facilitate the exploration of self-adaptive strategies in MLS will enrich the learning experience for users.

Expanding EcoMLS: The EcoMLS approach, aimed at optimizing energy efficiency in MLS without compromising model confidence, suggests several future directions. These include broadening the application of EcoMLS to include sustainable computing initiatives, particularly through its integration with edge computing and lightweight AI models, to promote energy efficiency across different computing paradigms. Another direction involves developing EcoMLS into a comprehensive tool for sustainability-aware decision-making, enabling the design and deployment of greener ML-enabled systems. Extending the impact of EcoMLS to new domains to demonstrate its versatility and effectiveness in promoting environmental sustainability in ML applications is also a key future direction.

Future directions with RelMLS: The RelMLS approach, with its focus on enhancing the adaptability and effectiveness of MLS for streaming data, presents opportunities for further innovation. Enhancing the adaptability and efficiency of RelMLS in processing streaming data, focusing on minimizing latency and maximizing accuracy in real-time decision-making scenarios, is a crucial area for future work. Investigating the application of RelMLS in high-stakes environments, where reliability and real-time processing are paramount, could lead to significant advancements. Additionally, exploring new methodologies for

dynamic model management in RelMLS to further improve its performance and reliability remains an important future direction.

In conclusion, our research contributions aims to lay the groundwork for future research in self-adaptation of MLS. By continuing to explore and innovate within this space, we can advance the state-of-the-art in MLS and contribute to the development of more resilient, efficient, and environmentally friendly technological solutions.

Bibliography

[1] Trend. https://tinyurl.com/2abwwdx7.

[2] N. Adams. Dataset shift in machine learning by j. quiñonero-candela; m. sugiyama; a. schwaighofer; n. d. lawrence. *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 173, 01 2010.

[3] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300. IEEE, 2019.

[4] D. Aparício, R. Barata, J. Bravo, J. T. Ascensão, and P. Bizarro. Arms: Automated rules management system for fraud detection, 2020.

[5] E. Barbierato and A. Gatti. Towards green ai. a methodological survey of the scientific literature. *IEEE Access*, pages 1–1, 2024.

[6] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 610–623, New York, NY, USA, 2021. Association for Computing Machinery.

[7] B. Branco, P. Abreu, A. S. Gomes, M. S. C. Almeida, J. T. Ascensão, and P. Bizarro. Interleaved sequence rnns for fraud detection. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '20. ACM, Aug. 2020.

[8] T. Bureš. Self-adaptation 2.0. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 262–263, 2021.

[9] R. Calinescu, R. Mirandola, D. Perez-Palacin, and D. Weyns. Understanding uncertainty in self-adaptive systems. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 242–251, 2020.

[10] J. Cámara, P. Correia, R. de Lemos, and M. Vieira. Empirical resilience evaluation of an architecture-based self-adaptive software system. In *Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures*, QoSA '14, page 63–72, New York, NY, USA, 2014. Association for Computing Machinery.

[11] J. Cámara, B. Schmerl, and D. Garlan. Software architecture and task plan co-adaptation for mobile service robots. In *2020 IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and*

Self-Managing Systems (SEAMS), SEAMS '20, page 125–136, New York, NY, USA, 2020. Association for Computing Machinery.

[12] J. Cámara, M. Silva, D. Garlan, and B. Schmerl. Explaining architectural design tradeoff spaces: A machine learning approach. In *Software Architecture: 15th European Conference, ECSA 2021, Virtual Event, Sweden, September 13-17, 2021, Proceedings*, pages 49–65. Springer, 2021.

[13] J. Cámara, J. Troya, A. Vallecillo, N. Bencomo, R. Calinescu, B. H. C. Cheng, D. Garlan, and B. Schmerl. The uncertainty interaction problem in self-adaptive systems. *Softw. Syst. Model.*, 21(4):1277–1294, aug 2022.

[14] Y. Cao and J. Yang. Towards making systems forget with machine unlearning. In *2015 IEEE Symposium on Security and Privacy*, pages 463–480, 2015.

[15] M. Casimiro, P. Romano, D. Garlan, G. A. Moreno, E. Kang, and M. Klein. Self-adaptation for machine learning based systems. In *ECSA (Companion)*, 2021.

[16] M. Casimiro, P. Romano, D. Garlan, and L. Rodrigues. Towards a framework for adapting machine learning components. In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 131–140, 2022.

[17] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Di Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. *Software Engineering for Self-Adaptive Systems: A Research Roadmap*, pages 1–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[18] S.-W. Cheng, D. Garlan, and B. Schmerl. Making self-adaptation an engineering reality. pages 158–173, 01 2005.

[19] S.-W. Cheng, V. Poladian, D. Garlan, and B. Schmerl. Improving architecture-based self-adaptation through resource prediction. 01 2009.

[20] J. Cleland-Huang, A. Agrawal, M. Vierhauser, M. Murphy, and M. Prieto. Extending mape-k to support human-machine teaming. In *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '22, page 120–131, New York, NY, USA, 2022. Association for Computing Machinery.

[21] J. Cámara, P. Correia, R. de Lemos, D. Garlan, P. Gomes, B. Schmerl, and R. Ventura. Evolving an adaptive industrial software system to use architecture-based self-adaptation. In *2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 13–22, 2013.

[22] J. Cámara, H. Muccini, and K. Vaidhyanathan. Quantitative verification-aided machine learning: A tandem approach for architecting self-adaptive iot systems. In *2020 IEEE International Conference on Software Architecture (ICSA)*, pages 11–22, 2020.

[23] W. D, P. H, G. T, K. R, and M. L. The mape-k architecture for self-adaptation systems. *Software Engineering for Adaptive and Pervasive Systems*, 1(1):15–29, 2006.

[24] R. de Lemos, D. Garlan, C. Ghezzi, H. Giese, J. Andersson, M. Litoiu, B. Schmerl, D. Weyns, L. Baresi, N. Bencomo, Y. Brun, J. Cámara, R. Calinescu, M. Cohen, A. Gorla, V. Grassi, L. Grunske, P. Inverardi, J.-M. Jézéquel, and F. Zambonelli. *Software Engineering for Self-Adaptive Systems: Research Challenges in the Provision of Assurances*, pages 1–29. 09 2017.

[25] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, et al. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, pages 1–32. Springer, 2013.

[26] S. Deng, S. Li, K. Xie, W. Song, X. Liao, A. Hao, and H. Qin. A global-local self-adaptive network for drone-view object detection. *IEEE Transactions on Image Processing*, 30:1556–1569, 2021.

[27] B. J. Erickson, P. Korfiatis, Z. Akkus, and T. L. Kline. Machine learning for medical imaging. *radiographics*, 37(2):505–515, 2017.

[28] K. Ervasti. A survey on network measurement: Concepts, techniques, and tools. *University of Helsinki*, 2016.

[29] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, 134:75–88, 2019.

[30] I. Gerostathopoulos, C. Raibulet, and P. Lago. Expressing the adaptation intent as a sustainability goal. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER '22, 2022.

[31] O. Gheibi and D. Weyns. Lifelong self-adaptation: self-adaptation meets lifelong machine learning. In *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '22. ACM, May 2022.

[32] O. Gheibi, D. Weyns, and F. Quin. Applying machine learning in self-adaptive systems: A systematic literature review. *ACM Trans. Auton. Adapt. Syst.*, 15(3), aug 2021.

[33] O. Gheibi, D. Weyns, and F. Quin. On the impact of applying machine learning in the decision-making of self-adaptive systems. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, May 2021.

[34] H. M. Gomes, J. Read, A. Bifet, J. P. Barddal, and J. a. Gama. Machine learning for streaming data: state of the art, challenges, and opportunities. *SIGKDD Explor. Newsl.*, 21(2):6–22, nov 2019.

[35] K. Goyal and J. Singhai. Texture-based self-adaptive moving object detection technique for complex scenes. *Computers & Electrical Engineering*, 70:275–283, 2018.

[36] J.-B. Hou, X. Zhu, and X.-C. Yin. Self-adaptive aspect ratio anchor for oriented object detection in remote sensing images. *Remote Sensing*, 13(7), 2021.

[37] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. In *2017 IEEE*

Conference on Computer Vision and Pattern Recognition (CVPR), pages 3296–3297, Los Alamitos, CA, USA, jul 2017. IEEE Computer Society.

[38] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58, 2011.

[39] M. U. Iftikhar, G. S. Ramachandran, P. Bollansée, D. Weyns, and D. Hughes. Deltaiot: A self-adaptive internet of things exemplar. In *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 76–82, 2017.

[40] H. Järvenpää, P. Lago, J. Bogner, G. Lewis, H. Muccini, and I. Ozkaya. A synthesis of green architectural tactics for ml-enabled systems. *arXiv preprint arXiv:2312.09610*, 2023.

[41] G. Jocher. Yolov5 by ultralytics, May 2020.

[42] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[43] C. Kinneer, Z. Coker, J. Wang, D. Garlan, and C. Le Goues. Managing uncertainty in self-adaptive systems with plan reuse and stochastic search. In *2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 40–50, 2018.

[44] C. Kinneer, R. Tonder, D. Garlan, and C. Goues. Building reusable repertoires for stochastic self-* planners. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 222–231, Los Alamitos, CA, USA, aug 2020. IEEE Computer Society.

[45] K. Kotar and R. Mottaghi. Interactron: Embodied adaptive object detection. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14840–14849, Los Alamitos, CA, USA, jun 2022. IEEE Computer Society.

[46] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive Mob. Comput.*, 17(PB):184–206, feb 2015.

[47] S. Kulkarni, A. Marda, and K. Vaidhyanathan. Towards self-adaptive machine learning-enabled systems through qos-aware model switching. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1721–1725, 2023.

[48] A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

[49] G. A. Lewis, I. Ozkaya, and X. Xu. Software architecture challenges for ml systems. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 634–638. IEEE, 2021.

[50] N. Li, J. Cámara, D. Garlan, B. Schmerl, and Z. Jin. Hey! preparing humans to do tasks in self-adaptive systems. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 48–58, 2021.

[51] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context, 2015.

[52] F. D. Macías-Escrivá, R. Haber, R. del Toro, and V. Hernandez. Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 40(18):7267–7279, 2013.

[53] A. Marda, S. Kulkarni, and K. Vaidhyanathan. Switch: An exemplar for evaluating self-adaptive ml-enabled systems, 2024.

[54] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A. M. Vollmer, and S. Wagner. Software engineering for ai-based systems: A survey. *ACM Trans. Softw. Eng. Methodol.*, 31(2), 2022.

[55] Y. Mehta, R. Xu, B. Lim, J. Wu, and J. Gao. A review for green energy machine learning and ai services. *Energies*, 16(15), 2023.

[56] N. C. Mendonça, D. Garlan, B. Schmerl, and J. Cámara. Generality vs. reusability in architecture-based self-adaptation: the case for self-adaptive microservices. In *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, ECSA '18, New York, NY, USA, 2018. Association for Computing Machinery.

[57] B. Miller, A. Kantchelian, M. C. Tschantz, S. Afroz, R. Bachwani, R. Faizullabhoy, L. Huang, V. Shankar, T. Wu, G. Yiu, et al. Reviewer integration and performance measurement for malware detection. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*, pages 122–141. Springer, 2016.

[58] F. A. Moghaddam, P. Lago, and I. C. Ban. Self-adaptation approaches for energy efficiency: a systematic literature review. In *Proceedings of the 6th International Workshop on Green and Sustainable Software*, GREENS '18. Association for Computing Machinery, 2018.

[59] F. A. Moghaddam, P. Lago, and I. C. Ban. Self-adaptation approaches for energy efficiency: a systematic literature review. In *Proceedings of the 6th International Workshop on Green and Sustainable Software*, New York, NY, USA, 2018. Association for Computing Machinery.

[60] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl. Flexible and efficient decision-making for proactive latency-aware self-adaptation. *ACM Trans. Auton. Adapt. Syst.*, 13(1), apr 2018.

[61] G. A. Moreno, J. Cámara, D. Garlan, and M. Klein. Uncertainty reduction in self-adaptive systems. In *2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 51–57, 2018.

[62] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl. Efficient decision-making under uncertainty for proactive self-adaptation. In *2016 IEEE International Conference on Autonomic Computing (ICAC)*, pages 147–156, 2016.

[63] G. A. Moreno, B. Schmerl, and D. Garlan. Swim: An exemplar for evaluation and comparison of self-adaptation approaches for web applications. In *2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 137–143, 2018.

[64] H. Muccini and K. Vaidhyanathan. Software architecture for ml-based systems: What exists and what lies ahead. In *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*, pages 121–128. IEEE, 2021.

[65] F. Pinto, M. O. P. Sampaio, and P. Bizarro. Automatic model monitoring for data streams, 2019.

[66] F. Quin, D. Weyns, and O. Gheibi. Decentralized self-adaptive systems: A mapping study. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, May 2021.

[67] S. Rabanser, S. Günnemann, and Z. C. Lipton. Failing loudly: An empirical study of methods for detecting dataset shift, 2019.

[68] RangiLyu. Nanodet-plus: Super fast and high accuracy lightweight anchor-free object detection model. `https://github.com/RangiLyu/nanodet`, 2021.

[69] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[70] S. Reichhuber and S. Tomforde. Active reinforcement learning – a roadmap towards curious classifier systems for self-adaptation, 2022.

[71] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner. Survey and benchmarking of machine learning accelerators. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, Sept. 2019.

[72] P. Rodriguez, C. Spanner, and E. Biersack. Analysis of web caching architectures: hierarchical and distributed caching. *Networking, IEEE/ACM Transactions on*, 9:404 – 418, 09 2001.

[73] B. Schmerl, J. Cámara, J. Gennari, D. Garlan, P. Casanova, G. A. Moreno, T. J. Glazier, and J. M. Barnes. Architecture-based self-protection: composing and reasoning about denial-of-service mitigations. In *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security*, HotSoS '14, New York, NY, USA, 2014. Association for Computing Machinery.

[74] Q. Song, C. Wang, Z. Jiang, Y. Wang, Y. Tai, C. Wang, J. Li, F. Huang, and Y. Wu. Rethinking counting and localization in crowds: A purely point-based framework. 2021.

[75] E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[76] M. Tedla, S. Kulkarni, and K. Vaidhyanathan. Ecomls: A self-adaptation approach for architecting green ml-enabled systems. Mar. 2024.

[77] A. Tundo, M. Mobilio, S. Ilager, I. Brandic, E. Bartocci, and L. Mariani. An energy-aware approach to design self-adaptive ai-based applications on the edge. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Computer Society, 2023.

[78] E. Umargono, J. E. Suseno, and S. Gunawan. K-means clustering optimization using the elbow method and early centroid determination based-on mean and median. In *Proceedings of the International Conferences on*

Information System and Technology, pages 234–240. SCITEPRESS—Science and Technology Publications Setubal, Portugal, 2019.

[79] K. Vaidhyanathan et al. Data-driven self-adaptive architecting using machine learning. 2021.

[80] R. Verdecchia, L. Cruz, J. Sallou, M. Lin, J. Wickenden, and E. Hotellier. Data-centric green ai an exploratory empirical study. In *2022 International Conference on ICT for Sustainability (ICT4S)*, pages 35–45, Los Alamitos, CA, USA, 2022. IEEE Computer Society.

[81] R. Verdecchia, J. Sallou, and L. Cruz. A systematic review of green ai. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, page e1507, 2023.

[82] D. Weyns. Software engineering of self-adaptive systems. *Handbook of software engineering*, pages 399–443, 2019.

[83] D. Weyns. *An Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective*. 02 2021.

[84] D. Weyns and R. Calinescu. Tele assistance: A self-adaptive service-based system exemplar. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 88–92, 2015.

[85] D. Weyns, I. Gerostathopoulos, N. Abbas, J. Andersson, S. Biffl, P. Brada, T. Bures, A. Di Salle, M. Galster, P. Lago, G. Lewis, M. Litoiu, A. Musil, J. Musil, P. Patros, and P. Pelliccione. Self-adaptation in industry: A survey. *ACM Trans. Auton. Adapt. Syst.*, 18(2), may 2023.

[86] R. Wohlrab and D. Garlan. A negotiation support system for defining utility functions for multi-stakeholder self-adaptive systems. *Requirements Engineering*, 01 2022.

[87] T. Wong, M. Wagner, and C. Treude. Self-adaptive systems: A systematic literature review across categories and domains. *Information and Software Technology*, 148:106934, 2022.

[88] Y. Wu, E. Dobriban, and S. B. Davidson. Deltagrad: Rapid retraining of machine learning models, 2020.

[89] J. Wuttke, Y. Brun, A. Gorla, and J. Ramaswamy. Traffic routing for evaluating self-adaptation. In *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 27–32. IEEE, 2012.

[90] T. Yigitcanlar, R. Mehmood, and J. M. Corchado. Green artificial intelligence: Towards an efficient, sustainable and equitable technology for smart cities and futures. *Sustainability*, 13(16), 2021.

[91] C. Zhang, Z. Li, J. Liu, P. Peng, Q. Ye, S. Lu, T. Huang, and Y. Tian. Self-guided adaptation: Progressive representation alignment for domain adaptive object detection. *IEEE Transactions on Multimedia*, PP:1–1, 05 2021.

[92] J. J. Zhang, S. Elnikety, S. Zarar, A. Gupta, and S. Garg. Model-switching: dealing with fluctuating workloads in machine-learning-as-a-service systems. In *Proceedings of the 12th USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'20, USA, 2020. USENIX Association.

[93] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye. Object detection in 20 years: A survey. *Proceedings of the IEEE*, 2023.