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

ABSTRACT

Addressing runtime uncertainties in Machine Learning-Enabled Systems (MLS) is crucial for maintaining Quality of Service (QoS). The Machine Learning Model Balancer is a concept that addresses these uncertainties by facilitating dynamic ML model switching, showing promise in improving QoS in MLS. Leveraging this concept, this paper introduces SWITCH, an exemplar developed to enhance self-adaptive capabilities in such systems through dynamic model switching in runtime. SWITCH is designed as a comprehensive web service, catering to a broad range of ML scenarios, with its implementation demonstrated through an object detection use case. SWITCH provides researchers a flexible platform to apply and evaluate their ML model switching strategies, aiming to enhance QoS in MLS. SWITCH features advanced input handling, real-time data processing, and logging for adaptation metrics. With its interactive realtime dashboard, SWITCH offers researchers a user-friendly interface for experiment management and system observability for MLS. This paper details SWITCH's architecture, self-adaptation strategies through ML model switching, and its empirical validation through case study, illustrating its potential to improve QoS in MLS. By enabling a hands-on approach to study adaptive behaviors in ML systems, SWITCH contributes a valuable tool to the SEAMS community for research into self-adaptive mechanisms and their practical applications.

KEYWORDS

Self-Adaptation, Self Adaptive Systems, Machine Learning-Enabled Systems, Exemplar, Quality of Service, Web Service, ML Model Balancer, Object Detection

ACM Reference Format:

. 2023. SWITCH: An Exemplar for Evaluating Self-Adaptive ML-Enabled Systems. In *Proceedings of SEAMS Conference (SEAMS '24).* ACM, Lisbon, Portugal, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Machine Learning-Enabled Systems (MLS), such as Google Bard [5] and ChatGPT [31], are increasingly prevalent, offering a diverse array of services powered by advancements in AI. However, these systems frequently encounter runtime uncertainties due to environmental factors, changing conditions, and complexities in their software architecture [7]. This evolving AI landscape in MLS underscores the necessity for adaptive software architecture [30] to effectively manage these uncertainties [17]. Traditionally, research in self-adaptive systems (SAS) have focused on non-ML-based systems [40], implementing tactics to modify software architecture or service quality in response to environmental shifts [17]. Nonetheless, recent progress underlines the importance of integrating SAS principles with AI, particularly within MLS [6]. The AdaMLS paper [25] introduces the Machine Learning Model Balancer concept, advocating for dynamic switching between ML models during runtime to manage uncertainties and optimize Quality of Service (QoS). This approach demonstrates how effectively switching between different ML models in runtime in response to operational demands can enhance system performance, recognizing the potential of self-adaptation in MLS. Despite AdaMLS's demonstration of this concept's practicality, the SEAMS community lacks practical tools for researchers to experiment with and refine self-adaptive strategies in MLS [19]. While existing literature and exemplars within the SEAMS community provide insights into self-adaptation, they primarily focus on non-ML systems and scenarios [19].

SWITCH, an exemplar for real-world ML-enabled systems, addresses this gap. It is showcased through object detection, a field that epitomizes significant ML advancements. SWITCH not only enables runtime ML model switching but also offers a comprehensive platform for handling varying loads, data drifts, and other typical uncertainties in MLS. Key features include input handling, real-time data processing, and a user-friendly dashboard, facilitating effective monitoring and experimentation in real-world scenarios. SWITCH is used in validating the AdaMLS, self-adaptation approach, illustrating its capacity to enhance QoS in MLS. Thus, SWITCH stands as a valuable tool for the self-adaptation and MLS research community, providing a unique and practical platform where researchers can thoroughly test, analyze, and refine their self-adaptation strategies, ensuring their efficacy and effectiveness before deployment in real-world ML scenarios. This paper details SWITCH's architecture, explores its self-adaptation strategies through ML model switching, and presents its empirical validation. SWITCH exemplar is available at Git Repository[1] [33], entire replication package at [1] and YouTube video of demonstration at : [2] [32].

The paper structure as follows: Section II provides an overview of SWITCH. Section III delves into the architecture and design of SWICTH. Section IV discusses System Usage and Adaptation. Section V presents empirical validation and case study. Section VI focuses technical challenges and solutions, Section VII discuss related work, Section VIII gives future directions and Section IX presents conclusion.

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 Machine Learning-Enabled Systems (MLS), offering a unique

SEAMS '24, April 2024, Lisbon, Portugal

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] https://anonymous.4open.science/r/switch-14E7/README.md
[2] https://youtu.be/ZIDE1v3jxeQ

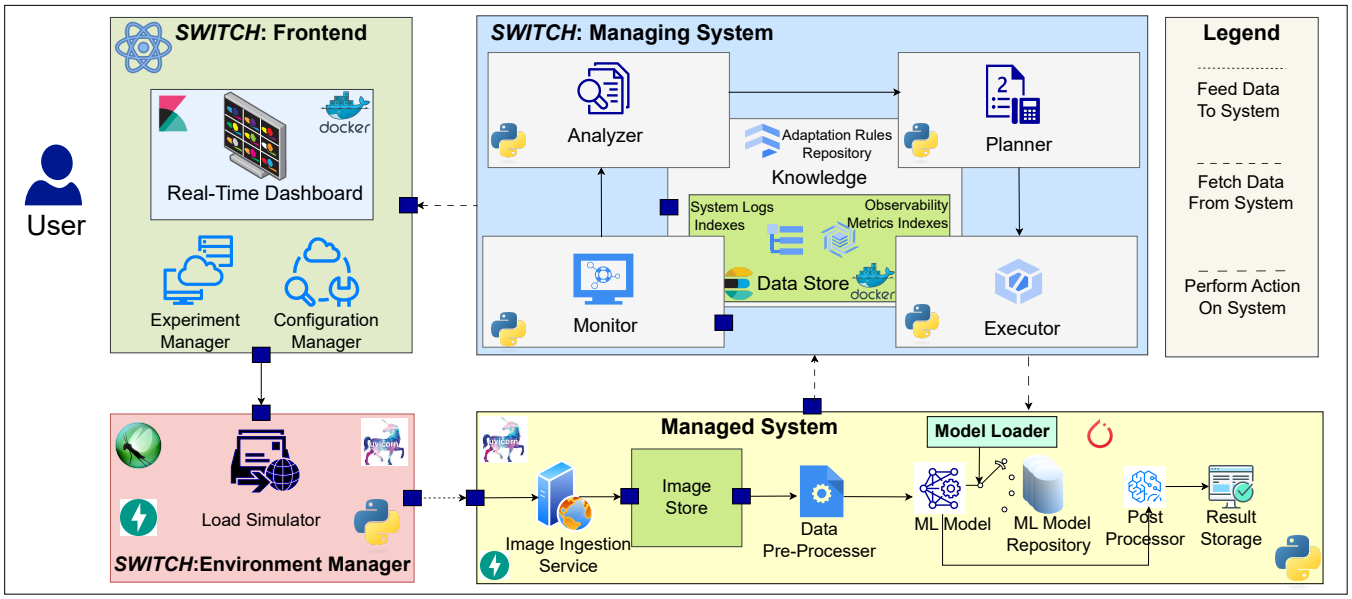

Figure 1: SWITCH : Architecture Diagram

simulation platform for dynamic model switching through software architecture-based self-adaptation through MAPE-K framework [39]. 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 an object detection use case. SWITCH integrates input handling via FastAPI [38], facilitating seamless and efficient user interactions. It employs state-of-the-art YOLOv5u object detection models [22] for real-time data processing, ensuring high accuracy and responsiveness. The system also features systematic logging of observability metrics and system logs in Elasticsearch [15], providing a robust framework for data management & analysis. A standout feature of SWITCH is its interactive, real-time dashboard, implemented using Kibana [16]. 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 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.

3 ARCHITECTURE AND DESIGN

SWITCH comprises core components like Managed System, Frontend, Environment Manager, and Managing System, each integral to the system's adaptability and user interaction.

3.1 Managed System

3.1.1 ***Image Ingestion Service:*** 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 [38], Uvicorn [4] 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 interactions in online ML systems for seamless subsequent processing.

3.1.2 ***Image Store:*** *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 this queue reflects the pending images to be processed as, providing a real-time view of the workload similar to operational queues in MLS deployments.

3.1.3 ***Data Preprocessor:*** *Data Preprocessor* picks the oldest unprocessed image from the *Image Store* for preprocessing. This component's responsibility is to prepare the image data for the object detection model. It opens and loads image data from a byte array into memory for model processing, ensuring data readiness for model inference.

3.1.4 ***Model Loader:*** *Model Loader* in SWITCH is a dynamic component that manages the *ML Model* in use as shown in Figure 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.

3.1.5 ***Model Repository:*** *Model Repository* of SWITCH houses YOLOv5nu, YOLOv5su, YOLOv5mu, YOLOv5lu & YOLOv5xu preloaded models of the YOLOv5u algorithm provided by Ultralytics [22], a state-of-the-art object detection system renowned for its accuracy and efficiency. YOLOv5u models, developed using the PyTorch framework [35] and trained on the COCO dataset [26], 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.

3.1.6 ***ML Model.*** In SWITCH, the *'ML Model'* refers to the currently active YOLOv5 model that is 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.

3.1.7 ***Post Processor:*** *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.

3.1.8 ***Result Storage:*** *Result Storage* is a temporary storage which manages processed data flow into the Elasticsearch-based [15] *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.

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.

3.2.1 ***Experiment Manager & Configuration Manager:*** Built with React [3], 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 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 [25] or Naive [25], 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 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.

3.2.2 ***Real-Time Dashboard:*** *Real-Time Dashboard*, developed using Kibana [16], is a key component of SWITCH's user interface. Integrated as an iframe within the React application and runs in docker container, it provides runtime visualization access to the latest system performance metrics, sourced directly from the Elasticsearch-based *Data Store* within the knowledge of the managing system through REST API endpoints. Kibana's powerful data visualization capabilities allow users to explore a wide array of metrics in an engaging and interactive manner. 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 and provide critical insights into the system's performance and adaptive strategies in run-time.

3.3 SWITCH: Environment Manager

SWITCH's Environment Manager features a *Load Simulator*, essential for replicating real-world API traffic showing load uncertainty in 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 [2], to emulate user behavior and manage incoming image data distribution. Locust's Python script simulate user interactions, 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 [36], 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 important for practitioners to analyze and enhance MLS systems' adaptive capabilities by assessing impact of their adaptation strategies in practical environments.

3.4 SWITCH: Managing System

3.4.1 ***Knowledge:*** *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 docker container. It continually receives data from the Managed System via REST API and stores into 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 efficient 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 system's performance in realtime through REST API.

3.4.2 ***Self-Adaptation Through MAPE-K Framework:*** Switch's self-adaptive capabilities are primarily demonstrated using the MAPE-K (Monitor, Analyze, Plan, Execute - Knowledge) framework, which is a standard approach for implementing self-adaptive systems. -***Monitor:*** Retrieves metrics from Knowledge using API calls for real-time system monitoring. - ***Analyzer:*** Analyzes the monitored data to assess whether adaptation is necessary. - ***Planner:*** Develops strategies based on the analysis for potential adaptations. - ***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 1 lists the key API endpoints that users can leverage to programmatically interact with SWITCH. These endpoints allow for operations for real-time adaptation and monitoring.

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

Table 1: API endpoints and their descriptions.

4 SYSTEM USAGE & ADAPTATION

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 [14] 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, a interactive web interface as shown in figure 2 and preloads all ML models in 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 Home Page to start the experiment: **1. 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. **2. Interarrival Rate File:** Users upload a .csv file for interarrival rates, allowing them to test the system's performance under varied real-world conditions. **3. Experiment ID:** Users assign an ID to their experiment, under which all related logs and metrics are organized and stored. **4. Select Self-Adaptation Strategy:** Through a dropdown 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. The user-centric design of SWITCH, combined with its operational flow, facilitates an immersive experience for exploring, experimenting and analyzing self adaptation strategies using model switching with MLS scenarios, mimicking real-world applications.

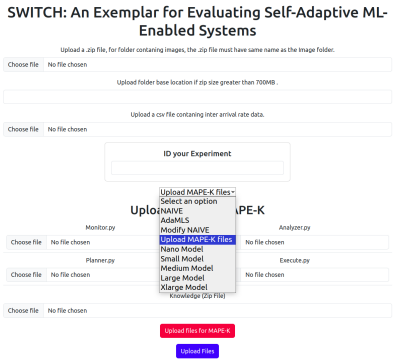

Figure 2: SWITCH User Interface : Home Page

4.2 Adaptation Strategies

Switch incorporates a range of self-adaptation strategies, notably the NAIVE and AdaMLS approaches, each uniquely enhancing the system's adaptability as explained in [25].

Single Model Strategies: These strategies 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 [25]. SWITCH provides AdaMLS implementation, enabling users to experiment with this advanced adaptation strategy.

Custom MAPE-K Strategies: SWITCH also supports the integration of custom self-adaptation strategies. Users can implement their own logic in python and seamlessly integrate it with the Managed System using the provided API guidelines (as shown in Listing 1 for Monitor as done through Elasticsearch and Listing 2 for executor). This feature allows users to explore and test their adaptation strategies, such as energy-saving approaches or context-aware strategies, within the MAPE-K framework.

In conclusion, SWITCH's diverse array of self-adaptation strategies makes it a valuable tool for researchers and practitioners aiming to test and evaluate different model switching strategies for various

objectives, like enhancing service quality or implementing context-aware adaptations in MLS. The following section presents results from running SWITCH with the AdaMLS approach, showcasing its efficacy in real-world scenarios.

Listing 1: Monitoring and Extracting System Metrics

```
# Fetching Past Metrics Averages from Elasticsearch
#Detailed listing is provided in git-repo
def fetch_past_n_metrics_average():
    fields = ["model_processing_time", "detection_boxes", "confidence"]
    doc_count = es.count(index=index_name)["count"]
    num_docs_to_fetch = min(num_documents, doc_count)
    query = {"size": num_docs_to_fetch, "sort": [{"log_id": {"order": "desc"}}]}
    response = es.search(index=index_name, body=query)
    field_values = {field: [] for field in fields}
    for hit in response["hits"]["hits"]:
        for field in fields:
            try:
                field_values[field].append(float(hit["_source"][field]))
            except ValueError:
                pass
    mean_values = {field: sum(field_values[field]) / len(field_values[field]) if field_values[field] else 0 for field in
        ↪ fields}
    return [mean_values[field] for field in fields]

# Extracting Current Model and Input Rate
def extract_metric(file_name):
    df = pd.read_csv(file_name, header=None)
    return df.to_numpy()[0][0]

monitor_dict = {"model": extract_metric("../model.csv"), "Input_rate": extract_metric("../monitor.csv")}
```

Listing 2: Model Switching Execution

```
def switch_model(model_name):
    with open("../model.csv", "w") as f:
        f.write(model_name)

def perform_action(act):
    model_names = ["yolov5nu", "yolov5su", "yolov5mu", "yolov5lu", "yolov5xu"]
    if 1 <= act <= 5:
        switch_model(model_names[act - 1])
    print("Adaptation completed.")
```

5 EMPIRICAL EVALUATION

Switch employs YOLOv5u models [22] for a wide range of object detection scenarios. These models are renowned for their accuracy and efficiency and are trained on the COCO dataset [26], which includes 80 object categories.

Case Study:- *1. General Detection:* Utilizing 10,000 images from the COCO 2017 Unlabelled dataset (1.6 GB) [26], Switch handles a wide variety of 80 categories, showcasing its capacity to deal with diverse general object detection tasks. - *2. Crowd Detection:* For crowd detection, Switch processes a subset of the CrowdHuman dataset [37] (3.3GB) containing 5,018 images. This dataset is particularly challenging due to the high density of human figures, emphasizing the system's capability in crowded environments. - *3. Traffic Detection:* Traffic object detection is tested using 1,000 images from the CityScapes Left 8-bit dataset [11] (2.3GB), focusing on detecting cars, buses, and trucks. This dataset highlights the system's applicability in urban traffic scenarios.

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 {...}*

5.1 Evaluation using AdaMLS Approach

The AdaMLS approach from [25] was directly applied in SWITCH, with tests on a 12th Gen Intel(R) Core(TM) i5-12500H -12 CORE system, results in Table 1. Load conditions were simulated using scaled FIFA98 logs [36]. Note: AdaMLS's rules, based on a different setup, can be recalibrated on users' systems before applying in SWITCH. For result interpretation, see AdaMLS paper [25]. The results in

Table 2: Performance Metrics and Model-wise Processing for General Object Detection

Performance Metrics across Different Scenarios			
Scenario	**General Object Detection**	**Vehicle Counting**	**Crowd Counting**
Total Images Processed	10000	1000	5018
Average Confidence	0.697711	0.670895	0.609580
Average CPU Consumption	19.962980	25.301000	19.148426
Total Objects Detected	47026	9736	60766
Average Model Processing Time (s)	0.032629	0.096568	0.047738
Average Image Processing Time (s)	0.251617	84.635964	50.005430
Average Utility	0.439868	-41.482587	-24.239873
Model-wise Processing for General Object Detection			
Model	**Total Images Processed**	**Avg Model Proc Time**	**Avg Image Proc Time**
yolov5mu	1244	0.025445	0.263286
yolov5nu	2653	0.010868	0.290487
yolov5su	4408	0.014886	0.275654
yolov5lu	1646	0.037947	0.244223
yolov5xu	49	0.066240	0.266817

Table 2 and Figure 3 reveal that under high loads, faster models, often preferred by AdaMLS, face longer wait times due to image processing. This suggests that as load increases, even quicker models can slow down, emphasizing the need for adaptable MLS strategies. These findings also point to the varying performance under different conditions. For instance, the noticeable drop in accuracy in certain scenarios highlights how data complexity and system uncertainties can impact MLS. Our tests, using FIFA98 logs [36], show the importance of SWITCH in managing these challenges and enhancing MLS adaptability.

SWITCH exemplar is available at Git Repository[3] [33], entire replication package, results and data-sets at [1] and YouTube video of demonstration at : [4] [32].

Effectiveness of SWITCH: The initial preloading of 5 YOLOv5u models in Switch consumes between 21.3 to 34.8 CPU, with a total load time of approximately 0.25 seconds and an energy consumption of about 10.56 joules (measured using pyrapl). Model switching in Switch is executed efficiently, taking around 100 microseconds. **Real-time Dashboard Insights:** Figure 4 shows Switch's dashboard with real-time metrics like confidence scores, models used,

[3]https://anonymous.4open.science/r/switch-14E7/README.md
[4]https://youtu.be/ZIDE1v3jxeQ

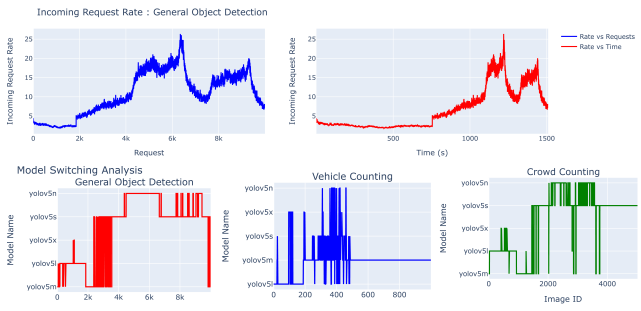

Figure 3: Load and Model Switching Graphs

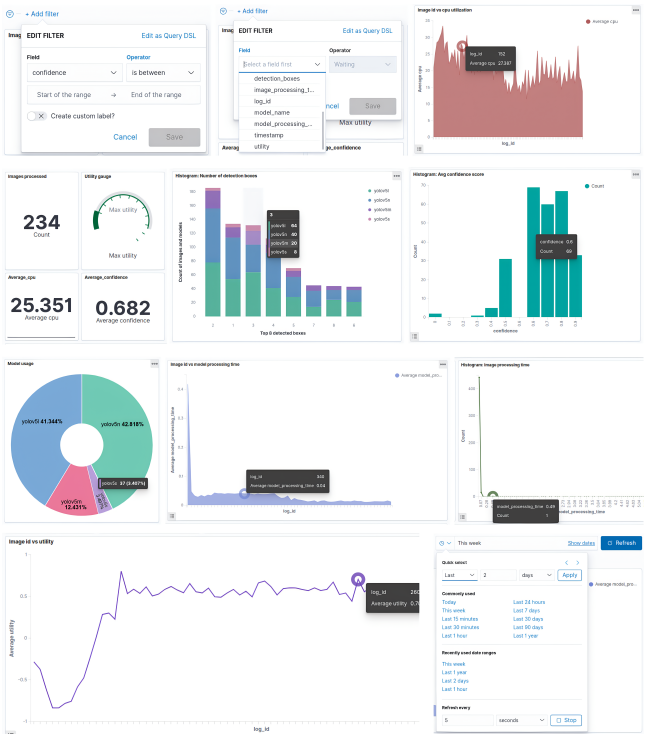

Figure 4: SWITCH : Runtime Dashboard

and image count. It allows users to apply filters and set refresh rates for detailed analysis. These insights underscore the importance of dynamic adaptation in MLS amid various uncertainties. Switch's model switching capability makes it a vital tool for practical self-adaptation strategy applications in MLS.

6 TECHNICAL CHALLENGES & SOLUTIONS

Integration of Diverse ML Models: To accommodate different YOLOv5u models, SWITCH preloads and organizes them in a dictionary by model name, enabling dynamic model switching based on real-time performance.

Efficient Real-Time Data Handling: Managing high-volume image data streams was streamlined using local storage for immediate processing and Elasticsearch for dashboard updates. This ensures efficient real-time data handling, visualization, and storage of logs and metrics for user access.

Interactive Dashboard Development: User-friendly and interactive dashboard was achieved by integrating Kibana & Elasticsearch, enhancing real-time data visualization & observability.

Platform Compatibility: Addressing compatibility issues for non-Linux systems, SWITCH is optimized for Linux environments. Users are recommended to use Linux-based systems or set up a Linux virtual environment for optimal performance.

These focused solutions collectively enhance the robustness, adaptability, and utility of SWITCH, making it a valuable tool in self-adaptive MLS research and applications.

7 RELATED WORK

Self-adaptive systems (SAS) traditionally focused on managing systems without embedded ML models [40]. These systems utilized tactics like architectural changes [17] or service quality adjustments [12] [29], in response to environmental changes [24]. However, the integration of ML components into managed systems introduces new challenges, especially regarding the variability of properties such as accuracy [9] [10] and quality of service [34] [8] [27]. The concept of self-adaptation in ML was initially introduced in [6], [30]. Further discussions and explorations in this direction are highlighted in [9] [10]. The field has seen approaches like AdaMLS improve Quality of Service (QoS) in ML-enabled systems through adaptive strategies [25].

Additionally, recent advancements in object detection, primarily focused on individual model enhancements [13, 18, 20, 21, 23, 41], have overlooked broader system adaptability aspects. Despite these developments, a review of the SEAMS community's repository of exemplars i.e. self-adaptive.org [19] reveals a significant gap in tools specifically for ML-enabled systems. SWITCH fills this gap, offering a platform for dynamic model switching and enhanced user experience, distinguishing itself from existing tools like SWIM [28]. This makes SWITCH a pioneering exemplar in SAS having a self adaptive ML-enabled system, enabling research, exploration & experimentation with self-adaptation strategies in a real-world ML context.

8 CONCLUSION & FUTURE WORK

This paper presents SWITCH, a pioneering tool designed for Machine Learning-Enabled Systems (MLS), with a current focus on runtime model-switching as its core functionality. SWITCH stands out for its principles of self-adaptation and dynamic responsiveness, featuring a scalable architecture, user-friendly interface, and a versatile dashboard that facilitates both experimentation and practical insights into MLS. Through empirical evaluations in object detection scenarios, SWITCH has demonstrated its adaptability and performance, showcasing the efficient management of varying conditions in MLS, underlines its potential and sets the stage for future enhancements.

In addition to its technical contributions, SWITCH serves as a valuable educational and research tool. It provides a platform for researchers, practitioners, and students to explore and understand the intricacies of self-adaptation in MLS. By enabling hands-on experimentation with different scenarios and strategies, SWITCH inspires innovative solutions and approaches in the evolving landscape of self-adaptive systems with deeper understanding of MLS.

While currently focused on model switching, future iterations of SWITCH aim to explore a broader spectrum of MLS tasks, including segmentation and user-driven customization beyond the MAPE-K framework. Additionally, further empirical studies and adaptations to address emerging MLS challenges and uncertainties are anticipated. Ultimately, the insights gained from SWITCH will guide the development of more versatile systems, exploring various adaptation strategies in MLS to meet both present and future demands in the field of self-adaptive systems and machine learning. This positions SWITCH as a significant contribution to the domain, emphasizing the critical role of adaptability in advancing MLS.

REFERENCES

[1] 2023. SWITCH Exemplar. https://drive.google.com/drive/folders/1gWFEp2KxCkYiI_sOSG7urD2CmOQ7Afbv?usp=sharing. Accessed: [Insert access date here].

[2] n.d.. Locust: An open-source load testing tool. https://locust.io/. Accessed: [Insert access date here].

[3] n.d.. React: A JavaScript library for building user interfaces. https://reactjs.org/. Accessed: [Insert access date here].

[4] n.d.. Uvicorn: The lightning-fast ASGI server. https://www.uvicorn.org/. Accessed: [Insert access date here].

[5] Google AI. 2023. Google Bard. https://bard.google.com/

[6] Tomáš Bureš. 2021. Self-Adaptation 2.0. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 262–263. https://doi.org/10.1109/SEAMS51251.2021.00046

[7] Javier Cámara, Javier Troya, Antonio Vallecillo, Nelly Bencomo, Radu Calinescu, Betty H. C. Cheng, David Garlan, and Bradley Schmerl. 2022. The Uncertainty Interaction Problem in Self-Adaptive Systems. *Softw. Syst. Model.* 21, 4 (aug 2022), 1277–1294. https://doi.org/10.1007/s10270-022-01037-6

[8] Yinzhi Cao, Alexander Yu, Andrew Aday, Eric Stahl, Jon Merwine, and Junfeng Yang. 2018. Efficient Repair of Polluted Machine Learning Systems via Causal Unlearning. 735–747. https://doi.org/10.1145/3196494.3196517

[9] Maria Casimiro, Paolo Romano, David Garlan, Gabriel A Moreno, Eunsuk Kang, and Mark Klein. 2021. Self-Adaptation for Machine Learning Based Systems.. In *ECSA (Companion)*.

[10] Maria Casimiro, Paolo Romano, David Garlan, and Luís Rodrigues. 2022. Towards a Framework for Adapting Machine Learning Components. In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. 131–140. https://doi.org/10.1109/ACSOS55765.2022.00031

[11] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. 2016. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[12] Javier Cámara, Henry Muccini, and Karthik Vaidhyanathan. 2020. Quantitative Verification-Aided Machine Learning: A Tandem Approach for Architecting Self-Adaptive IoT Systems. In *2020 IEEE International Conference on Software Architecture (ICSA)*. 11–22. https://doi.org/10.1109/ICSA47634.2020.00010

[13] Sutao Deng, Shuai Li, Ke Xie, Wenfeng Song, Xiao Liao, Aimin Hao, and Hong Qin. 2021. A Global-Local Self-Adaptive Network for Drone-View Object Detection. *IEEE Transactions on Image Processing* 30 (2021), 1556–1569. https://doi.org/10.1109/TIP.2020.3045636

[14] Docker. n.d.. Docker Desktop: The #1 Containerization Tool for Developers. https://www.docker.com/products/docker-desktop/. Accessed: [Insert access date here].

[15] Elastic. n.d.. Elasticsearch: The Official Distributed Search & Analytics Engine. https://www.elastic.co/elasticsearch. Accessed: [Insert access date here].

[16] Elastic. n.d.. Kibana Guide [8.11]. https://www.elastic.co/guide/en/kibana/current/index.html. Accessed: [Insert access date here].

[17] Omid Gheibi, Danny Weyns, and Federico Quin. 2021. Applying Machine Learning in Self-Adaptive Systems: A Systematic Literature Review. *ACM Trans. Auton. Adapt. Syst.* 15, 3, Article 9 (aug 2021), 37 pages. https://doi.org/10.1145/3469440

[18] Kalpana Goyal and Jyoti Singhai. 2018. Texture-based self-adaptive moving object detection technique for complex scenes. *Computers & Electrical Engineering* 70 (2018), 275–283. https://doi.org/10.1016/j.compeleceng.2016.05.017

[19] Hasso Plattner Institute. n.d.. Exemplars – Software Engineering for Self-Adaptive Systems. https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/. Accessed: [Insert access date here].

[20] Jie-Bo Hou, Xiaobin Zhu, and Xu-Cheng Yin. 2021. Self-Adaptive Aspect Ratio Anchor for Oriented Object Detection in Remote Sensing Images. *Remote Sensing* 13, 7 (2021). https://doi.org/10.3390/rs13071318

[21] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. 2017. Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 3296–3297. https://doi.org/10.1109/CVPR.2017.351

[22] Glenn Jocher. 2020. YOLOv5 by Ultralytics. https://doi.org/10.5281/zenodo.3908559

[23] K. Kotar and R. Mottaghi. 2022. Interactron: Embodied Adaptive Object Detection. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 14840–14849. https://doi.org/10.1109/CVPR52688.2022.01444

[24] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. 2015. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing* 17 (2015), 184–206. https://doi.org/10.1016/j.pmcj.2014.09.009 10 years of Pervasive Computing' In Honor of Chatschik Bisdikian.

[25] Shubham Kulkarni, Arya Marda, and Karthik Vaidhyanathan. 2023. Towards Self-Adaptive Machine Learning-Enabled Systems Through QoS-Aware Model Switching. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1721–1725. https://doi.org/10.1109/ASE56229.2023.00172

[26] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll'a r, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. *CoRR* abs/1405.0312 (2014). arXiv:1405.0312 http://arxiv.org/abs/1405.0312

[27] Pedro Mendes, Maria Casimiro, Paolo Romano, and David Garlan. 2020. TrimTuner: Efficient Optimization of Machine Learning Jobs in the Cloud via Sub-Sampling. In *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 1–8. https://doi.org/10.1109/MASCOTS50786.2020.9285971

[28] Gabriel A. Moreno, Bradley Schmerl, and David Garlan. 2018. 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)*. 137–143.

[29] Henry Muccini and Karthik Vaidhyanathan. 2019. A Machine Learning-Driven Approach for Proactive Decision Making in Adaptive Architectures. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. 242–245. https://doi.org/10.1109/ICSA-C.2019.00050

[30] Henry Muccini and Karthik Vaidhyanathan. 2021. 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)*. 121–128. https://doi.org/10.1109/WAIN52551.2021.00026

[31] OpenAI. 2021. ChatGPT. https://openai.com/research/chatgpt.

[32] Author or Channel Name. Year of Publication. Title of the YouTube Video. https://youtu.be/ZIDE1v3jxeQ. Accessed: [Insert access date here].

[33] Author(s) or Organization. 2023. SWITCH Exemplar. https://anonymous.4open.science/r/switch-14E7/README.md. Accessed: [Insert access date here].

[34] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. 2019. Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/8558cb408c1d76621371888657d2eb1d-Paper.pdf

[35] PyTorch. n.d.. PyTorch: An open source machine learning framework. https://pytorch.org/. Accessed: [Insert access date here].

[36] Pablo Rodriguez, Christian Spanner, and Ernst Biersack. 2001. Analysis of Web caching architectures: hierarchical and distributed caching. *Networking, IEEE/ACM Transactions on* 9 (09 2001), 404 – 418. https://doi.org/10.1109/90.944339

[37] Shuai Shao, Zijian Zhao, Boxun Li, Tete Xiao, Gang Yu, Xiangyu Zhang, and Jian Sun. 2018. CrowdHuman: A Benchmark for Detecting Human in a Crowd. *arXiv preprint arXiv:1805.00123* (2018).

[38] Tiangolo. n.d.. FastAPI. https://fastapi.tiangolo.com/. Accessed: [Insert access date here].

[39] Parunak-H. Gurgencz T. Krikhaar R. & Moorman L. Weyns, D. 2006. The MAPE-K architecture for Self-adaptation systems. *Software Engineering for Adaptive and Pervasive Systems* 1, 1 (2006), 15–29.

[40] Terence Wong, Markus Wagner, and Christoph Treude. 2022. Self-adaptive systems: A systematic literature review across categories and domains. *Information and Software Technology* 148 (2022), 106934. https://doi.org/10.1016/j.infsof.2022.106934

[41] Chong Zhang, Zongxian Li, Jingjing Liu, Peixi Peng, Qixiang Ye, Shijian Lu, Tiejun Huang, and Yonghong Tian. 2021. Self-Guided Adaptation: Progressive Representation Alignment for Domain Adaptive Object Detection. *IEEE Transactions on Multimedia* PP (05 2021), 1–1. https://doi.org/10.1109/TMM.2021.3078141