AUTOMATION AND INTELLIGENCE IN IT OPERATION MANAGEMENT: MACHINE LEARNING FOR CAPACITY PLANNING AND LOAD TESTING OPTIMIZATION

Arthur Vitui

Under the supervision of Dr. Tse-Hsun (Peter) Chen

A Thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Doctor of Philosophy
Concordia University
Montréal, Québec, Canada

April 2025

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATESTUDIES

This is to certify that the thesis prepared

By: Arthur Vitui

Entitled: Automation and Intelligence in ITOM: Machine Learning for Capacity Planning and Load Testing Optimization

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

Doctor of Philosophy (Computer Science)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________________Chair
Dr. Manar Amayri

______________________________External Examiner
Dr. Marin Litoiu

______________________________Arms-Length Examiner
Dr. Ching Yee Suen

______________________________Examiner
Dr. Weiyi Ian Shang

______________________________Examiner
Dr. Diego Costa

______________________________Thesis Supervisor
Dr. Tse-Hsun Peter Chen

Approved by ______________________________
Dr. Sabine Bergler, Graduate Program Director

4/24/2025

____
Dr. Mourad Debbabi, Dean, Gina Cody School of Engineering and Computer Science

Abstract

Automation and Intelligence in IT Operation Management: Machine Learning for Capacity Planning and Load Testing Optimization

Arthur Vitui, Ph.D.
Concordia University, 2025

The increasing complexity and scale of modern IT infrastructures necessitate innovative strategies to maintain efficiency, reliability, and cost effectiveness. Large scale industrial systems require precise capacity planning to manage fluctuating demands, prevent downtime, and operate within optimal cost parameters. However, traditional capacity planning methods often fall short in today's dynamic environments. This dissertation introduces an agentic approach to AIOps (Artificial Intelligence for IT Operations) aimed at enhancing the maintenance and operational stability of large scale systems. Effective capacity planning is essential for stable system operations. Over provisioning leads to resource waste, while under provisioning can cause failures and diminished performance. By utilizing load testing data and advanced machine learning (ML) models, we propose a blueprint process that optimizes system capacity planning. Integrating ML into this process enhances predictive capabilities, enabling proactive resource scaling, reducing costs, and increasing system resilience. A significant challenge in optimizing this process is the inefficiency and time consuming nature of traditional load testing. Existing methodologies often require substantial manual effort and considerable time to simulate large scale workloads. To address this, we propose a framework that streamlines load testing through automation and early stopping rules based on spike detection techniques for system Key Performance Indicators (KPIs). By leveraging a system's ability to predict KPI spikes, we can dynamically adjust capacity as needed. We aim to integrate these processes into tools utilized by LLM (Large Language Model) agents within an AIOps system. These tools will act as intermediaries for monitoring and maintaining large scale systems. This integration will establish a fully managed architecture, where AIOps agents enhance the IT operations team's ability to perform proactive maintenance, respond

to new incidents, autonomously monitor system health, predict potential issues, and implement proactive measures to maintain optimal performance. This dissertation presents a novel approach to enhancing efficiency in large scale systems by combining automation and load testing improvements with machine learning and LLM agents. By developing a comprehensive, scalable framework, this research seeks to reduce operational overhead and establish a new standard for IT system management and load testing practices within the Software Development Life Cycle (SDLC) in industrial settings.

To my beloved wife.

Acknowledgments

I express my deepest gratitude to my supervisor, Dr. Peter Chen, whose guidance, patience, and wisdom have been invaluable throughout the course of this research. His commitment to pushing the boundaries of knowledge while providing me with the support and encouragement I needed at every step is something that I deeply appreciate. This work would not have reached its potential without his insight and steady mentorship.

I also extend my sincere thanks to the members of my dissertation committee: Dr. Marin Litoiu, Dr. Weiyi (Ian) Shang, Dr. Ching Yee Suen, and Dr. Diego Elias Costa. Their thoughtful feedback, constructive criticism, and invaluable suggestions helped refine and strengthen the direction of my work. I am grateful for their time and dedication, which contributed significantly to the quality of this thesis.

I am deeply grateful to my wife, Mihaela, for her unwavering support. Her belief in my abilities, her patience during the long hours, and her constant encouragement provided me with the strength to continue, especially during the most challenging moments of this journey. I am forever grateful for her love and understanding, without which this achievement would not have been possible.

Lastly, I want to acknowledge my loyal companion, Newton. His joyful presence and unconditional affection brought balance and happiness to my life throughout this endeavor. The simple moments spent with him provided much needed respite and reminded me of the importance of taking care of oneself amid rigorous academic pursuits.

To all those who have supported me on this path, your contributions have been integral to this accomplishment. I offer my heartfelt thanks.

Related Publications

In all chapters and related publications of this dissertation, I made contributions including formulating the initial research idea, conducting surveys of related work and research background, implementing the code, performing case studies, analyzing the results, and writing and finalizing the work. My co-author assisted me by reviewing the initial idea, referencing related work, reviewing the writing, and providing feedback.

Earlier versions of the chapters in this dissertation were published and submitted as follows:

1. *MLASP: Machine learning assisted capacity planning: An industrial experience report* (Chapter 3)
   Arthur Vitui, Tse-Hsun (Peter) Chen, Empirical Software Engineering (EMSE), 2021. Volume 26, Articles 87 and 109.

2. *MLOLET - Machine Learning Optimized Load and Endurance Testing: An industrial experience report* (Chapter 4)
   Arthur Vitui, Tse-Hsun (Peter) Chen, IEEE/ACM International Conference on Automated Software Engineering (ASE), 2024. Sacramento, CA, USA.

3. *Empowering AIOps: Leveraging Large Language Models for IT Operations Management*(Chapter 5)
   Arthur Vitui, Tse-Hsun (Peter) Chen, **Under review for** IEEE/ACM International Conference on Automated Software Engineering (ASE), 2025, Seoul, Republic of Korea.

Table of Contents

List of Tables ix

List of Figures xi

1 Introduction 1

1.1 Research Statement 4
1.2 Research Overview 5
1.3 Capacity Planning Challenge 5
1.4 Load Testing Cost Efficiency Challenge 7
1.5 AIOps Challenge 9
1.6 Contributions 11

2 Background and Related Works 13

2.1 Capacity Planning For Large Scale Systems 13
2.2 Traffic Spike Detection and Load Testing Enhancements for Large Scale Systems 15
2.3 Using Large Language Models for IT Operations Management 17
2.4 Evaluation metrics for Predictive Machine Learning Models in Capacity Planning and Load Testing Optimization 18
2.5 Evaluation metrics for Large Language Models 19

I Capacity Planning For Large Scale Systems 21

3 Enhancing the Effectiveness of Capacity Planning for Large Scale Systems 22

3.1 Motivation 23

3.2 Case Study Setup 25
3.3 Case Study Results 39
3.4 Discussion and Implication of Our Findings 49
3.5 Threats to Validity 53

II Load and Endurance Testing For Large Scale Systems 55

4 Enhancing the Effectiveness of Load Testing for Large Scale Systems 56
4.1 Motivation 58
4.2 Case Study Setup 59
4.3 Case Study Results 70
4.4 Discussion 76
4.5 Threats to Validity 79

III Large Language Model Agents for IT Operations Management 81

5 Using Large Language Models for adding AIOps Capabilities to Large Scale Systems 82
5.1 Motivation 83
5.2 Case Study Setup 83
5.3 Case Study Results 94
5.4 Discussion 112
5.5 Threats to Validity 117

6 Conclusion 119
6.1 Contributions 120
6.2 Future Research Directions 121

List of Tables

1 Model evaluation results when using the entire data with separated training and testing. We excluded the MAE, MSE and RMSE for the enterprise systems due to NDA. 41
2 Mean of model evaluation results when using the entire data with a 10-fold cross-validation. We excluded the MAE, MSE and RMSE for the enterprise systems due to NDA. 42
3 Model evaluation results when using **a subset of the entire data** with separated training and testing. We excluded the MAE, MSE and RMSE for the enterprise systems due to NDA. 47
4 Mean of model evaluation results when using **a subset of the entire data** with separated training and testing and leave-one-out cross-validation technique on the training dataset. We excluded the MAE, MSE and RMSE for the enterprise systems due to NDA. 50
5 Enterprise System - One step forecasting results (MLOLET-RQ1). . . 72
6 Open Source System - One step forecasting results (MLOLET-RQ1). 72
7 Enterprise System - Forecasting results in an online setting using a previously trained model on new system configurations (MLOLET-RQ2). 74
8 Open Source System - Forecasting results in an online setting using a previously trained model on new system configurations (MLOLET-RQ2). 74
9 Enterprise System - Results of forecasting capabilities for different model architectures (MLOLET-RQ3). 76
10 Open Source System - Results of forecasting capabilities for different model architectures (MLOLET-RQ3). 77

11 Large Language Model Agents for AIOps Evaluation Queries and aasociated tools for responding. Category shows the category of the question, where SR means Simple Reasoning (use at most one tool) and AR means Advanced Reasoning (use multiple tools). 92
12 RQ1 - Summary of Large Language Model Agents solving task accuracy in AIOps Context grouped by type of reasoning: Simple Reasoning, SR, (use at most one tool), and Advanced Reasoning, AR (use multiple tools). 97
13 RQ1 - Detailed Results for Large Language Model Agents solving task accuracy in AIOps Context. 98
14 RQ2 - Summary of Large Language Model Agents solving task average response times (in seconds) in AIOps Context. 102
15 RQ2 - Detailed Results for Large Language Model Agents solving task response times (in seconds) in AIOps Context. 103
16 RQ3 - Summary Average token count (verbosity) of Large Language Model Agents solving tasks in AIOps Context. 108
17 RQ3 - Average token count (verbosity) of Large Language Model Agents solving task in AIOps Context. 110
18 Query Q-24 correctly solved by GPT 4-turbo powered agent. 113

List of Figures

1 MLASP Process with its three stages: (1) Automated Load Testing, (2) Feature Engineering and ML Training, and (3) Model On-Line Serving. 26

2 A high-level overview of our approach and load test execution setup. . 36

3 An example of the predicted v.s. actual throughput for the open-source system from the MLP Neural Net model results built using the complete dataset. 43

4 Distributions of the percentage deviation between the predicted and actual value for the open-source system using the complete dataset with separated training and testing dataset. 44

5 An example of train v.s. validation loss for the open-source system from the MLP Neural Net model results built using a subset of the complete dataset. 45

6 Distributions of the percentage deviation between the predicted and actual value for the open-source system using a subset of data to train the models. 48

7 Feature Importance: Open source system parameters that have the most impact over the load test results. 51

8 MLOLET framework overview. 60

9 Traffic KPI information captured from a test application running under different load and configuration settings. 63

10 MLOLET traffic forecasting process overview. 64

11 MLOLET Models Data windowing. 69

12 Gartner's AIOps platform conceptualization [2]. 84

13 An example workflow for LLM performance evaluation in an AIOps context. 92

Chapter 1

Introduction

Modern large-scale systems are expected to handle a vast number of concurrent requests every day. Any malfunctions or service degradations—resulting from sudden traffic spikes or sustained load pressure—can cause companies to lose millions or even billions of dollars [32]. Therefore, capacity planning and traffic spike detection and prediction are critical components in the quality assurance process of these systems. Consequently, IT Operations teams must proactively ensure that services remain operational by properly sizing the software ecosystem and swiftly detecting and addressing any traffic spikes. Ideally, these measures should be taken before they lead to service disruptions.

Load testing is essential for determining system capacity and is a key part of the software development process. Its main goal is to ensure that the system behaves correctly under load conditions. By simulating real-world usage, load tests collect data during and after the simulation, helping load test engineers and software developers evaluate the system's performance and ability to handle capacity [56].

To address the functional complexity of modern large-scale systems, software developers offer configuration parameters to provide additional flexibility. Load test engineers apply load to the system and analyze its behavior and performance [21]. The idea is to change the values of configuration parameters—such as thread pool sizes, buffer sizes, and so on—without modifying the source code. During the parameter tuning process, load test engineers also need to consider different hardware configurations, both in terms of compute resource availability and horizontal scaling options.

Large-scale systems often consist of multiple components, each of which may have hundreds of configurable parameters and deployment settings [45, 70]. Due to limited project resources and tight deadlines before releases, it is impossible and impractical for load test engineers to evaluate all possible configuration settings during load tests. This constraint prevents them from providing a comprehensive assessment of the system's capacity [42, 43, 45]. As a result, load test engineers typically test only a few combinations of parameter values from the default deployment settings, using an ad hoc selection process [70]. Moreover, the default parameters are only minimally adjusted and rely on the domain knowledge of the load test engineer. Without a thorough understanding of how parameter values affect system performance, this can lead to load-related issues—such as high response times that may cause timeouts or even component crashes—improperly utilized resources, or service level agreement violations [20, 128].

Many studies have proposed various techniques for finding optimal parameter values [9, 42, 43, 45, 101]. However, capacity planning differs from parameter optimization because the latter often focuses on tuning a single instance of a software component under a small load (e.g., a few seconds of load). In contrast, capacity planning considers all the software components in the system to determine the system's capacity using key performance indicators (KPIs), given a load, a set of parameters for each component, and a deployment setting. The goal is to provide customers and product managers with tuning guidelines to meet service level agreements (SLAs) in industrial settings for long-running loads.

Running long load tests presents another challenge in modern software development, where release cycles are only a few weeks long. Executing lengthy load tests becomes expensive—not just due to the significant resources required but also because of the time needed to reserve them. Therefore, it would be more economical to stop failing load tests as early as possible. Additionally, it is important for load test engineers to forecast the system's performance trends based on load test results. This ability would allow them to take appropriate actions to ensure the system's stability in the event of sudden load spikes.

Given the importance and critical nature of the aforementioned technical processes, our research aims to expand the techniques applicable in industrial settings. This expansion seeks to enhance the quality of the resulting software systems and improve the cost and environmental efficiency of using such techniques.

The challenges previously mentioned are encountered daily by companies involved in the development and delivery of large-scale solutions. Thanks to our industrial partner, Ericsson Inc., we had access to such environments to conduct empirical studies and propose new techniques to address these issues. Due to a Non-Disclosure Agreement (NDA) with our partner, we are unable to disclose the complete details of the subject systems. Consequently, we have validated our proposed approaches using open-source software or custom-developed mock software that we have made publicly available. This enables others to reuse our tools in their projects and experiments, ensuring that our experiments can be reproduced.

To address the capacity prediction problem, we propose an approach called MLASP (Machine Learning Assisted Capacity Planning) [114]. This process framework accelerates the load testing process through automation and then uses machine learning algorithms to model the system's KPIs. These models can assist in tuning the configuration parameters of the software ecosystem to achieve the desired KPIs. The MLASP framework offers non-intrusive methods for metrics collection and configuration management. Combined with its automation features, this makes MLASP an easily adaptable and adoptable blueprint for many software systems without requiring code changes to the underlying software.

To solve the problem of failing load tests, we propose an approach called MLOLET (Machine Learning Optimized Load and Endurance Testing) [115]. MLOLET leverages time series analysis and forecasting for spike detection capabilities to predict emerging traffic trends. It then uses a rule based decision algorithm to decide whether a load test is failing.

The management of complex large-scale systems remains a significant challenge, even with the introduction of new frameworks that leverage machine learning (ML) to assist in task resolution. This difficulty arises primarily due to the extensive scripts and configurations required for the overall tuning of such systems. Recent advancements in natural language processing, particularly the emergence of Large Language Models (LLMs), present new opportunities for IT Operations Management (ITOM). By integrating LLMs to develop tool-based agents and assistants, organizations can introduce varying levels of autonomy in AI-driven IT Operations, including tasks such as configuration management for capacity planning. This approach helps ensure that the KPIs of monitored software systems consistently meet desired SLAs. Ultimately, businesses will benefit from more reliable services, efficient operations management, and overall cost savings through this proactive and adaptive system optimization.

1.1 Research Statement

Industrial systems are often deployed on a large scale to handle millions of requests daily. Some of these systems provide real-time, mission-critical services, making it essential to maintain sufficient capacity to manage sudden spikes in demand. However, determining the optimal capacity and dynamically adjusting system parameters to ensure that Service Level Agreements (SLAs) and Key Performance Indicators (KPIs) stay within business-defined thresholds is both challenging and costly, requiring significant time, resources, and financial investment. An under-provisioned system can negatively impact the business by causing substantial revenue loss during outages. Conversely, an over-provisioned system may lead to revenue loss due to unnecessary resource consumption and also increase the environmental footprint by raising carbon emissions. Balancing these factors is crucial for maintaining operational efficiency, ensuring reliability, and minimizing costs. Therefore, effective capacity planning and adaptive system management are critical for the sustainable and profitable operation of large-scale industrial systems. Based on these challenges and prior research, in this thesis we focus further enhancing software development lifecycle (SDLC) tasks as well as ITOM tasks as follows:

> This thesis aims to to improve the practical efficiency of load testing within the software development lifecycle through the use of automation and non-intrusive data collection techniques, as well as machine learning. It leverages machine learning methodologies, combined with Large Language Models (LLMs), to develop actionable AIOps solutions. These solutions are tailored to tackle real-world capacity planning challenges in large-scale systems, fostering more effective resource management and improved operational scalability.

Using non-intrusive data collection techniques is essential to our objective, as it allows our proposed methods to generalize across a wider spectrum of software applications and systems. In the next section, we present the research overview and its objectives that form the focus of this dissertation.

1.2 Research Overview

Overall, this dissertation presents techniques to enhance the efficiency of processes related to both the software development life cycle (SDLC) management of large-scale application ecosystems and their operational aspects. For the SDLC, we focus on improving the load testing process through automation and introducing early stopping techniques for failing load tests using spike detection. For capacity planning in IT operations, we propose a novel framework that begins during the SDLC and leverages load testing data to create models that can predict configurations based on business needs. Critical to our objective is using non-intrusive data collection techniques so that our proposed methods generalize for a larger spectrum of software applications and systems. Lastly, the emergence of Large Language Models (LLMs) has opened the door to a new era of chatbots and assistants that can aid in IT operations management.

This dissertation is divided into three main parts. The first two provide perspectives on the load testing process and its data. In particular, Chapter 3 proposes a new framework for capacity planning using load testing data, while Chapter 4 proposes a novel framework for increasing the efficiency of the load testing process itself. The third part, Chapter 5, provides perspectives in the use of LLM agents in IT operations management of modern software ecosystems.

1.3 Capacity Planning Challenge

As previously stated, load testing is essential for determining system capacity. Its goal is to ensure that the system behaves correctly under load (e.g., simulated real-world usage) and to help load test engineers and developers evaluate system performance and capacity [21].

Due to the complexity of modern large-scale systems and deployment methods, developers often provide configuration parameters for additional flexibility. Load test engineers execute the same load against the system under different settings to analyze its behavior and expected performance [21]. Developers or load test engineers can adjust system performance by changing the values of configuration parameters (e.g., increasing the size of thread pools) without modifying the source code. When tuning parameter values, load test engineers also need to consider different deployment settings (e.g., horizontal scaling by adding more worker nodes) or hardware configurations (e.g., vertical scaling by adding more computational power) that customers employ.

Large-scale software systems may consist of multiple components, each with up to hundreds of configurable parameters and deployment settings [44, 69]. This creates a huge search space for parameter combinations and due to limited resources and tight time constraints before releases, it is impossible and impractical to perform load testing on all possible combinations of configuration parameter values to provide a comprehensive overview of the system's capacity [42, 43, 44]. In practice, load test engineers typically test only a handful of parameter combinations under the default deployment setting, and their selection process remains ad hoc [69]. For example, they often use default configuration parameter values provided by developers and may only make marginal modifications during load tests.

Such configuration tuning is inefficient and largely depends on the load test engineers' domain knowledge of the systems, which may not be up to date as the systems evolve. Failure to understand how parameter values affect system capacity under specific deployment settings can lead to load-related issues (e.g., high response times or even crashes), underutilization of resources, or violations of service level agreements [20, 128].

Parameter configuration tuning for software systems has been a significant area of interest in the research community over the years [9, 20, 42, 43, 45, 70, 101]. However, these studies primarily focus on finding the optimal parameter values in a single setting—that is, on one instance of the tested software application—and mostly involve tuning binary system configuration parameters. Recent advances in neural networks have led to testing some of the earlier state-of-the-art approaches against newer algorithms [45], still within single-instance use cases.

These prior studies mainly focus on techniques that optimize parameter values to maximize performance of software applications. As stated before, although parameter optimization and capacity planning might appear similar, there are significant differences between them. Parameter optimization typically focuses on finding the optimal performance of a single instance of a system (e.g., on one machine) using a relatively small load (e.g., lasting only seconds). In contrast, capacity planning primary focus is to determine the system's capacity—such as key performance indicators (KPIs)—given a specific load, system configuration, and deployment setting. While capacity planning aims to minimize the resource usage as its secondary objective, it does not guarantee that the proposed configuration is the most performant one, as opposed to parameter optimization. The idea here is to have a minimal, potentially redundant, stable system that is able to perform long term under continuous load and produce the desired KPIs and meet the Service Level Agreements (SLA). Additionally, the goal is to provide guidelines so that different settings may be applied when the KPIs and SLAs change. Therefore, capacity planning is usually conducted in a larger test environment that mimics a production deployment and involves running long-duration loads.

Chapter 3 addresses this challenge by proposing a new framework that uses automation, non-intrusive information retrieval and machine learning. The proposed process has been tested in an industrial setting on two large scale enterprise systems, as well as on two open source systems.

1.4 Load Testing Cost Efficiency Challenge

Load and endurance testing are crucial in the software development life cycle and during operations because they ensure that software systems can handle expected and unexpected workloads over time. Running workloads over extended periods of time may cause malfunctions or service degradation resulting from spikes which may cause companies losses in the millions or even billions of dollars [32]. Load testing simulates real-world user demand on the system, helping developers identify performance bottlenecks and optimize resource utilization before deployment. Endurance testing, also known as soak testing, evaluates the system's performance over an extended period to detect issues like memory leaks, resource exhaustion, or performance degradation that may not appear in shorter tests. There are, however, two main challenges when considering the execution of load and endurance testing processes and monitoring the production system's performance. As a first challenge, due to the project time limits, detection of failing load/endurance tests must occur as early as possible in the execution so that the load test engineers may stop their execution, and save resources (time and money). These tests may be failing for various reasons, such as:

- A slowdown in the overall system performance can occur due to bottlenecks or overload in any of the software system's components. For example, the system's throughput might drop below a defined threshold for a specific duration—for instance, if in the last 5 minutes the throughput fell below 50 transactions per second (TPS).

- Intermittent information loss represents a worsening of the previous condition where the system's processing speed has decreased and is also losing some of the incoming requests. For example, in the last five minutes, the throughput dropped below 50 transactions per second (TPS), and/or three percent of the messages were lost.

- Experiencing a malfunction or critical error in one or more components of the software ecosystem—which is a worsening of the previous condition—can ultimately lead to incorrect behavior throughout the entire system. For example, if the database backend becomes unavailable, all processing operations result in errors.

We can see that all of the above situations have in common the fact that they are measured by KPI values over time and the changes in these values hold the key to determine whether a load/endurance test is executing correctly or not. In other words, we are talking about spike detection in time series data. This also means that, during load testing, different types of spikes can occur within the software ecosystem, potentially leading to adverse effects. This is a serious challenge that needs to be addressed.

The second challenge, is the one faced this time by the operations team. They need to know a load increase might occur so they can take preventive measures to maintain the software system's agreed-upon KPIs and SLAs.

We encountered both these challenges when working with our industrial partner. In order to address them, we proposed a novel load testing process and framework. Similar to the capacity planning challenge, we rely on automation, non-intrusive data collection mechanisms and machine learning used in an online setting for the building blocks of the suggested load/endurance testing process improvements. Chapter 4 presents the details and specifics of the suggested improvements.

1.5 AIOps Challenge

The integration of Artificial Intelligence (AI) into IT Operations Management (ITOM), often referred to as AIOps [2], holds significant promise for automating processes, improving efficiency, and enhancing decision-making. However, implementing AI in IT operations presents several challenges that organizations must address to realize its full potential. Below we present some of the most important challenges:

1. **Data Quality and Diversity:** AI systems rely heavily on large volumes of high-quality data to function effectively. In IT operations, data is generated from various sources such as logs, metrics, events, and alerts. This data is often unstructured, noisy, and inconsistent, making it difficult for AI algorithms to process accurately. Ensuring data cleanliness, relevance, and consistency requires significant effort in data preprocessing and management [22, 39, 62, 68, 79].

2. **Complexity of IT Environments:** Modern IT infrastructures are highly complex and heterogeneous, encompassing on-premises systems, cloud services, virtualized environments, and legacy systems [21, 22, 39, 79]. Integrating AI solutions into these environments without disrupting existing services is a substantial challenge. AI models must be adaptable to various platforms and technologies, increasing the complexity of deployment and maintenance [22, 39, 62].

3. **Interpretability and Transparency:** Many AI models, particularly those based on deep learning, are often considered "black boxes" due to their lack of explainability [39, 50, 60, 62, 79]. In IT operations, understanding the reasoning behind AI-driven insights is crucial for trust, compliance, and effective decision-making. The inability to interpret model outputs can hinder adoption, as IT professionals may be reluctant to rely on recommendations they cannot fully understand.

4. **Skill Gaps and Expertise:** Implementing AI in IT operations requires a unique blend of expertise in both AI technologies and IT domain knowledge [39,

62, 79]. There is a shortage of professionals who possess this combination of skills, making it challenging for organizations to develop, deploy, and maintain AI solutions effectively. This skills gap can slow down AI initiatives and reduce their overall effectiveness.

5. **Cultural Resistance and Change Management:** The introduction of AI technologies can be met with resistance from IT staff who may fear job displacement or distrust automated systems. Overcoming this resistance requires effective change management strategies, including education, clear communication about the role of AI, and involving staff in the implementation process to build trust and acceptance [39, 62].

The above list is not exhaustive as there may be other considerations that slow down adoption including security and privacy concerns, ethical and legal considerations as well as cost and resource allocation [39, 62, 79].

Addressing these challenges requires a comprehensive approach, that requires development of frameworks, tools and procedures [39, 62]:

- Implement robust data governance practices to ensure data quality and accessibility. Employ data preprocessing techniques to clean and normalize data for AI consumption.

- Upgrade infrastructure to support the computational demands of AI models. Utilize cloud services and scalable architectures to manage resource allocation efficiently.

- Develop and adopt AI models that provide transparency and interpretability, enabling IT professionals to understand and trust AI-driven insights.

- Invest in training and development programs to build internal expertise. Encourage collaboration between AI specialists and IT operations staff to foster knowledge sharing.

- Engage stakeholders throughout the organization. Communicate the benefits of AI and involve staff in planning and implementation to reduce resistance.

The emergence of Large Language Models (LLMs) may be key in addressing some of the challenges, especially when dealing with data quality, IT environments complexity and skill gaps challenges given their advances natural language understanding abilities. These features may help organizations analyze vast amounts of unstructured data, such as logs, incident reports and system documentation and is motivating our research. We aim to combine the traditional predictive ML models with generative AI models such as the LLMs and propose new ways to address some of the earlier mentioned AIOps challenges. Chapter 5 presents the details and specifics of the suggested improvements.

1.6 Contributions

In this dissertation, we study how to improve the load testing process and how using load test results and automation contribute to improved capacity planning techniques. We also study how the load testing data collected during this process may be used to increase the efficiency and thus reduce the overall cost of the load and endurance testing processes. Additionally, we explore the utilization of generative AI with large language models agents to extend the capabilities of ITOM practices and increase efficiencies through AIOps where LLMs and predictive models are used together to solve everyday operational challenges. Our contributions are as follows:

1. We introduce MLASP [114], a framework that combines automation with predictive online machine learning models to enhance capacity planning for large-scale industrial systems. This framework may be used during the software development phase (managed by project teams) and the production phase (managed by IT operations teams). We demonstrate how load testing data can be progressively used to help developers identify the most sensitive parameters in the software ecosystem and expedite the load testing process. Additionally, we show how automation and non-intrusive data collection techniques can be generalized to increase efficiency and reduce costs throughout the software development life cycle (SDLC).

2. We propose MLOLET [115], a novel process that employs predictive machine learning models in an online setting. This process further assists load testing and operational teams in improving the efficiency of load and endurance testing. Additionally, IT operations can use these same models to predict traffic trends, allowing them to anticipate when system tuning is needed to meet traffic demands and uphold service level agreements (SLAs).

3. We propose the use of generative AI and predictive ML as a means to improve AIOps adoption for large scale industrial systems and allow organizations to achieve higher levels of autonomy and adaptability in their IT operations, leading to improved performance, scalability, and overall service quality.

Chapter 2

Background and Related Works

In this section, we review existing literature and methodologies that use machine learning based approaches related to capacity planning, load testing optimization, traffic trend prediction. We also review the same for the application of AIOps in industrial settings. We then describe the commonly used metrics for evaluation to provide a comprehensive understanding of the effectiveness of these approaches.

2.1 Capacity Planning For Large Scale Systems

As previously mentioned, the research community has displayed a great interest for running software with optimal configuration parameters. Based on the direction of the approach, the research can be grouped as follows [9]:

- Analytical optimization: makes use of mathematical models to calculate the effect of the configuration parameters over the software system. This method is often used in the early development cycles of the software system [20, 101].
- Measurement based optimization - makes use of statistical approaches [42, 43]
- Search based optimization - this direction follows a black box optimization approach using various search algorithms [70].
- Learning based configuration optimization - makes use of different machine learning techniques [9, 101, 108].

Guo et al. [42, 43] use statistical learning approaches to predict system performance by randomly sampling various sets of system configuration parameters. However, their methods primarily focus on tuning binary system parameters. With the increasing prominence of neural networks, Ha and Zhang [45] have proposed a novel approach for performance prediction using deep sparse neural networks, called DeepPerf. In their research, they conducted experiments on eleven open-source projects and compared their method with existing state-of-the-art approaches. They determined that their proposed method outperforms existing techniques.

Sayyad et al. [101] proposed an approach called the Indicator-Based Evolutionary Algorithm (IBEA) for finding optimal configuration models in very large software systems, such as the Linux kernel, which may have thousands of parameters. Their method uses heuristics to determine a subset of configuration parameters that affect specific parts of the system.

Chen et al. [20] utilize log analysis to understand system execution. With this information, they employ Petri net models to recommend caching configurations. AutoConfig, by Bao et al. [9], proposed automated configuration for distributed message systems (DMS) such as Apache Kafka [6] and RabbitMQ [96]. While their work focuses on the proposed type of systems, it may be extended to other systems too. When it comes to large-scale industrial settings, Li et al. [70] share their experience from an industrial environment where they worked with their partner to include autonomic computing capabilities to address performance configuration tuning.

Unlike prior studies, our proposed approach, MLASP [114], considers the entire set of configuration parameters, including environment and deployment settings. Our focus is to assist load testers and operations engineers with capacity prediction, rather than seeking the optimal configuration parameter for every component of the software ecosystem. Additionally, we propose a blueprint for integrating automated test pipelines.

Another key aspect of our approach is data collection: we concentrate on non-intrusive information retrieval that does not require code changes to the underlying system. This is important because, in industrial settings, source code profiling or sampling is often not feasible due to performance overhead or the use of closed-source systems where the source code is unavailable. This means lower-level design documents are not always accessible, so the emphasis is on system integration aspects.

As a result, a white-box approach is not always possible or may only be viable to a limited extent.

2.2 Traffic Spike Detection and Load Testing Enhancements for Large Scale Systems

Over the past few years, event forecasting and anomaly detection have also been of great interest in the research community. However, they have rarely been studied and applied simultaneously in industrial environments due to their different characteristics. To identify anomalies, there are two general methods: online detection and offline detection. Given our research objective to increase the efficiency of load testing, we focus exclusively on online detection methods. While spikes can be considered a subclass of anomalies, they have special characteristics—such as extended duration—that set them apart from typical anomalies, which are generally brief deviations from the normal or expected behavior of the system. It is important to note that we view load testing data as a series of events over time. Therefore, we will discuss related work from two areas: time-series forecasting using deep learning models and time-series-based spike detection in load test data.

2.2.1 Time-series Forecasting Using Deep Learning Models

Modern business applications frequently encounter daily challenges in event forecasting. To address various business problems, prior research has developed and evaluated architectures based on machine learning models.

Wei et al. [119] have demonstrated that Long Short-Term Memory (LSTM) based auto-encoders (AE) models outperform other model architectures when applied to modeling and predicting road traffic flow. Similarly, Laptev et al. [65] and Zhu and Laptev [132] have also used an LSTM-AE (Long Short-Term Memory Autoencoder) architecture at Uber to forecast the number of trips during special events like Christmas or New Year's Eve. In addition to event forecasting, they applied the model for anomaly prediction during real-time data collection, predicting anomalies when the forecasts fell outside the expected interval. Our proposal, MLOLET [115], differs from these prior research as we focus on a different domain – load testing of software systems. As in the above mentioned work, we also plan on trying different machine learning architectures.

An example of such an architecture is the Temporal Convolutional Network (TCN). Lin et al. [73] demonstrated the effectiveness of TCNs in predicting long-term time series events. While their approach combined two types of data for model training, our work differs by focusing on a single dimension for training data—specifically, one key performance indicator (KPI) at a time.

Combining different types of networks for distinct purposes is another promising approach for time series forecasting, as demonstrated by Shen et al. with their SeriesNet architecture [105]. This approach employs two network architectures: an LSTM-based network in the first step, followed by a causal convolutional network (CN) in the second step. Shen et al. suggest that LSTMs are effective for learning holistic features and reducing the dimensionality of multi-conditional data, while the CN focuses on capturing patterns across different time intervals. Our work differs from theirs as we evaluate only one architecture at a time. However, their proposed integration of the two-network architectures could be incorporated into MLOLET [115], given its extensibility feature.

2.2.2 Time-series based Spike Detection for Load Tests

Chen et al, [16], proposed SPIKE as a method to predict cloud resource usage spikes using regression trees. In their approach, they used supervised learning to classify any data point above a certain threshold as an anomaly, identifying it as a traffic event contributing to a spike. While we plan to use similar modeling features as Chen and his team (e.g., service response time, transaction throughput, etc.), we intend to adopt a different approach—specifically, unsupervised learning—for the spike detection component, as we will not do a classification based on fixed thresholds.

Wen and Keyes [120] talk about the criticality of anomaly detection in automated monitoring systems. Through their work, they demonstrate the significant role that transfer learning can play when using Convolutional Neural Network (CNN) architectures, particularly in extending partially trained models to other systems. In our research, we plan to leverage this idea by testing how models trained on specific time series data perform when system conditions have changed. In other words, we aim to assess how well previously trained models can predict spikes and forecast system traffic when the application is configured with a new, previously unseen configuration. To verify this assumption, we plan to use several model architectures, including CNNs.

2.3 Using Large Language Models for IT Operations Management

Integrating Large Language Models (LLMs) into Artificial Intelligence for IT Operations (AIOps) is an emerging field poised to revolutionize the management and maintenance of IT systems. LLMs like OpenAI's GPT-4 and Anthropic's Claude have demonstrated exceptional abilities in understanding and generating human-like text, which can be leveraged to enhance various aspects of IT operations.

There are different research areas on utilizing LLMs for AIOps. One such area is related to log analysis, which includes sub-areas like log parsing [57, 80, 81, 125], log anomaly detection [46, 74] and logging statement generation [71, 124]. IT systems generate vast amounts of unstructured and complex log data. Researchers are exploring how Large Language Models (LLMs) can process and interpret this data to identify patterns, detect anomalies, and predict potential system failures. By understanding the context within log messages, LLMs can proactively identify issues before they escalate, thus reducing downtime and improving system reliability. Our work differs from prior research because we are not targeting logs but focusing on remediation workflows and procedures.

Another research path refers to automating incident management and response [41, 55, 100, 129]. Large Language Models (LLMs) can be trained to interpret alerts, correlate events, and suggest remediation steps. They are capable of generating incident reports, summarizing key findings, and even automating communication between IT teams. This not only accelerates the incident resolution process but also reduces the cognitive load on IT personnel, allowing them to focus on more strategic tasks. While our work can be applied to incident management, the agents we develop—with their associated toolsets—can also be used for preventive maintenance. Moreover, to the best of our knowledge, our research is the first to integrate predictive machine learning models with LLMs in AIOps specifically for capacity planning purposes.

2.4 Evaluation metrics for Predictive Machine Learning Models in Capacity Planning and Load Testing Optimization

The choice of metrics to evaluate machine learning models depends heavily on the type of problem being addressed, as each metric provides unique insights into model performance. Each metric emphasizes different aspects of performance, making it crucial to select the most appropriate ones to ensure the model aligns with the specific goals and requirements of the problem being solved. As we've seen earlier, the capacity planning challenge is a regression problem and load testing optimization relies on a time series forecasting problem. Below we provide the names and definitions of most commonly used metrics to evaluate these types of machine learning models.

1. **Median Percentage Deviation** MPD, is the median of the measured percentage deviation between the predicted and actual target (system throughput) The percentage deviation for a point in this set (given n points in the testing set) is the fraction of the difference between the actual and the predicted throughput from the actual, for each point i in the set. Thus, the MPD is calculated as:

$$MPD = Median(\frac{actual_i - predicted_i}{actual_i}). \tag{1}$$

2. **Mean Absolute Percentage Error** (MAPE), also known as mean absolute percentage deviation (MAPD), is a measure of prediction accuracy of a forecasting method and measures the size of the error in percentage terms. It is calculated by the formula (given n points in the testing set):

$$MAPE = \frac{1}{n}\sum_{i=1}^{n} \mid \frac{actual_i - predicted_i}{actual_i} \mid \tag{2}$$

3. **Mean Absolute Error** (MAE) is the average of the absolute errors, which is the difference between the measured/predicted value and the actual value and it is defined by the following formula (given n points in the testing set):

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \mid actual_i - predicted_i \mid \tag{3}$$

4. **Mean Squared Error** (MSE) measures the squared average distance between the real data and the predicted data and it is defined by the following formula (given n points in the testing set):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(actual_i - predicted_i)^2 \tag{4}$$

5. **Root Mean Squared Error** (RMSE) is the square root of the mean squared error, thus defined as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(actual_i - predicted_i)^2} \tag{5}$$

6. **R2 score**, or the coefficient of determination is a statistical measure of how close the data are to the fitted regression line and it is defined by the following formula:

$$\begin{gathered} R^2 = \frac{Variance\_explained\_by\_the\_model}{Total\_variance} = \frac{SS_{tot} - SS_{res}}{SS_{tot}}, \\ SS_{tot} = sum\_of\_total\_squares, \\ SS_{res} = sum\_of\_squares\_of\_residuals \end{gathered} \tag{6}$$

   The SS_{res} is also known as the the unexplained variation.

7. **Pearson Correlation Coefficient** (PCC) is a measure of the linear correlation (or dependence) between two data sets. In our case, we applied this metric only for evaluation of the time series models performance, therefore the two data sets in question are the true values set and the predicted set. In summary, the PCC is defined as the ratio between the covariance of the two variables and product of their standard deviations as follows:

$$\begin{gathered} \rho_{act,pred} = \frac{cov(act, pred)}{\sigma_{act}\sigma_{pred}} = \\ \frac{\mathbb{E}[(Actual - \mu_{act})(Predicted - \mu_{pred})]}{\sigma_{act}\sigma_{pred}} \end{gathered} \tag{7}$$

2.5 Evaluation metrics for Large Language Models

Large Language Models (LLMs) are evaluated using a variety of metrics that assess their performance across different dimensions, depending on the specific tasks they are designed to perform. Since LLMs are technically text generators that predict the next word in the sequence, some of the metrics used to evaluate LLMs are from traditional Natural Language Processing (NLP) metrics or the rise of the pretrained models (e.g., BERT [28]), as highlighted in prior studies [51, 82].

According to Sai et. al [99], LLMs should be evaluated by the task they perform (e.g. classification, question and answering, summarization), therefore many other metrics can be used. They also suggest that human expert evaluation that rate a model's outputs based on criteria like fluency, coherence, relevance, and appropriateness are essential for open-ended generation tasks where quantitative metrics may not capture nuances. In addition, there are also system performance metrics that need to be taken into account. As a result, given their contextual nature, below we present next only the metrics used in our research to determine the performance of LLM agents used in an AIOps context:

1. **Compliance with Instructions**: Measures how well the model follows user instructions or prompts. Since agents make use of tools, with the help of this metric we are measuring the LLMs capacity to understand what tools it needs to use and in what order.

2. **Accuracy**: Measures the ratio of correctly predicted instances to the total instances. Although this metric is usually applicable for classification tasks, we use it to measure, when asked repeatedly, the agent has been able to correctly solve a task and provide the correct answer. The correctness of the answer is made by human expert evaluation.

3. **Human Evaluation**: Involves human judges rating the model's outputs based on criteria like fluency, coherence, relevance, and appropriateness. This is essential for open-ended generation tasks where quantitative metrics may not capture nuances, or for expert systems evaluation where the result is a chain of actions decided by the chain of thought of the LLM [23, 117].

4. **Latency and Throughput**: Measures the time taken to generate responses and the number of responses generated per unit time. This is important for real time applications where the responsiveness is critical.

5. **Cost**: Measures the number of tokens by the LLM in the chain of thought to process the user request, reach a conclusion and format the final response.

Part I

Capacity Planning For Large Scale Systems

Chapter 3

Enhancing the Effectiveness of Capacity Planning for Large Scale Systems

This chapter presents an empirical study that examines the challenges of capacity planning for large scale industrial systems and our proposed blueprint for a framework and process that may be used both during the project phase and the production phase. The blueprint we propose is extensible and generalizes on any configurable software systems.

Prior research [32] has highlighted the significant economic impact on projects and companies when applications cannot handle sustained load pressures and traffic spikes. Similarly, the research community has shown that finding optimal parameter values is a complex and time-consuming task [9, 20, 42, 43, 45, 70, 101].

However, although optimization is important, in industrial settings it makes more sense to focus on optimizing Key Performance Indicators (KPIs) and Service Level Agreements (SLAs), since these metrics ensure the correct functionality of the system. Therefore, concentrating on capacity planning is more meaningful for the business. This means that optimization efforts should target achieving optimal capacity for the entire software ecosystem, rather than solely optimizing individual software configuration parameters—even though the two are, to some extent, directly linked for obvious reasons.

We, therefore propose MLASP [114], a framework that uses automation and machine learning to address the capacity planning in industrial settings starting from the project phase and continue into the production phase.

An earlier version of this chapter has been published at Empirical Software Engineering (EMSE), 2021, Volume 26, Article Number 87. Vitui and Chen [114]

3.1 Motivation

The motivation for our research was derived from the collaboration with our industrial partner, Ericsson Inc. We present in this section the challenges faced by our partner during the capacity planning process.

Load test engineers at Ericsson Inc. are tasked with testing and verifying the capacity of software systems within a short time frame—the release cycle is only six weeks. Software development teams rely on the load testing team's results to determine whether the new release can be deployed to production. Depending on these results, a new Service Level Agreement (SLA) may also be provided if needed. However, given the complexity of the software systems and the hundreds of performance-related configurations, it is difficult for the load testing teams to execute comprehensive load tests within the allotted time frame.

Ericsson's systems consist of multiple components that combine both third-party software and in-house developed software. The in-house components are maintained by numerous developers and can encompass millions of lines of code. Depending on requirements driven by capacity needs and project economics, these components are grouped together either within the same application server or across different ones. This approach addresses the need for increased capacity and redundancy. Sometimes, redundancy is geographically distributed across several data centers. Each component of the software system has various configuration parameters to provide increased operational flexibility. However, this can affect the system's performance and capacity at both single-instance and clustered levels. The task of the load test engineers is to estimate the software system's capacity by considering not only the components' configuration values but also the hardware settings, and to provide guidelines that meet the performance and service level agreements expected by the customer.

Generally, load test engineers do not know how each configuration parameter—or combination of parameters—affects overall system performance. They rely on default values provided by software developers. Similarly, developers may not know what type of hardware or virtualization technology will be used to run the software. A prior study by Li et al. [70] shows that developers may not even understand the effects of parameter value combinations, so default values are sometimes provided based on guessing or prior experience with similar systems. Load test engineers cannot afford to test all possible combinations of parameter values, so they rely on their experience and test only the most significant parameters.

Once an initial set of parameters has been selected, the load testing teams need to perform additional endurance testing to ensure the software system's stability under sustained load. Throughout the endurance testing phase, the software components are continuously monitored for capacity metrics—such as throughput and other KPIs—as configuration values change.

One major drawback in capacity and performance evaluation is that when system components are modified—such as through code changes, deployment settings, or hardware configuration changes—the effects of the configuration parameters may also change. This issue is known as the n-way feature interaction problem. Rerunning all load tests to verify the effects of different parameter combinations becomes very time-consuming and costly. Moreover, the number of parameters can increase exponentially when new components are added to the software system. This makes it impossible to test a sufficiently wide range of parameter values to confidently assert that a given set leads to the desired capacity capable of sustaining SLAs under heavy load for extended periods.

To meet time constraints between releases, load test engineers might consider accelerating testing through parallelism by adding more computing resources, if available. However, this approach may not be feasible for large-scale solutions due to the time required to deploy these resources, not to mention the significant cost increase that the project would incur.

To summarize, poorly configured software can lead to suboptimal resource utilization, resulting in performance issues and revenue loss. It can also cause deployments to be larger than necessary, increasing capital expenditure costs for hardware, electricity, and other resources. Capacity planning for large-scale systems is an expensive and time-consuming process. These points highlight the critical importance of having an efficient capacity planning and configuration analysis process for large-scale systems. In this section, we present a novel machine learning-based approach to help load test engineers reduce the number of test runs needed to study and verify a system's capacity.

3.2 Case Study Setup

In this section, we present details about the studied systems. We then present the methodology for the proposed framework blueprint.

3.2.1 MLASP Framework Proposal

Our objective for improving the capacity planning process is to provide a generic process blueprint that can be adopted in an industrial setting with minimal effort. The intention is to have a process that leverages existing and integrated toolsets used by load testing engineers and developers for various tasks within their current capacity planning and parameter tuning processes, preferably without requiring any code changes. To achieve this, we propose a three-stage process called Machine Learning Assisted System Performance and Capacity Planning (MLASP) [114] to effectively address capacity planning challenges.

MLASP is a three stage process and its overview is presented in Figure 1. In the first stage we aim to increase the load testing efficiency by the means of automation. As mentioned before, we focus on non-intrusive data collection mechanisms. This is imperative to follow as a generic approach which does not require access to the original source code of a subject software component to extract valuable metrics. The second phase presents the machine learning modeling and its associated activities: data cleaning and feature engineering, model training and validation. The third and final part is about using a trained model to answer the two applicable business situations: "what if?" and "find a valid configuration for a desired KPI" (i.e., throughput). Noteworthy is the fact here that due to the nature of the problem we're trying to solve, it may be impossible to find a configuration set that yields precisely a KPI, therefore we must consider searching for a configuration set that is close enough to a desired target, in other words, it is within an acceptable percentage deviation from

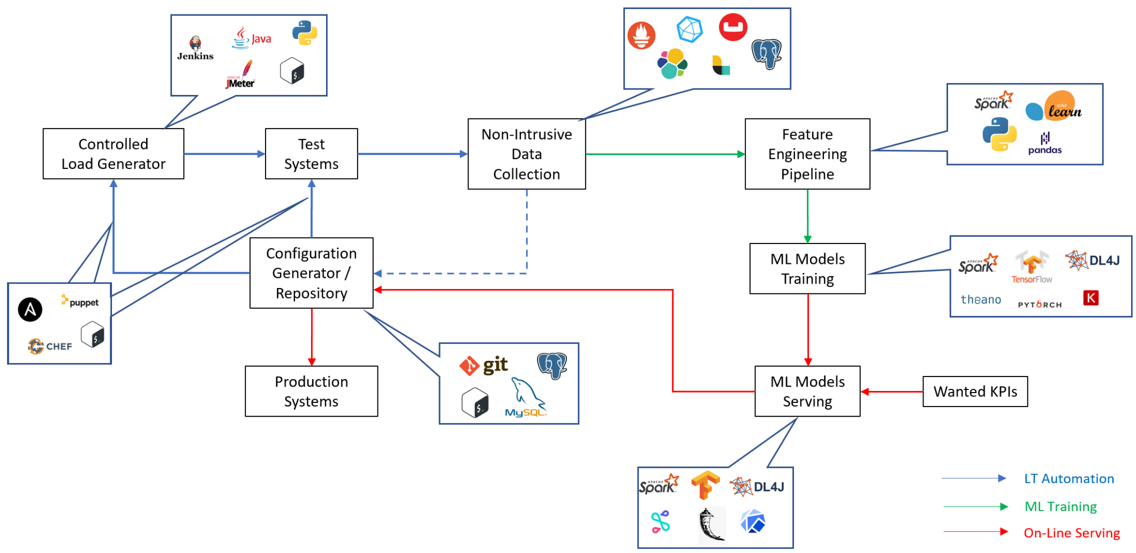

Figure 1: MLASP Process with its three stages: (1) Automated Load Testing, (2) Feature Engineering and ML Training, and (3) Model On-Line Serving.

the desired KPI value.

Our proposed approach to automated load testing considers the following aspects of the overall process:

- **Automatic component configuration**: at the beginning of each test cycle, random values are generated and applied to the selected configuration parameters. This operation is performed for every selected component.

- **Automatic cluster configuration**: at the beginning of each test cycle, within the available resource boundaries, each component deployment is randomly scaled.

- **Automatic invocation of a dynamically configurable load driver**: at the beginning of each test cycle, a load driver is programmed to generate and apply constant or variable load over the subject system as required by the business test case scenarios.

- **Automatic and consolidated data collection**: throughout the execution of the load test and as well as at its completion data is collected from the components and underlying infrastructure using non-intrusive methods. These data points form the desired KPI metrics data sets.

Key to controlling the automation of the load testing part of the MLASP framework is the random configuration parameter value generator which may be implemented in various ways: as a generic script, or as a custom application written in several programming languages. Alternatively, for regression testing purposes, prior configurations may be retrieved from persistent storage solutions such as relational databases (e.g., MySQL [87], PostgreSQL [91], etc.), no-sql databases (e.g. CouchDB [7], etc.), or even a file versioning system. A new randomly generated, or a previously defined configuration set is then applied to the target components (including the load generator) by using either specialized tools (e.g., Ansible [5], Puppet [93], or Chef [15]), or by custom scripts/applications. The load generator may also be implemented through various ways: it may be a custom developed application or an existing specialized tool such as JMeter [34], Jenkins [54], Soap-UI [107], etc. To collect metrics using a non-intrusive approach, the software systems (applications and the underlying infrastructure) must expose beforehand the KPIs of interest. These KPI values maybe collected through logs (if available), or, by exposing counters that may be queried by external agents, specialized for collecting such information. Most modern systems follow the practice of exposing counters for querying either logs, for tools such as Logstash [77] or Elasticsearch [29], or through specialized web services where the information may be collected using specialized agents like JMX [58] (for Java application only) or Prometheus (for any application that exposes Prometheus style counters). The metrics information collected may also be aggregated and stored in various types of databases (relational, time-series, no-sql).

The next phase in the proposed framework is the Machine Learning engineering stage. This consists of a feature engineering pipeline and model training and validation step. Similar to the previous stage, there are many available tools for each step in the stage (data engineering, model training and validation) such as: Apache Spark [8], DeepLearning4J [27], Tensorflow [111], PyTorch [95], scikit-learn [103], etc.

The final stage of the MLASP blueprint is the model serving and inferencing phase. Depending on the tools and libraries the model was created with, the final trained and validated model may be served by frameworks such as Apache Spark [8], Seldon [104], Flask [33], DeepLearning4J [27], etc. The resulting inferencing endpoint may then be used for the two earlier mentioned business scenarios:

- What if – in this case the load test engineer provides the model with concrete parameter values and finds out the predicted KPI information from the model.

- Find valid configurations within provided ranges of parameter values that meet a desired KPI value within an acceptable threshold. This scenario is very useful for production capacity planning when a robust model (that has been trained on a significant amount of data) is queried repeatedly to provide possible sets of configuration parameter and deployment setting values. Using specialized knowledge of the system, the load test engineers may then select one of the proposed configuration sets and apply them to the software ecosystem.

Due to NDA reasons, we cannot disclose the integrations and tool sets our industrial partner has used in each phase of the MLASP process. However, as we described above, several options are available for each stage. Furthermore, the proposed blueprint is flexible may be extended or adapted to incorporate additional steps as needed.

3.2.1.1 Feature Engineering and Selection

Feature selection and engineering are the precursor steps for machine learning model training. They are necessary as they ensure that quality data is provided to the machine learning models, which are very sensitive to this type of information. For example, as prior studies show, multicolinearity may affect the stability of the model (e.g., overfitting) [35, 47]. Therefore, we conduct an analysis to find correlated features and only keep one of them in case a correlation of over 0.8 is detected [36].

As prior studies indicate [106, 108], some machine learning algorithms are sensitive to the scale of the input data. There are various scaling techniques available, and in our experiments, we use the StandardScaler and MinMaxScaler implementations from the scikit-learn [103] library. The StandardScaler subtracts the mean and scales the data to have unit variance, while the MinMaxScaler rescales the dataset so that the entire feature set falls within the range [0, 1].

Also, for training purposes we consider only the configurations of successful load tests, meaning that during a load test there were no errors recorded in the system (i.e. all request were responded with 2xx OK type response).

It is important to note that the outcome of the feature selection process is closely tied to a specific set of configuration parameters. This means that whenever a new non-constant parameter is added to the system, it becomes necessary to recreate and retrain the model. The reason is that the existing model may no longer provide accurate predictions if it does not take into account the newly introduced parameters.

3.2.1.2 Applying Machine Learning Techniques

To model and predict KPI responses based on a given set of parameter values, we utilize different machine learning algorithms and architectures. This approach allows us to capture various relationships between the features and the target variables, as different models may excel in different aspects. By applying multiple techniques, we study how effectively each algorithm models the given data. We apply the following types of models:

Tree-based models:

- Random Forests - they are an ensemble learning algorithm that are constructing multiple decision trees. Random Forests are less sensitive to outliers and can identify important features [12].

- Extreme Gradient Boosting (XGBoost) - they are an optimizied version of the gradient boosting machine learning method that is based on decision trees. Prior research pointed out that this method is highly efficient for modelling regression problems [18, 86, 90].

Deep Neural Network models:

- Multi-Layer Perceptron (MLP) - it is the base unit of a fully connected feed forward neural network. Such a network is capable of modeling data with non-linear relationship [130]. As pointed out by Cybenko's theorem [17], MLPs are universal approximators and are suitable for regression problems. It is important to consider this factor in model selection because universal approximators can extrapolate and make predictions using data that falls outside the sets and ranges used during training. In contrast, tree-based methods do not perform well when asked to make predictions on data that falls outside the feature ranges they were trained on.

- Convolutional Neural Networks (CNNs) are regularized versions of Multi-Layer Perceptron (MLP) networks. Although they were originally created for image analysis [38], CNNs have proven to be very effective at modeling regression problems, as highlighted by prior research [66].

- Long Short-Term Memory (LSTM) networks [130] are a variant of recurrent neural networks that capture and remember the order of data in sequences during model training [30, 123].

Traditional model:

- The Linear Regression (LR) model serves as a fundamental baseline in machine learning for regression problems. We use this model as a reference point to compare its prediction performance with the more advanced models we mentioned above.

For each model, we use the same training and testing data sets. Following the Pareto principle [3, 126], we split the training data into 80% for actual model training and 20% for validation. We then test the models using the test data sets.

3.2.1.3 Training Machine Learning Models

In this section we discuss aspects about ML model training. In our approach, we implemented a sequential model hyperparameter tuning process, which can result in longer training times. Specifically, we observed that searching for optimal hyperparameters in deep neural network models can take a significant amount of time (e.g., several hours for the open-source system) due to their complexity and the large number of parameters that require optimization. While deep neural networks generally deliver excellent prediction results when sufficient training data is available, load test engineers may still opt for algorithms like XGBoost. This is because XGBoost offers significantly shorter training times while still providing strong prediction performance.

Based on our experience, as more test data becomes available, the optimal model architecture (e.g., the number of neurons per layer in Neural Network models) or hyperparameter values may change. For instance, we observed that the optimal hyperparameter values identified in research questions RQ1 and RQ2 differed when using our model hyperparameter tuning approaches. Consequently, in addition to online training, it is necessary to periodically retrain the models to determine new optimal hyperparameter values. Similar findings were observed in the enterprise systems, prompting our industrial collaborators to incorporate model retraining and hyperparameter tuning into their CI pipeline once the number of new load tests exceeds a certain threshold. This practice has been shown to improve prediction accuracy and better capture the relationship between the system's configuration parameters and throughput.

Similar to the approach by Ha and Zhang [45], we developed an iterative method to determine the optimal depth for the neural network that yields the best performance on the validation subset. This iterative search involves testing network architectures with different widths, such as 32, 64, and 128 neurons per layer, respectively. We begin the evaluation with no hidden layers and incrementally increase the network depth by adding one layer at a time, up to a predetermined maximum number of layers (15 in our case). For each iteration, we record the following information:

- **Model performance metrics:** Metrics such as R2 score and Mean Absolute Error (MAE) are calculated for the trained network's performance on the test data. These metrics help visualize the evolution of the loss function's performance with respect to the number of hidden layers.

- **Training and validation data values:** We store the training and validation data values for each iteration to visualize their history. This allows us to identify signs of overfitting or underfitting, such as when the training results deviate significantly from the validation results.

- **Predicted throughput:** The predicted throughput values for the test data are recorded to evaluate the model's predictive performance.

- **Model details:** The trained model's parameters and associated details are saved in a file containing the best training score. These files allow us to reload the fully trained model and deploy it within the CI process for further validation and use.

This comprehensive recording process ensures that we can effectively evaluate and refine our neural network architectures for optimal performance.

The process of expanding network depth can be either manual or automated. In the manual expansion process, ML model developers follow an iterative approach to determine the optimal maximum depth for neural networks or decision trees. For example, during the first trial, the maximum depth may be set to $N = 5$. After reviewing the results, developers can decide whether the performance is acceptable or if further modeling is required. If additional modeling is needed, the maximum depth can be increased (e.g., to $N = 10$ or $N = 15$).

Although the manual process is straightforward to implement, it may not be the most efficient method in terms of training time. To optimize the depth expansion process, developers can adopt an automated approach [45]. This involves continuously increasing the depth as long as the model performance metrics improve beyond a certain threshold. For instance, the process can stop if the performance improvement is less than $x\%$ after adding N additional layers, or if the results deteriorate by more than $y\%$ after adding M additional layers. In our experiments, we chose a manual approach for the open-source system, while we employed an automated approach for the enterprise system.

During this iterative process, to optimize model performance we also tune other neural network-specific parameters, including the following:

- **Batch size:** Determines the number of samples from the training set that the model processes before adjusting its internal parameters (weights).
- **Epochs:** Specifies the number of complete passes (forward and backward) through the entire dataset during the learning process. One epoch corresponds to a single full pass through the dataset.
- **Regularization values:** Adjusts the L1 and L2 regularization parameters [88] to help prevent overfitting by adding a penalty for large weights.

Regularization is a technique used to prevent overfitting by adding a penalty to the model's parameters (excluding the intercept). This penalty is incorporated into the loss function, encouraging the model to generalize the data and avoid overfitting. Prior research [88] recommends evaluating model performance using two standard types of regularization procedures:

- **Lasso regression [112]:** Applies a penalty that minimizes the sum of the absolute values of the model parameters (weights). This method is also known as the L1 norm.

- **Ridge regression [89]:** Uses a penalty that minimizes the sum of the squares of the model parameters. This method is also known as the L2 norm.

These methods help ensure that the model remains robust and performs well on unseen data.

For tree-based algorithms, we also use an iterative approach to determine the optimal number of trees as well as the L1 and L2 regularization values to achieve better prediction results. We gradually increased the number of trees until there was little to no improvement in prediction performance. Similarly, we adjusted the L1 and L2 regularization values incrementally to optimize the model's performance.

To further study if our models suffer from any overfitting, we also conduct a 10-fold cross validation for all models on the existing test data. We find that when using 10-fold cross validation, the models show similar prediction results as before.

3.2.2 Studied Systems

We conduct our experiments on two sets of studied systems: enterprise and open source.

3.2.2.1 Enterprise Systems

We evaluate and integrate our methodology on two large-scale enterprise systems developed by our industrial partner, Ericsson Inc. Due to a non-disclosure agreement (NDA), we cannot provide detailed information about these systems. However, we can offer a high-level description: these systems consist of millions of lines of code and are maintained by a large team of software engineers. They provide business-to-business (B2B) and business-to-consumer (B2C) telecommunication services related to messaging, location-based services, and payments. These systems utilize Ericsson's in-house developed products, third-party commercial products, and some open-source software. They process tens of millions of requests daily, serving millions of customers around the world, including some in mission-critical operations. Owing to their critical nature, these systems are deployed with both local and geographically distributed redundancy.

3.2.2.2 Open Source Systems

To validate our findings and ensure reproducibility, we also conduct experiments on Apache Kafka [6]. Kafka is a distributed messaging streaming (DMS) system that operates in clusters of one or more servers known as brokers. These clusters can expand within a single data center or across multiple data centers. Within these clusters, streams of records (also called messages) are stored. Similar messages are grouped into topics, which are further divided into partitions. These partitions can be distributed and replicated across any number of brokers in the cluster for redundancy.

We use Kafka as our first open-source subject system as it has similar high level features as the enterprise systems as follows:

- Kafka is a highly configurable system.
- Kafka is supports non-intrusive key performance indicator measurements and dynamic configuration changes (i.e., using JMX).
- Its performance depends on both vertical and horizontal settings.
- It supports both local and geographically distributed redundancy.

We perform our experiments with Kafka on a system using Amazon Web Services (AWS) cloud resources, not using the AWS managed Kafka services. In our experiments we use three different environments (i.e. deployment settings):

- one broker with one topic having one partition and without replication.
- one broker with one topic having two partitions and without replication.
- two brokers with one topic having two cross replicated partitions, meaning each broker has one active partition and the replica of the other broker.

In addition to the Apache Kafka open source test environment, we verify the experiments and provide a full scale reproduction of our proposed framework, MLASP, in a Kubernetes [61] environment implementation, namely Red Hat OpenShift [97]. In this context, we provide comprehensive, step-by-step instructions and source code for setting up the test environment. This includes the load testing framework, data collection processes, machine learning (ML) procedures, and example operational clients for utilizing the trained ML models. The two scenarios in which the ML models are applied are: (1) a "what if" analysis, and (2) finding a valid configuration to reach a target KPI within an acceptable margin. The complete instruction set and source code are available in a GitHub-hosted project called MLASP on OpenShift [84]. This project uses WireMock [121] as the test subject—a popular open-source, generic web service mocking application that can be adapted and extended to a wide range of business applications. Through this project, we validate our experiments on an additional open-source test subject and demonstrate our proposed blueprint on a commercial-grade open-source software system. Noteworthy is the apsect that this second open-source system verification based on Red Hat OpenShift was not part of the original MLASP [114] research. The results presented in Section 3.3 refer only to the ones from the MLASP [114] publication.

3.2.3 Performing Load Tests

We presented earlier in Figure 1 the three stage process of MLASP. Next, in Figure 2 we depict the overview of the load test execution setup. We use for each studied system appropriate load scripts to generate traffic and exercise the system. We take into consideration that a system may contain multiple components, therefore, we collect the set of configuration parameters and the corresponding throughput in each component. As mentioned, we rely on non-intrusive data collection and system control techniques for reconfiguration such as specialized APIs (web-based APIs, RESTful or SOAP, or Java Management Extensions - JMX), or periodic reloading of configuration files. Throughput information about the studied software system may be collected from logs or from API queries. If logs are used, depending on the their nature and stored information, expert knowledge may be required to perform the calculations of the desired throughput information. During these load tests, the input payload was constant during the execution of the test, however different tests had different payloads as different business request types were tested.

Next, we detail the process of running load tests for the studied systems.

3.2.3.1 Enterprise Systems.

The two enterprise systems are deployed on a large scale. Each system contains several different products that work together to fulfill specific business requirements. Some of these systems offer out-of-the-box capabilities for querying dynamically collected

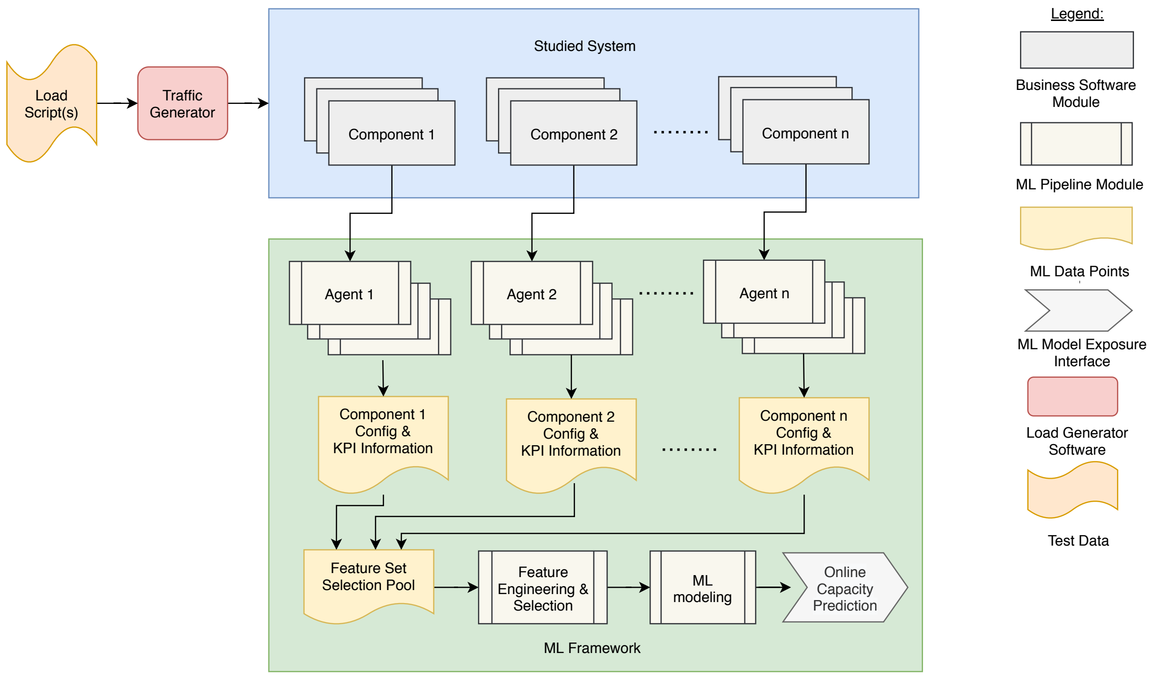

Figure 2: A high-level overview of our approach and load test execution setup.

performance metrics (i.e., for calculating throughput), while others require additional analysis to extract the necessary metrics.

To handle the distributed information, we aggregated the metrics based on timestamps. In large-scale deployments, the various components must be highly synchronized. We leveraged the synchronous properties of the enterprise systems to extract information from each component at the same synchronized time intervals. Once the data collection endpoints were in place, the load test engineers executed load tests by following a defined in-house process.

Due to the nature of the studied systems, load test engineers use in-house custom scripts to generate and execute the load. The test generation and execution processes are integrated into the Continuous Integration (CI) pipeline. At the end of each test, we collect and calculate the throughput of the studied systems under the executed load, along with the given configuration parameter values and the corresponding deployment settings.

3.2.3.2 Open Source System

As previously mentioned, concerning the studied open source systems, we shall only refer to the one described in the existing MLASP [114] publication, namely the Apache Kafka [6].

We consider three sets of configuration parameters in our study: environment settings, broker (i.e., Apache Kafka), and load-generator (i.e., custom application).

Following recommendations from previous academic research focused on Kafka performance [10, 67], as well as from commercial support suppliers for Kafka [24, 25], we select and vary the most important Kafka broker configuration parameters in our tests [6]:

- background.threads: The number of threads the broke may use for various background processing tasks. The range of values we used in our experiments for this parameter was [5-30].

- num.io.threads: The number of threads that the broker uses for processing requests that refer to disk I/O. The range of values we used in our experiments for this parameter was [4-16].

- num.network.threads: The number of threads that the broker uses for exchanging requests and response message with the network. The range of values we used in our experiments for this parameter was [3-6].

- num.replica.fetchers: The number of fetcher threads used to replicate messages from a source broker. Increasing this value can increase the degree of I/O parallelism in the follower broker. The range of values we used in our experiments for this parameter was [1-2].

For each environment setting, we vary the Kafka broker configuration parameters as well as the load-driver parameters, using the same sequence of values from the selected range applicable to each parameter.

Note that the three previously mentioned environment settings concerning the Kafka cluster setup (i.e., the number of brokers and the number of partitions per topic) are also part of the configuration parameters. From the load generator's standpoint, we use the message size and the number of client threads publishing messages into Kafka as input variables. The measured target is the client-side throughput in relation to the server-side capabilities of the target Kafka environment. In other words, given a specific configuration tuple (i.e., broker settings, environment, load generator parameters), we calculate the throughput as the number of messages that can be published to Kafka within 30 minutes. The load test results are aggregated per environment and are used to build the machine learning models for capacity prediction.

For the actual load test preparation and execution, we follow the steps below for the open-source system:

- We used the load-driver to run tests for a fixed duration (i.e., 30 minutes).
- At the beginning of each test run, we update the configuration of both the load-driver as well as of the Kafka brokers. We generate a total number of 300 distinct configurations.
- We executed load tests using the same 300 distinct configurations on each of the three different Kafka environments (as described in Section 3.2.2.2). In total, we executed 900 load tests with a total machine execution time of over 18 days.

We utilized expert knowledge to determine which configuration parameters to tune on the Kafka broker side for each load test scenario [10, 24, 25, 67]. The range used to vary each configuration parameter was defined as a list of values described by the function $f(x) = nx$, where both n and x are positive integers. This range included values both higher and lower than the default configuration value for the parameter x. The upper and lower bounds of the range (i.e., n) were selected based on domain knowledge for each configuration parameter considered in the test scenarios. For example, in the open-source environment, the BackgroundThreads parameter values of the Kafka broker were modeled by the function $f(x) = 5x$, where $x \in [1, 6]$ and the NumIoThreads parameter values were modeled by $f(x) = 4x$, where $x \in [1, 4]$.

Using a custom-developed Java-based command-line application as the load driver, we controlled the number of client threads used to send messages to the Kafka brokers, as well as the size of the messages being delivered. The load-driver application recorded the configuration parameter values and the measured throughput (i.e., the KPI for the studied open-source system) in a log file for each test run.

On the Kafka broker side, we developed an agent that collects Kafka throughput metrics at one-minute intervals. For each test run, we aggregated the broker's throughput to correlate it with the load-driver's recorded throughput, allowing us to calculate the overall system throughput under a specific set of configuration parameter values. The model's source code and the test results for the open-source systems can be found online [83].

Although we cannot disclose the details of the enterprise systems due to our NDA, we follow a similar process by running load tests with varying configuration parameter values. We use customized scripts for test execution and data collection, which are integrated into the Continuous Integration (CI) process used by Ericsson Inc.

3.3 Case Study Results

In this section, we discuss the results of our research questions (RQs). For each RQ, we present the motivation, our approach and the results.

To comply with the NDA requirements set by our industrial partner, we present capacity planning information in a generic form applicable to both the enterprise and open-source systems, focusing on application throughput. We define throughput as the number of messages or requests processed by the software system within a specified time period.

3.3.1 RQ1: What is the prediction accuracy on the system throughput given variable system configuration parameters?

Motivation: Modern large-scale systems can be highly complex, comprising multiple integrated components, each with a varying number of configuration parameters. Consequently, there may be hundreds of combinations of configuration parameter values, environment settings, and deployment options that can be applied to the system. Studying the effects of these parameter combinations requires load testing, which helps identify the configuration sets that support a desired capacity. However, this is a lengthy process. In addition to running these tests, load test engineers need to invest significant time in analyzing the results and creating baseline recommendations for specific KPI targets, such as throughput.

Approach: To simplify and reduce the time required to accomplish this task, we propose using a series of generic machine learning models to determine which one performs best on the provided data sets. To asses the performance of the models we use metrics defined in Section 2.4. We perform the training and valiadion of the models using the Pareto principle as described in Section 3.2.1.2. We also perform ten-fold cross validation of our models to verify their robustness.

Results: We have found that the machine learning models can predict the Key Performance Indicator (KPI), such as throughput, with high accuracy. Specifically, XGBoost and fully connected neural networks (MLP neural networks) achieve the best prediction results, and our findings are consistent across the studied systems. In Table 1 and Table 2, we summarize the results for both one-pass and ten-fold cross-validation training. We observe that some models exhibit high accuracy due to low values in the model evaluation metrics—for example, the Mean Absolute Percentage Error (MAPE) ranges from 0.68% to 4.5% for models like XGBoost and MLP neural networks. In contrast, other models do not capture the relationship between the configuration values and throughput as effectively; for instance, MAPE increases up to 62% for architectures like CNN, LSTM, and classic linear regression.

Similarly, when we examine the Mean Absolute Error (MAE), which measures the average magnitude of the errors, we see values between 2,200 and 5,500 in the case of the open-source system for most ML models (due to our NDA, we cannot disclose the values for the enterprise systems). Considering that the number of processed messages ranges from 100,000 to 700,000, an MAE of 2,200 to 5,500 is considered very small.

As an example, Figure 3 shows the predicted vs actual throughput across various test runs from the MLP Neural Net model of the open-source system. We can easily see that there is a very high overlap between the predicted and actual throughput (i.e., there is a near-perfect alignment between the two types of dots in the figure).

Figure 4 illustrates the distribution, where each point represents the deviation value for one load test result. The results show that the percentage deviation is small for all models in the open-source systems, except for linear regression models. This finding indicates that, in the open-source system, the model predictions are consistent across various load tests given different sets of configuration parameter values. We exclude the detailed distribution for the enterprise systems due to NDA constraints. However, as shown in Table 1, the median percentage deviation is higher for models

Table 1: Model evaluation results when using the entire data with separated training and testing. We excluded the MAE, MSE and RMSE for the enterprise systems due to NDA.

System Type	**Model**	**R2 Score**	**Median (%) Deviation**	**MAPE**	**MAE**	**MSE**	**RMSE**
Open-Src.	XGBoost	0.99964	0.0865 %	0.8060 %	2,193	2.4028e7	4,901
	Random F.	0.99958	-0.1828 %	0.6617 %	2,266	2.8253e7	5,315
	MLP NN	0.99969	0.1648 %	0.6822 %	2,110	2.0594e7	4,538
	CNN	0.99949	0.1590 %	1.1940 %	3,555	3.4482e7	5,872
	LSTM NN	0.99902	0.2583 %	1.4341 %	5,586	6.6280e7	8,141
	Linear Regr.	0.90930	7.3951 %	41.3498 %	66,759	6.1744e9	78,579
Entprz. 1	XGBoost	0.99324	0.6323 %	2.3596 %		—	
	Random F.	0.95078	-0.1450 %	4.5820 %			
	MLP NN	0.99512	2.2929 %	2.3435 %			
	CNN	0.98317	-1.0807 %	3.5182 %			
	LSTM NN	0.92908	-16.0805 %	8.0132 %			
	Linear Regr.	0.98830	4.6561 %	21.3450 %			
Entprz. 2	XGBoost	0.99973	0.8091 %	4.5485 %		—	
	Random F.	0.99647	1.5526 %	67.5510 %			
	MLP NN	0.99991	1.0868 %	3.3082 %			
	CNN	0.99067	-8.1808 %	23.6534 %			
	LSTM NN	0.97589	-5.9882 %	62.2091 %			
	Linear Regr.	0.92218	4.3187 %	10.4637 %			

Table 2: Mean of model evaluation results when using the entire data with a 10-fold cross-validation. We excluded the MAE, MSE and RMSE for the enterprise systems due to NDA.

System Type	**Model**	**R2 Score**	**Median (%) Deviation**	**MAPE**	**MAE**	**MSE**	**RMSE**
	XGBoost	0.99967	0.0473 %	0.9716 %	2,360	2.2792e7	4,742
	Random F.	0.99963	0.0789 %	0.7673 %	2,295	2.5318e7	5,004
	MLP NN	0.99921	0.4544 %	1.8316 %	4,250	5.4934e7	7,320
Open-Src.	CNN	0.99181	-0.6284 %	6.8631 %	14,891	5.7416e8	22,665
	LSTM NN	0.99759	0.4520 %	1.9627 %	7,286	1.7208e8	11,423
	Linear Regr.	0.91294	4.2547 %	42.1498 %	64,867	6.0917e9	77,905
	XGBoost	0.96281	1.4391 %	3.7316 %			
	Random F.	0.92250	-1.5916 %	7.0361 %		—	
Entprz. 1	MLP NN	0.95681	3.3302 %	3.0288 %			
	CNN	0.91407	1.6746 %	5.8063 %			
	LSTM NN	0.90522	-18.5884 %	10.0907 %			
	Linear Regr.	0.91538	4.5912 %	19.1878 %			
	XGBoost	0.94997	2.2235 %	8.1909 %			
	Random F.	0.90843	-1.3599 %	57.8177 %		—	
Entprz. 2	MLP NN	0.95196	-3.4797 %	6.9008 %			
	CNN	0.91448	9.1220 %	33.5975 %			
	LSTM NN	0.93642	4.1062 %	69.3470 %			
	Linear Regr.	0.85118	4.5878 %	27.0634 %			

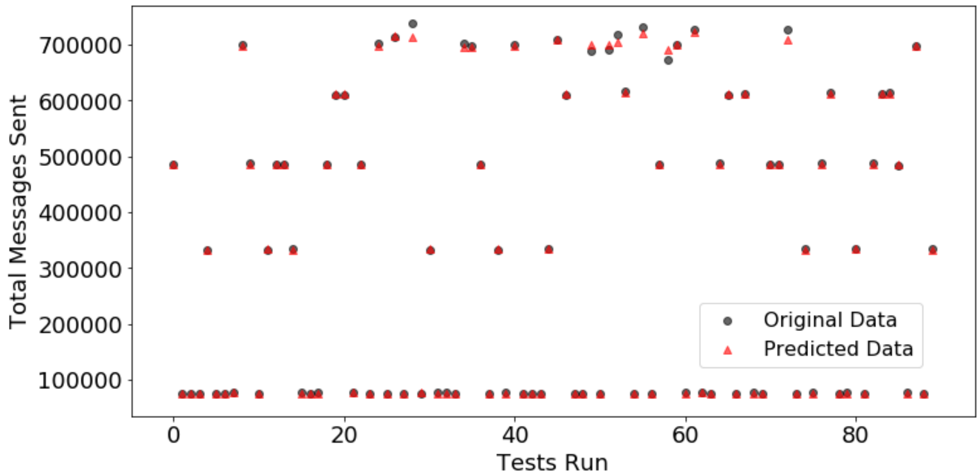

Figure 3: An example of the predicted v.s. actual throughput for the open-source system from the MLP Neural Net model results built using the complete dataset.

such as CNN, LSTM, and linear regression. Our findings demonstrate that CNN, LSTM, and linear regression may result in a higher deviation between the predicted and actual throughput. In contrast, XGBoost and MLP Neural Networks have the highest R2 scores and relatively low deviations in the prediction results.

As mentioned earlier, to further investigate whether our models suffer from overfitting, we conducted a ten-fold cross-validation. Table 2 presents the mean of the metrics obtained after applying a ten-fold cross-validation (following the same data splitting and training process). Our findings show that, using ten-fold cross-validation, the models exhibit similar prediction results as before. Figure 5 displays an example of the training versus validation loss for the MLP model obtained after training on the training set. The convergence of the training and validation loss indicates that the model is not suffering from overfitting.

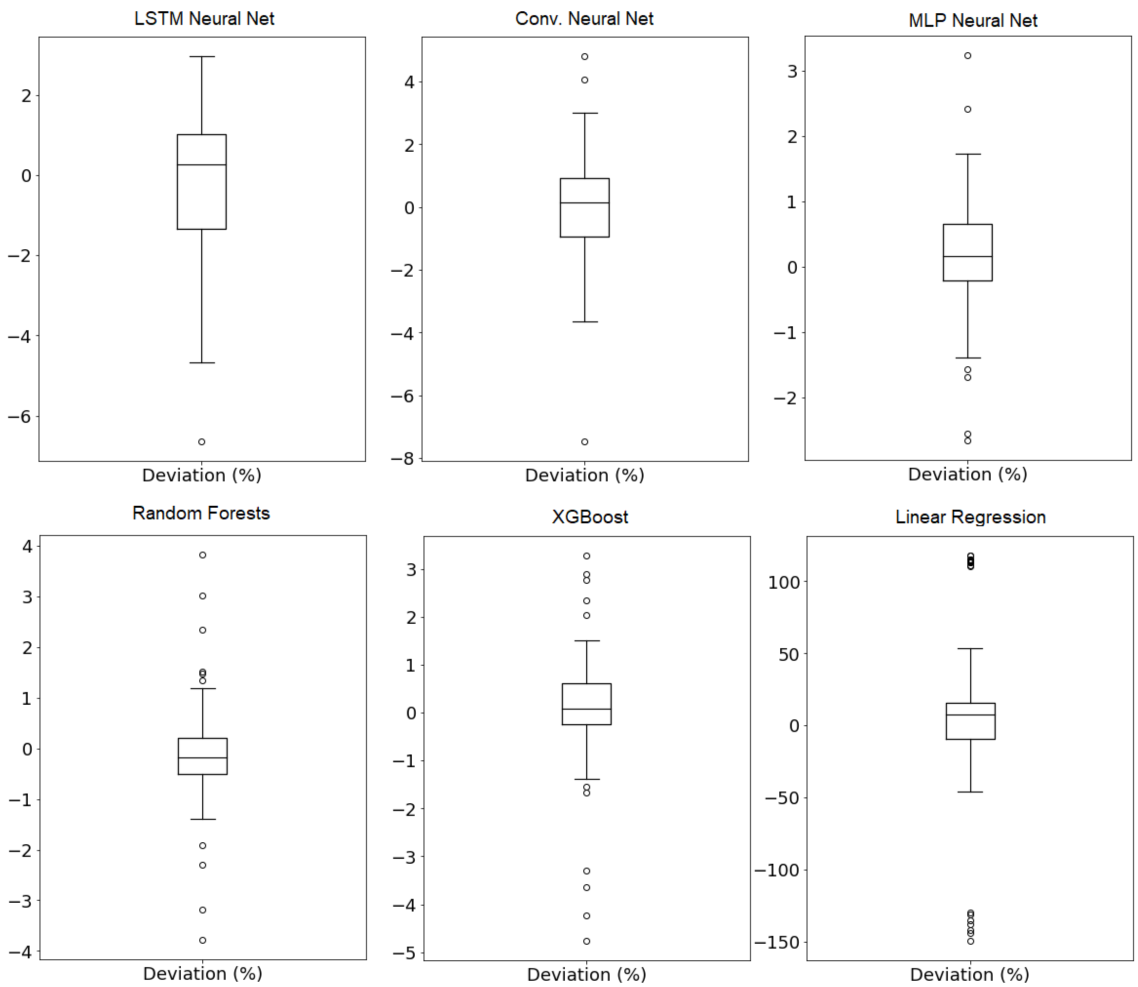

Figure 4: Distributions of the percentage deviation between the predicted and actual value for the open-source system using the complete dataset with separated training and testing dataset.

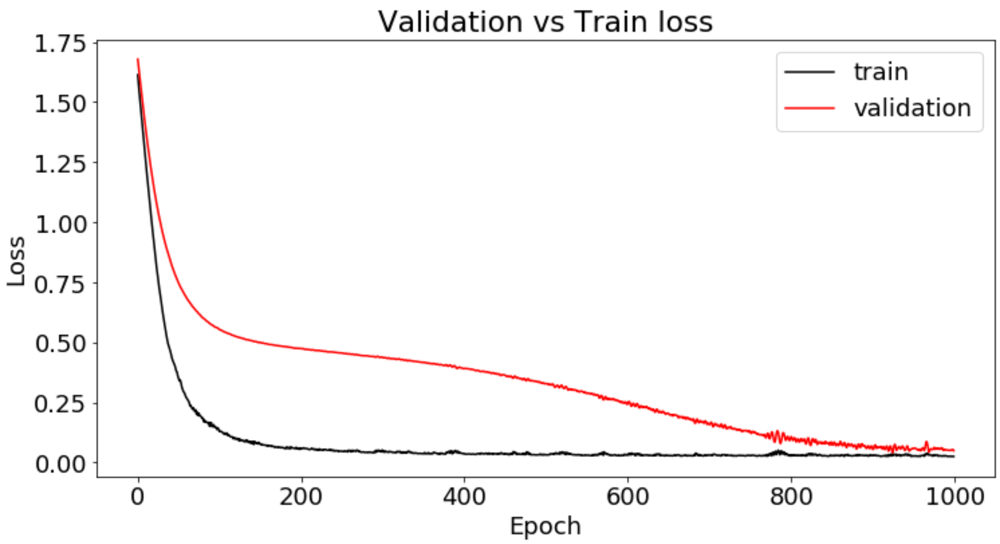

Figure 5: An example of train v.s. validation loss for the open-source system from the MLP Neural Net model results built using a subset of the complete dataset.

Our prediction models can forecast a system's throughput given a set of configuration parameter values with very high accuracy. Specifically, machine learning models like XGBoost and Multi-Layer Perceptron (MLP) neural networks achieve the best prediction results, with Mean Absolute Percentage Error (MAPE) values ranging from 0.81% to 8.2% and a low median percentage deviation. However, other ML models—such as linear regression, Convolutional Neural Networks (CNN), and Long Short-Term Memory (LSTM) networks—have lower prediction accuracy and exhibit higher variation in prediction results across the studied systems.

3.3.2 RQ2: What is the prediction accuracy by training the models using a small number of test runs with different configuration parameter values?

Motivation: Predictive ML models usually require large sets of data in order to train [4, 98] in order to provide useful predictions. Since running load tests can be resource intensive, load test engineers may not be able to run a large number of load tests with a wide range of configuration parameter values in order to gather a large data set for building a high performance capacity prediction model. Therefore, in this research question (RQ), we investigate whether we can accurately predict system throughput using machine learning models trained on a smaller number of test runs. The results would provide insights into the test execution time that can be saved by using our approach to predict system throughput given different configuration parameter values.

Approach: For the open-source system, we used a small subset of data—specifically, 30 test results out of a total of 900. These 30 test results were randomly selected to build the models. We then evaluated the models on several hundred randomly selected test results from the remaining data. We followed a similar process for the enterprise systems, using only a small subset of the test results to build the models. To evaluate the models, as in the first research question (RQ1), we reported the R2 score, Mean Absolute Error (MAE), and the percentage deviation between the predicted and actual throughput.

Results: We observe that when we used only a subset of the data for model training, XGBoost achieves the best prediction results with the lowest variability compared to other machine learning models. Table 3 shows the prediction results when using a subset of the test results to train the model. We find that, in general, the prediction accuracy has decreased compared to using the entire dataset.

The Mean Absolute Error (MAE) values for the open-source systems have increased to a range of 4,154 to 64,867, compared to 2,110 to 67,751 in RQ1. Similarly, the Mean Absolute Percentage Error (MAPE) values for the open-source systems range between 1.06% and 36.44%, depending on the algorithm used to solve the regression problem, where lower values indicate higher prediction accuracy. Notably, XGBoost achieves a MAPE of 1.1% to 2.8%, demonstrating that the prediction results are comparable to those obtained when training the models using the entire dataset. Our findings show that even though the overall prediction performance has decreased, the models can still achieve good prediction results. We observe that some algorithms are less stable, exhibiting a higher percentage deviation compared to others (Table 3). Specifically, LSTM models are the least stable and have the highest median percentage deviation across all studied systems, ranging from -13% to 6.5%. Additionally, as shown in Figure 6, there are certain test results in the open-source system where the percentage deviation between the predicted and actual throughput is very high (i.e., outliers in the box plot).

Table 3: Model evaluation results when using **a subset of the entire data** with separated training and testing. We excluded the MAE, MSE and RMSE for the enterprise systems due to NDA.

System Type	**Model**	**R2 Score**	**Median (%) Deviation**	**MAPE**	**MAE**	**MSE**	**RMSE**
Open-Src.	XGBoost	0.99889	0.5694 %	1.0672 %	4,154	7.8966e7	8,886
	Random F.	0.99893	0.5315 %	1.4713 %	5,539	7.6122e7	8,724
	MLP NN	0.99953	2.1227 %	2.6546 %	4,392	3.3086e7	5,752
	CNN	0.96168	-2.2649 %	15.5894 %	29,567	2.7289e9	52,239
	LSTM NN	0.96830	-13.0955 %	14.7153 %	37,675	2.1557e9	46,430
	Linear Regr.	0.89828	10.4876 %	36.4431 %	67,751	7.2433e9	85,108
Entprz. 1	XGBoost	0.99159	0.0262 %	2.7578 %		—	
	Random F.	0.94208	-0.6794 %	5.0949 %			
	MLP NN	0.98547	1.5053 %	5.7127 %			
	CNN	0.98269	1.3599 %	4.5767 %			
	LSTM NN	0.90235	6.5161 %	17.8877 %			
	Linear Regr.	0.96475	8.4154 %	14.4674 %			
Entprz. 2	XGBoost	0.99954	-1.0259 %	1.7781 %		—	
	Random F.	0.99367	-1.2040 %	7.8301 %			
	MLP NN	0.99192	7.1343 %	13.3365 %			
	CNN	0.94616	-4.2667 %	36.1945 %			
	LSTM NN	0.95059	-13.1491 %	13.9580 %			
	Linear Regr.	0.96084	2.5049 %	8.18635 %			

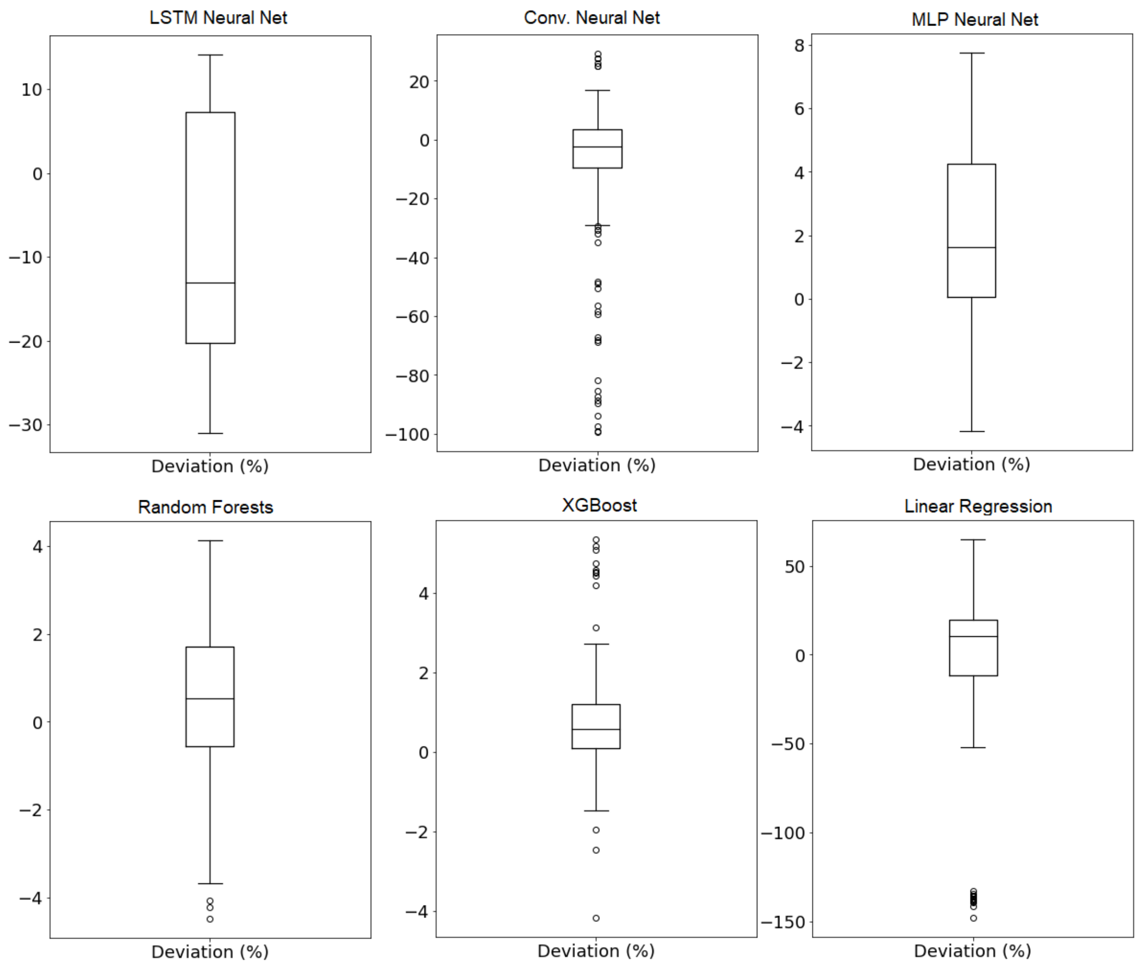

Figure 6: Distributions of the percentage deviation between the predicted and actual value for the open-source system using a subset of data to train the models.

Table 4 presents the mean evaluation metrics obtained when applying leave-one-out cross-validation on the subset dataset. We observe that for some models (e.g., LSTM), the MAPE achieved in leave-one-out cross-validation is significantly different from that obtained using separate training and testing subsets. This suggests that, due to the smaller size of the training and testing data, some models may experience overfitting issues (i.e., different runs yield varied results in leave-one-out cross-validation). Nevertheless, XGBoost models achieve very similar results when using both separate training and testing datasets and leave-one-out cross-validation. Our findings indicate that XGBoost models provide the best prediction results and are the most stable when trained on a subset of data. Therefore, future studies and practitioners may prefer XGBoost models for capacity prediction when only limited training data is available.

MLASP can still achieve excellent prediction results for throughput when trained on a small subset of data (i.e., 3% of the open-source data and a subset of the enterprise systems data). We also find that XGBoost provides the best prediction results—with MAPE values ranging from 1% to 3% across all studied systems—and exhibits the most stable performance when the training data is limited.

3.4 Discussion and Implication of Our Findings

In this section, we delve into the lessons learned from conducting our experiments and integrating our approach into the enterprise systems. We highlight key insights and practical implications that emerged during this process. We also highlight some key findings from studying the Apache Kafka open source system.

3.4.1 Understanding the Effect of Configuration Parameters

As discussed in prior research [69], we find that developers may not fully understand the impact of a configuration parameter once the system is deployed in a distributed and complex setting. Consequently, load test engineers need to spend considerable time testing the system under various configuration parameter values, which can be costly or even impractical. By building models, we can comprehend the importance of each feature (i.e., configuration parameter) in relation to the target variable (i.e.,

Table 4: Mean of model evaluation results when using **a subset of the entire data** with separated training and testing and leave-one-out cross-validation technique on the training dataset. We excluded the MAE, MSE and RMSE for the enterprise systems due to NDA.

System Type	Model	R2 Score	Median (%) Deviation	MAPE	MAE	MSE	RMSE
Open-Src.	XGBoost	0.99883	0.5632 %	1.0795 %	4,242	8.3032e7	9,092
	Random F.	0.99877	0.7213 %	1.5241 %	5,857	8.7052e7	9,252
	MLP NN	0.99660	-4.0275 %	7.8962 %	11,654	2.4191e8	14,454
	CNN	0.88445	-4.8323 %	23.8084 %	51,175	8.2286e9	87,672
	LSTM NN	0.83048	-33.5486 %	47.4998 %	73,846	12.149e9	90,930
	Linear Regr.	0.89742	9.2974 %	36.4756 %	67,810	7.3047e9	85,466
Entprz. 1	XGBoost	0.96354	0.0879 %	3.6777 %		—	
	Random F.	0.89339	0.7440 %	7.3675 %			
	MLP NN	0.91848	1.9285 %	5.7111 %			
	CNN	0.88238	2.6036 %	7.0554 %			
	LSTM NN	0.89979	7.2277 %	25.8928 %			
	Linear Regr.	0.86504	9.6058 %	21.4257 %			
Entprz. 2	XGBoost	0.98516	0.5902 %	2.0413 %		—	
	Random F.	0.94357	0.7911 %	13.3980 %			
	MLP NN	0.92409	8.1647 %	19.3214 %			
	CNN	0.88860	-6.4085 %	48.8870 %			
	LSTM NN	0.92140	-15.6541 %	21.1542 %			
	Linear Regr.	0.82025	3.0043 %	11.4730 %			

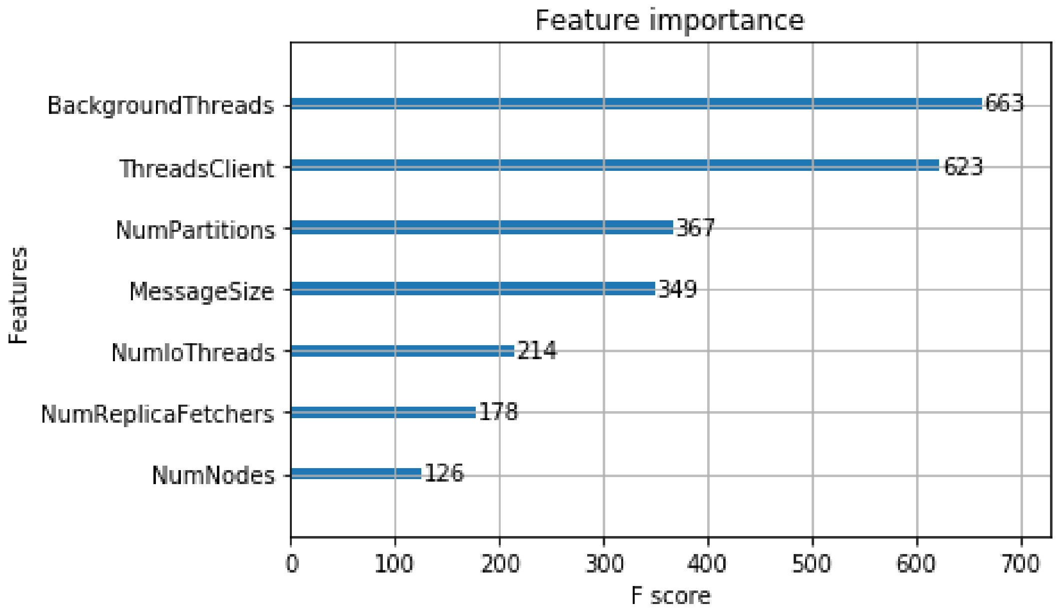

Figure 7: Feature Importance: Open source system parameters that have the most impact over the load test results.

KPI). This approach allows us to detect upper boundaries where increasing the parameter values further does not significantly affect the KPI.

The importance of the features can be determined directly from the ML models using specialized libraries that provide such insight. Figure reffig:mlasp-feature-importance, depicts the feature importance of the parameters for the open source system, correlated with the performed load test data.

For example, in the open-source system, we observe that the throughput remains unchanged once the number of background threads in a Kafka broker reaches a certain threshold. After examining the feature importance in the model, we find that the combinational effect of configuration parameters is more significant than any single parameter. In addition to the number of threads, the model results also indicate that message size is an important feature. Our investigation reveals that the overall input/output operations per second (IOPS) of the disk and network greatly influence the total number of messages the system can process, regardless of how many background threads we use. By utilizing MLASP, we can model the combinational effects of configuration parameters and environment/deployment settings while running fewer load tests. This approach enables us to provide more informed suggestions to load test engineers and developers regarding system capacity.

We observed similar benefits in the enterprise systems. Specifically, MLASP not only enhances our understanding of the importance of configuration parameters but also provides strong evidence for fine-tuning scaling strategies. For example, knowing the IOPS limits and requirements of a system aids in virtual machine allocation and relocation strategies for multi-tenant hardware within private clouds and data centers. The models also assist in test automation by reducing the number of tests needed to tune or verify less important configuration parameters, leading to an overall reduction in load testing time. Overall, we find that MLASP can lead to significant cost savings, streamline the CI process, and enhance the efficiency of load testing activities.

3.4.2 Integrating MLASP in Industrial Setting

There are various methods to integrate MLASP into industrial settings, such as modifying the source code to include self-monitoring and self-tuning functionalities. Ultimately, MLASP employs a non-intrusive approach for data collection and configuration parameter tuning. As discussed in Section 3.2, we calculate system throughput by leveraging readily available information. Our approach requires no changes to the system's source code and does not affect the testing activities that are part of the enterprise systems' software development process. This non-intrusive method was highly regarded by the project management team, as it avoided adding additional costs to the project. Consequently, it did not jeopardize timely delivery plans and release roadmaps, which ultimately led to the integration of MLASP. Future studies may consider such a non-intrusive approach to increase the adoption of the developed methods.

Beyond delivering excellent prediction results, MLASP also assists load test engineers in identifying combinations of configuration parameter values that yield specific outputs. When integrating our approach with the industrial system, we discussed with our industrial partner—including project management and technical personnel—whether the models could be used for capacity planning in addition to capacity prediction. For example, our industrial partner might be interested in knowing the expected throughput if the number of deployed nodes decreases by two. If the model can provide accurate predictions by inputting the desired variable values into the trained models, it can offer an accurate estimated outcome. Essentially, MLASP helps load test engineers model the scalability of the software system (e.g., determining the types and amounts of resources needed to achieve a desired throughput based on forecasts of future traffic needs). Such inverse prediction or classification is commonly used in statistics and machine learning to understand the effect of variables and to aid in business decisions [1, 19, 69]. MLASP also helps load test engineers identify potential physical limitations of the system hardware by studying the correlation between features (i.e., configuration parameters) and throughput, as discussed in Section 3.2. This information was highly valuable for reporting purposes to both management and development teams when simulating "what-if" scenarios. The results from these simulations highlighted the relationships between different parameters—such as underlying infrastructure and software configurations—and provided important insights to development teams on where to focus improvement efforts in the next iteration. Project management also gained a clearer understanding of the roadmap and were able to create risk mitigation plans more quickly (e.g., determining when to order additional hardware and when and what communications should be given to the customer regarding the roadmap).

Given the high level of interest from our industrial partner, we also adapted this tool for use with the open-source system. We added the adapted code to the public GitHub repository, sharing the data and algorithms [83].

During our integration process, we also enhanced the existing load testing process by improving automation. Specifically, we integrated the previously mentioned random configuration generator tool and the machine learning model training capability. Due to the NDA, we cannot disclose which tools are used in each phase by our industrial partner.

3.5 Threats to Validity

In this section, we discuss the threats to validity related to the MLASP framework.

3.5.1 External Validity

We conducted our experiments on both open-source and enterprise systems. Ultimately, MLASP was well received by our industrial partner, and the prototype has been integrated into their software development process. Although our findings are consistent across the studied systems, these results may not generalize to other systems. We believe that the approach may be easily extended to any type of IT system that takes in a request, takes some time to process it and provides a response (the input and output payloads may differ in structure and size).

3.5.2 Internal Validity

Since enterprise systems are continuously evolving, changes in the source code can affect the system's throughput. Therefore, we aimed to use the same release of the system throughout the process to minimize the impact of new code changes. For the open-source system, we did not use any system KPI metrics (e.g., CPU usage), as the underlying hardware and virtual server capabilities may experience some noise (e.g., context switches or garbage collection overhead). Therefore, we chose to model the KPI that better reflects users' perception of the system—namely, throughput.

3.5.3 Construct Validity

Previous studies [44, 88] have shown that parameters in machine learning models can significantly affect model performance. Therefore, in our experiments, we applied a semi-automated analysis to tune a wide range of model parameters using a validation set. Other potential issues with machine learning models include multicollinearity and overfitting. To mitigate these risks, we split the data into training, validation, and testing sets to avoid training biases. We also applied different regularization methods to reduce the possibility of overfitting. Our prediction results on external test datasets demonstrate that our models perform very well and are similar to the results obtained on the validation set. Thus, the models are not suffering from the problem of overfitting. However there is an aspect to consider that due to the technique we used, the models are not guaranteed to cover the worst case scenarios.

Part II

Load and Endurance Testing For Large Scale Systems

Chapter 4

Enhancing the Effectiveness of Load Testing for Large Scale Systems

This chapter presents an empirical study that examines the challenges of the load testing process for large scale industrial systems and our proposed blueprint for a framework and process that may be used both during the project phase and the production phase. The blueprint we propose is extensible and generalizes on any configurable software systems.

We have learned from prior studies [32] how costly load-related issues can be for companies. Therefore, having proper capacity planning is critical for businesses. However, capacity planning heavily depends on load testing, which, in turn, may require significant resources and time; thus, having an efficient load and endurance testing process is also crucial for the software development life cycle (SDLC).

Additionally, software systems under load may exhibit spikes in various metrics. Neglecting these spikes has often led to more serious problems causing outages [40]. Therefore, it is important to understand the nature of spikes, their causes, and their potential ripple effects within the system. In other words, having a good spike detection system in place can help correlate events among the different components of a software ecosystem, making it easier to isolate and study their effects. This aspect is extremely useful when spikes cause ripple effects that destabilize later parts of the ecosystem, as those events are hard to reproduce—especially when the starting conditions are unknown (e.g., a spike in the load of a component may have caused a buffer overrun, leading to loss of messages or corruption of other data).

An earlier version of this chapter has been published in the Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering - Pages 1956–1966 · Oct 28, 2024. Vitui and Chen [115]

4.1 Motivation

In this section, we present the challenges faced by our industrial partner, Ericsson Inc., during endurance testing execution.

Load test engineers are required to perform both load and endurance testing. An endurance test is a specialized load test that aims to observe how a system behaves under prolonged load conditions, potentially running for hours or even days [21, 56]. The first challenge is for load test engineers to identify as early as possible when a load or endurance test is failing and to stop its execution, given the limited time and resources available for the project. Endurance tests may fail for various reasons, such as:

- **Slowdown of the end-to-end system performance** due to bottlenecks or overload in any number of components within the software system. For example, the system's throughput drops below a defined threshold for a specific duration—for instance, in the last 5 minutes, the system throughput fell below 50 transactions per second (TPS).
- **Intermittent information loss**, which is an escalation of the previous condition where the system's processing speed has decreased and it is also losing some of the received requests. For example, in the last five minutes, the throughput decreased below 50 TPS, and/or three percent of the messages were lost.
- **Encountering a malfunction or critical error** in one or more components of the software ecosystem—an exacerbation of the previous condition—that ultimately leads to erroneous behavior across the entire software ecosystem. For example, if the database backend becomes unavailable, all processing results in errors.

This means that different types of spikes can occur in the software ecosystem during load testing, potentially leading to adverse effects. This is a serious challenge that needs to be addressed.

Another challenge, faced by the operations team, is to anticipate when an increase in load might occur so that preventive measures can be taken to maintain the agreed Key Performance Indicators (KPIs) for the software system. In other words, there is a spike-capacity forecasting challenge that needs to be solved.

4.2 Case Study Setup

4.2.1 MLOLET Framework Proposal

Our objective in improving the load and endurance testing process is to provide a generic process blueprint that can be adopted in an industrial setting with minimal effort.

We propose enhancing the load and endurance testing procedures by incorporating an online spike detection mechanism for the load test data. We introduce an approach called MLOLET (Machine Learning Optimized Load and Endurance Testing) that aims to detect spikes in KPI data using an online method. Being an online detection system means that MLOLET is expected to detect spikes while the load test is being executed. Having an online spike detection system is the first step in addressing the initial challenge presented by the load testing team of our industrial partner.

As mentioned before, an endurance test may fail if the system experiences a large number of spikes (e.g., the system's response time is outside the normal distribution) within a given period. With this insight, load test engineers can stop failing tests early, saving time and costs in the testing process.

To solve this challenge, we suggest including a configurable component in the load testing practice that can provide the following functions:

- **Alert** (by email or on a central monitoring system dashboard) the load testing team if the number of anomalous events in the load or endurance test exceeds a certain threshold.
- **Stop** the load or endurance test if the number of spikes in the test exceeds a certain threshold.
- **Begin** a new load or endurance testing cycle if the number of spikes in the test exceeds a certain threshold.

To address the second challenge, we propose a time-series-based load testing framework that:

- **Forecasts** the system's trend in an online setting.
- **Increases** the efficiency and project economics of the load testing team.

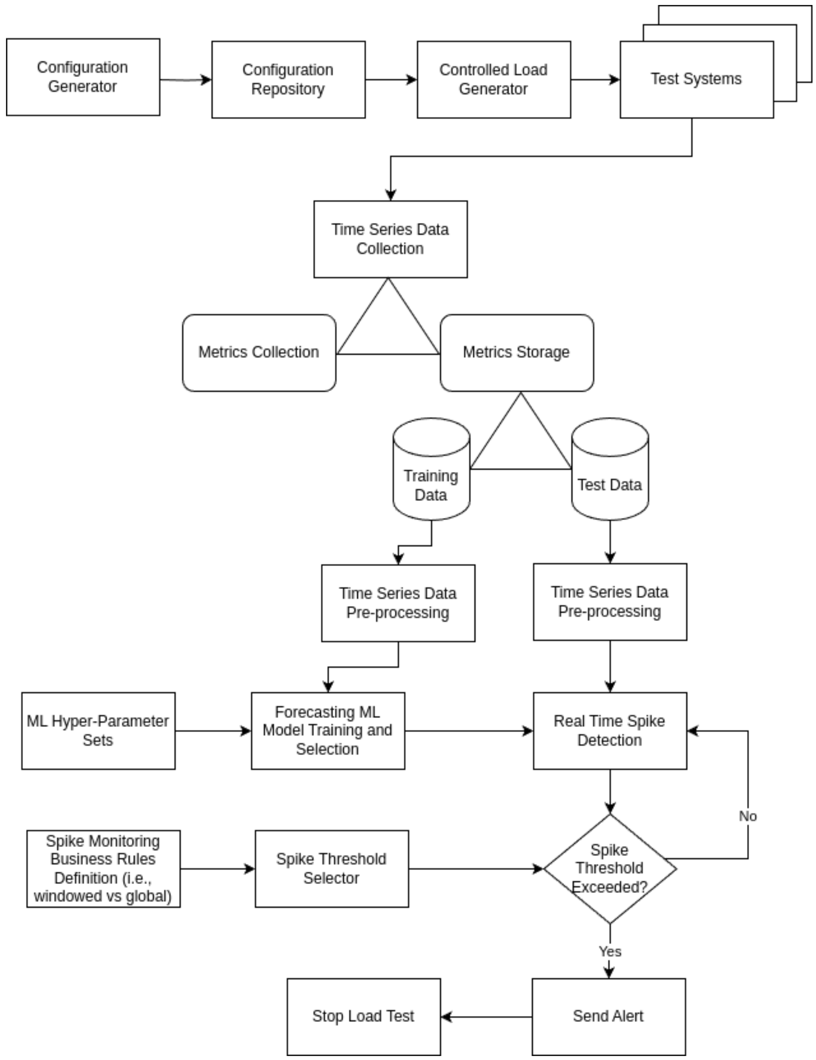

Figure 8: MLOLET framework overview.

- **Improves** the operational performance and economics of the operations team regarding service upkeep and associated SLA fulfillment.

Figure 8 depicts the overall process of MLOLET.

One of the most important building blocks of the framework is the 'Real Time Spike Detection' one. The pseudo-code for this procedure is detailed in Algorithm 1.

Various tools are available for each stage of the process blueprint. Defining a tool-agnostic process is important because it increases the likelihood of adopting the framework in an industrial setting, where, depending on business needs and regulations, certain tools must be used for specific processes. This is very similar to the

Algorithm 1 MLOLET Real Time Spike Detection

Require: Y_pred_train ▷ The list of predictions on the train data
Require: Y_true_train ▷ The true values of the train data
Require: $error\_selector$ ▷ The error function selector
Require: y_pred_test ▷ The prediction of the test/new data
Require: y_true_test ▷ The true value of the test/new data
Require: $is\_train\_configuration\_flag$ ▷ Is 1 if the test values are from the training set

```
function CALCULATE_ERROR(Y_true, Y_pred)
    if error_selector is Absolute_Error then
        errors ← | Y_true − Y_pred |
    else if error_selector is Squared_Error then
        errors ← (Y_true − Y_pred)^2
    end if
    return errors
end function
errors ← CALCULATE_ERROR(Y_true_train, Y_pred_train)
err_mean ← mean(errors)
std_deviaton_err ← std(errors)
up_threshold ← err_mean + 3 ∗ std_deviation_err
low_threshold ← err_mean − 3 ∗ std_deviation_err
repeat
    deviation ← CALCULATE_ERROR(y_true_test, y_pred_test)
    if is_train_configuration_flag ≠ 1 then
        deviation ← deviation + std_deviation_err
    end if
    if low_threshold ≤ deviation ≤ up_threshold then
        spikes ← unchanged
    else
        spikes ← increased
    end if
until there is new data.
```

MLASP [114] process we defined in Section 3.2.1. In fact, the automation part of the load testing process defined by MLASP is vary much applicable here, with the distinction that we are interested in collecting different type of metrics. We describe next the possible tools tailored to the purposes of MLOLET.

The configuration generator is a component-specific tool that can be developed by software engineers—testers and developers alike—using any available scripting or programming language (e.g., C/C++, Python, Bash scripting, Java, Ruby). Its purpose is to provide a base of meaningful configuration sets for testing and load testing purposes of a specific software component within the overall software system.

The generated configurations can be stored in a repository, which may be a database (relational or NoSQL), a Git repository, or simply a folder containing text-like documents (e.g., YAML, CSV, XML, or JSON formatted files).

Subsequently, the controlled load generator will use the configuration repository to select available configurations, apply them to the tested components, and then initiate a load test. Automation is key in this case, and there are several specialized tools available to accomplish the necessary steps. For example, specific component configuration settings (i.e., the parameters of the tested software component) and component scalability values within the test environment can be controlled by specialized tools such as Ansible [5], Chef [15], or Puppet [93], or by in-house automated scripts or configurator applications. Selecting the right tool for the job will depend on the specific behavior of a software component with regards to reconfiguration: some software components may expose a reconfiguration interface (e.g., JMX interface) that accepts commands to adjust internal parameter settings. Other software components may require setting specific values in a configuration file and might need a restart if the configuration is not automatically reloaded upon detecting a file change.

The load testing step can be accomplished using either custom-developed applications or specialized tools such as JMeter [34], Jenkins [54], SoapUI [107], etc.

To maximize system portability, we collect the desired metrics—including both computing nodes' resources and the application's processing metrics (throughput per unit of time and associated average response time)—in a non-intrusive manner. Methods for this include log parsing and aggregation using tools such as LogStash [77] or ElasticSearch [29], or even through in-house developed scripts. The collected metrics can then be stored in various types of databases (relational, time-series, NoSQL), or even as simple text files (e.g., CSV, JSON-formatted documents).

Next, we propose using machine learning to help solve the previously mentioned challenges. There are many libraries available for data preprocessing and machine learning-based modeling, such as Apache Spark [8], TensorFlow [111], PyTorch [95], etc. Once a model is trained and validated, model serving can also be achieved through various tools: Apache Spark [8], Flask [33], Seldon [104], etc. This step is, however, somewhat dependent on the modeling tools and approach selected in the previous step.

It is noteworthy that MLOLET's spike detection relies on an online detection approach. Machine learning offers several architectures that function in an online setting; however, this is not a prerequisite, as other techniques may be used as long as they operate in an online setting (e.g., the VARMA statistical method [78]).

Another important aspect of the spike detection is that we aim do detect both upwards and downwards occurrences. This is important as a sudden drop may be an indicator of a malfunction, where the sudden increase may be an indicator of a capacity problem (a bottleneck processing flow). In Figure 9, we see traffic samples from an application running under three different load and configuration settings. In the first example the systems seems to struggle with requests processing and eventually fails; the second scenario shows that everything performs as expected, where the third one shows that despite some troubles (spikes), the system is eventually recovering and able to continue performing nominally.

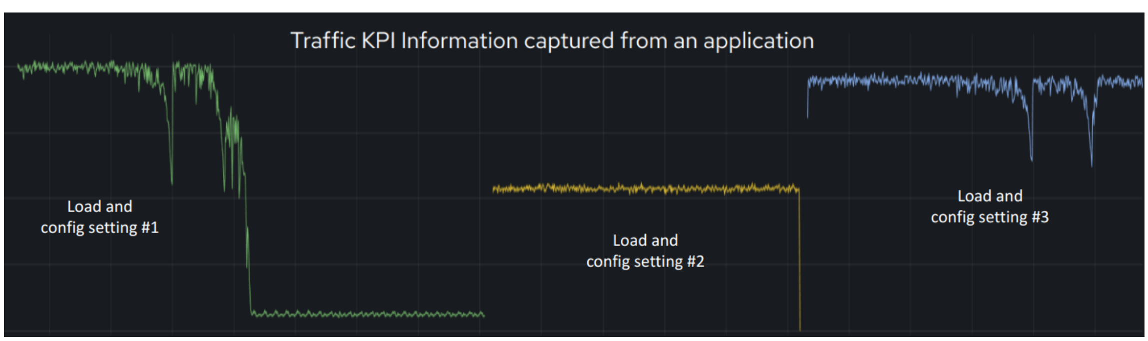

Figure 9: Traffic KPI information captured from a test application running under different load and configuration settings.

4.2.1.1 Using MLOLET Models for Traffic Trend Prediction

In Figure 8, we presented our proposed process for using spike detection to early-stop failing load and endurance tests. However, this only addresses the first challenge. To tackle the second challenge, we assess how suitable the models trained with the existing load testing data are for predicting future trends in the load testing sequence. The proposed process is depicted in Figure 10.

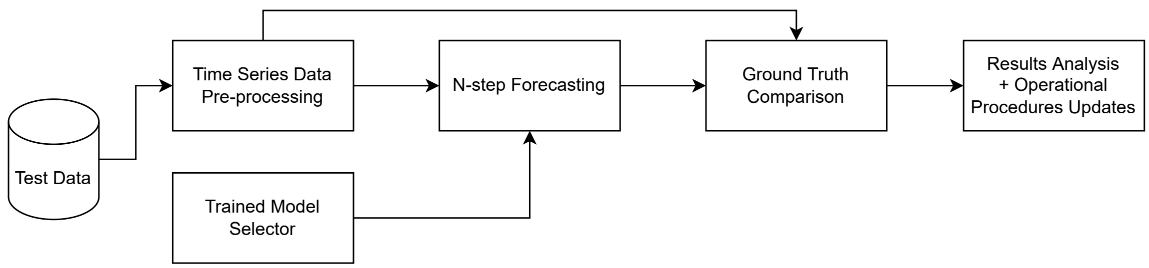

Figure 10: MLOLET traffic forecasting process overview.

Engineers can test how different models perform in trend forecasting by comparing their predictions with the ground truth after a predetermined number of steps or iterations (denoted by 'N'). Based on the conclusions drawn from this comparison, engineers can devise new or update existing operational procedures to ensure that the Service Level Agreements (SLAs) of the software systems are maintained within the desired boundaries.

4.2.1.2 Defining Spikes

Spike detection involves identifying data points that significantly deviate from the majority of the data. Anomalies are highly context-dependent, meaning that what constitutes a spike can vary based on the situation. Generally, a spike is an event that does not conform to any established pattern or group within the data. In the context of time series, a single value may be considered a spike based on its timing and the preceding values. For example, if an application's CPU load typically hovers around 10%, a sudden jump to 70% would be considered an extreme event or spike. However, if the CPU load consistently remains around 70%, that jump is no longer unusual and is instead part of normal behavior.

Based on this example, we found that simple threshold-based monitoring is insufficient to handle complex real-world situations. More advanced methods are needed.

One such method uses the 3-sigma statistical rule (also known as the empirical rule) to determine if the next point in a time series is a spike. The 3-sigma rule is a common guideline in science, stating that in a normal distribution, most values fall within three standard deviations of the mean [52, 109, 131]. Therefore, we first measure the mean and the standard deviation (σ) of the data points in the training data. Then, we define our error thresholds as: $\epsilon_{1,2} = \text{mean} \pm 3\sigma$. Any data point value outside of this threshold interval is considered a spike. As described earlier by Algorithm 1, the data points considered here are actually the calculated error between the true value of the measured indicator and its predicted value (produced by the employed ML algorithm).

As previously mentioned, our experiments use point-in-time values from two KPI indicators: request throughput rate and request response time, collected from the software system. These data sets are gathered during load testing cycles, which may vary in duration and load, as shown in the overall process diagram (Figure 8). Each load testing cycle therefore provides two distinct time series that we can use for model training.

4.2.1.3 Performing Load Tests

We performed load tests using various configurations. During these tests, we monitored system configuration variables, including both internal component settings and deployment factors (such as the number of component instances). Additionally, we recorded the actual load applied to the system for each configuration.

The load applied to the systems used several types of request that had different payloads, namely different structure and overall size (e.g. 50, 150 and 255 bytes respectively). These requests were randomly selected. The response payloads for these requests had also different payload sizes, and each payload depended on the result of processed request.

With these settings, we collected time-series data for each key application metric—specifically, throughput per unit of time and average response time—by running multiple load tests with different workload types and configurations. This data allows us to explore whether we can train a time-series model using data from one specific test and then evaluate the model on other tests with different configurations.

4.2.1.4 Model Classes in MLOLET

Based on MLOLET's requirements for online testing and evaluation, two types of spike detection algorithms are suitable for time-series analysis: traditional statistical models and artificial neural network-based models.

Importantly, the MLOLET process does not mandate the use of a specific model. Instead, it is designed to be horizontally extensible, allowing multiple models to be used simultaneously if needed for different scenarios.

Traditional (or statistical) models. We use the Vector Auto-Regression Moving Average (VARMA) [78] as our traditional reference model. The VARMA method extends the ARMA model to handle multivariate stationary time series by combining Vector Auto-Regression (VAR) and Vector Moving Averages (VMA) to predict the next step in multivariate time series data. We have chosen VARMA as a reference model because two of its characteristics make it well-suited for comparison with newer deep learning algorithms.

First, like deep neural networks (DNNs [130]), the VARMA algorithm can take a custom-defined input time series sequence to predict the next value. Second, the length of the input sequence in VARMA can be adjusted as a hyperparameter during training, similar to how it is done in DNNs. These two similarities make it easy to use VARMA in an online prediction setting, just as you would with a DNN-based model. Because of these shared features, comparing the results of VARMA and neural network-based models becomes more straightforward.

Artificial Neural Network (ANN) based models. Artificial neural networks are highly effective at generalizing across different types of datasets and are equally adept at modeling time series data. Previous studies [53, 72, 110] have demonstrated that specific types of neural networks are particularly well-suited for time series analysis. In our tests, we evaluated the following architectures: Convolutional Neural Networks (CNN) [130], classical Recurrent Neural Networks (RNN) [130], and variants such as Long Short-Term Memory networks (LSTM) [130] as well as Bidirectional Recurrent Neural Networks [14]. For more advanced architectures, we also evaluated ResNet [49]-type models and LSTM-AutoEncoder [132]-based approaches.

A Recurrent Neural Network (RNN) is a type of artificial neural network (ANN) where connections between nodes form a directed graph along a temporal sequence. Unlike traditional feedforward networks, RNNs have internal memory that allows them to process sequences of varying lengths by retaining information from previous steps in the sequence. An LSTM is a variant of RNN that remembers the order of data in sequences during the training process [30, 123].

A key limitation of standard RNNs is that they can only utilize past information without access to future context. Bidirectional Recurrent Neural Networks (B-RNNs) [102] overcome this limitation by employing two separate recurrent layers. One processes the input sequence in the forward direction, while the other processes it in reverse. Both layers are connected to the same output, enabling the network to consider both past and future contexts for a more comprehensive understanding of the input. Bidirectional LSTMs (B-LSTMs) combine the principles of bidirectional networks and LSTMs [14].

As highlighted by Zhu and Laptev [132], neural networks can encounter prediction uncertainties, which may result in false anomaly alerts. To mitigate this, they proposed an autoencoder-based architecture. Similarly, in our case, we consider spikes as anomalies, even though they may not always qualify as such, as discussed earlier. The encoder-decoder layers in the autoencoder are designed to extract key features from the input time series, leveraging the common use of autoencoders for dimensionality reduction and feature extraction [11]. Essentially, the autoencoder functions as an intelligent feature extraction mechanism [132].

Another type of network that excels at feature extraction is the Convolutional Neural Network (CNN), renowned for its ability to automatically detect important features without human intervention [37, 59]. This capability allows CNNs to learn patterns from sequential data, making them valuable for both spike detection and forecasting.

ResNet (Residual Networks), introduced by Microsoft Research in 2015 [49], is an advanced Convolutional Neural Network (CNN) architecture designed to address the vanishing and exploding gradient problems in deep neural networks. According to He et al. [49], ResNet stands as one of the most sophisticated CNN architectures available. Its effectiveness in time series analysis has been demonstrated by Fawaz et al. [53].

We evaluated the efficiency of the different models by using the same input sequence length for each, while optimizing each model according to its own set of hyperparameters.

4.2.1.5 Data Processing and Machine Learning Models Tuning

Neural network inputs are scale sensitive, as prior studies have shown [106, 108]. Therefore, several normalization methods have been utilized in defining our approach: MinMaxScaler provided by the scikit-learn [103] library and logarithmic based scaling (i.e., log transformation). Note that we retrain the models using the different scaling methods separately and only report the model that achieves the best performance.

The MinMaxScaler rescales the dataset so that the entire feature set is in the range [0, 1]. Since in general traffic data is a positively skewed distribution, we also considered log transformation as a second option for data normalization [113].

Furthermore, we also apply grid search optimization for hyperparameter tuning in the case of the neural network based models.

We used the same range of values for all the parameters within the hyperparameter tuning step. The following common neural network parameters have been tuned in the process: number of neurons per layer [100 - 250], the learning rate [10e-4, 10e-3, 10e-2], and the number of hidden layers [1 - 10]. For each pass we consider different data window sizes, in the range [5 - 120]. An input data window (T) is the number of time steps to use for predicting the next value in the series. The offset represents the number of steps forecasted from the selected data window (e.g., an offset of one is equivalent with having one forecasted value). Figure 11 provides a visual representation of the data windowing concepts with $T = 6$, the input width, label $width = 1$ as the length of the time unit and $offset = 1$ being the number of steps forecasted.

We evaluated our time-series forecasting models using several metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE) and the Pearson Correlation Coefficient (PCC) value. The definitions of these metrics have been provided in Section 2.4.

4.2.2 Studied Systems

We verify our experiments on two sets of studied systems: enterprise and open source.

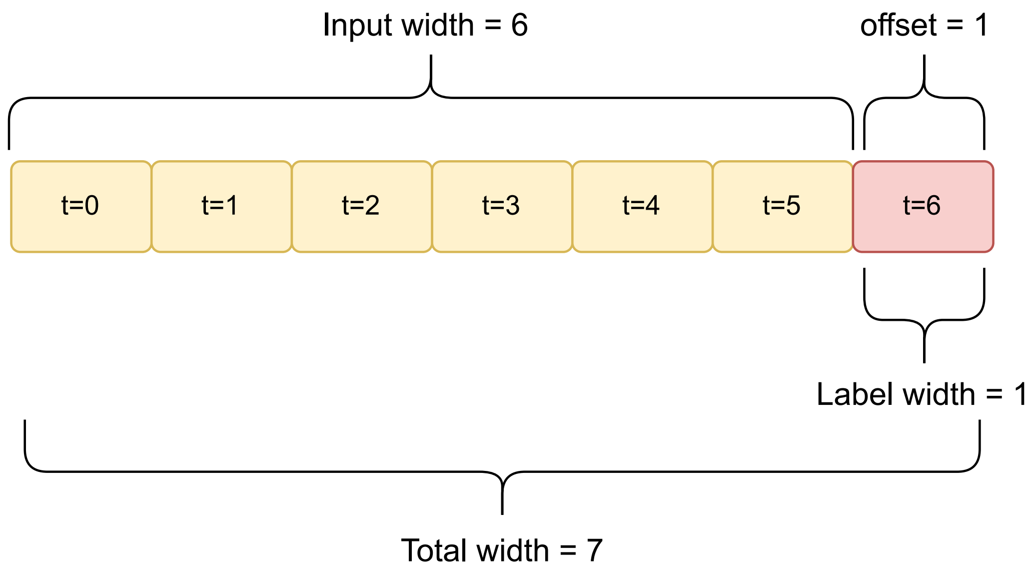

Figure 11: MLOLET Models Data windowing.

4.2.2.1 Enterprise Systems

In this section, we present, within the limits of the NDA boundaries, the studied enterprise system and the available metrics that may be monitored in order to implement our proposed framework. The system offers Business-to-Business (B2B) and Business-to-Consumer (B2C) messaging services, combining Ericsson's in-house products, open-source software, and third-party commercial solutions. Since these functions are business-critical, the system is built for high availability with local redundancy across multiple data centers. It is used daily by millions of users worldwide and processes tens of millions of requests.

In general, there are many metrics available to measure a complex system's performance when running load testing. Common performance metrics include CPU, memory, and disk usage, and application specific metrics such as the throughput per unit of time (transactions per second - TPS, and transactions per minute TPM, etc.) and the associated average component response time for the selected unit of time.

To generalize the approach we imply the use of the following two metrics: 1) how many requests the software component is capable of processing within a desired time unit (i.e., *throughput*); and 2) how fast the software component is processing those requests (i.e., *response time*).

4.2.2.2 Open Source System

To validate our findings and ensure reproducibility, we also run our experiments on open-source software. We use a custom developed load generator, code available in this GitHub repository [76]. The reason we developed our custom load generator is to have a fine tuned control over the load, including the capability of emitting spikes as well as collect near-real time using Prometheus [92] metrics information about the load test such as number of processed messages (both successful and erroneous) and average latency of the responses. These three functions together are typically not available in commercial or other open source software. The test subject is a generic black box application created with the help of an open source mock application framework, WireMock [121]. We extended the framework with Prometheus [92] metrics generator and a global random string payload ResponseTransformer. The test application responds to requests by providing a string payload of a specified length, with a response delay following a user-controlled uniform distribution. Depending on the deployment configuration, the application may recursively call itself several times. This behavior, modeled in a generic way using a black-box approach, simulates a wide range of business applications, which typically share the same process: receiving a request, processing it, and returning a response. The source code for the test subject is available in the following GitHub repository [122].

The test results related to MLOLET and the Machine Learning model prototypes used by MLOLET are also available online [85].

4.3 Case Study Results

In this section, we discuss the results of our three research questions (RQs). To uphold the NDA requirements with our industrial partner we present only limited information from the enterprise system. We model generic KPI information applicable to a large number of systems both enterprise and open source, namely throughput and average response time.

4.3.1 RQ1: How do different models compare in one-step forecasting?

Motivation: To implement the early stopping mechanism proposed by the MLOLET framework, we first need to identify a highly effective forecasting model. The best-performing model becomes the baseline for our subsequent research questions. Identifying such a model and knowing its limitations is very important for the subsequent research questions.

Approach: We randomly select one load testing event. We use this event to train and test several different machine learning model architectures as defined in Section 4.2.1.4. We follow the process described in Section 4.2.1 to ensure consistency. Our primary interest is in the one-step time forecasting accuracy of the various machine learning algorithms and architectures.

Results: We find that, in general, all the models provide a good one step forecasting results. Table 5 summarizes, for the enterprise system, the best prediction result for every type of model that we experimented, where Table 6 provides the same information for the open source system. We used the normalized values of the datasets (e.g., KPIs) when calculating the model results. We find that VARMA achieves the worst results in terms of all the evaluated metrics (i.e., MAE is 0.1156 and MSE is 0.0743). On the other hand, LSTM-based models achieve the best results (i.e., MAE ranges from 0.0386 to 0.0804, and MSE ranges from 0.0131 to 0.0207). In contrast, CNN and ResNet have a slightly higher MAE and MSE compared to Autoencoder LSTM.

From the results we see that in general, the ML models are able to predict the normalized KPIs in the next time step with good accuracy. Although VARMA delivers the worst results, its training and execution time are much faster than DNN models. In situations where computing resources are limited, VARMA could still be a good option for online training and application. However, advanced neural networks remain superior for delivering more accurate one-step predictions. While we cannot disclose the specific ML model configuration values we identified for the enterprise system due to an NDA, we found that choosing the correct time window size, T, is crucial.

Table 5: Enterprise System - One step forecasting results (MLOLET-RQ1).

Model	MAE	MSE	MAPE (%)	PCC
VARMA	0.1156	0.0743	8.290	0.5965
RNN	0.0644	0.0227	1.347	0.8217
LSTM	0.0756	0.0207	1.641	0.8611
Bidirectional LSTM	0.0804	0.0254	1.720	0.843
Autoencoder LSTM	0.0386	0.0131	1.0826	0.8999
CNN	0.07173	0.0282	1.494	0.8094
ResNet	0.0687	0.0177	1.457	0.8851

Table 6: Open Source System - One step forecasting results (MLOLET-RQ1).

Model	MAE	MSE	MAPE (%)	PCC
VARMA	0.0303	0.0236	26.6544	0.9888
RNN	0.0058	0.0010	3.5584	0.9888
LSTM	0.0072	0.0010	3.4166	0.9887
Bidirectional LSTM	0.0083	0.0010	6.0844	0.9887
Autoencoder LSTM	0.0078	0.0010	12.7623	0.9890
CNN	0.0086	0.0010	6.9289	0.9894
ResNet	0.0177	0.0013	55.3107	0.9854

For 1-step forecasting, the LSTM-Autoencoder model architecture achieves the best result. Noteworthy is that depending on the required accuracy of the detection and from the project perspective, simpler models, like VARMA, may also provide reasonable results.

4.3.2 RQ2: What is the generalizability of the forecasting model when system deployment settings change?

Motivation: Spike detection mechanisms are typically designed for static systems, meaning the system's internal configuration remains unchanged regardless of the applied load. This setup doesn't account for changes in the number of nodes added for redundancy or capacity scaling. In load testing, however, applications or ecosystems of multiple applications often require parameter tuning. It would be inefficient for load test engineers to retrain the model every time the internal configuration changes. Therefore, we compare the forecasting performance of each model both when the system's internal configuration changes and when the type of business request changes.

Approach: Each load test varies either the system configuration or the applied load or both. For this research question, we take a model trained with one of the randomly selected load tests data and verify the forecasting capabilities on all the remaining data sets. In other words, we take the model trained on a specific configuration and apply it for all other configurations and request types.

Results: Tables 7 and 8 summarize the prediction metrics for each model, on the respective studied system. Similar to RQ1, we normalize the values of the dataset when calculating the prediction metrics. Overall, we observe an increase in errors compared to the results from RQ1. However, the average errors remain relatively small, and most models show similar performance. Similar to RQ1, we find that, even though VARMA has one of the worst performance, the prediction results are still reasonable (i.e., MAE of 0.1236 and MSR of 0.0775). Therefore, in the case of limited resources, practitioners may still consider VARMA as a possible choice. We also see an increase in percentage error in convolution based models (CNN and ResNet). We attribute this to the data training requirements of these models, that these models may require more training data to correctly recognize the patterns in the datasets.

Table 7: Enterprise System - Forecasting results in an online setting using a previously trained model on new system configurations (MLOLET-RQ2).

Model	MAE	MSE	MAPE (%)	PCC
VARMA	0.1236	0.0775	8.376	0.5825
RNN	0.2231	0.1534	7.213	0.7339
LSTM	0.9777	0.9218	2.603	0.8008
Bidirectional LSTM	0.9752	0.1354	1.928	0.7941
Autoencoder LSTM	0.0610	0.1179	3.7872	0.7524
CNN	0.8649	0.0785	10.885	0.8458
ResNet	0.3561	0.1604	6.721	0.8023

Table 8: Open Source System - Forecasting results in an online setting using a previously trained model on new system configurations (MLOLET-RQ2).

Model	MAE	MSE	MAPE (%)	PCC
VARMA	0.0401	0.0779	6.2584	0.9613
RNN	0.0690	0.0880	12.4745	0.9529
LSTM	0.0154	0.0037	4.7358	0.9525
Bidirectional LSTM	0.0124	0.0038	3.8204	0.9519
Autoencoder LSTM	0.0129	0.0036	3.8595	0.9528
CNN	0.0258	0.0044	8.5224	0.9451
ResNet	0.0751	0.0190	24.6865	0.8659

When testing a model on a different configuration we find that LSTM based architectures still achieve the best results. Our findings indicate that practitioners should consider LSTM based models for having the best prediction accuracy.

4.3.3 RQ3: How far in time can MLOLET forecast future trends?

Motivation: As mentioned before, the operations teams are interested in knowing how far models can predict traffic trends. Such forecasting may provide insight to operations teams, and load testing teams, on how to adapt the resources configuration to account for the new load.

Approach: To address this problem, a significant number of model coefficients must be determined, along with the appropriate size of the sliding window. This is crucial for defining a model that uses the last X time steps to predict the next N time steps. If a model can accurately forecast far enough into the future, the operations team can gain valuable insights that help them manage the software system more efficiently by scaling it up or down based on projected traffic requirements. The process flow has been presented in Section 4.2.1.1.

Results: While we are unable to share the detailed results due to an NDA, we can share our findings on the maximum forecast steps (in seconds) that maintained reasonable accuracy in our experiment. (Table 9). We provide, however, the MAE and MSE details for the open source system (Table 10). For our subject system, our test results show that many models have difficulties forecasting beyond 5-15 seconds in the future. One exception is the AutoEncoder-LSTM model, for which we obtained acceptable forecasts up to 120 seconds, followed by the B-LSTM architecture, for which we obtained forecasts up to 30 seconds. Even though the forecasting capabilities of some models may seem limited (5-15 seconds), they can still be practical in real-world applications since many industry systems support online reconfiguration features (such as JMX-based systems) or can be restarted in under 5 seconds, particularly in containerized or Kubernetes-based environments.

To generalize, for an automated cutover process, we observe that n-step forecasting remains effective as long as the MAE and/or MSE do not exceed ten times the baseline measurements obtained in RQ2 4.3.2. However, different models exhibit varying

Table 9: Enterprise System - Results of forecasting capabilities for different model architectures (MLOLET-RQ3).

Model Name	**Max Forecast Steps (seconds)**
VARMA	5
RNN	10
LSTM	10
Bidirectional LSTM	30
AutoEncoder LSTM	120
CNN	10
ResNet	15

thresholds for acceptable performance, indicating that the cut-off values at which the forecasting remains reliable may differ across models.

We also observed that increasing the input data window size, T (the number of time steps used to predict the next value in the series), does not always improve performance. Additionally, a larger T typically leads to a more complex network, often requiring more hidden layers rather than just more neurons per layer.

When forecasting traffic trends for operations, LSTM-Autoencoder models have the longest trend prediction accuracy. This performance is however tightly coupled with the input data window size, and, using more time steps in the training process may not always help improve the model's accuracy.

4.4 Discussion

In this section, we discuss the feedback from our industrial partner for the experiments we conducted on the enterprise systems.

4.4.1 The Effects of Spikes and Online Spike Detection Observations

Addressing spike misclassification is a crucial part of model training, where different metrics—such as Mean Absolute Error (MAE) or Mean Squared Error (MSE)—are

Table 10: Open Source System - Results of forecasting capabilities for different model architectures (MLOLET-RQ3).

Model Name	Max Forecast Steps (seconds)	MAE	MSE
VARMA	15	0.2827	0.3178
RNN	30	0.1198	0.0369
LSTM	30	0.1072	0.0342
Bidirectional LSTM	45	0.1228	0.0330
AutoEncoder LSTM	120	0.5149	0.3170
CNN	15	0.1717	0.0441
ResNet	10	0.1572	0.0395

used to determine dynamic thresholds. These metrics enable us to control the sensitivity of spike detection relative to the previous data point (or sequence of points) observed by the model, which are used to predict the next value in the sequence. In our experiments, we found that using the Mean Absolute Error as a measure of prediction deviation provided better results for KPI spike detection.

Additionally, software systems under load may exhibit spikes across various metrics. Neglecting these spikes can often lead to more serious issues, including outages [40]. Therefore, it is important to understand the nature of the spikes, their causes, and possible ripple effects within the system. In other words, having a robust spike detection system in place and performing detection for various Key Performance Indicators (KPIs) simultaneously can help correlate events among the different components of a software ecosystem, making it easier to isolate and study their effects.

These factors play a significant role in the software development lifecycle (SDLC) and solution operational processes. In the SDLC, improving the efficiency of the load test cycle enables developers to gain accurate insights into when events occur. This aspect is extremely useful when spikes cause ripple effects that destabilize later parts of the ecosystem, as such events are hard to reproduce without knowing the initial conditions (e.g., a spike in the load of a component may have caused a buffer overrun, leading to loss of messages or data corruption).

Another important aspect of handling spikes and their ripple effects is knowing when to act on them and determining the count threshold that signifies a failing load test, necessitating a decision to stop it. This often requires expert knowledge of the software system to correlate different data points and logs collected from the system. The challenge lies in understanding how the ripple effects of spikes may propagate through specific components or affect other components. For instance, suppose that during a load test, three detected problems do not occur simultaneously. If the load test is stopped early after detecting the first problem, would the other two problems still occur in subsequent load test runs, or were they consequences of the first error? By correcting the first error, would we avoid encountering the others? This issue does not generalize easily, and in the case of our industrial partner, it was often determined that by detecting a certain problem, test engineers could, using expert knowledge, predict where the next problem might occur. Therefore, performing an early stop was more beneficial to the overall project economics.

4.4.1.1 Operational Insights

To reduce load testing time while enhancing process efficiency through spike detection-based techniques, it is most effective to use a model that can be applied across different configurations and multiple business case scenarios. In our experiments, we demonstrated that it is possible to train a model using load testing data from one system configuration and then utilize that trained model for spike detection in other configurations and business scenarios. We observed that when using different configurations, the signal reconstruction from the time series data improves when the predictions are adjusted with the mean of the training data.

Overall, the outcomes of these tests had two significant implications for our industrial partner:

- For future iterations, it is not necessary to train a spike detection model for each scenario, thereby reducing the overall effort of the load testing process and improving project economics.
- The trained model can be deployed in production by the operations team, provided that the load test data resembles production load conditions.

4.5 Threats to Validity

4.5.1 External Validity

Threats to external validity relates to the generalizability of our findings. We conducted our experiments on a large enterprise system composed of numerous components with different business functions. To ensure reproducibility, we also tested our assumptions on an open-source system that generically models a vast number of business applications using a black-box approach, characterized by the common process of receiving a request, processing it, and returning a response. Future studies may be necessary to evaluate our approach on other, more specific systems.

4.5.2 Internal Validity

Since the enterprise system is continuously evolving, changes to the source code from one release to another can affect the system's overall throughput, depending on the specific modifications made. In our tests, we used the same release of the system to avoid side effects introduced by such code changes.

4.5.3 Construct Validity

Prior studies [44, 88] have shown that model performance is closely tied to machine learning parameters. In our experiments, we used grid search as a technique to tune these parameters and reduce bias.

While we mitigated multicollinearity and overfitting by splitting the data into training, validation, and testing sets, it is still possible that the models may suffer from overfitting. However, in our experiments on external validation (as described in Section 4.3.2), we found that the models continued to perform well.

We evaluated ML models as presented in Section 4.3. Although there may be other models that could achieve better results, the models we implemented were sufficient to address the challenges we encountered. Another aspect to consider is that due to the technique we used, the models are not guaranteed to cover the worst case scenarios. We also assumed that the different request types used during the load testing process were well distributed and the payloads did not drastically differ in size. It is possible that in case this assumption were not satisfied that the models would produce a lower accuracy in predictions. Future studies may consider including other ML models to evaluate their effectiveness.

Part III

Large Language Model Agents for IT Operations Management

Chapter 5

Using Large Language Models for adding AIOps Capabilities to Large Scale Systems

This chapter presents an empirical study that examines the adoption of Large Language Models (LLMs) to help with IT Operations management in a form of AIOps [2] agentic approach.

This approach enhances traditional AIOps by providing a deeper contextual understanding of system behaviors and automating processes such as incident management, capacity planning, and resource allocation. LLMs, and LLM based agents, can also facilitate communication between teams and systems by generating human-readable reports, explanations, and action plans. In essence, the integration of LLMs into IT operations transforms how businesses handle infrastructure and operations, driving greater efficiency, reducing downtime, and improving the overall stability of the software ecosystem.

The study focuses a viable example applicable in the industry for Kubernetes [61] based platforms, more precisely, applied on the Red Hat OpenShift [97] platform. We also incorporate in the LLM agents tool set models from our prior studies.

This chapter has submitted for publishing in Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering, Seoul, Republic of Korea

5.1 Motivation

Red Hat OpenShift [97] is the industry's leading Kubernetes based hybrid cloud application platform. It is an enterprise grade Platform as a Service (PaaS) product for containerized and virtualized application deployment at scale. One challenge with deployment scaling within the Red Hat OpenShift platform is that out of the box it offers only horizontal auto-scalers applicable only for Kubernetes pods. This approach is often inefficient or insufficient as prior research shows [13, 26, 48]. A major issue with linear scaling is that distributed applications are usually non-linear in nature and therefore pure horizontal scaling inevitably leads to over-provisioning of resources. By using MLASP's dynamic capacity scaling we can address this issue given that MLASP can have many models that run in parallel, addressing the needs of different applications. Enhancing the application traffic monitoring with forecasting models and capacity planning models we lay the foundation for dynamic scaling.

Furthermore, the platform allows control over Kubernetes resource deployments and scaling, but this is not a simple task. Additionally, since the platform is designed to host thousands of applications and run them at scale, integrating and monitoring these applications and their deployments in IT operations requires significant effort.

The emergence of Large Language Models brings us a new generation of assistants in form of agents and chatbots that IT Operations teams can use to simplify and accelerate their work, namely for application reconfiguration and scaling.

To better understand how these different items can work together we present in Figure 12 the AIOps platform conceptualization in the view of the Gartner analyst who coined the term in 2016 [2].

5.2 Case Study Setup

5.2.1 Studied Systems

We validate our experiments on an open-source enterprise-grade Kubernetes system, namely the Red Hat OpenShift [97] application platform. We use this platform to host our custom workloads, which serve as test subjects for our experiments. The test subject application we employ is the generic WireMock extension application described in Section 4.2.2.2, whose source code is available in a GitHub repository [122].

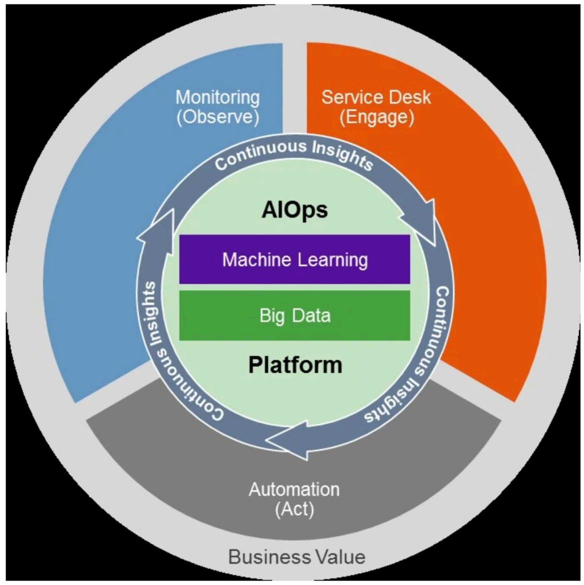

Figure 12: Gartner's AIOps platform conceptualization [2].

We selected Red Hat OpenShift as our test subject for agentic AIOps integration as OpenShift is considered a state-of-the-art Kubernetes environment and Platform as a Service (PaaS) because it extends Kubernetes with additional features that simplify and enhance the development, deployment, and management of containerized applications. OpenShift provides an integrated set of tools for developers, including built-in CI/CD pipelines, automated builds, and application catalogs, which streamline the application lifecycle. It offers advanced security features, such as integrated authentication, authorization, and policy management, ensuring enterprise-grade compliance and governance. Furthermore, OpenShift supports hybrid and multi-cloud deployments, allowing organizations to run applications consistently across on-premises and cloud environments. Its robust ecosystem and developer-friendly interfaces make it an ideal platform for accelerating the delivery of modern, cloud-native applications.

5.2.2 Large Language Model Agents

We use Large Language Models in the form of ReAct [127] (Reason and Act) type agents equipped with various tools to perform IT Operations management functions inside the Red Hat OpenShift platform.

ReAct is a framework designed to enhance Large Language Model (LLM)-based agents by enabling them to perform both reasoning and action tasks in a unified manner. ReAct-based LLM agents combine the natural language understanding and generation capabilities of LLMs with the ability to interact with external tools and environments, allowing for more sophisticated and context-aware responses.

Some of the key features of ReAct based LLM Agents include:

- **Integrated Reasoning and Acting**: ReAct agents interleave reasoning steps (thought processes) with action steps (interactions with the environment). This integration allows the agent to think through a problem, take necessary actions (like querying a database or using an API), and refine its understanding based on the results.

- **Chain-of-Thought Prompting** [118]: The agents use chain-of-thought prompting techniques to generate intermediate reasoning steps. This approach makes the reasoning process transparent and improves the agent's ability to handle complex tasks that require multi-step solutions.

- **Tool and API Utilization**: ReAct agents can leverage external tools, APIs, and databases to gather real-time information or perform computations beyond their trained knowledge. This capability extends the agent's functionality, enabling it to provide up-to-date and accurate responses.

- **Enhanced Problem Solving Abilities**: By combining reasoning with action, the agents can tackle tasks that involve planning, decision-making, and dynamic interactions. This makes them suitable for applications like virtual assistants, automated research, and complex data analysis.

- **Improved Transparency and Interpretability**: The interleaving of reasoning and action steps allows developers and users to trace the agent's thought process. This transparency is valuable for debugging, refining agent behavior, and ensuring alignment with user intentions.

The typical ReAct instructions provided in the LangGraph [64] library (which we used for conducting our experiments) is presented below:

```
Answer the following questions as best you can.
You have access to the following tools:

{tools}

Use the following format:

Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [{tool_names}]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!

Question: {input}
Thought:{agent_scratchpad}
```

With this approach, by leveraging the natural language processing capabilities of the LLM, the IT Operations teams shall have a transformative approach to managing deployed applications and resources. The LLM agents can interpret user commands and by having access to OpenShift's APIs and other tools (such as capacity planning models from MLASP [114]) will be able to execute complex workflows without manual intervention. This agentic approach enhances operational efficiency by reducing the cognitive load on IT teams, enabling faster issue resolution, and facilitating continuous deployment and scaling of applications. Furthermore, LLM agents can provide contextual recommendations and predictive maintenance, ensuring that applications remain resilient and performant within the dynamic environments managed by OpenShift. Ultimately, this integration fosters a more intelligent, responsive, and scalable

IT infrastructure, driving superior business outcomes.

As an example, let's walk through the process of how an LLM Agent chat bot equipped with a Retrieval Augmented Generation (RAG) tool can help in IT Operations by efficiently summarizing procedures from extensive documentation:

1. **User Query**: An IT professional interacts with the chatbot by asking a question or requesting a summary of a procedure: "How do I reset a user's password in Active Directory?"

2. **Query Interpretation**: The LLM interprets the query, identifying key entities and intent. It recognizes that the user needs a step-by-step procedure for resetting a password in Active Directory. It will parse the list of available tool descriptions to identify which tool could possibly be used to provide an answer. It identifies a RAG tool for Active Directory documentation that may provide relevant content.

3. **Document Retrieval**: The RAG tool searches the organization's documentation repositories for relevant content. It uses semantic search to find the most pertinent procedures, even if exact keywords aren't matched. It retrieves previously ingested content which may be sections from manuals, knowledge bases, or previous incident resolutions.

4. **Content Summarization**: The LLM processes the retrieved documents to generate a concise summary. It extracts essential steps and presents them in an easy-to-understand format and filters out irrelevant information to focus on actionable instructions.

5. **Response Delivery**: The chat bot presents the summarized procedure to the user in a conversational manner that may include numbered steps, warnings, or prerequisites. Here is an example response for our earlier user query:

    - Open the Active Directory Users and Computers console.
    - Locate and right-click the user's account.
    - Select 'Reset Password' from the context menu.
    - Enter the new password and confirm.
    - Ensure 'User must change password at next logon' is checked if required.

- Click 'OK' to apply the changes.

Looking at the earlier example, we can easily identify benefits for IT Operations, such as:

1. **Time Efficiency**: Reduces the time spent searching through extensive documentation. Provides immediate access to necessary procedures, enhancing productivity.

2. **Consistency and Accuracy**: Ensures that all team members follow standardized procedures. Minimizes errors by providing up-to-date and verified instructions.

3. **Knowledge Sharing**: Acts as a centralized knowledge repository accessible via natural language queries. Aids in onboarding new team members by simplifying access to operational procedures.

4. **Scalability**: Handles multiple simultaneous queries without degradation in performance. Supports large IT teams and complex operational environments.

5. **Multilingual Support**: Translate documentation and responses as needed, therefore providing support in multiple languages is the IT team is global or multilingual.

6. **Integration with ITSM Tools**: depending on the available tools, LLMs can help automate ticket creation/updates/closures, retrieve system information or application KPI information, execute application or system adjustments.

Although there are clear and powerful benefits from using LLM agents we must still have the following considerations when implementing them:

1. **Safety and Ethical Use**: Developers must ensure the agent acts responsibly, especially when interacting with external systems that can alter data or perform transactions.

2. **Error Handling**: Developers must implement robust mechanisms to handle failures in external tools or incorrect data.

3. **Privacy and Security**: Developers must ensure user data protection and secure interactions with APIs and services.

5.2.3 Integrating Tools for LLM Agents

We conduct our experiments with the help of the LangChain [63] and LangGraph [64] frameworks. We use this approach as these frameworks provide a higher level abstraction for using different Large Language Models (different providers and different variants from the same provider). This aspect is important to maintain consistency across the tests, given that different LLMs use different prompt formats and information is expected to passed in a specific way. In other words, by using LangChain and LangGraph as LLM access libraries, we avoid writing custom code for conversing with all the different language models.

We test the LLM agents in the form of chat bots as this is one of the easiest ways to generalize their integration within IT Operations Management. Our chatbot, implemented as a Knative application, serves as an interface for IT operations professionals by leveraging an LLM and specialized tools to process requests. The chatbot's key components include: (1) A user interface for receiving input, which was bypassed in our testing as we used predefined queries. (2) An LLM client module that connects to an LLM inference server. While an optional memory component can enable conversational functionality, it may sometimes disrupt reasoning processes (see Section 5.4).

For our experiments, we developed a set of custom tools, denoted as $T < n >$, where $n \in [1, 9]$, using the Python programming language and supporting libraries. These tools are designed to assist with various ITOM tasks, including capacity planning, extracting procedure summaries, retrieving platform deployment and configuration information, and extracting platform and application KPIs for a specified datetime range, with outputs in both CSV and graphical formats. In the next section, we present the user query and tool list association the LLMs must use in order to correctly respond to the each query. Below is the list of tools we created for our experiments:

1. **T1, mlasp-tool:** to integrate the MLASP [114] based capacity planning ML model for the WireMock based custom workload application. This tool generates a set of parameter configuration to support a desired KPI value within a given precision boundary for the WireMock application. It searches for the parameter configurations a given number of epochs.

2. **T2,rag-tool:** a tool that gives the LLM the ability to search through a specialized vector database that contains encoded documentation about the Red Hat OpenShift AI operator, including procedure description and howto information. The LLM can inspect this database to obtain information based out of the received query and then summarize a response to the user.

3. **T3, time-tool:** Tool to calculate the timestamp, the iso formatted string and the timezone string of the requested time information. It returns a pydantic object containing the timestamp value, the ISO formatted string of the date time value, the timezone string.

4. **T4, list-operators-tool:** to find out information about operators installed within a namespace. The response may contain information such as the name of the operator, its version and deployment status.

5. **T5, pod-summary-tool:** Tool used to summarize information about the pods that exist in a namespace. It returns an object containing the name of namespace and pod state and count information. For the running pods it also returns its name and if available any service information such as service name, service ports and route.

6. **T6, service-summary-tool:** Tool used to summarize services information in an OpenShift namespace. It returns an object containing the name of namespace and a list of the available services and their properties such as name, port numbers and route information.

7. **T7, prometheus-metric-names-tool:** Tool to List available metric names in a Prometheus instance using an input filter. The input filter name and value are expected as input (e.g. input filter name is 'namespace' and filter value is 'demo'). Returns a list containing the available metric names.

8. **T8, prometheus-metric-data-range-tool:** Tool used to list the application metric values and associated timestamps between a start and an end timestamp interval for a given metric name stored within a Prometheus instance. It returns a pydantic object containing the list of the desired application metric values and associated timestamp information.

9. **T9, plot-prometheus-metric-range-data-as-file-tool:** Tool used to create a file with the plot of the instantaneous rate (irate) of an application metric values and associated timestamps between a start and an end timestamp interval for a given metric name stored within a Prometheus instance. It returns a string containing the name of the file containing the plot.

The tools mentioned above allow us to evaluate a limited scope of IT Operations Management (ITOM) operations and scenarios. Nonetheless, these use cases are adequate to assess and report on the performance of various LLMs. Naturally, this toolset can be extended with additional tools, thereby enhancing the ITOM management capabilities of the LLM-powered AI assistant and enabling support for a broader range of operational tasks.

5.2.4 Evaluating Agents in AIOps Context

We test the LLM agents in the form of chat bots as this is one of the easiest ways to generalize their integration within IT Operations Management. We provide the same list of questions as detailed in Table 11. We form this list as a mix of general purpose queries (e.g., Q-01, Q-02, Q-08, etc.) and very specific platform queries (e.g., Q-05, Q-10, Q-13, etc.) as well as target application management questions (e.g., Q-21, Q-23, Q-24, etc.).

To respond to these queries, the LLMs must rely on either their training data or the provided list of tools. We classify these queries into two categories: **Simple Reasoning (SR)**—where the LLM generates a response based solely on its training data or by utilizing a maximum of one tool; and **Advanced Reasoning (AR)**—where the LLM identifies and utilizes multiple tools, constructing a workflow that ensures the tools are executed in the correct sequence. In advanced reasoning scenarios, the LLM must also format the data appropriately before invoking a tool, with the possibility of using the same tool multiple times with different inputs.

The intent is to see how different models understand the questions and use either their internal knowledge or access the right tools in the right order in order to produce an answer. We ask these questions to different large language models and different variants of the same model and assess their responses for correctness, while collecting KPI information such as response time and average tokens consumed to produce an answer. We evaluate our queries on the following list of large language models: from

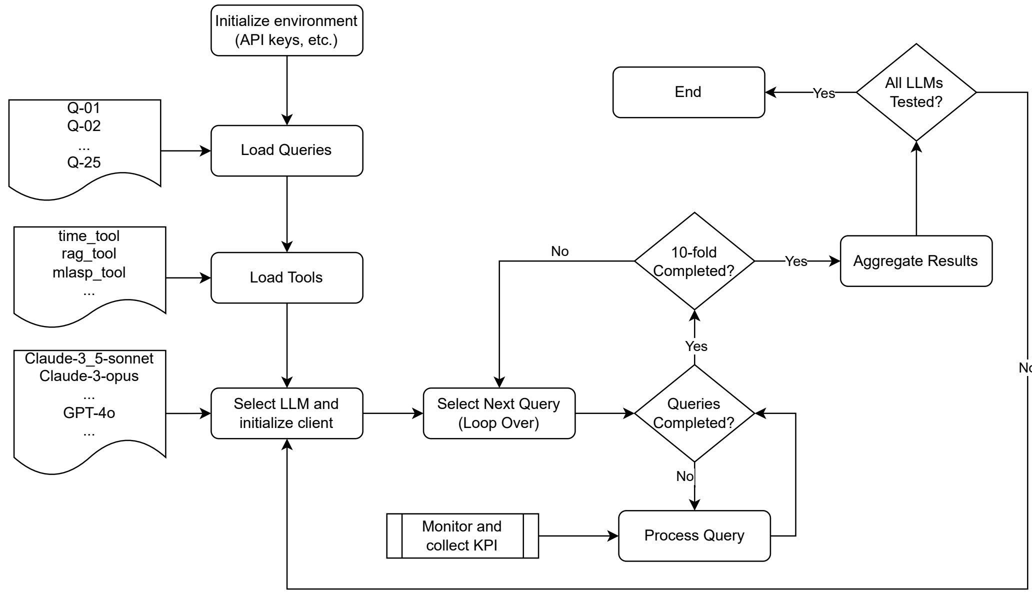

Figure 13: An example workflow for LLM performance evaluation in an AIOps context.

the OpenAI family: GPT 3.5 Turbo, GPT 4-o, GPT 4-o Mini, and GPT 4 Turbo; from the Anthropic family: Claude 3.5 Sonnet, Claude 3 Haiku, and Claude 3 Opus; from the Mistral family: Mistral Largest, Mixtral 8x22B, and Mistral Small 7B.

To assess a model's robustness and consistency in providing answers, we ask each model the same question ten times. The overall evaluation process is depicted in Figure 13.

Table 11: Large Language Model Agents for AIOps Evaluation Queries and aasociated tools for responding. Category shows the category of the question, where SR means Simple Reasoning (use at most one tool) and AR means Advanced Reasoning (use multiple tools).

Q#.	**Cat.**	**Tools**	**Query Text**
Q-01	SR	-	Hi, who are you?
Q-02	SR	-	What tools do you have access to?
Q-03	SR	-	Give me the list of tools you have access to.

Q#.	Cat.	Tools	Query Text
Q-04	SR	-	Give me the list and a short description of the tools you have access to.
Q-05	SR	T4	What operators are in namespace demo?
Q-06	SR	T4	What operators are in namespace demo? Please provide only the name and the version for each operator.
Q-07	SR	T2	How can I create a Data Science Project?
Q-08	SR	-	Can you describe Paris in 100 words or less?
Q-09	SR	-	Is there a river?
Q-10	SR	T5	Tell me about the pods in namespace demo.
Q-11	SR	T5	Give me a summary of the running pods in namespace demo. Please include service and route information in the response.
Q-12	SR	T5	Give me the complete summary of the pods in namespace demo.
Q-13	SR	T5	Give me a summary of the running pods in namespace demo. Give me only the names and the route if they have one.
Q-14	SR	T3	What day is today?
Q-15	SR	T3	What is the current date time?
Q-16	SR	T3	What is the current timestamp?
Q-17	SR	T3	What is the timestamp and date time for 3 hours ago?
Q-18	SR	T3	What is the timestamp and date time for 3 hours from now?
Q-19	SR	T3	What is the timestamp and date time for 3 hours ago?
Q-20	SR	T6	Is there a prometheus service running in namespace demo? If so, give me its name and port values.
Q-21	AR	T6, T7	Find out the service name and port number of the Prometheus service running in namespace demo. Then use that information to retrieve the list of metrics filtered by namespace demo.

Q#.	Cat.	Tools	Query Text
Q-22	AR	T6, T7	Find out the Prometheus service name and port number running in namespace demo. Give me all the metrics stored by it that have a name that starts with load_generator.
Q-23	SR	T1	What configuration of WireMock supports a throughput KPI of 307 within a 2.9 percent precision? Search for 100 epochs to find the result.
Q-24	AR	T3, T6, T9	Find out the Prometheus service name and port number running in namespace demo. Use it to to plot all the prometheus metric data for the metric load_generator_total_msg starting 40 days ago until now. Return only the file name and nothing else.
Q-25	AR	T3, T6, T8	Find out the Prometheus service name and port number running in namespace demo. Use that to get all the prometheus metric data for the metric load_generator_total_msg starting 40 days ago until now. Print out only the metric values and their associated timestamp as a CSV table.

5.3 Case Study Results

In this section, we discuss the results of our three research questions (RQs).

5.3.1 RQ1: How accurately do different LLMs perform on a set of IT Operations tasks?

Motivation: Large Language Models are different from one another from several perspectives:

- **Training Data:** Different LLMs are trained on varying datasets, which can affect their knowledge base, language proficiency, and domain expertise.
- **Model Size:** LLMs differ in the number of parameters (millions or billions),

which affects their complexity, performance, and ability to understand nuanced language.

- **Architecture:** While most LLMs are built on transformer-based architectures, specific variations in model design (for example, GPT, BERT, T5) lead to differences in processing language and generating outputs.

- **Specialization:** Some LLMs are fine-tuned for specific tasks (e.g., medical, legal, or technical language), making them more effective in particular domains.

- **Inference Speed:** The response time of LLMs varies based on their architecture and model size, with smaller models generally being faster but less accurate.

- **Context Window:** LLMs differ in how much previous conversation or text they can retain in their context, affecting their ability to maintain coherence over long exchanges.

- **Fine-tuning and Adaptability:** Some LLMs are more easily fine-tuned or adapted to specific use cases or custom datasets, while others may be less flexible.

Given these differences we want to evaluate how accurately different models perform on the user queries detailed in Table 11.

Approach: We perform the procedure described in Section 5.2.4 and depicted in Figure 13.

Results: Table 12 presents the summary of the accuracy for the models we evaluated, based on the testing process described in our approach. We observe that some models excel in simple reasoning tasks, others perform better in advanced reasoning tasks, while a few demonstrate consistent performance across both types of tasks. Notably, the Anthropic model family outperforms others in simple reasoning, whereas OpenAI's models excel in advanced reasoning tasks. For advanced reasoning specifically, Claude 3.5 Sonnet and Claude 3 Opus exhibit high accuracy rates but tend to lack the depth and completeness found in the responses provided by GPT-4 models.

We notice that, in general, larger models perform better. This is expected for a number of reasons:

- Representation Capacity: Larger models have more parameters, allowing them to capture complex patterns and nuances in language data. This enables them to understand idiomatic expressions, rare words, and intricate linguistic structures. With increased capacity, models can differentiate between subtle differences in context and meaning, leading to more accurate and relevant responses.

- Generalization Ability: Larger LLMs are trained on vast amounts of data, encompassing diverse topics and styles. This broad training helps them generalize better to new and unseen inputs.

- Contextual Understanding: Bigger models can maintain context over longer stretches of text, which is crucial to understanding and generating coherent responses in conversations or long documents. They are also better at resolving ambiguities by considering more extensive contextual information.

- Complex Reasoning and Inference: Multistep reasoning allows the model to perform more complex reasoning tasks, like mathematical calculations, logical inferences, and following intricate instructions. Larger models can better grasp relationships between concepts, aiding in tasks like question answering and problem solving. They are also better at understanding and generating figurative language.

However, smaller models are also quite efficient when it comes to reasoning and tool selection to complete a task.

Table 13 provides the detailed results of the accuracy report of the models we tested following the process mentioned in our approach. We provide further details and explanation on various aspects of the model responses and their accuracy in the Discussion section, Section 5.4, where we shall explain why models such as ChatGPT 3.5 Turbo (once a state-of-the-art model) are struggling with certain queries that seem quite straightforward (e.g., query Q7).

Table 12: RQ1 - Summary of Large Language Model Agents solving task accuracy in AIOps Context grouped by type of reasoning: Simple Reasoning, SR, (use at most one tool), and Advanced Reasoning, AR (use multiple tools).

Model	**SR Task** (0..1 tools)	**AR Task** (2+ tools)
Claude 3.5 Sonnet	95.23%	95%
Claude 3 Haiku	89.52%	30%
Claude 3 Opus	90%	95%
Mistral Largest	94.76%	45%
Mixtral 8x22B	2.38%	0%
Mistral Small 7B	76.19%	0%
GPT 3.5 Turbo	85.71%	0%
GPT 4-o	87.14%	100%
GPT 4-o Mini	85.71%	77.5%
GPT 4 Turbo	85.71%	90%

With the exception of Mixtral 8x22B, all tested LLMs followed the ReAct principles and effectively utilized the available tools when the instructions were clear. Among the models, the GPT family achieved the best overall performance in advanced reasoning tasks, with GPT 4-o excelling in accuracy and reliability. The Claude model family, on the other hand, performed best on simple reasoning tasks. In contrast, Mixtral 8x22B exhibited the poorest performance, frequently hallucinating responses and failing to leverage the tools provided. Being a closed model, we cannot say why it hallucinated most of the responses and did not use the available tools to form the responses.

Table 13: RQ1 - Detailed Results for Large Language Model Agents solving task accuracy in AIOps Context.

Query No.	Claude 3.5 Sonnet	Claude 3 Haiku	Claude 3 Opus	Mistral Largest	Mixtral 8x22B	Mistral Small 7B	GPT 3.5 turbo	GPT 4-o	GPT 4-o mini	GPT 4 turbo
Q-01	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
Q-02	100%	90%	100%	100%	100%	100%	100%	100%	100%	100%
Q-03	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
Q-04	100%	100%	100%	100%	100%	100%	0%	100%	100%	100%
Q-05	100%	90%	100%	100%	0%	100%	100%	100%	100%	100%
Q-06	100%	100%	100%	100%	0%	100%	100%	100%	100%	100%
Q-07	100%	0%	0%	90%	0%	0%	100%	30%	0%	0%
Q-08	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
Q-09	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Q-10	100%	100%	100%	100%	0%	100%	100%	100%	100%	100%
Q-11	100%	100%	100%	100%	0%	100%	100%	100%	100%	100%
Q-12	100%	100%	100%	100%	0%	100%	100%	100%	100%	100%
Q-13	100%	100%	100%	100%	0%	100%	100%	100%	100%	100%
Q-14	100%	100%	100%	100%	0%	0%	100%	0%	0%	0%
Q-15	100%	100%	100%	100%	0%	0%	100%	100%	100%	100%
Q-16	100%	100%	100%	100%	0%	100%	100%	100%	100%	100%
Q-17	100%	100%	100%	100%	0%	100%	100%	100%	100%	100%
Q-18	100%	100%	100%	100%	0%	100%	100%	100%	100%	100%

Query No.	Claude 3.5 Sonnet	Claude 3 Haiku	Claude 3 Opus	Mistral Largest	Mixtral 8x22B	Mistral Small 7B	GPT 3.5 turbo	GPT 4-o	GPT 4-o mini	GPT 4 turbo
Q-19	100%	100%	100%	100%	0%	100%	100%	100%	100%	100%
Q-20	100%	100%	100%	100%	0%	0%	0%	100%	100%	100%
Q-21	80%	10%	0%	0%	0%	0%	0%	100%	100%	100%
Q-22	100%	50%	100%	100%	0%	0%	0%	100%	100%	100%
Q-23	100%	100%	90%	100%	0%	100%	100%	100%	100%	100%
Q-24	100%	30%	100%	80%	0%	0%	0%	100%	100%	80%
Q-25	100%*	30%*	100%*	0%	0%	0%	0%	100%	10%	80%

5.3.2 RQ2: How fast do different LLMs perform on a set of IT Operations tasks?

Motivation: Building on the earlier mentioned differences between LLMs, we will measure how quickly various models respond to user requests. Specifically, we record the 50th percentile (median), 90th percentile, and maximum response times (in seconds) for models that vary in size, architecture, context window, and training data. This measurement is important because some tasks may be critical in time, particularly in real-time applications such as chatbots, virtual assistants, and customer support systems. Preventive maintenance in applications is another area where quick responses are critical. Slow responses can lead to user frustration, decreased engagement, and a perception of unreliability. Evaluating the LLM's performance over repeated requests using the 50th-percentile (median) and 90th-percentile response times helps in understanding both typical and worst-case scenarios. The 50th percentile indicates the response time that half of the users experience or is faster, reflecting the general performance. The 90th percentile shows the response time below which 90% of the requests are served, highlighting the tail-end delays that can impact user satisfaction. Analyzing these percentiles allows developers to identify latency issues and optimize the system to ensure consistent and timely responses for most users.

Approach: We perform the procedure described in Section 5.2.4 and depicted in Figure 13.

Results: In Table 14 we present the results summary for the performance metrics we observed and described in the motivation part of this research question. The summary is aligned to the type of reasoning required to perform the task, namely simple reasoning (SR) or advanced reasoning (AR). The detailed results are provided in Table 15.

Response times must always be evaluated alongside accuracy to provide meaningful insights into a model's performance. This combined assessment is particularly important in scenarios where both speed and correctness are critical. Without such context, faster response times might overshadow inaccuracies, or highly accurate responses might fail to meet real-time requirements. Considering this, we observe that for **SR queries**, OpenAI models generally respond the fastest, closely followed by Anthropic models, with Claude 3 Haiku standing out as the fastest responder based on P-50 response times. The Mistral family models, excluding Mixtral 8X22B, also deliver competitive response times for SR tasks. For **AR queries**, response times among models capable of solving these tasks are relatively consistent.

Smaller models within a family generally respond approximately 50% faster than their larger counterparts. When evaluating both response time and accuracy, OpenAI models, particularly GPT-4o, stand out as the fastest and most reliable performers. Conversely, the Mistral models, especially Mixtral 8x22B, performed the worst. Although Mixtral 8x22B returned hallucinated responses quickly, it failed to resolve any **AR queries**. However, for **SR queries**, the Mistral Small 7B model remains a viable option due to its reasonable accuracy, as observed in RQ1. Its relatively fast response times combined with acceptable accuracy make it suitable for specific ITOM operations scenarios where high precision is not the primary requirement.

We notice that in general larger models take more time to respond, especially where the query requires several reasoning and action steps in order to determine the response (e.g., Q-21, Q-22, Q-24, and Q-25). These queries require the LLM to piece together several instructions from the input and decide which tools and in which order to use. In some cases, the results of one tool are input for the next; therefore, the model performance is also dependent on the performance of the tools themselves. We must remember that faster here does not necessarily mean better, since response times must be correlated with the accuracy aspect of the associated responses. This is in particular visible for the Mixtral 8x22B model, which has the smallest response times. However, when correlated with the RQ1, we can see that this model has a very low accuracy as it failed to respond to the majority of the queries.

Response times should always be assessed in context and alongside the accuracy of responses to ensure a comprehensive evaluation. Within this framework, OpenAI models generally exhibit the fastest response times for both **SR** and **AR** queries. However, when considering the P-50 metric specifically, Claude 3 Haiku emerges as the fastest model for **SR queries**, while GPT-4o demonstrates the fastest performance for **AR queries**.

Table 14: RQ2 - Summary of Large Language Model Agents solving task average response times (in seconds) in AIOps Context.

Model	**Metric** (Average)	**SR Task** (0..1 tools)	**AR Task** (2+ tools)
Claude 3.5 Sonnet	P-50	6.41	19.14
	P-90	7.12	20.54
	Max	7.52	21.01
Claude 3 Haiku	P-50	3.14	9.07
	P-90	4.38	16.08
	Max	5.16	16.97
Claude 3 Opus	P-50	18.18	48.38
	P-90	20.97	55.12
	Max	22.73	57.54
Mistral Largest	P-50	8.36	71.95
	P-90	12.31	88.04
	Max	17.17	94.67
Mixtral 8x22B	P-50	4.73	6.88
	P-90	5.72	7.17
	Max	11.00	7.50
Mistral Small 7B	P-50	4.72	9.12
	P-90	5.17	9.42
	Max	5.44	9.66
GPT 3.5 Turbo	P-50	3.45	6.99
	P-90	4.25	7.35
	Max	4.50	7.71
GPT 4-o	P-50	4.47	46.12
	P-90	5.69	54.42
	Max	7.13	69.40
GPT 4-o Mini	P-50	4.13	22.08
	P-90	6.06	28.61
	Max	9.91	46.00
GPT 4 Turbo	P-50	9.17	48.62
	P-90	10.53	54.99
	Max	11.47	57.10

Table 15: RQ2 - Detailed Results for Large Language Model Agents solving task response times (in seconds) in AIOps Context.

Query No.	Metric	Claude 3.5 Sonnet	Claude 3 Haiku	Claude 3 Opus	Mistral Largest	Mixtral 8x22B	Mistral Small 7B	GPT 3.5 turbo	GPT 4-o	GPT 4-o mini	GPT 4 turbo
Q-01	P-50	1.81	0.77	7.07	1.16	2.06	1.38	0.71	0.87	0.93	1.85
	P-90	2.04	1.22	9.50	1.62	2.34	1.70	0.87	1.29	1.20	2.54
	Max	2.07	2.16	12.97	2.07	2.77	2.23	1.18	3.13	1.23	2.73
Q-02	P-50	6.39	2.81	23.05	8.38	7.56	6.32	2.02	5.15	3.68	13.33
	P-90	6.72	4.45	31.52	12.32	7.82	6.76	2.57	5.95	6.44	16.86
	Max	6.83	4.94	39.60	18.01	7.87	7.03	3.13	8.07	14.86	17.72
Q-03	P-50	3.86	2.79	20.63	11.10	5.17	2.72	2.06	4.77	4.40	10.96
	P-90	6.22	5.28	27.50	27.96	5.82	2.95	2.38	5.67	6.39	12.56
	Max	6.35	6.84	28.23	42.01	6.22	3.26	2.44	6.00	9.74	12.64
Q-04	P-50	7.00	4.13	24.35	25.17	7.97	6.21	1.64	8.39	5.48	14.84
	P-90	8.46	4.97	27.31	33.17	8.24	6.96	1.91	10.49	10.80	16.30
	Max	9.03	7.47	33.90	34.49	8.26	7.18	1.95	15.49	43.94	19.60
Q-05	P-50	5.99	2.9	17.73	5.31	2.29	6.42	3.77	5.47	4.08	12.63
	P-90	7.37	3.88	20.60	6.62	2.85	6.89	4.36	6.43	6.84	13.62
	Max	7.49	4.13	20.94	7.53	3.11	7.08	4.38	6.67	10.80	13.62
Q-06	P-50	4.95	2.09	16.37	5.53	2.53	4.86	4.02	4.12	3.76	9.48
	P-90	5.18	2.36	18.53	9.46	2.85	5.11	4.73	4.98	5.09	11.73

Query No.	Metric	Claude 3.5 Sonnet	Claude 3 Haiku	Claude 3 Opus	Mistral Largest	Mixtral 8x22B	Mistral Small 7B	GPT 3.5 turbo	GPT 4-o	GPT 4-o mini	GPT 4 turbo
	Max	5.27	2.52	18.92	9.54	3.14	5.64	4.76	5.59	6.45	12.84
Q-07	P-50	7.53	3.14	23.69	6.46	7.08	5.56	3.59	8.19	9.46	25.81
	P-90	8.62	4.95	26.87	11.74	7.99	6.36	4.08	11.28	17.28	28.15
	Max	8.79	5.77	32.01	11.88	9.10	6.72	4.58	13.66	17.55	29.08
Q-08	P-50	3.97	1.96	12.45	3.83	1.95	2.61	2.34	2.00	1.78	5.68
	P-90	4.47	3.71	13.30	6.50	2.58	2.76	3.43	2.51	2.31	6.66
	Max	4.54	4.23	13.46	6.74	2.66	2.82	3.80	3.02	3.71	6.86
Q-09	P-50	2.64	1.08	11.38	23.71	2.20	1.83	1.64	0.87	0.98	1.60
	P-90	2.93	2.17	13.67	28.07	2.63	1.92	1.72	1.79	2.20	2.66
	Max	3.07	2.30	13.76	51.20	2.71	2.00	1.95	1.99	2.74	3.88
Q-10	P-50	10.61	6.36	22.38	13.58	6.09	8.53	8.28	10.31	8.98	16.17
	P-90	10.97	7.06	23.45	18.39	6.36	8.85	10.60	12.01	14.82	18.78
	Max	11.03	7.27	23.72	25.84	6.86	8.93	10.90	15.89	29.50	19.26
Q-11	P-50	10.52	5.93	22.81	12.27	4.70	9.01	8.19	8.23	8.67	15.19
	P-90	11.17	8.42	23.28	14.58	4.89	10.02	8.94	9.36	9.86	16.32
	Max	11.61	10.04	23.49	22.14	5.16	10.30	9.08	10.58	10.31	17.42
Q-12	P-50	11.04	6.21	22.00	11.30	3.52	8.64	7.74	8.71	8.81	15.78
	P-90	11.72	6.98	26.90	19.09	15.42	8.88	8.57	12.04	10.93	17.68
	Max	12.36	7.17	27.72	20.55	120.12	9.04	8.92	16.60	11.75	20.88
	P-50	8.33	5.45	20.05	8.17	3.37	6.94	6.69	6.65	6.46	12.14

Query No.	Metric	Claude 3.5 Sonnet	Claude 3 Haiku	Claude 3 Opus	Mistral Largest	Mixtral 8x22B	Mistral Small 7B	GPT 3.5 turbo	GPT 4-o	GPT 4-o mini	GPT 4 turbo
Q-13	P-90	9.07	6.89	22.06	15.11	3.74	7.54	7.14	8.09	8.43	13.01
	Max	9.32	7.27	22.96	15.48	4.49	7.74	7.58	9.36	11.30	13.76
Q-14	P-50	5.03	1.77	15.89	2.66	5.26	1.39	1.59	0.57	0.67	0.90
	P-90	5.77	1.85	17.29	3.23	5.82	1.91	1.86	0.60	1.08	1.09
	Max	6.27	2.15	18.10	3.46	6.14	2.58	2.03	0.62	1.09	1.11
Q-15	P-50	5.39	1.86	16.80	3.44	2.53	2.55	1.77	2.07	1.70	3.53
	P-90	5.81	3.32	17.29	5.11	3.28	2.97	1.95	2.54	2.49	4.81
	Max	5.83	4.02	18.10	10.68	3.38	3.11	2.03	3.02	6.39	5.57
Q-16	P-50	5.55	2.29	16.24	3.81	3.33	2.98	2.06	2.19	1.97	4.41
	P-90	5.94	3.83	19.09	4.88	3.51	3.40	2.33	3.30	3.43	5.17
	Max	6.10	5.85	19.31	5.15	3.69	3.52	2.50	4.39	3.62	5.69
Q-17	P-50	6.36	2.32	16.42	4.60	8.77	3.18	2.10	2.44	2.06	4.34
	P-90	6.69	2.98	18.86	5.64	9.47	3.54	2.24	3.17	2.37	5.13
	Max	7.01	3.53	18.97	6.36	9.54	3.60	2.64	3.62	2.43	7.40
Q-18	P-50	5.73	2.04	17.34	3.87	5.45	3.17	2.16	2.19	2.47	4.67
	P-90	6.92	2.19	19.91	5.60	5.90	3.74	2.34	3.07	3.46	5.29
	Max	6.95	2.78	20.39	5.80	6.35	3.96	2.52	5.46	4.51	5.34
Q-19	P-50	6.19	2.29	16.35	5.81	8.84	3.13	2.09	2.27	2.13	4.38
	P-90	6.42	4.48	18.52	8.56	9.52	3.59	2.53	2.84	2.59	5.79

Query No.	Metric	Claude 3.5 Sonnet	Claude 3 Haiku	Claude 3 Opus	Mistral Largest	Mixtral 8x22B	Mistral Small 7B	GPT 3.5 turbo	GPT 4-o	GPT 4-o mini	GPT 4 turbo
	Max	6.69	6.49	20.99	14.72	9.54	3.80	2.91	2.87	4.29	6.64
Q-20	P-50	5.31	2.25	15.62	6.58	5.18	4.17	1.97	2.12	2.54	3.91
	P-90	5.72	3.31	18.28	8.42	5.37	4.77	8.02	4.75	2.84	4.90
	Max	6.04	3.45	19.57	8.63	5.41	5.68	8.20	5.82	2.84	6.00
Q-21	P-50	17.68	5.90	37.57	124.03	6.28	9.22	3.54	75.95	66.59	21.11
	P-90	20.51	6.97	42.13	126.67	6.60	9.42	3.76	91.96	75.10	26.04
	Max	20.72	7.06	47.22	126.71	6.75	9.64	3.99	118.36	77.52	28.39
Q-22	P-50	16.51	7.79	41.35	23.20	6.83	11.14	8.10	9.15	9.86	18.68
	P-90	17.74	10.38	47.66	26.49	7.27	11.81	8.23	10.37	11.68	20.98
	Max	18.43	12.43	49.04	32.98	8.02	12.13	8.23	12.20	12.22	22.43
Q-23	P-50	10.43	5.47	23.17	8.74	3.44	7.46	6.12	6.26	5.77	11.03
	P-90	11.38	7.62	25.48	12.47	3.76	7.89	6.71	7.33	6.61	12.10
	Max	15.19	7.96	28.87	38.28	4.54	8.01	7.06	7.99	9.01	12.79
Q-24	P-50	10.84	4.41	37.27	8.42	6.99	10.99	8.13	6.38	7.69	13.19
	P-90	11.20	9.94	42.08	24.29	7.20	11.21	8.59	8.00	13.26	14.88
	Max	11.74	10.99	44.95	37.83	7.59	11.28	8.60	8.67	14.55	19.17
Q-25	P-50	31.54	18.18	77.33	132.14	7.40	5.12	8.19	92.99	4.16	141.51
	P-90	32.72	37.02	88.60	174.69	7.62	5.22	8.80	107.35	14.42	158.07
	Max	33.15	37.39	88.96	181.16	7.64	5.58	10.03	138.38	79.72	158.40

5.3.3 RQ3: How verbose do different LLMs perform on a set of IT Operations tasks?

Motivation: Building on the differences between LLMs mentioned earlier, we will assess how verbose each model is in responding to user requests. This will be measured by calculating the average number of tokens that a model uses to answer a specific request. We will determine this average by repeating the same request ten times and recording the token count for each response. Similarly to the previous research questions, our observations are made for different models that vary in size, architecture, context window, and training data. This measurement is important as it not only affects the response time (more tokens means more time to stream back a response, more time for a human to spend on absorbing a response), but also affects the overall operational costs for models hosted by third party providers where access is paid on the number of utilized tokens. Additionally, a more verbose model with a limited context window may also be limited in being able to answer complex questions or execute workflows.

Approach: We perform the procedure described in Section 5.2.4 and depicted in Figure 13.

Results: In Table 16 we present the summary of the results for the verbosity of the different LLMs solving the AIOps-related tasks, grouped by the type of reasoning required: simple reasoning (SR), or advanced reasoning (AR). The detailed results are provided in Table 17. For the detailed results (Table 17), the numbers represent the average token count, rounded to closest integer, for the ten-fold execution of each query in Table 11.

As observed in previous research questions, a model's verbosity is strongly tied to its ability to solve tasks effectively. Larger models generally produce more verbose responses. Additionally, verbosity levels vary across models from different providers, even within the same class (e.g., Anthropic's Claude 3.5 Sonnet, Mistral's largest model, and OpenAI's GPT 4-turbo). These differences reflect provider-specific approaches to response generation and structuring.

Similarly to the prior research question, the verbosity of a model is tightly coupled with its ability to solve a task. We notice that larger models are generally more verbose in their answers. The verbosity also varies for different model suppliers,

Table 16: RQ3 - Summary Average token count (verbosity) of Large Language Model Agents solving tasks in AIOps Context.

Model	**SR Task** (0..1 tools)	**AR Task** (2+ tools)
Claude 3.5 Sonnet	5420.3	42768.1
Claude 3 Haiku	5564.6	46535.7
Claude 3 Opus	5892.6	44648.5
Mistral Largest	6066.8	39877.6
Mixtral 8X22B	2803.4	3007.3
Mistral Small 7B	4162.7	4619.5
GPT 3.5 Turbo	3242.3	2722.6
GPT 4-o	3031.6	25965
GPT 4-o Mini	3065	20900.7
GPT 4 Turbo	3081.3	24308.5

despite the respective models' belonging to similar model classes. (e.g., Anthropic's Claude 3.5 Sonnet versus Mistral Largerst versus OpenAi's GPT 4-turbo).

In general, Anthropic's Claude family models are the most verbose, despite their responses being less complete or accurate compared to OpenAI's GPT models. For instance, Claude models consistently truncated their responses to query Q-25, whereas GPT 4-turbo delivered a complete response. The verbosity of Mistral family models is often comparable to that of the Claude family but only when they provided correct responses. This demonstrates that verbosity does not always align with response quality or completeness.

In RQ1 we mentioned that queries Q-04 and Q-21 were not resolved by the GPT 3.5 turbo model due to a model internal processing error (as reported by the integration library). The correlation between the token count and result of the query is especially visible for these two queries as in RQ3 we see the count of tokens at zero, despite the model taking some times before an error is provided, as per the results from RQ2.

Overall, the OpenAI model family is the most efficient in token usage for both SR and AR queries, whereas the Anthropic models are the most verbose. This difference is important because token usage directly affects the operational costs of employing the models for ITOM tasks. Higher verbosity can substantially increase costs, making such models less desirable from a project cost management perspective.

Table 17: RQ3 - Average token count (verbosity) of Large Language Model Agents solving task in AIOps Context.

Query No.	Claude 3.5 Sonnet	Claude 3 Haiku	Claude 3 Opus	Mistral Largest	Mixtral 8x22B	Mistral Small 7B	GPT 3.5 turbo	GPT 4-o	GPT 4-o mini	GPT 4 turbo
Q-01	2841	2789	3147	2519	2609	2550	1713	1662	1679	1728
Q-02	3189	5668	5065	2833	3021	2831	1810	1950	1917	2059
Q-03	3056	6123	5101	6958	2857	2636	1813	1951	1919	2004
Q-04	3265	6223	4752	7841	3061	2848	0*	2174	2011	2097
Q-05	6386	6021	6859	5488	2627	5685	3870	3856	3850	3982
Q-06	6307	6093	6849	5597	2664	5582	3912	3801	3803	3915
Q-07	7178	6203	3927	5697	2993	2799	4197	3083	2210	2386
Q-08	3003	2934	3252	2628	2621	2633	1831	1754	1764	1840
Q-09	2894	2823	3243	7064	2621	2580	4297	1664	1673	1720
Q-10	6505	6173	6946	5798	2877	5673	3988	3942	3922	4018
Q-11	6574	6230	6972	5792	2826	5712	3998	3944	3920	4025
Q-12	6566	6200	6925	5727	2466	5677	3973	3927	3965	4022
Q-13	6369	6174	6928	5621	2727	5617	3916	3813	3807	3930
Q-14	6044	5783	6605	5201	2843	2552	3508	1646	1646	1701
Q-15	6055	5809	6697	5239	2658	2623	3533	3423	3417	3527
Q-16	6094	5829	6655	5247	2696	5267	3550	3437	3438	3554
Q-17	6152	5890	6709	5277	3117	5286	3566	3457	3457	3570
Q-18	6142	5867	6721	5279	2868	5290	3569	3460	3460	3575

Query No.	Claude 3.5 Sonnet	Claude 3 Haiku	Claude 3 Opus	Mistral Largest	Mixtral 8x22B	Mistral Small 7B	GPT 3.5 turbo	GPT 4-o	GPT 4-o mini	GPT 4 turbo
Q-19	6146	5886	6685	5278	3117	5287	3565	3457	3457	3570
Q-20	6187	5949	6724	20770	2859	2729	3693	3587	5374	3696
Q-21	34865	34023	35594	21463	2950	5062	0*	25882	28049	20108
Q-22	34822	25803	35862	33235	2994	5426	3601	11173	26946	21859
Q-23	6877	6191	6985	5551	2745	5562	3788	3678	3679	3791
Q-24	17319	12892	19398	19867	3021	5180	3638	11612	15648	11163
Q-25	84066	113426	87740	84946	3065	2810	3650	55193	12960	44105

5.4 Discussion

In this section, we discuss the key observations of developing tools for LLM agents and integrating them into the Red Hat OpenShift [97] platform.

To summarize our testing approach, we used the LangChain [63] and LangGraph [64] framework for agent creation and LLM integration. Additionally, we used for integration the Pydantic [94] library which allowed us to use the object-oriented programming (OOP) approach for data exchange with the different LLMs. For the agent setup, we used the default ReAct implementation from the LangGraph library, without any other tuning. We used this approach to ensure that we do not introduce any bias, by prompt manipulation, to any of the LLMs we tested. As mentioned in Section 5.2.4, we used a set of different questions in our tests that ranged from general knowledge, to platform knowledge (RedHat OpenShift), product documentation - through retrieval augmented generation (RAG), and application configuration (using MLASP [114] predictions to configure an mock application for a desired KPI throughput).

Chat agents may be implemented with a memory component (to retain aspects of earlier queries) or without it. Using a memory component for the chat assistant may seem beneficial, as it may follow up on queries from the user and behave like an ongoing conversational agent. This proved to work well for queries Q-08 and Q-09. However, it completely provided unexpected results when used later for queries Q-15 through Q-25 when the LLM did not calculate updated timestamp information anymore. To avoid this behavior, we disabled the memory component of the agent, which allowed correct calculations for the real IT operations queries and an incorrect response 100% of the time for query Q-09, as the LLMs were no longer capable of inferring that the Q-09 query may be in context with query Q-08 (expected response for query Q-09 would have been the LLM mentioning the river Seine). Since asking the agent to always calculate timestamps may decrease the user experience, we opted to remove the memory component from our agent setup. Another reason why we opted to remove the chat history feature (through removing the memory component) was that some models have a shorter context window which is directly affected by the length of the chat history feature. Also, when performing IT operations tasks, the response expected from the LLM may not be usable in future queries (e.g., Q-24 returns the name of a file). We recognize that in some cases the existence of a chat history may be beneficial. For example, after obtaining the response for Q-23 the user may want to use the agent to apply the response on the system (if such a tool were available to the agent).

Some of the queries we used for testing the agents' reasoning capabilities require detecting and using in the correct order of several tools, in some cases using the same tool repeatedly withing the same workflow. As an example, Table 18 describes the correct reasoning and steps taken by the LLM to solve the task. The LLM that provided the response was OpenAi's GPT 4-turbo.

Table 18: Query Q-24 correctly solved by GPT 4-turbo powered agent.

Step	**Agent activity**
1	Input: Find out the Prometheus service name and port number running in namespace demo. Use it to to plot all the prometheus metric data for the metric load_generator_total_msg starting 48 hours ago until now. Return only the content string of the tool and nothing else.
2	Entering new AgentExecutor chain... To solve the question, I need to first identify the Prometheus service name and port number in the "demo" namespace. Then, I will use this information to plot the Prometheus metric data for "load_generator_total_msg" from 48 hours ago until now.
3	Action: Summarize_Services_Information_In_OpenShift_Namespace Action Input: {"namespace": "demo"} namespace = 'demo' svc_summary = [ServiceInfo (name = 'grafana-demo-service', ports = [PortInfo (port=3000, name = 'grafana', protocol = 'TCP')], route = 'http://grafana-demo.apps.cluster-dnjmk.dnjmk.sandbox1590.opentlc.com/'), ServiceInfo (name = 'influxdb', ports = [PortInfo (port = 8086, name = 'No name available', protocol = 'TCP')], route = 'unavailable'), ServiceInfo (name = 'load-generator', ports = [PortInfo (port = 9090, name = 'metrics-app', protocol = 'TCP'), PortInfo (port = 9100, name = 'metrics-node', protocol = 'TCP')], route = 'unavailable'), ServiceInfo (name = 'prometheus-operated', ports = [PortInfo (port = 9090, name = 'web', protocol = 'TCP'), PortInfo (port = 10901, name = 'grpc', protocol = 'TCP')], route = 'unavailable')]

Step	Agent activity
4	From the observation, the Prometheus service in the "demo" namespace is named "prometheus-operated" and it uses port 9090 for web access. Next, I need to get the current time and the time 48 hours ago to define the time range for plotting the metric data.
5	Action: Get_timestamp_and_time_ISO Action Input: {"time_value": "now", "time_metric": "seconds", "ago_flag": 0} timestamp = 1730500568.411993 date_time_iso_format_string = '2024-11-01T18:36:08.411993-04:00' timezone = 'America/New_York'
6	Now I have the current timestamp. Next, I need to calculate the timestamp for 48 hours ago.
7	Action: Get_timestamp_and_time_ISO Action Input: {"time_value": 48, "time_metric": "hours", "ago_flag": 1} timestamp = 1730327770.333979 date_time_iso_format_string = '2024-10-30T18:36:10.333979-04:00' timezone = 'America/New_York'
8	Now I have the timestamp for 48 hours ago. With both timestamps, I can proceed to plot the metric data for "load_generator_total_msg" from 48 hours ago until now using the Prometheus service details.
9	Action: File_create_plot_irate Action Input: {"prom_service": "prometheus-operated", "prom_namespace": "demo", "prom_port": 9090, "metric_name": "load_generator_total_msg", "metric_range_start": 1730327770.333979, "metric_range_end": 1730500568.411993} file_name='FILE-plot-load_generator_total_msg-1730327770-1730500568.png'
10	I now know the final answer Final Answer: FILE-plot-load_generator_total_msg-1730327770-1730500568.png

We can see that in the first step, the agent receives the input query. In the second step, the agent provides a brief summary of the reasoning it needs to use to solve the task. We identify a first tool that the agent will use in solving the task in the third step where the *Summarize_Services_Information_In_OpenShift_Namespace* tool shall be used to find the Prometheus service name and port number of the Prometheus operator instance running in the specified namespace, in our case called *demo*. The response from the tool returns as a list of Pydantic objects. These items are then parsed by the LLM and provided as an observation in step four. In the same step, the LLM informs what the next steps will be. In steps five through eight, it uses twice the *Get_timestamp_and_time_ISO* tool in order to obtain the timestamps for the time range needed in step nine by the *File_create_plot_irate* tool. Naturally, the tool execution in this step returns the plot as a file which is then used by the agent executor in step ten to draw the conclusion that the response is now complete and final, thus returning the file name as its final answer.

As depicted in Table 12, we can see that seven of the ten models we tested were able to respond with various degree of accuracy to Q-24, six models providing over 80% accuracy. Looking into the chains call logs we can see that when the models fail they do for the following reasons:

- The model hallucinates the date range instead of using the tool to calculate the time interval.

- The model simply hallucinates a response by saying what the user should do to obtain the response.

- The model does not create the right workflow and chain of tools that it should use and responds that it did not find anything.

In some cases, for the Anthropic model family, the LLM does not determine the correct order in which to use the tools (particularly the timestamp calculation tool), but it eventually manages to correct itself. While this demonstrates a positive capability—the model can recover from erroneous reasoning—it may incur significant costs in both response time and the number of tokens consumed. Ultimately, this approach could be economically inefficient for the operations team.

Surprisingly, although Q-25 is almost identical in terms of reasoning with Q-24, with the exception of the GPT 4-turbo and GPT 4-o models, all other models encountered some challenges in providing the response. The challenges were as follows:

- Truncate the response (to some extent) - the case of Anthropic models. The impact of truncation depends on how the user intends to utilize the resulting information. If the truncated result is solely for review or display purposes, it may not pose an issue. However, if the result is subsequently used as input for a reporting tool to perform additional calculations, the truncation should be regarded as an error in processing.

- Improper use of tools (especially the timestamp calculation tool) - Anthropic Claude 3 Haiku, OpenAi's GPT 3.5 turbo and GPT 4-o mini
- Hallucinate a response: MistralAI's Mixtral 8x22B and Mistral Small
- Timeout from the model - Mistral Largest.

MistralAI also provides an LLM variant that functions as a mixture of experts in the Mixtral8 family (with different weight variants). However, our tests—following the framework and approach outlined in Section 5.2.4—showed that this model performed the worst. It frequently produced hallucinations related to IT operations tasks and failed to use the provided tools for calculations. As the model is closed-source, we cannot determine the root cause of this behavior. We recommend future studies that involve further tuning the agentic approach and reevaluating this model's performance.

We further report that in some cases the GPT 3.5 turbo model has failed to provide a response since the response exceeded the context window of the model.

To summarize, the best-performing models were the larger ones in the GPT-4 family, namely GPT-4 turbo and GPT-4-o. Notably, when we adjusted the Q-07 query to include the product name from the documentation stored in the RAG database, the model's performance on this query improved, highlighting once again the importance of specificity in the query prompt. The worst-performing model was Mixtral 8x22B, while Mistral Small 7B delivered acceptable results in most cases. This makes Mistral Small 7B a suitable candidate for local deployments, where improvements in performance and cost are relevant, as discussed in RQ2 5.3.2 and RQ3 5.3.3.

Overall, when we conducted an anonymous survey at various events showcasing AIOps capabilities for managing a Kubernetes-based infrastructure using an agentic approach with large language models, the respondents provided positive feedback. The concept was well received by diverse groups within the software industry, including development, infrastructure, and operations teams. This feedback aligns with current industry trends, where a more agentic AI-based approach is increasingly sought to manage operational tasks. The code and results of the experiments from this study are available in the following GitHub repository [31].

5.5 Threats to Validity

5.5.1 External Validity

Threats to external validity concern the generalizability of our findings. Our experiments were conducted using a single family of frameworks for Large Language Model integration: LangChain [63] and LangGraph [64]. These libraries provide abstractions over the implementation details required to interact with specific LLMs by leveraging the SDK (software development kit) provided by the model provider. However, other frameworks, such as LLamaIndex [75], offer similar functionalities and could be worth exploring in future studies.

Additionally, abstraction frameworks like LangChain and LangGraph may introduce limitations compared to using the native SDKs of specific models. This is because these frameworks aim to provide a unified interface that works across multiple models, potentially overlooking unique features offered by model-specific SDKs. As a result, not utilizing the specialized functionalities available in a dedicated SDK could negatively affect the performance of the overall agent.

5.5.2 Internal Validity

As we mentioned earlier, we performed our experiments using only the LangChain [63] and LangGraph [64] libraries, the python programming language variant. We did not test the behavior using the Javascript variants of the integration libraries.

5.5.3 Construct Validity

We conducted our experiments using specific versions of the LangChain [63] and LangGraph [64] libraries, specifically versions *0.2.12* and *0.2.10*, respectively. It is possible that newer versions of these libraries may behave differently, potentially offering enhancements to the client-side instance of the LLMs. Another aspect of our experimental setup is the reliance on the serving capabilities provided by the

LLM model owners (OpenAI, Anthropic, MistralAI) and their assurance of runtime compatibility with the integration libraries we employed.

As Anthropic and OpenAI models are fully closed, we could not test their agentic functionality in custom, locally deployed runtimes. While testing the MistralAI model family in a local environment is feasible, the compute requirements (both CPU and GPU) for larger models made it cost-prohibitive. However, we tested Mistral Small 7B locally using a vLLM [116] runtime but were unable to replicate the results obtained from the MistralAI endpoints. This discrepancy was due to a missing feature in the vLLM-LangChain integration library at the time of our experiments, specifically the *bind_tools* function. Further testing may be feasible with other runtime servers or with updated vLLM serving runtimes once the missing functionality becomes available. Future studies will aim to address these limitations.

Chapter 6

Conclusion

In this chapter we summarize the contributions of this dissertation. We also propose future work related to AIOps and agentic AIOps practises for large scale systems.

We presented a series of empirical studies, each one covering a particular goal in the scope of improvements to load testing, capacity planning and to operational efficiency of software systems with emphasis on large scale and distributed systems.

Load testing is a critical activity in the software development lifecycle (SDLC) as it ensures that the software system behaves correctly within defined SLA ranges under loads as close as possible to real world usage. As a result, it is recommended that load test engineers run as many tests as possible using different loads and system configurations. Furthermore, the results of load testing are also critical in calculating the required software, environment and deployment settings for the components of a software ecosystem required to have a desired capacity for business processing.

The use of frameworks that accelerate and offer cost savings to businesses when performing the tasks related to both load testing and capacity planning are highly appreciated, especially when the frameworks are extensible, and non-intrusive in their approach. Using a non-intrusive approach for KPI information retrieval is very important as it simplifies the integration of these methods with existing software ecosystems. Including Machine Learning models in the proposed frameworks leverages cutting edge technologies for solving the challenges raised by the use of traditional methods.

Furthermore, the rise of Large Language Models opens new possibilities for managing large scale systems. Incorporating LLM powered agents equipped with tools can lift operational burdens. Having LLM powered agents equipped with predictive machine learning (ML) tools is a game changer for modern IT operations and business processes. These agents combine the natural language understanding and contextual reasoning capabilities of LLMs with the powerful data analysis and forecasting abilities of predictive ML models. This synergy enables them not only to interpret complex queries, but also to proactively offer actionable insights and predictions to uphold service level agreements (SLAs). By integrating predictive tools, these agents can go beyond reactive responses, offering a forward-thinking approach that enhances decision making, reduces downtime, and optimizes resource utilization, ultimately improving overall operational efficiency.

6.1 Contributions

In this dissertation we focus on improvements to different processes that are part of the software development life cycle (SDLC) for distributed large scale systems. In particular we focus on improvements to load testing, capacity planning and use of AIOps for IT Operations management. Our studies managed to fill in process gaps in the practice of the aforementioned processes. Below, we outline the key contributions:

1. **MLASP - Machine Learning Assisted Capacity Planning** [114] is an industrial report that introduces a novel approach to capacity planning for large-scale industrial systems using automation and machine learning. A central aspect of this approach is the use of non-intrusive data collection, which enhances its applicability to a broader range of systems, including closed-source ones. Our findings demonstrate that leveraging machine learning (ML) enables highly accurate predictions of key performance indicators (KPIs), with the difference between predicted and actual throughput being less than 1%. Additionally, we show that effective results can be achieved even when training on a smaller subset of data—for instance, just 3% of the total data in the case of the open-source system. By adopting MLASP, our industrial partner experienced substantial cost savings in the load testing process and reduced operational costs in production environments.

2. **MLOLET - Machine Learning Optimized Load and Endurance Testing** [115] is an industrial report presenting a novel approach to improving the

efficiency of load and endurance testing processes through the early detection and termination of failing tests. This approach leverages machine learning (ML) for time series modeling to identify spikes, which are then validated against business rules to assess the health of ongoing tests. Our findings indicate that adopting MLOLET in an industrial environment can lead to significant cost savings across multiple areas. For instance, it can reduce the setup and execution time for load testing by up to 50% and cut the time required for analyzing load test data for spikes by up to 65%. Moreover, the number of load and endurance tests that can be conducted within the project timeline may increase by up to 100%, depending on the specific business scenario. Lastly, we show that time series models trained on load testing data may be used in production environments to help with IT Operations management processes.

3. **Using Large Language Models for Adding AIOps Capabilities to Large-Scale Systems** - an empirical study evaluating the effectiveness of LLM-powered agents in performing IT Operations management tasks, including capacity planning. The study assesses the agents' performance based on their accuracy in resolving user queries, the time taken to address these queries, and their associated costs (measured in token usage). We compare the performance of various state-of-the-art models across different scenarios. Our findings indicate that OpenAI's fourth-generation GPT family consistently delivers the best results within the proposed testing framework. However, other models also demonstrate acceptable performance, depending on the specific scenario and business requirements.

6.2 Future Research Directions

While we have introduced approaches to enhance the efficiency of various IT operations management processes, several challenges remain that warrant further exploration. These challenges are discussed in the following sections.

6.2.1 Evaluating Agentic frameworks

In Chapter 5, we discussed that our LLM-powered agents were implemented using one family of frameworks: LangChain [63] and LangGraph [64]. While these libraries support a variety of models, other libraries, such as LLamaIndex [75], also offer similar capabilities and may be worth investigating. Additionally, constructing agents using native SDKs could provide certain benefits compared to generic abstraction frameworks like LangChain and LangGraph. Leveraging dedicated functionalities available in native SDKs might enhance performance and flexibility, offering advantages over the generalized features provided by abstraction frameworks.

6.2.2 Improving Cost Efficiency for Agentic AIOps

As discussed in Chapter 5, the use of LLM-powered agents in IT Operations management can be expensive, both in terms of the time required for the agent to generate a response and the financial cost associated with token usage. As LLMs become more advanced and open-source models increasingly emerge, it would be valuable to investigate the performance of these agents when deployed locally, on the same platform (e.g., RedHat OpenShift [97]) where tools, data and other managed workloads are hosted. Running LLMs locally could potentially lead to significant operational expenditure (Opex) savings. This cost reduction could enable the testing and deployment of autonomous monitoring agents capable of continuous 24x7 application ecosystem oversight. Such agents could also utilize tools to perform preventive maintenance, including adjustment of parameters on the fly (e.g. use the ideas from MLOLET and MLASP simultaneously) and recovery actions for monitored applications. This approach could broaden the accessibility and scalability of agents for AIOps purposes, offering enhanced operational efficiency and reduced costs.

Bibliography

[1] C. Aggarwal, C. Chen, and Jiawei Han. The inverse classification problem. *Journal of Computer Science and Technology*, 25:458–468, 2010.

[2] AIOps. Applying aiops platforms to broader datasets will create unique business insights, 2016. https://www.gartner.com/en/documents/3364418.

[3] Ayedh H. ALQahtani and Andrew Whyte. Estimation of life-cycle costs of buildings: regression vs artificial neural network. 2016.

[4] Ateret Anaby-Tavor, Boaz Carmeli, Esther Goldbraich, Amir Kantor, George Kour, Segev Shlomov, Naama Tepper, and Naama Zwerdling. Do not have enough data? deep learning to the rescue! In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7383–7390, 2020.

[5] Ansible. Ansible - automation for everyone, 2019. https://www.ansible.com/.

[6] Apache. Apache kafka - a distributed streaming platform, 2019. https://kafka.apache.org/.

[7] Apache CouchDb. Apachecouchdb - the nosql database, 2019. https://couchdb.apache.org/.

[8] Apache Spark. Apache spark - unified engine for large-scale data analytics, 2019. https://spark.apache.org/.

[9] Liang Bao, Xin Liu, Ziheng Xu, and Baoyin Fang. Autoconfig: Automatic configuration tuning for distributed message systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, 2018.

[10] Liang Bao, Xin Liu, Ziheng Xu, and Baoyin Fang. Autoconfig: automatic configuration tuning for distributed message systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, pages 29–40, 2018.

[11] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009. ISSN 1935-8237. doi: 10.1561/2200000006. URL `http://dx.doi.org/10.1561/2200000006`.

[12] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[13] Eric A. Brewer. Towards robust distributed systems (abstract). In *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 2000. URL `https://api.semanticscholar.org/CorpusID:207631444`.

[14] Raymond Brueckner and Björn Schuller. Social signal classification using deep blstm recurrent neural networks. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4823–4827, 2014.

[15] chef. Chef - configuration management automation software, 2019. https://www.chef.io/products/chef-infrastructure-management.

[16] Jianfeng Chen, Joymallya Chakraborty, Philip V. Clark, Kevin Haverlock, Snehit Cherian, and Tim Menzies. Predicting breakdowns in cloud services (with spike). *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019.

[17] Tianping Chen, Hong Chen, and Ruey-wen Liu. A constructive proof and an extension of cybenko's approximation theorem. In Connie Page and Raoul LePage, editors, *Computing Science and Statistics*, pages 163–168, New York, NY, 1992. Springer New York.

[18] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[19] Tse-Hsun Chen, Stephen W. Thomas, Meiyappan Nagappan, and Ahmed E. Hassan. Explaining software defects using topic models. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, page 189–198, 2012.

[20] Tse-Hsun Chen, Weiyi Shang, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. Cacheoptimizer: Helping developers configure caching frameworks for hibernate-based database-centric web applications. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, page 666–677, 2016.

[21] Tse-Hsun Chen, Mark D Syer, Weiyi Shang, Zhen Ming Jiang, Ahmed E Hassan, Mohamed Nasser, and Parminder Flora. Analytics-driven load testing: An industrial experience report on load testing of large-scale systems. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pages 243–252, 2017.

[22] Qian Cheng, Doyen Sahoo, Amrita Saha, Wenzhuo Yang, Chenghao Liu, Gerald Woo, Manpreet Singh, Silvio Saverese, and Steven CH Hoi. Ai for it operations (aiops) on cloud platforms: Reviews, opportunities and challenges. *arXiv preprint arXiv:2304.04661*, 2023.

[23] Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. A survey of chain of thought reasoning: Advances, frontiers and future. *arXiv preprint arXiv:2309.15402*, 2023.

[24] Cloudera Documentation. Configuring apache kafka for performance and resource management, 2018. https://docs.cloudera.com/documentation/kafka/latest/topics/kafka_performance.html.

[25] Confluent Blogs. Optimizing your apache kafka deployment, 2017. https://www.confluent.io/blog/optimizing-apache-kafka-deployment/.

[26] George Coulouris. Distributed systems : concepts and design / george coulouris ... [et al.]. 2011. URL `https://api.semanticscholar.org/CorpusID:155813208`.

[27] DeepLearning4J. Deeplearning4j - jvm based deep learning, 2019. https://deeplearning4j.konduit.ai/.

[28] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL `https://arxiv.org/abs/1810.04805`.

[29] elasticsearch. Elasticsearch - the distributed, restful search and analytics engine, 2019. https://www.elastic.co/elasticsearch.

[30] Tolga Ergen and Suleyman S Kozat. Online training of lstm networks in distributed systems for variable length data sequences. *IEEE transactions on neural networks and learning systems*, 29(10):5159–5165, 2017.

[31] Evaluating LLM Agents for AIOps on Red Hat OpenShift. Evaluating llm agents for aiops on red hat openshift - a full implementation example of agents with langgraph and langchain on red hat openshift, 2024. https://github.com/eartvit/llm-agents-on-ocp.

[32] FastCompany. How one second could cost Amazon 1.6 billion sales. `http://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales`, 2016. Last accessed March 3 2016.

[33] Flask. Flask - the micro web framework, 2019. https://flask.palletsprojects.com/.

[34] Apache Software Foundation. Apache jmeter, 2019. https://jmeter.apache.org/.

[35] Lynn Friedman and Melanie Wall. Graphical views of suppression and multicollinearity in multiple linear regression. *The American Statistician*, 59:127–136, 02 2005. doi: 10.1198/000313005X41337.

[36] Agustin Garcia Asuero, Ana Sayago, and Gustavo Gonzalez. The correlation coefficient: An overview. *Critical Reviews in Analytical Chemistry - CRIT REV ANAL CHEM*, 36:41–59, 01 2006. doi: 10.1080/10408340500526766.

[37] Dario García-Gasulla, Ferran Parés, Armand Vilalta, Jonathan Moreno, Eduard Ayguadé, Jesús Labarta, Ulises Cortés, and Toyotaro Suzumura. On the

behavior of convolutional nets for feature extraction. *J. Artif. Intell. Res.*, 61: 563–592, 2018.

[38] Antonio Giulli and Sujit Pal. *Deep Learning with Keras*. Packt Publishing Ltd., 2017. ISBN 978-1-78712-842-2.

[39] Maloy Jyoti Goswami. Challenges and solutions in integrating ai with multi-cloud architectures.

[40] Haryadi S. Gunawi, Mingzhe Hao, Riza O. Suminto, Agung Laksono, Anang D. Satria, Jeffry Adityatama, and Kurnia J. Eliazar. Why does the cloud stop computing?: Lessons from hundreds of service outages. *Proceedings of the Seventh ACM Symposium on Cloud Computing*, 2016.

[41] Hongcheng Guo, Jian Yang, Jiaheng Liu, Liqun Yang, Linzheng Chai, Jiaqi Bai, Junran Peng, Xiaorong Hu, Chao Chen, Dongfeng Zhang, Xu Shi, Tieqiao Zheng, Liangfan Zheng, Bo Zhang, Ke Xu, and Zhoujun Li. Owl: A large language model for it operations. *ArXiv*, abs/2309.09298, 2023. URL `https://api.semanticscholar.org/CorpusID:262043747`.

[42] Jianmei Guo, K. Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wasowski. Variability-aware performance prediction: A statistical learning approach. *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 301–311, 2013. URL `https://api.semanticscholar.org/CorpusID:14770655`.

[43] Jianmei Guo, Dingyu Yang, Norbert Siegmund, Sven Apel, Atrisha Sarkar, Pavel Valov, K. Czarnecki, Andrzej Wasowski, and Huiqun Yu. Data-efficient performance learning for configurable systems. *Empirical Software Engineering*, 23:1826–1867, 2018. URL `https://api.semanticscholar.org/CorpusID:3282867`.

[44] Huong Ha and Hongyu Zhang. Deepperf: Performance prediction for configurable software with deep sparse neural network. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, pages 1095–1106, 2019.

[45] Huong Ha and Hongyu Zhang. Deepperf: Performance prediction for configurable software with deep sparse neural network. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, pages 1095–1106, 2019.

[46] Fateme Hadadi, Qinghua Xu, Domenico Bianculli, and Lionel C. Briand. Anomaly detection on unstable logs with gpt models. *ArXiv*, abs/2406.07467, 2024. URL `https://api.semanticscholar.org/CorpusID:270379599`.

[47] Frank E. Harrell. *Regression Modeling Strategies*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387952322.

[48] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Bin Gani, and Samee Ullah Khan. The rise of "big data" on cloud computing: Review and open research issues. *Inf. Syst.*, 47:98–115, 2015. URL `https://api.semanticscholar.org/CorpusID:205488005`.

[49] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[50] Anna Hedström, Leander Weber, Dilyara Bareeva, Daniel Krakowczyk, Franz Motzkus, Wojciech Samek, Sebastian Lapuschkin, and Marina M. C. Höhne. Quantus: An explainable ai toolkit for responsible evaluation of neural network explanations and beyond, 2023. URL `https://arxiv.org/abs/2202.06861`.

[51] Taojun Hu and Xiao-Hua Zhou. Unveiling llm evaluation focused on metrics: Challenges and solutions, 2024. URL `https://arxiv.org/abs/2404.09135`.

[52] Xiaolei Hua, Lin Zhu, Shenglin Zhang, Zeyan Li, Su Wang, Dong Zhou, Shuo Wang, and Chao Deng. Genad: General representations of multivariate time seriesfor anomaly detection. *ArXiv*, abs/2202.04250, 2022.

[53] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963, 2019.

[54] Jenkins. Jenkins - the open source automation server, 2019. https://www.jenkins.io/.

[55] Yuxuan Jiang, Chaoyun Zhang, Shilin He, Zhihao Yang, Minghua Ma, Si Qin, Yu Kang, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. Xpert: Empowering incident management with query recommendations via large language models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400702174. doi: 10.1145/3597503.3639081. URL `https://doi.org/10.1145/3597503.3639081`.

[56] Z. M. Jiang and A. E. Hassan. A survey on load testing of large-scale software systems. *IEEE Transactions on Software Engineering*, 41(11):1091–1118, 2015.

[57] Zhihan Jiang, Jinyang Liu, Junjie Huang, Yichen Li, Yintong Huo, Jia-Yuan Gu, Zhuangbin Chen, Jieming Zhu, and Michael R. Lyu. A large-scale evaluation for log parsing techniques: How far are we? *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023. URL `https://api.semanticscholar.org/CorpusID:261049500`.

[58] JMXTerm. Interactive terminal based jmx client, 2017. https://docs.cyclopsgroup.org/welcome.

[59] Manjunath Jogin, Mohana, M S Madhulika, G D Divya, R K Meghana, and S Apoorva. Feature extraction using convolution neural networks (cnn) and deep learning. In *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pages 2319–2323, 2018. doi: 10.1109/RTEICT42901.2018.9012507.

[60] Maximilian Kohlbrenner, Alexander Bauer, Shinichi Nakajima, Alexander Binder, Wojciech Samek, and Sebastian Lapuschkin. Towards best practice in explaining neural network decisions with lrp, 2020. URL `https://arxiv.org/abs/1910.09840`.

[61] Kubernetes. Kubernetes - production-grade container orchestration, 2022. https://kubernetes.io/.

[62] Bharath Kumar. Challenges and solutions for integrating ai with multi-cloud architectures. *International Journal of Multidisciplinary Innovation and Research Methodology, ISSN: 2960-2068*, 1(1):71–77, 2022.

[63] LangChain. Langchain - a framework for developing applications powered by large language models (llms)., 2024. https://python.langchain.com/docs/introduction/.

[64] LangGraph. Langgraph - building agents as graphs, 2024. https://langchain-ai.github.io/langgraph/.

[65] Nikolay Laptev, Jason Yosinski, Li Erran Li, and Slawek Smyl. Time-series extreme event forecasting with neural networks at uber. In *International conference on machine learning*, volume 34, pages 1–5. sn, 2017.

[66] S. Lathuiliére, P. Mesejo, X. Alameda-Pineda, and R. Horaud. A comprehensive analysis of deep regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2019.

[67] P. Le Noac'h, A. Costan, and L. Bougé. A performance evaluation of apache kafka in support of big data streaming applications. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 4803–4806, 2017.

[68] Anna Levin, Shelly Garion, Elliot K. Kolodner, Dean H. Lorenz, Katherine Barabash, Mike Kugler, and Niall McShane. Aiops for a cloud object storage service. In *2019 IEEE International Congress on Big Data (BigDataCongress)*, page 165–169. IEEE, July 2019. doi: 10.1109/bigdatacongress.2019.00036. URL `http://dx.doi.org/10.1109/BigDataCongress.2019.00036`.

[69] Heng Li, Tse-Hsun (Peter) Chen, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. Adopting autonomic computing capabilities in existing large-scale systems: An industrial experience report. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '18, pages 1–10, 2018.

[70] Heng Li, Tse-Hsun (Peter) Chen, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. Adopting autonomic computing capabilities in existing large-scale systems: An industrial experience report. In *Proceedings of the 40th*

International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP '18, pages 1–10, 2018.

[71] Yichen Li, Yintong Huo, Renyi Zhong, Zhihan Jiang, Jinyang Liu, Junjie Huang, Jiazhen Gu, Pinjia He, and Michael R. Lyu. Go static: Contextualized logging statement generation. *ArXiv*, abs/2402.12958, 2024. URL `https://api.semanticscholar.org/CorpusID:267760100`.

[72] Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, 379(2194):20200209, 2021.

[73] Wen-Hui Lin, Ping Wang, Kuo-Ming Chao, Hsiao-Chung Lin, Zong-Yu Yang, and Yu-Huang Lai. Wind power forecasting with deep learning networks: Time-series forecasting. *Applied Sciences*, 11(21), 2021. ISSN 2076-3417. doi: 10.3390/app112110335. URL `https://www.mdpi.com/2076-3417/11/21/10335`.

[74] Yilun Liu, Shimin Tao, Weibin Meng, Jingyu Wang, Wenbing Ma, Yuhang Chen, Yanqing Zhao, Hao Yang, and Yanfei Jiang. Interpretable online log analysis using large language models with prompt strategies. *2024 IEEE/ACM 32nd International Conference on Program Comprehension (ICPC)*, pages 35–46, 2023. URL `https://api.semanticscholar.org/CorpusID:260900274`.

[75] LlamaIndex. Llamaindex - a framework for building context-augmented generative ai applications with llms including agents and workflows., 2024. https://docs.llamaindex.ai/en/stable/.

[76] Load Generator. Load generator - load testing controller with spike load capability, 2023. https://github.com/eartvit/load-generator.

[77] logstash. Logstash - collect, parse, transform logs, 2019. https://www.elastic.co/logstash.

[78] Helmut Lütkepohl. *Specification and Checking the Adequacy of VARMA Models*, pages 493–514. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-27752-1. doi: 10.1007/978-3-540-27752-1_13. URL `https://doi.org/10.1007/978-3-540-27752-1_13`.

[79] Lucy Ellen Lwakatare, Aiswarya Raj, Ivica Crnkovic, Jan Bosch, and Helena Holmström Olsson. Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions. *Information and Software Technology*, 127:106368, 2020. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2020.106368. URL `https://www.sciencedirect.com/science/article/pii/S0950584920301373`.

[80] Zeyang Ma, An Ran Chen, Dong Jae Kim, Tse-Husn Chen, and Shaowei Wang. Llmparser: An exploratory study on using large language models for log parsing. *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, pages 1209–1221, 2024. URL `https://api.semanticscholar.org/CorpusID:269123285`.

[81] Zeyang Ma, Dong Jae Kim, and Tse-Husn Chen. Openlogparser: Unsupervised parsing with open-source large language models. *ArXiv*, abs/2408.01585, 2024. URL `https://api.semanticscholar.org/CorpusID:271709638`.

[82] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey, 2024. URL `https://arxiv.org/abs/2402.06196`.

[83] MLASP. Mlasp - open source system experimental data, 2020. https://github.com/SPEAR-SE/mlasp.

[84] MLASP on OpenShift. Mlasp on openshift - a full implementation example of the mlasp framework, 2022. https://github.com/eartvit/mlasp-on-ocp.

[85] MLOLET on OpenShift. Mlolet on openshift - a full implementation example of the mlolet framework on red hat openshift, 2024. https://github.com/eartvit/mlolet-on-ocp.

[86] Pablo Montero-Manso, George Athanasopoulos, Rob J Hyndman, and Thiyanga S Talagala. Fforma: Feature-based forecast model averaging. *International Journal of Forecasting*, 36(1):86–92, 2020.

[87] MySQL. Mysql - an open source rdbms database, 2019. https://www.mysql.com/products/.

[88] Andrew Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 78, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138385. doi: 10.1145/1015330.1015435. URL `https://doi.org/10.1145/1015330.1015435`.

[89] Kamal Nigam, John Lafferty, and Andrew McCallum. Using maximum entropy for text classification. In *IJCAI-99 workshop on machine learning for information filtering*, volume 1, pages 61–67. Stockholom, Sweden, 1999.

[90] Bingyue Pan. Application of xgboost algorithm in hourly pm2.5 concentration prediction. *IOP Conference Series: Earth and Environmental Science*, 113: 012127, 02 2018. doi: 10.1088/1755-1315/113/1/012127.

[91] PostgreSQL. Postgresql - the world's most advanced open source relational database, 2019. https://www.postgresql.org/.

[92] Prometheus. Prometheus - monitoring system and timeseries database, 2022. https://prometheus.io/.

[93] Puppet. Puppet - infrastructure automation compliance at enterprise scale, 2019. https://www.puppet.com/.

[94] Pydantic. Pydantic - a widely used data validation library for python., 2024. https://docs.pydantic.dev/latest/.

[95] PyTorch. Pytorch - an end to end machine learning framework, 2019. https://pytorch.org/.

[96] Rabbit MQ. Rabbit mq - an open source message broker system, 2020. https://www.rabbitmq.com/.

[97] Red Hat OpenShift. Red hat openshift - the industry's leading hybrid cloud application platform powered by kubernetes, 2022. https://www.redhat.com/en/technologies/cloud-computing/openshift.

[98] Yuji Roh, Geon Heo, and Steven Euijong Whang. A survey on data collection for machine learning: a big data-ai integration perspective. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1328–1347, 2019.

[99] Ananya B. Sai, Akash Kumar Mohankumar, and Mitesh M. Khapra. A survey of evaluation metrics used for nlg systems, 2020. URL `https://arxiv.org/abs/2008.12009`.

[100] Komal Sarda, Zakeya Namrud, Marin Litoiu, Larisa Shwartz, and Ian Watts. Leveraging large language models for the auto-remediation of microservice applications: An experimental study. In *SIGSOFT FSE Companion*, 2024. URL `https://api.semanticscholar.org/CorpusID:271099029`.

[101] Abdel Salam Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. Scalable product line configuration: A straw to break the camel's back. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, ASE'13, page 465474. IEEE Press, 2013. ISBN 9781479902156.

[102] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. doi: 10.1109/78.650093.

[103] SciKit-Learn. Scikit learn - machine learning in python, 2019. https://pypi.org/project/psutil.

[104] Seldon. Seldon - mlops for the enterprise, 2019. https://www.seldon.io/.

[105] Zhipeng Shen, Yuanming Zhang, Jiawei Lu, Jun Xu, and Gang Xiao. A novel time series forecasting model with deep learning. *Neurocomputing*, 396:302–313, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2018.12.084. URL `https://www.sciencedirect.com/science/article/pii/S0925231219304461`.

[106] Bikesh Kumar Singh, Kesari Verma, and A. S. Thoke. Investigations on impact of feature normalization techniques on classifier's performance in breast tumor classification. *International Journal of Computer Applications*, 116:11–15, 2015.

[107] SoapUI. Soapui - the world's most widely-used automated testing tool for soap and rest apis, 2019. https://www.soapui.org/.

[108] J. Sola and Joaquin Sevilla. Importance of input data normalization for the application of neural networks to complex industrial problems. *Nuclear Science, IEEE Transactions on*, 44:1464 – 1468, 07 1997. doi: 10.1109/23.589532.

[109] Siwoon Son, Myeong-Seon Gil, Yang-Sae Moon, and Hee-Sun Won. Anomaly detection of hadoop log data using moving average and 3-sigma. 2016.

[110] Yang Syu, Chien-Min Wang, and Yong-Yi Fanjiang. A survey of time-aware dynamic qos forecasting research, its future challenges and research directions. In *International Conference on Services Computing*, pages 36–50. Springer, 2018.

[111] TensorFlow. Tensorflow - an end-to-end open source machine learning platform, 2019. https://www.tensorflow.org/.

[112] Robert Tibshirani. Regression shrinkage selection via the lasso. *Journal of the Royal Statistical Society Series B*, 73:273–282, 06 2011. doi: 10.2307/41262671.

[113] John W Tukey et al. *Exploratory data analysis*, volume 2. Reading, MA, 1977.

[114] Arthur Vitui and Tse-Hsun Peter Chen. Mlasp: Machine learning assisted capacity planning. *Empirical Software Engineering*, 26:1–27, 2021. URL `https://api.semanticscholar.org/CorpusID:236930377`.

[115] Arthur Vitui and Tse-Hsun Peter Chen. Mlolet - machine learning optimized load and endurance testing: An industrial experience report. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE'24, page 465474. IEEE Press, 2024. ISBN 979-8-4007-1248-7/24/10.

[116] vLLM. vllm - a fast and easy-to-use library for llm inference and serving., 2024. https://docs.vllm.ai/en/latest/.

[117] Chaojie Wang, Yishi Xu, Zhong Peng, Chenxi Zhang, Bo Chen, Xinrun Wang, Lei Feng, and Bo An. keqing: knowledge-based question answering is a nature chain-of-thought mentor of llm. *arXiv preprint arXiv:2401.00426*, 2023.

[118] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL `https://arxiv.org/abs/2201.11903`.

[119] Wangyang Wei, Honghai Wu, and Huadong Ma. An autoencoder and lstm-based traffic flow prediction method. *Sensors (Basel, Switzerland)*, 19, 2019.

[120] Tailai Wen and Roy Keyes. Time series anomaly detection using convolutional neural networks and transfer learning. *ArXiv*, abs/1905.13628, 2019.

[121] WireMock. Wiremock: Mock the apis you depend on, 2022. https://wiremock.org/.

[122] WireMock Metrics. Wiremock metrics - extended wiremock with prometheus metrics and global random string payload responsetransformer, 2023. https://github.com/eartvit/wiremock-metrics2.

[123] Martin Wöllmer, Florian Eyben, Björn Schuller, Ellen Douglas-Cowie, and Roddy Cowie. Data-driven clustering in emotional space for affect recognition using discriminatively trained lstm networks. In *Proc. Interspeech 2009, Brighton, UK*, pages 1595–1598, 2009.

[124] Junjielong Xu, Ziang Cui, Yuan Zhao, Xu Zhang, Shilin He, Pinjia He, Liqun Li, Yu Kang, Qingwei Lin, Yingnong Dang, S. Rajmohan, and Dongmei Zhang. Unilog: Automatic logging via llm and in-context learning. *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, pages 1–12, 2024. URL `https://api.semanticscholar.org/CorpusID:267523731`.

[125] Junjielong Xu, Ruichun Yang, Yintong Huo, Chengyu Zhang, and Pinjia He. Divlog: Log parsing with prompt enhanced in-context learning. *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, pages 2457–2468, 2024. URL `https://api.semanticscholar.org/CorpusID:269123195`.

[126] Yun Xu and Royston Goodacre. On splitting training and validation set: A comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning. *Journal of Analysis and Testing*, 2, 10 2018. doi: 10.1007/s41664-018-0068-2.

[127] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL `https://arxiv.org/abs/2210.03629`.

[128] Zuoning Yin, Xiao Ma, Jing Zheng, Yuanyuan Zhou, Lakshmi N. Bairavasundaram, and Shankar Pasupathy. An empirical study on configuration errors in commercial and open source systems. SOSP '11, page 159–172, 2011.

[129] Zhaoyang Yu, Ming-Jie Ma, Chaoyun Zhang, Si Qin, Yu Kang, Chetan Bansal, S. Rajmohan, Yingnong Dang, Changhua Pei, Dan Pei, Qingwei Lin, and Dongmei Zhang. Monitorassistant: Simplifying cloud service monitoring via large language models. In *SIGSOFT FSE Companion*, 2024. URL `https://api.semanticscholar.org/CorpusID:271098855`.

[130] Giancarlo Zaccone, Md. Rezaul Karim, and Ahmed Menshawy. *Deep Learning with TensorFlow*. Packt Publishing Ltd., 2017. ISBN 978-1-78646-978-6.

[131] Chunkai Zhang and Ao Yin. Anomaly detection algorithm based on subspace local density estimation. *Int. J. Web Serv. Res.*, 16:44–58, 2019.

[132] Lingxue Zhu and Nikolay Pavlovich Laptev. Deep and confident prediction for time series at uber. *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 103–110, 2017.