

TECHNICAL UNIVERSITY OF CRETE, GREECE

**SCHOOL OF ELECTRICAL AND COMPUTER
ENGINEERING**

**Collection and Analysis of Datasets for AI-Driven
Networking Algorithms**



**ΠΟΛΥΤΕΧΝΕΙΟ
ΚΡΗΤΗΣ**

Georgios Skoulas

Thesis Committee

Supervisor: *Professor Thrasyvoulos Spyropoulos (ECE)*

Professor George Karystinos (ECE)

Professor Dionysios Christopoulos (ECE)

Chania, February 2025

Abstract

The integration of artificial intelligence (AI) into networking systems has ushered in a new era of efficiency, scalability, and intelligence in managing modern communication infrastructures. However, the effectiveness of AI-driven networking algorithms is intrinsically tied to the quality and relevance of the datasets used for their development and evaluation. This thesis focuses on the collection, preprocessing, and analysis of datasets derived from diverse domains, including cloud computing, 4G networks, and online platforms like YouTube, to enable the design of advanced AI-driven algorithms.

We present a comprehensive study of four key datasets: Helios and Philly, representing GPU-based cloud computing workloads, a 4G LTE dataset capturing cellular network performance under varying mobility conditions, and a YouTube dataset encompassing user engagement metrics. Each dataset is meticulously preprocessed and analyzed to address challenges such as non-stationarity, heavy-tailed distributions, and missing data. Time-series analysis techniques, including rolling mean, autocorrelation function (ACF), and the Augmented Dickey-Fuller (ADF) test, are applied to uncover statistical properties and enhance data suitability for predictive modeling.

This work also demonstrates the practical applications of these datasets by developing predictive models using techniques such as ARIMA and neural networks. The models are evaluated for their ability to forecast key performance metrics, optimize resource allocation, and enhance the reliability of networking systems. Additionally, insights from the analysis inform strategies for improving system performance and developing error-resilient scheduling policies in GPU clusters and cellular networks.

The findings of this thesis underscore the critical role of robust datasets in advancing AI-driven networking algorithms. By addressing the challenges of data collection and preprocessing and showcasing their potential in real-world scenarios, this work contributes to the foundation for future innovations in intelligent networking systems.

Περίληψη

Η ενσωμάτωση της τεχνητής νοημοσύνης στα δικτυακά συστήματα έχει εγκαινιάσει μια νέα εποχή αποδοτικότητας, επεκτασιμότητας και ευφυΐας στη διαχείριση των σύγχρονων επικοινωνιακών υποδομών. Ωστόσο, η αποτελεσματικότητα των αλγορίθμων δικτύωσης που βασίζονται στην τεχνητή νοημοσύνη εξαρτάται άμεσα από την ποιότητα των δεδομένων που χρησιμοποιούνται για την ανάπτυξη και αξιολόγησή τους. Η παρούσα διπλωματική εργασία εστιάζει στη συλλογή, προεπεξεργασία και ανάλυση συνόλων δεδομένων από διάφορους τομείς, όπως το cloud computing, τα δίκτυα 4G και πλατφόρμες όπως το YouTube, με στόχο τον σχεδιασμό προηγμένων αλγορίθμων που βασίζονται στην τεχνητή νοημοσύνη.

Παρουσιάζεται μια εκτενής μελέτη τεσσάρων βασικών συνόλων δεδομένων: Helios και Philly, τα οποία αντιπροσωπεύουν φόρτους εργασίας βασισμένους σε GPU στο cloud computing· ένα σύνολο δεδομένων 4G LTE που καταγράφει την απόδοση κυψελοειδών δικτύων υπό διάφορες συνθήκες κινητικότητας· και ένα σύνολο δεδομένων από το Youtube που περιλαμβάνει μετρήσεις αλληλεπίδρασης χρηστών. Κάθε σύνολο δεδομένων προεπεξεργάζεται και αναλύεται διεξοδικά για την αντιμετώπιση προκλήσεων, όπως η μη στασιμότητα, οι βαριές ουρές κατανομών και τα ελλιπή δεδομένα. Τεχνικές ανάλυσης χρονοσειρών, όπως η κυλιόμενη μέση τιμή, η συνάρτηση αυτοσυσχέτισης και το τεστ Augmented Dickey-Fuller (ADF), εφαρμόζονται για την εξαγωγή στατιστικών ιδιοτήτων και τη βελτίωση της καταλληλότητας των δεδομένων για μοντέλα προβλέψεων.

Επιπλέον, η εργασία καταδεικνύει τις πρακτικές εφαρμογές αυτών των συνόλων δεδομένων μέσω της ανάπτυξης μοντέλων, όπως το ARIMA και τα νευρωνικά δίκτυα. Τα μοντέλα αξιολογούνται για την ικανότητά τους να προβλέπουν σύμφωνα με δείκτες απόδοσης, να βελτιστοποιούν την κατανομή πόρων και να ενισχύουν την αξιοπιστία των δικτυακών συστημάτων. Παράλληλα, οι γνώσεις που απορρέουν από την ανάλυση συμβάλλουν στη διαμόρφωση στρατηγικών βελτίωσης της απόδοσης συστημάτων και στην ανάπτυξη πολιτικών προγραμματισμού με αντοχή σε σφάλματα σε συμπλέγματα GPU και κυψελοειδή δίκτυα.

Τα ευρήματα αυτής της διπλωματικής εργασίας αναδεικνύουν τον κρίσιμο ρόλο των αξιόπιστων συνόλων δεδομένων στην προώθηση των αλγορίθμων δικτύωσης που βασίζονται στην τεχνητή νοημοσύνη. Μέσα από την αντιμετώπιση προκλήσεων συλλογής και προεπεξεργασίας δεδομένων και την επίδειξη της χρησιμότητάς τους σε πραγματικά σενάρια, η παρούσα εργασία συνεισφέρει στις βάσεις για μελλοντικές καινοτομίες στα ευφυή δικτυακά συστήματα.

Acknowledgements

First and foremost, i would like to express my deep gratitude to my supervisor, Prof. Thrasyvoulos Spyropoulos for his invaluable guidance throughout my thesis. I would also like to thank my thesis committee members, Prof. George Karystinos and Prof. Dionysios Christopoulos, for dedicating their valuable time and their helpful comments. Finally, i wish to thank all of my friends and family for their constant encouragement and support throughout this academic journey.

Contents

1	Introduction	1
1.1	Thesis Outline	2
1.2	Contributions	2
2	Theoretical Background	4
2.1	Cloud Computing	4
2.1.1	Introduction to Cloud Computing	4
2.1.2	Introduction to Cloud Computing Datasets	7
2.2	4G Networks	11
2.2.1	Introduction to 4G	11
2.2.2	Introduction to our 4G Dataset	12
2.3	Youtube	13
2.3.1	Introduction to Youtube	13
2.3.2	Introduction to our Youtube Dataset	14
2.4	Related Work	15
3	Time Series Analysis	17
3.1	Introduction to Time Series Analysis	17
3.1.1	Stationarity	17
3.1.2	Correlation	18
3.1.3	Cumulative Distribution Function	18
3.1.4	Autocorrelation Function Plot	19
3.1.5	Rolling Mean Analysis	20
3.1.6	Augmented Dickey-Fuller Test	21
3.2	Cloud Datasets	21
3.2.1	Helios Preprocessing	21
3.2.2	Helios Results	23
3.2.3	Philly Preprocessing	31
3.2.4	Philly Results	33
3.3	4G Dataset	40

3.3.1	Preprocessing of Dataset	40
3.3.2	4G Dataset Results	41
3.4	Youtube Dataset	45
3.4.1	Youtube Preprocessing	45
3.4.2	Youtube Dataset Results	45
4	Practical Applications	51
4.1	Predictive Models	51
4.1.1	ARIMA	52
4.1.2	Neural Networks	53
4.2	Prediction in 4G Dataset	53
4.2.1	Prediction using ARIMA	53
4.2.2	Prediction Using Neural Networks	55
4.2.3	Results	57
4.2.4	Key Takeaways	58
4.3	Parallel Prediction of Time Series	59
4.3.1	DeepCog	59
4.3.2	Experiments	60
4.3.3	Results	61
4.3.4	Key Takeaways	67
5	Conclusions and Future Work	68
5.1	Conclusions	68
5.2	Future Work	68

List of Figures

2.1	PaaS vs SaaS vs IaaS	6
3.1	Duration Series for Uranus Cluster	22
3.2	Queue Series for Uranus Cluster	23
3.3	CCDF Earth duration	23
3.4	CCDF Earth queue	24
3.5	CCDF Saturn VC duration	25
3.6	CCDF Saturn VC queue	25
3.7	Rolling Mean Analysis Helios	27
3.8	Rolling Mean Analysis Helios VC	28
3.9	Transformed Duration	29
3.10	Autocorrelation of Transformed Timeseries	29
3.11	Helios Correlation	31
3.12	Binned Duration of Philly Traces	32
3.13	Binned Queue of Philly Traces	32
3.14	CCDF Philly GPU cluster Duration	33
3.15	CCDF Philly GPU cluster Queue	34
3.16	CCDF Philly VC1 Duration	34
3.17	CCDF Philly VC1 Queue	35
3.18	CCDF Philly VC2 Duration	35
3.19	CCDF Philly VC2 Queue	36
3.20	Timeseries for VC1 of Philly	37
3.21	Timeseries for VC2 of Philly	37
3.22	Transformed Timeseries along with Rolling mean	38
3.23	Autocorrelation Function Plot Transformed Duration Philly	39
3.24	Philly Correlation	40
3.25	Throughput for Pedestrian and Static	41
3.26	Throughput for Car and Train	41
3.27	Transformed Throughput for Car and Pedestrian	42
3.28	Autocorrelation Function for Car and Pedestrian	43

3.29	Transformed Plots in Train Mobility Pattern	44
3.30	4G LTE Correlation	45
3.31	Views of Viral Video	46
3.32	Views of Average Video	46
3.33	Views of Low-View Video	46
3.34	Autocorrelation Function Plot High Mid Views	47
3.35	Transformed Timeseries for Views	48
3.36	Autocorrelation of Transformed Videos	48
3.37	Youtube Dataset Correlation	50
4.1	ARIMA Model (Initial Timeseries)	54
4.2	Results of LSTM	56
4.3	Results of GRU	56
4.4	Results of FCNN	56
4.5	Results of GBD	57
4.6	Results of Random Forrest	57
4.7	LSTM vs Deepcog GPU0	62
4.8	LSTM vs Deepcog GPU1	62
4.9	LSTM vs Deepcog GPU2	63
4.10	LSTM vs Deepcog GPU3	63
4.11	LSTM vs Deepcog GPU0	64
4.12	LSTM vs Deepcog GPU1	65
4.13	LSTM vs Deepcog CPU	65
4.14	LSTM vs Deepcog Memory	66

List of Tables

2.1	Properties of Helios Clusters	8
3.1	Results of ADF Test for Helios Virtual Clusters	30
3.2	Results of ADF Test for Philly	38
3.3	Results of ADF Test for 4G Dataset	43
3.4	Results of ADF Test for Youtube Dataset	47
4.1	Mean Evaluation Metrics for 20 Experiments	58
4.2	Comparison of Mean Errors between DeepCog and LSTM Models	63
4.3	Comparison of Experiment B Mean Errors DeepCog and LSTM Models . . .	66

Chapter 1

Introduction

The rapid proliferation of artificial intelligence (AI) and machine learning (ML) technologies has revolutionized various domains, from healthcare and finance to transportation and communications. In the realm of networking, AI-driven algorithms are increasingly pivotal in optimizing resource allocation, improving system performance, and enabling predictive insights. These advancements rely heavily on the availability of high-quality datasets, which form the foundation for training, testing, and evaluating AI models.

Networking environments, particularly those characterized by distributed systems, dynamic resource demands, and varying user behaviors, present unique challenges for AI integration. From cloud computing platforms to cellular networks, these systems generate vast amounts of data, encompassing metrics like resource utilization, throughput, latency, and user engagement. Analyzing such datasets is essential for designing intelligent algorithms that can adapt to changing conditions, predict future states, and enhance the overall efficiency and reliability of network operations.

The area of networking, both wireless and wired, has experienced increasing interest in data-driven solutions and AI-based algorithms. Recent research has demonstrated the potential of machine learning techniques in addressing complex networking challenges such as traffic prediction, anomaly detection, and resource optimization. For example, Jeon et al. [1] analyzed large-scale GPU clusters for deep neural network (DNN) training workloads, highlighting the growing relevance of data-driven methodologies. Similarly, Hu et al. [2] characterized deep learning workloads in GPU datacenters, showcasing the role of predictive models in improving efficiency. These studies underscore the transformative potential of AI in networking.

However, unlike traditional ML problems such as image classification, which benefit from widely used, large benchmark datasets, networking problems and AI-based solutions lack equivalent resources. The scarcity of large, standardized datasets in networking is a significant bottleneck for innovation. This is primarily due to several challenges. One major challenge is the limited deployment of data-driven schemes in real networks. Most opera-

tional networks still rely on traditional, rule-based approaches, leading to fewer data-driven implementations. Another challenge is the reluctance of major operators to share data. Concerns over privacy, security, and competitive advantage deter operators from releasing detailed network data. Last but not least, the cost and complexity of collecting this kind of datasets is very high, because gathering and labeling network data requires significant resources and expertise. These challenges highlight the urgent need for collaborative efforts to create and share high-quality datasets, enabling the development of AI-driven networking solutions.

The datasets analyzed in this work provide unique opportunities to drive advancements in various aspects of networking. The Philly Dataset enables research into resource allocation and scheduling in multi-tenant GPU clusters, with potential applications in improving datacenter efficiency. The Helios Dataset facilitates the study of workload behavior and resource utilization in cloud computing, supporting the development of energy-efficient scheduling algorithms. The 4G LTE Dataset offers insights into cellular network performance under different mobility conditions, aiding in adaptive resource management and mobility-aware network optimizations. Last but not least DeepCog demonstrates the potential of deep learning models in real-time workload prediction and dynamic scheduling, contributing to cognitive network management.

1.1 Thesis Outline

The structure of this thesis is as follows: Chapter 2 provides a theoretical background, introducing the core concepts of cloud computing, 4G networks, and YouTube data. Chapter 3 delves into time-series analysis, highlighting techniques for preprocessing and exploring key statistical patterns within datasets. Chapter 4 focuses on the practical applications of these datasets, including the development of predictive models and the evaluation of their performance. Finally, Chapter 5 concludes with a discussion of the findings and potential directions for future research.

1.2 Contributions

Our contributions in this thesis are as follows. First we collected three publicly available the Philly dataset, that was used in this paper [1], the Helios traces, explained in this paper [2] and a 4G LTE dataset explained in [3]. After that a detailed examination of these datasets, we highlighted their unique characteristics, challenges, and potential applications. Additionally, the processed and the raw datasets are both available in a Google Drive repository for ease of access and reproducibility. By having this repository available these datasets can be used for future research and work.

Following that, application of advanced time series analysis methods to uncover statistical properties, such as stationarity and autocorrelation, in network performance metrics. Last but not least, we implemented various predictive models from simple statistical models like ARIMA [19] to more advanced machine learning techniques like neural networks [18]. Our aim was to predict key performance metrics and optimize resource allocation, to understand better how modern loads work, in order to be able to design better scheduling algorithms in the future. We also implemented a seminar predictive model explained in this paper [13], by using Convolutional Neural Networks instead of sequential NNs. This implementations is used to identify, if the existence of correlation between resource allocation metrics can aid us to make more accurate prediction for future loads.

By addressing the challenges of data scarcity and demonstrating the value of AI-driven insights, this work aims to lay the foundation for future innovations in intelligent networking systems. In summary, this thesis seeks to bridge the gap between data collection and AI-driven decision-making in networking, emphasizing the critical role of datasets in driving innovation and improving system performance.

Chapter 2

Theoretical Background

In this chapter we discuss the theoretical background required for this thesis.

2.1 Cloud Computing

Cloud computing can be defined as the delivery of on-demand computing services over the internet (cloud) on a pay as you go basis. Users can access and use computing resources (storage, processing power, and software applications) whenever they need them, while paying only what they use. This means that the user does not need to own and maintain the hardware. This model offers a variety of advantages such as flexibility, scalability, and cost efficiency, making it a popular choice for businesses and individuals alike. Without cloud computing, organizations would need to invest in and maintain their own on-premises servers, which involves several significant responsibilities and costs. This means that cloud computing simplifies IT management for businesses by focusing only on their core activities and leaving the security and maintenance issues to the cloud providers.

2.1.1 Introduction to Cloud Computing

Cloud computing offers distinct advantages that set it apart from traditional computing models. One primary characteristic of cloud computing is the on-demand self-service, where users can access cloud services and computing resources, whenever they need them (on-demand), without requiring direct interaction with the service provider (self-service). This capability offers significant flexibility and convenience. Another important characteristic is the broad network access. This means that cloud services are accessible over the internet through a variety of devices. This ensures that the costumers can engage with the cloud resources remotely, enhancing mobility and comfort.

Another fundamental feature of cloud computing is resource pooling. Cloud providers

create a shared infrastructure, where resources such as processing power, storage and memory are pooled together and are allocated on-demand. This means that multiple customers use the same physical hardware but their applications and data remain isolated and secure. Security and isolation is achieved through virtualization and other isolation techniques (such as Containers). Moreover, cloud computing is widely known for the rapid elasticity of its resources. This means that resources can be rapidly scaled up or down according to the demand. Lastly, cloud computing operates on a measured service basis, where resource usage is monitored, controlled, and reported, allowing users to pay only for the resources they consume. This usage-based pricing model promotes cost efficiency and allows organizations to manage their expenses effectively.

In cloud computing there are 3 different service models IaaS, PaaS and SaaS. Infrastructure as a Service (IaaS) provides customers with computing resources over the internet. Specifically, this model allows its users to manage their own IT infrastructure, as they have access to resources such as Virtual Machines, storage and network resources. One major benefit of this IaaS is that it allows users to adjust resources up or down to handle fluctuating workloads efficiently. By eliminating the need for upfront hardware investment, IaaS also enhances cost efficiency, as organizations pay only for resources that they use. In this model, the cloud provider's only responsibility is maintaining the physical equipment, while users manage the operating system and any applications they might install and run.

Platform as a Service (PaaS) is basically a platform where developers can manage and run their applications without worrying about the underlying infrastructure. PaaS is designed to support developers in creating, deploying, and managing applications without needing to worry about servers, storage or the operating system. In this model the cloud provider offers a variety of tools, libraries, databases and development frameworks for cloud application development. The Software as a Service (SaaS) model delivers software applications hosted on the cloud, typically available on a subscription basis. Like in the Platform as a Service (PaaS) model, where users do not manage the underlying infrastructure, SaaS goes further by relieving users of all responsibilities related to updates, maintenance, and security patches. All of these aspects are managed entirely by the provider, ensuring that applications remain up-to-date and secure without any action required by the user. SaaS applications are highly accessible, as they can be used from any internet-connected device, making this model particularly advantageous for remote and mobile work environments.

The three service models are closely interconnected, with each model tailored to support specific user groups and purposes. The SaaS model primarily targets end users, including both individual consumers and businesses. PaaS focuses on engaging with software developers and software companies, while IaaS is designed for IT professionals who manage and maintain hardware resources.

Cloud computing also offers different deployment models. The most commonly used are the Public, Private and Hybrid Cloud. The Public Cloud model allows broad access to cloud services and resources over the internet, enabling anyone to use these systems as needed.

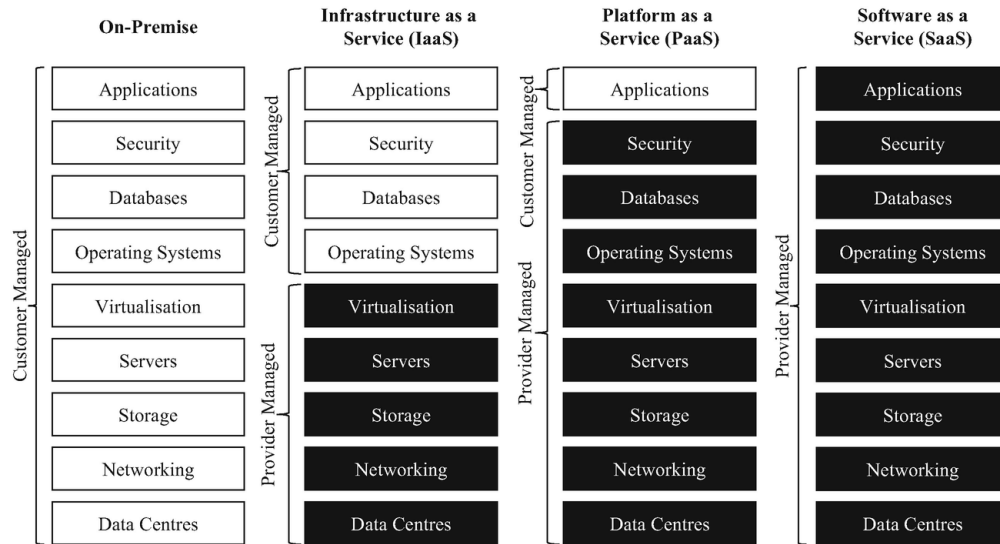


Figure 2.1: PaaS vs SaaS vs IaaS

Provided by third-party vendors, public clouds are managed and maintained entirely by the provider, relieving users of most infrastructure responsibilities. This model is highly scalable, offers reduced infrastructure costs, and requires minimal maintenance, making it an ideal choice for organizations seeking a cost-effective solution without the need for extensive IT management. However, because resources in a public cloud are shared across multiple users, security and privacy can be a concern, particularly for organizations handling sensitive data or regulated information.

In a Private Cloud environment, resources are dedicated to a single organization, providing a more secure and controlled infrastructure than the shared model of a public cloud. A private cloud can be hosted on-premises or through a third-party provider, with no shared hardware, which enhances data security and privacy. Only authorized users within the organization can access the private cloud's services, ensuring a high level of control over sensitive information. Private clouds offer the advantage of fully customizable infrastructure, allowing organizations to tailor resources to specific needs and compliance requirements. However, they typically come with higher costs and maintenance demands and offer limited scalability compared to public cloud options.

The Hybrid Cloud model merges aspects of both private and public cloud environments, allowing organizations to capitalize on the scalability and flexibility of public clouds while maintaining control over sensitive data within a private cloud. This arrangement is particularly advantageous for organizations with dynamic or seasonal workloads, as they can scale resources up or down in the public cloud as needed, all while keeping critical operations and confidential data securely hosted in the private cloud. However, one significant drawback

of the hybrid cloud model is its complexity. Coordinating and managing both private and public infrastructures can be challenging, often requiring advanced IT expertise and robust management tools to ensure seamless integration, consistent operation, and security across both platforms.

In recent years, advancements in Machine Learning (ML) and Artificial Intelligence (AI) have led to significant improvements in cloud computing. Graphics Processing Units (GPUs) were originally designed for rendering graphics in video games, but nowadays have become a critical component in modern GPU data centers. Due to their architecture GPUs are capable on running multiple tasks simultaneously and efficiently. This makes GPUs highly suited for parallel processing tasks, such as handling large datasets and training Deep Neural Networks (DNNs).

GPU datacenters are a specialized type of cloud computing infrastructure. Unlike traditional CPU-based datacenters, GPU datacenters are optimized to handle thousands of tasks simultaneously. They also provide cost-effectiveness by reducing the time needed to complete tasks, thereby lowering operational expenses. Because GPUs process large datasets more efficiently than CPUs, fewer resources are required to achieve the same outcomes, making GPU datacenters a financially viable option for many organizations.

2.1.2 Introduction to Cloud Computing Datasets

In our case the first 2 datasets that we used were collected from a GPU data center and contained training workloads from Deep Neural Networks (DNN). GPUs are expensive resources and that is why understanding and optimizing the use of resources is essential. The initial phase of our study involved the collection of datasets essential for the subsequent analysis. The process of acquiring high-quality, specialized datasets can be challenging due to limited options among available open-source resources. The first two datasets, that were collected are available on GitHub, fall into the field of cloud computing. The names of these datasets are Helios Trace and Philly Traces.

Helios Trace

The Helios Traces dataset, accessible through the HeliosData GitHub Repository is a comprehensive collection of job traces from a large-scale GPU datacenter operated by SenseTime, a leading AI company. This dataset is designed to support research into the behavior and optimization of deep learning workloads within high-performance GPU clusters, offering a unique resource for understanding workload patterns and enhancing datacenter management strategies.

The Helios dataset includes over 3.36 million job traces collected across four independent GPU clusters, named Venus, Earth, Saturn, and Uranus, over a six-month period. These clusters, which together contain over 6,400 GPUs and span hundreds of compute nodes,

support a wide array of deep learning tasks essential for AI applications. These tasks range from model training and inference to data preprocessing, covering diverse fields such as computer vision and natural language processing. The workload heterogeneity within this dataset reflects both production and research activities, giving researchers valuable insights into the operational dynamics of a real-world GPU datacenter.

Each cluster in the Helios dataset is organized into multiple virtual clusters (VCs) to support multi-tenancy, with each VC dedicated to specific user groups and configured for isolated resource management. This setup allows for detailed analysis of resource utilization patterns, queue times, and job completion rates within different operational contexts. The traces contain granular information about each job, such as execution time, CPU and GPU usage, job status (completed, canceled, failed), and queue delay. This level of detail enables researchers to examine not only the performance of individual jobs but also the efficiency of the overall system, identifying potential bottlenecks and resource allocation inefficiencies. The properties of each Helios cluster are summarized in Table 2.1, providing an overview of the cluster specifications and configurations.

	Earth	Venus	Saturn	Uranus	Total
# of VCs	25	27	28	25	105
# of Nodes	143	133	262	264	802
# of GPUs	1144	1064	2096	2112	6416
# of Jobs	873k	247k	1753k	490k	3363k

Table 2.1: Properties of Helios Clusters

The initial datasets contain information about the scheduler’s logs and the resources demand of the job. The dataset consists of the following columns:

- **job_id**: Unique identifier for each job.
- **user**: user that submitted the job.
- **vc**: the virtual cluster that the was submitted in.
- **state**: state is the status of the job upon termination. Job can end in 5 statuses (COMPLETED, CANCELED, FAILED, TIMEOUT and NODE_FAIL)
- **gpu_num**: Number of GPUs allocated for the job.
- **cpu_num**: Number of CPUs allocated for the job.
- **node_num**: Number of nodes allocated for the job.
- **submit_time**: Time when the job was submitted.

- **duration:** Duration of the job execution in seconds.
- **queue:** Time spent in the queue before job execution.

The Helios dataset is particularly useful for exploring job scheduling strategies, resource allocation methods, and energy-saving techniques in GPU datacenters. For instance, the dataset includes metrics that allow for the analysis of tail distributions of queue times, which can provide insights into peak load conditions and the effectiveness of various scheduling approaches. Additionally, the dataset’s comprehensive nature makes it suitable for developing and testing predictive models that could optimize job scheduling and reduce energy consumption by dynamically adjusting resource allocation based on predicted demand. With its depth and diversity, the Helios Traces dataset offers a valuable resource for advancing the state of knowledge in GPU datacenter management, helping to pave the way for more efficient, reliable, and scalable infrastructure for deep learning.

Philly Traces

The Philly Traces dataset, available at Philly Traces GitHub Repository, is a large-scale, multi-tenant dataset from a GPU cluster managed by Microsoft Research, specifically designed for training deep neural networks (DNNs). This dataset captures a rich array of job traces collected over a two-month period, encompassing approximately 96,000 jobs run across a cluster equipped with thousands of GPUs. The data is intended to support research into the optimization and efficient management of DNN training on shared GPU resources, providing real-world insights into scheduling, resource utilization, and system behavior under different workloads.

The Philly cluster serves a diverse range of users, including multiple research and production groups, each with specific resource quotas and computational needs. The dataset includes detailed information about each job, such as job type, requested resources (e.g., number of GPUs), queuing and execution times, and status upon completion (e.g., success, failure, or killed by user). In addition, the Philly dataset provides logs from the Apache YARN scheduler and Ganglia monitoring system, which track CPU and GPU usage, memory, network throughput, and overall system load in one-minute intervals, enabling precise, minute-by-minute analysis of hardware utilization and job runtime efficiency across various job configurations.

The Philly dataset contains two primary file types: scheduler logs and hardware performance counters. These files provide insights into job scheduling, resource utilization, and hardware performance, with each type capturing the following eight key variables:

- **vc:** Represents the hashed identifier of the virtual cluster where the job was executed.
- **machine_id:** Denotes the unique identifier of the machine that processed the job (referred to as "server" in some parts of the paper).

- **submitted_time**: The timestamp marking when a computational job was initially submitted for execution.
- **start_time**: Records the actual start time of the job, given in datetime format.
- **end_time**: The completion time of the job, also noted in datetime format.
- **status**: Indicates the final status of each job, which may be one of three possible outcomes: Pass, Killed, or Failed.
- **gpu_utils**: Contains per-minute GPU utilization data for each machine, as monitored by the nvidia-smi tool. Machines can have between 1 and 8 GPUs.
- **cpu_util**: Contains per-minute utilization metrics for the server’s CPU.
- **mem_util**: Includes per-minute memory usage statistics for each server.

Having the `submitted_time`, `start_time`, and `end_time` of each job allows us to derive essential time metrics associated with job processing. One such metric is the queuing delay, defined as the time a job spends in the queue before execution begins, measured in seconds. The queuing delay is calculated by subtracting the `submitted_time` from the `start_time`, as shown below:

$$queue = start_time - submitted_time$$

Another critical time metric is the execution time of each job, referred to as run time. This is the duration of the job’s execution from start to completion, measured in seconds. Specifically run time is the time that the job needs to finish (do not contain the time that the job spent in queue) and can be calculated as follows:

$$duration = end_time - start_time$$

A unique aspect of the Philly Traces dataset is its focus on the challenges associated with managing DNN workloads in a multi-tenant environment. For instance, the data highlights how gang scheduling (scheduling jobs in groups that must start simultaneously) and locality constraints (the need to place certain jobs close to each other on the network) affect resource fragmentation, queuing times, and GPU utilization. Locality constraints are particularly relevant for DNN training, as they help reduce synchronization delays by ensuring that GPUs in the same job are connected via high-bandwidth links. However, strict locality requirements can lead to extended queuing times, especially for large jobs that require multiple GPUs.

The dataset also allows for a detailed examination of job failures, which are significant in the context of GPU clusters due to their high cost and resource intensity. About 30% of the jobs in the Philly dataset either fail or are terminated prematurely, often due to issues like incorrect configurations, memory limitations, or data format inconsistencies. The dataset

includes information on failure frequencies, root causes, and the time-to-failure for different job types, providing valuable data for developing predictive failure models and designing error-resilient scheduling policies.

By providing a granular view of job scheduling, resource allocation, and failure modes in a production-grade GPU cluster, the Philly Traces dataset is a valuable tool for researchers and system designers working to improve the scalability, efficiency, and reliability of GPU-based deep learning infrastructures. This dataset facilitates the exploration of real-world scheduling challenges and provides the foundational data necessary to inform future innovations in distributed DNN training environments.

For GPU datacenters and cloud computing environments (like the Philly dataset) we want to have a better understanding of multi CPU/GPU workloads. By doing an extensive analysis of those workloads, our aim is to understand how these workloads work so that we can design better scheduling algorithms, which will minimize the queue and duration times of jobs, optimize resource utilization and reduce system fragmentation. This dataset is particularly valuable for exploring trade-offs between resource locality and job efficiency, paving the way for innovations in GPU cluster management and distributed DNN training systems.

2.2 4G Networks

2.2.1 Introduction to 4G

Since the advent of cellular networks in the late 1970s, the use of mobile networks has increased rapidly. The development of mobile networks has been very rapid extending the initial role of voice communication. Today mobile devices have become an essential and inextricable part of our every day life. Communication tasks evolved into browsing the Web, watching videos, and performing office work. In the last ten years, the conversion of the mobile phone to mobile broadband has accelerated rapidly, with mobile data traffic surging eighteen folds in just five years. Over the past years, mobile data usage has seen explosive growth.

The success of 4G paved the way for the development of 5G, the next generation of mobile networks. While 4G remains widely used, the demand for even higher speeds, lower latency, and support for massive Internet of Things (IoT) and ultra-reliable applications led to the introduction of 5G. Many aspects of 4G technology, particularly LTE-Advanced, provide a foundation for 5G development, allowing for a gradual transition as carriers upgrade their networks. As 5G is a relatively new technology, with its initial launch in Greece in 2020, there is a limited availability of high-quality datasets for comprehensive analysis. Therefore, we have chosen to focus on 4G networks in this thesis, where data is more readily accessible and mature, allowing for a more thorough and reliable study.

2.2.2 Introduction to our 4G Dataset

The 4G LTE dataset, accessible on Kaggle, is a robust dataset containing 4G LTE traces with both channel and context metrics, created to support in-depth studies of cellular network performance under various mobility conditions. Collected from two major Irish mobile operators, this dataset spans different mobility scenarios, including static, pedestrian, car, bus, and train environments, allowing researchers to analyze cellular network behavior across both urban and rural settings.

The dataset consists of 135 traces, with each trace lasting approximately fifteen minutes, and throughput values captured at one-second intervals. Using the G-NetTrack Pro mobile application, a non-rooted Android-based network monitoring tool, the dataset records a wide array of key performance indicators (KPIs) and context metrics. Key KPIs include:

- **RSRQ (Reference Signal Received Quality)**: Reflects the quality of the received signal by capturing interference levels across all resource elements.
- **RSRP (Reference Signal Received Power)**: Measures signal strength over specific resource symbols, essential for assessing coverage.
- **RSSI (Received Signal Strength Indicator)**: Represents the total received power, including interference, and is used to determine signal reliability.
- **SNR (Signal-to-Noise Ratio)**: Quantifies signal quality in relation to background noise, which is critical for high data rate transmission.
- **CQI (Channel Quality Indicator)**: Provides feedback from the user equipment (UE) to the eNodeB, helping in the selection of appropriate modulation and coding schemes based on channel conditions.

Alongside these KPIs, the dataset records GPS coordinates, velocity, cell ID, uplink and downlink throughput, and handover events, providing a detailed spatial and temporal context for each data point. These columns are explained in the following

- **GPS Coordinates**: Records the geographical position of the user equipment, providing spatial context for each data point (Latitude and Longitude)
- **Velocity**: Measures the speed of the user equipment, which is important for analyzing mobility patterns and their effect on network performance.
- **Cell ID**: Identifies the serving cell of the user equipment, aiding in tracking connectivity and handover events.
- **Uplink and Downlink Throughput**: Captures the data transmission rates in both uplink and downlink directions, essential for understanding data flow and network load.

- **Handover Events:** Logs instances where the user equipment switches from one cell to another, providing insights into network reliability and the management of continuous connectivity.

This combination of channel and context metrics enables a multi-dimensional analysis of the factors impacting network quality, such as interference, signal strength variations, and the effects of user mobility on throughput and latency.

While the LTE dataset provides a substantial amount of information, there are some limitations to consider. The dataset’s real-world component is constrained by a one-second sampling interval due to the limitations of G-NetTrack Pro and Android APIs. Furthermore, not all KPIs are available for every sample, and some cell locations rely on the OpenCellID database, which may have missing entries for certain areas. These limitations can introduce slight gaps in data continuity but do not significantly affect the dataset’s suitability for high-level research and exploratory analysis.

The LTE dataset provides invaluable insights into how cellular networks handle mobility and varying signal conditions. By analyzing network performance across different movement scenarios, researchers can develop adaptive resource management techniques, improve handover efficiency, and optimize connectivity strategies in future wireless networks. Furthermore, the dataset serves as a foundation for building AI-driven predictive models for proactive network management, helping operators enhance service reliability and user experience.

2.3 Youtube

2.3.1 Introduction to Youtube

YouTube is one of the largest online video-sharing platforms globally, launched in 2005 and now owned by Google. It serves as a central hub for content creators, viewers, and businesses, offering an extensive range of videos, from entertainment and education to product marketing and personal vlogs. As of 2024, YouTube has over 2 billion monthly active users, uploading more than 500 hours of video content every minute.

The platform’s widespread popularity and massive volume of user interactions make it a valuable source for data analysis. Metrics such as views, likes, comments, and dislikes provide insights into user engagement and content trends. These metrics can be leveraged for applications such as content recommendation systems, targeted advertising, and network resource optimization.

The study of YouTube-related data, particularly time-series metrics like view counts, plays a crucial role in understanding user behavior patterns and content dynamics, which are pivotal for developing AI-driven networking algorithms that can optimize content delivery and user experience.

2.3.2 Introduction to our Youtube Dataset

The fourth and final dataset was retrieved by the Youtube API. This Youtube dataset contains a per hour count (views, likes, dislikes, comments and favorites) of videos that were uploaded in April of 2018 on the platform of Youtube. There are count observations about 1611 different videos, which means that in the trace there are approximately 1.1 millions entries. This dataset contains 2 different excel files. The first file is called `count_observation_upload` and it mainly contains per hour statistics about every video.

- **Time:** Time that the statistics were saved
- **videoId:** The ID of the video
- **commentCount:** Number of comments of the video at the time that the data was collected.
- **dislikeCount:** Number of dislikes of the video at the time that the data was collected
- **viewCount:** Number of views of the video at the time that the data was collected
- **likeCount:** Number of likes of the video at the time that the data was collected

The second file is called `video_characteristics_upload` which is a very interesting dataset that contains information such as the channel that uploaded the video, tags and the category of the video. These data can be used to uncover trends

- **videoId:** ID of the video
- **title:** title of the video
- **categoryId:** The ID of the category that the video belongs to
- **channelId:** The ID of the channel that uploaded the video
- **channelTitle:** The name of the channel that uploaded the video
- **duration:** The length of the video
- **publishedAt:** The date that the video was uploaded in the platform
- **tags:** a list of tags associated with the video

This Youtube dataset provides valuable insights about user interactions and metrics in widely famous platform like Youtube. By analyzing these metrics we can uncover various trends and patterns about user interaction with the platform. We can also create various predictive models like trying to predict the number of views by the end of a time period.

2.4 Related Work

A significant body of research has focused on exploring either traditional optimization methods or machine learning approaches for the deployment GPU clusters or optimizing the performance of both wired and wireless networks. The analysis of multi-tenant GPU clusters for DNN training workloads was comprehensively explored in [1]. The authors presented an in-depth study on the behavior and optimization of deep learning jobs in large-scale GPU datacenters. Their work focused on characterizing the unique challenges of heterogeneous workloads and providing insights into resource scheduling strategies.

In [2], the authors examined the characterization and prediction of deep learning workloads in large-scale GPU datacenters. By leveraging ML techniques, they modeled workload demands and predicted resource utilization trends. This work introduced statistical methods for better understanding GPU resource allocation. However, unlike our framework, it did not integrate reinforcement learning (RL) to adaptively respond to workload variability in real time.

The 4G LTE dataset with channel and context metrics was introduced in [3]. This dataset provides detailed insights into cellular network performance across diverse mobility scenarios, including static, pedestrian, and vehicular settings. It enabled the analysis of key performance indicators like throughput, signal quality, and handover events, serving as a cornerstone for network optimization studies. While [3] primarily focused on descriptive analytics, our framework extends its application by incorporating predictive modeling and decision-making strategies, which might improve network slice performance.

In those datasets various time series analysis techniques were implemented to better understand the nature of the datasets. Time series analysis has become a very common procedure to characterize the stability of the statistical properties of time series. In [20], [21] the procedures to characterize whether a time series is stationary or not are explained.

The Autoregressive Integrated Moving Average (ARIMA) model is a widely used statistical analysis technique for time series forecasting. In [19] the researchers used ARIMA to predict various energy consumption time series, which might contain both linear and non-linear relationships as in our case. By analyzing historical network data, ARIMA models can forecast future network behavior, enabling proactive management and optimization. In our case, ARIMA has been applied to predict network throughput, assisting in congestion avoidance and efficient resource allocation.

However, while ARIMA models are effective for linear time series data, they may face limitations when dealing with non-linear patterns commonly present in network traffic and this is the reason why we used Deep Neural Networks to predict the network's throughput. In [18] a distributed deep neural networks (DDNNs) has been proposed to operate over distributed computing hierarchies, including the cloud, edge, and end devices, allowing for localized inference and improved system fault tolerance

A groundbreaking work of AI for network management, particularly in environments

implementing network slicing is, DeepCog: Cognitive Network Management in Sliced 5G Networks with Deep Learning, was proposed in [13]. The original work on DeepCog demonstrated how deep learning architectures could address complex challenges at multiple levels of mobile network operation, including scheduling slice traffic at the RAN, resource allocation in the network core, and admission control for new slices. Their findings illustrated the significant potential gains achievable through integrating AI into network slicing, highlighting the transformative impact of such technologies on next-generation zero-touch mobile networks. Building on this foundation, this thesis extends the application of DeepCog to cloud computing environments. Specifically, we applied DeepCog to analyze and predict high-correlation workloads in the Philly dataset, focusing on GPU machine-level data. By leveraging DeepCog’s predictive capabilities, we explored resource utilization patterns and optimized workload scheduling in multi-tenant GPU clusters. This work demonstrates the versatility of DeepCog beyond mobile networks and underscores its utility in addressing challenges in large-scale cloud computing systems.

Chapter 3

Time Series Analysis

3.1 Introduction to Time Series Analysis

In this thesis, we studied time series extensively. The first step in this process is to understand what a time series is. The definition of a time series is as follows:

Definition 1 A ***time series*** is a sequence of observations x_t collected in chronological order over time t

Time series can also be categorized as either continuous or discrete. In continuous time series, observations are recorded continuously over time, while in discrete time series, measurements are taken at specific time intervals. Any variable that changes over time can be considered a time series. Certain forecasting methods rely on historical values of a time series to make predictions, aiming to uncover patterns and trends that enable accurate forecasting of future values. However, the success of these forecasting methods is not guaranteed, as each time series may possess unique characteristics, and some predictive models may perform better on certain types of data than others. Thus, it is essential to first analyze and understand the properties of a time series before selecting an appropriate forecasting approach.

3.1.1 Stationarity

A key property of a time series is its stationarity. In many predictive models, it is assumed that certain characteristics of the series, such as the mean or variance, remain consistent over time. There are two widely recognized types of stationarity strict stationarity and weak stationarity, each with its own definition and practical implications.

Definition 2 A time series is said to be ***strictly stationary*** if its statistical distribution is unchanged after any arbitrary shift in time

This definition, however, is quite stringent and restrictive, so it is more common to use the concept of weak stationarity.

Definition 3 *A time series is **weakly stationary** if it has a finite second moment, a constant mean, and a covariance that depends only on the distance between observations (lags), rather than on absolute time*

The second definition is preferred due to the strictness of strict stationarity, making weak stationarity more practical and widely applicable in time series analysis.

3.1.2 Correlation

Correlation analysis is a fundamental statistical technique used to identify the strength and direction of the linear relationship between two variables. This analysis provides valuable insights into how variables relate to one another, helping to reveal underlying patterns and dependencies. In predictive modeling, an essential step known as feature selection involves choosing the variables (or "features") that will enhance model performance. By performing correlation analysis on all features in a dataset, we can identify those that are highly correlated with the target variable or other key features, often leading to improved model accuracy and effectiveness.

In our analysis we used the Pearson Correlation Coefficient to calculate the linear relationship between variables. Pearson Correlation is a commonly tool used to calculate the linear relationship between variables. The results of this analysis range from -1 to 1. If a value is smaller than 0 and bigger than -1 then the 2 variables exhibit negative correlation ($-1 \leq r < 0$). This means that whenever the one variable tends to increase then the other variable will go in the opposite direction (will decrease). If a value is closer to -1 then the correlation between the 2 variables becomes stronger. If the result of the Correlation is equal to zero then the 2 variables have no linear relationship. The last case is when the result of the correlation is between 0 and 1 ($0 < r \leq 1$) then the 2 variables have positive correlation. This will mean that whenever the value of one variable changes then the other variable will tend to change in the same direction. Values closer to 1 will mean that the correlation becomes stronger.

3.1.3 Cumulative Distribution Function

In cloud computing datasets, the exploration of tail distributions for key statistical metrics (such as arrival time, queue time, and run time) is essential. Analyzing these tail distributions provides deeper insights into system behavior, especially during peak or extreme conditions. This approach not only enhances our understanding of the system but also aids in identifying critical patterns in tail behavior that can impact overall performance, resource allocation, and service quality.

The main reason the CDF is essential in our analysis of tail distributions is that it provides a comprehensive view of data across the entire range of values, including extreme values. By using the CDF, we can accurately determine the probability of rare but impactful events (such as high queue times), which are critical for understanding system behavior under peak conditions. These insights are invaluable for enhancing the reliability and efficiency of cloud environments, as they allow us to anticipate and mitigate potential performance issues during high-demand periods, ultimately leading to more robust and responsive systems.

Definition 4 *The **Cumulative Distribution Function (CDF)** of a random variable X can be mathematically be defined as the probability that that X will take a value less than or equal of a specific value x . The following equation explains the definition of CDF:*

$$F_X(x) = P(X \leq x), \quad x \in \mathbb{R}$$

In our analysis we used the Complementary Cumulative Distribution Function (CCDF) to explore the tail distributions.

Definition 5 *The **Complementary Cumulative Distribution Function (CCDF)** of a random variable X can be mathematically defined as the probability that X will take a value bigger than a specific value x . CCDF can also be calculated by subtracting the equation for CDF by one. The following equation explains the definition of CDF:*

$$\bar{F}_X(x) = P(X > x) = 1 - F_X(x), \quad x \in \mathbb{R}$$

Using the CCDF allows us to directly observe and quantify the likelihood of extreme values, which often lie in the tail of the distribution. Unlike the CDF, which accumulates probabilities from lower values, the CCDF highlights the higher values and makes it easier to see how frequently large, impactful events—such as prolonged queue times—occur. This approach is particularly useful in cloud computing, where understanding and mitigating the risks associated with long-tail events is essential for optimizing resource allocation, improving service quality, and ensuring system resilience.

3.1.4 Autocorrelation Function Plot

This subsection explains the purpose of the autocorrelation function and how it aids in analyzing the time dependencies within your cloud computing and networking datasets. The autocorrelation function (ACF) is a statistical tool that quantifies the degree of correlation between a time series and its lagged versions, providing insights into the persistence of patterns over time.

In our analysis, the ACF is valuable for examining how key metrics in cloud computing and networking datasets, such as queue time and run time, throughput, are influenced by

their past values. By identifying these temporal dependencies, the ACF helps us uncover underlying patterns and trends, enabling a better understanding of how the system behaves over time and under various conditions. This understanding is essential for effective resource management and optimizing performance in cloud environments.

Autocorrelation reveals the underlying structure of a dataset by highlighting trends, seasonality, or cyclic patterns. For instance, if queue times or run times exhibit high autocorrelation, it suggests the presence of persistent trends or recurring patterns in these metrics. Recognizing these patterns is critical for optimizing resource allocation, as it allows us to anticipate periods of high demand or system bottlenecks. By leveraging this information, cloud and network administrators can proactively adjust resources to maintain performance and enhance system efficiency.

In our analysis, the ACF plot is used to assess the stationarity of key metrics in cloud computing and networking datasets. By observing the ACF plot we identify if a timeseries is stationary or non-stationary. A stationary time series typically shows significant autocorrelations only at lower lags, with correlations rapidly decreasing to near zero for higher lags. A non-stationary series often exhibits high autocorrelation values that decay slowly over increasing lags, indicating the presence of long-term trends or non-constant variance over time.

The ACF plot at a given lag k measures the correlation between observations in a time series that are k time steps apart. The ACF at lag k , denoted as ρ_k can be mathematically defined as:

$$\rho_k = \frac{\sum_{t=1}^{N-k} (X_t - \mu)(X_{t+k} - \mu)}{\sum_{t=1}^N (X_t - \mu)^2}$$

where X_t is the value of the time series at time t , \bar{X} is the mean of the time series and n is the total number of observations

3.1.5 Rolling Mean Analysis

Rolling Mean Analysis is a technique in time series analysis that involves calculating the average value of the data within a specified window as it moves (or "rolls") across the time series. This rolling average allows for the observation of changes in the mean over time and helps detect any trends or patterns that may exist within the series. By examining how the mean changes (or remains stable) over different windows, analysts can gain insights into the stationarity of the time series, which is a key requirement for many time series models.

In a stationary time series, the rolling mean should remain relatively constant across time, as a stationary series exhibits no long-term trends or seasonal patterns. In contrast, a non-stationary series, which might contain trends, seasonality, or structural changes, often shows a rolling mean that fluctuates or drifts over time. This fluctuation in the rolling mean could indicate underlying trends that violate stationarity assumptions, which could affect model accuracy if not addressed through techniques like differencing or detrending.

The rolling mean is calculated by choosing a specific window size, such as a 100-point or 200-point window, and taking the mean of the values within that window as it moves sequentially along the time series. Larger windows tend to smooth out short-term fluctuations, revealing longer-term trends or patterns, while smaller windows capture more immediate changes but may be more sensitive to noise. The choice of window size is therefore critical and often depends on the nature of the data and the specific insights needed from the analysis.

In the context of this thesis, rolling mean analysis is particularly significant for large datasets, such as those in cloud computing, where there are hundreds of thousands of entries. With such high-frequency data, rolling mean analysis can help identify subtle shifts in mean behavior that may be masked in traditional analysis. For instance, a consistent change in the rolling mean over time might signal a gradual shift in resource usage patterns, latency, or other performance metrics critical to cloud environments. However, for smaller datasets, such as those with around 1,000 entries, the rolling mean analysis provides limited additional insight due to the lack of sufficient data points for capturing meaningful trends over long windows.

Overall, rolling mean analysis serves as a valuable tool for understanding time series stationarity and trends. When combined with other techniques like the Autocorrelation Function (ACF), it provides a robust foundation for identifying stationarity and preparing the data for further analysis or model development.

3.1.6 Augmented Dickey-Fuller Test

Augmented Dickey-Fuller Test is a statistical test that assesses the stationarity of a timeseries by checking for the presence of a unit root. The ADF Test is based on the Null Hypothesis, which supports that the presence of a unit root will automatically mean that the time series is non-stationary. On the other hand, the lack of a unit root will mean that the data are stationary.

The results of the ADF Test are a p-value, a Test Statistic and some critical values at all significance levels (1%, 5% and 10%). So for example if the Test statistic is smaller than the critical value at 1% and the p-value is smaller than a selected significance level (commonly 0.05) we can be with 99% certainty that our time series is stationary.

3.2 Cloud Datasets

3.2.1 Helios Preprocessing

Helios Traces consists of 4 different traces, one for every GPU cluster (Earth, Saturn, Uranus and Venus). The Helios dataset includes logs that span multiple virtual clusters,

which represent isolated workloads operating within the same physical cluster. These virtual clusters are crucial for understanding how resources are utilized and managed in cloud and distributed systems. To ensure meaningful analysis and to capture workload-specific trends, the dataset was divided into individual subsets, with each subset representing the activity of a specific virtual cluster.

The division was performed using the virtual cluster identifier (VC ID), which allowed for the extraction of data specific to each VC. There are 5 different timeseries available for each VC (number of GPUs, CPUs and Nodes and Duration and Queue time for each job). Each VC information was stored in a separate csv file.

The second part of the preprocessing was to create to adjust the granularity of the samples. Helios dataset include time-series data like duration and queue time of a job along with the time that the job was submitted. This means that the dataset lacks a standardized temporal granularity, with job entries recorded at irregular intervals. To enable consistent time-series analysis and comparison, it was necessary to preprocess the data by aggregating it into uniform time bins.

After experimenting with the time between each bins 20 minute intervals was selected as it gave us the most consistent time bins. This step is pivotal for ensuring the dataset aligns with the objectives of this thesis, enabling meaningful insights into workload behavior and resource utilization patterns. To account for the presence of extreme values, particularly in the queue time series, we applied a filtering process to remove outliers. A practical threshold was established, excluding durations exceeding 5000 minutes and queue times greater than 1000 minutes. Additionally, due to the extensive volume of data, we focused on the month of June to allow for more focused and meaningful observations. The x-axis in our visualizations was adjusted to represent the days of the month numerically, facilitating a clearer temporal analysis. Notably, in June, the 1st corresponds to a Monday, and the sequence progresses such that the 7th falls on a Sunday. The final time series for duration and queue time are shown in Figures 3.1 and 3.2 respectively in both normal scale and in log scale.

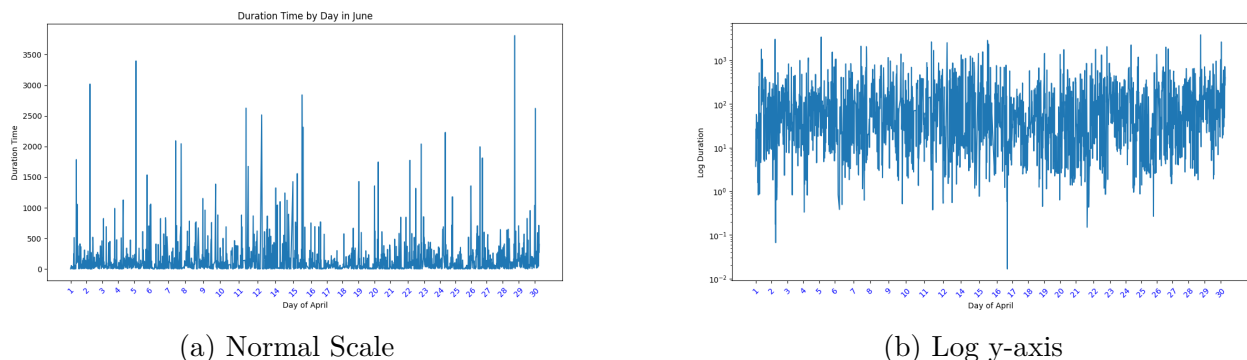


Figure 3.1: Duration Series for Uranus Cluster

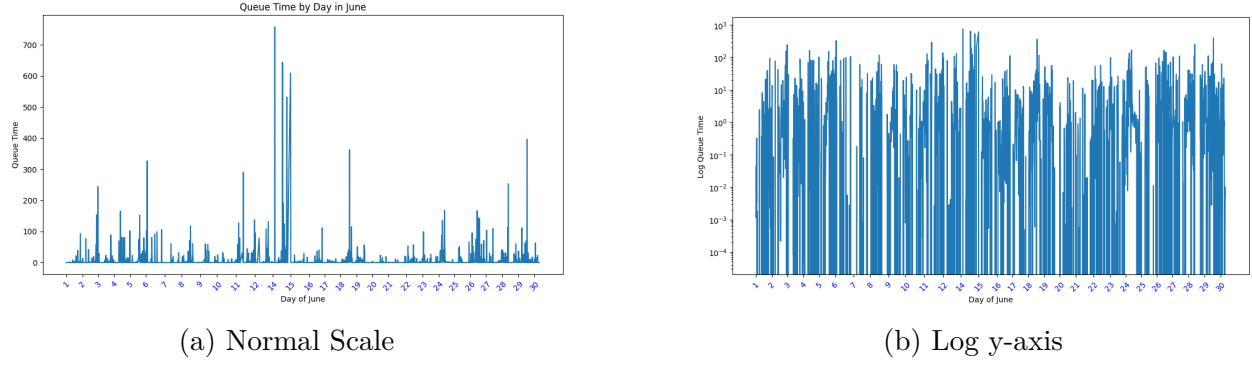


Figure 3.2: Queue Series for Uranus Cluster

3.2.2 Helios Results

CCDF Analysis for GPU clusters

At first the analysis was performed in the physical GPU clusters as a whole. We plotted the CCDF of our data along with the CCDF of some known distributions (like Pareto and Log-Normal, Gamma, k-Weibull). This step was implemented to compare our data with these distributions and see if our CCDF follow a light or heavy-tail distribution. In Figures 3.3 and 3.4 the CCDF of duration and queue is plotted for Earth cluster (every cluster has similar distribution) in Helios along with the CCDF of the known distributions.

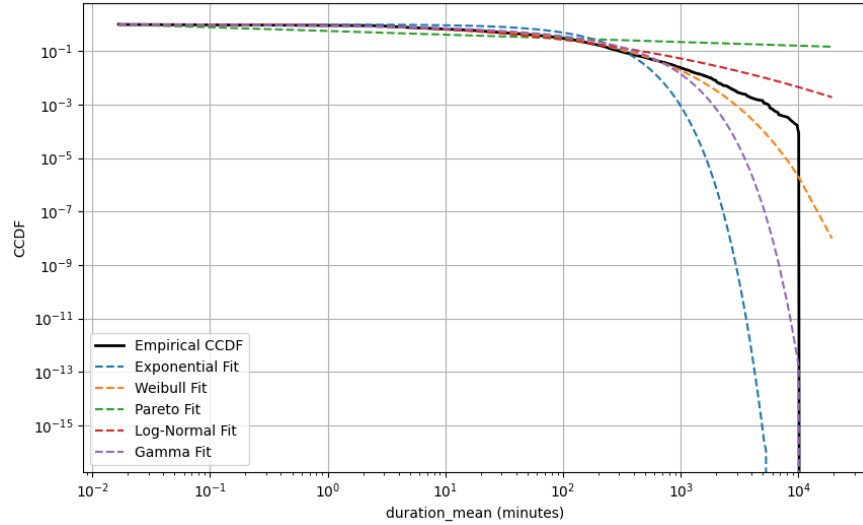


Figure 3.3: CCDF Earth duration

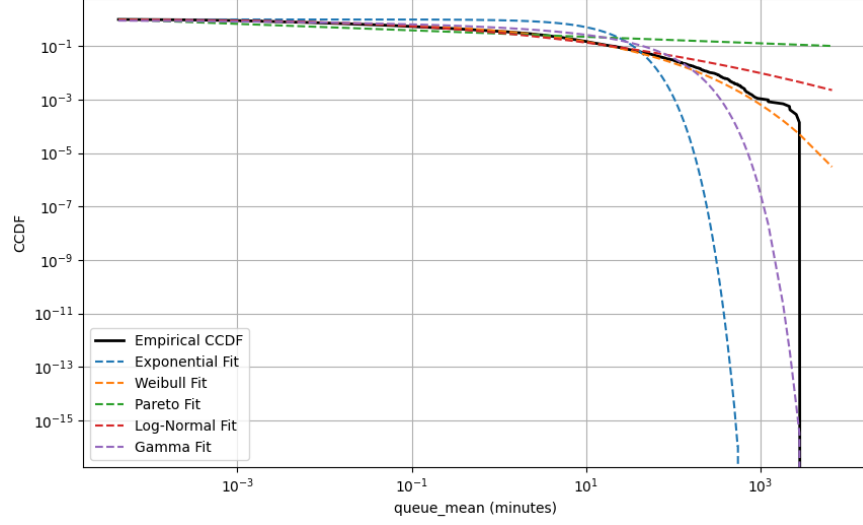


Figure 3.4: CCDF Earth queue

The analysis of the tail distributions for both duration and queue times reveals a characteristic heavy-tailed behavior. This is evident from the Complementary Cumulative Distribution Function (CCDF) plots, which display a gradual decay across larger values. Such behavior indicates that, within this dataset, extreme values—such as prolonged queue times or extended durations—are not merely outliers but occur with considerable frequency. This indicates that there are non-negligible probabilities of very high queue times, suggesting long-tail behavior, which is common in workload distributions. In our case, Weibull and Log-Normal fits are the closest to the empirical data at the tail. The Exponential and Gamma distributions decay too rapidly and fail to capture the heavy-tailed nature of the queue times.

The long tails observed in both queue time and duration indicate the presence of heavy-tailed behavior, a common characteristic in cloud and distributed systems. This suggests that infrequent but extreme events can disproportionately impact overall system performance. Additionally, a small subset of jobs or workloads tends to dominate resource consumption, highlighting significant variability in workload characteristics. This heavy-tailed nature underscores the importance of tailored resource management and scheduling strategies to address these extremes effectively.

CCDF Analysis for Virtual Clusters

In the same spirit of the previous subsection we opted to analyze the CCDF of some of the biggest Virtual Cluster for timeseries such as duration and queue. In Figures 3.5 and 3.6 we plotted the CCDF of the biggest Virtual Cluster for Saturn which one of the busiest

VC in the whole of Helios.

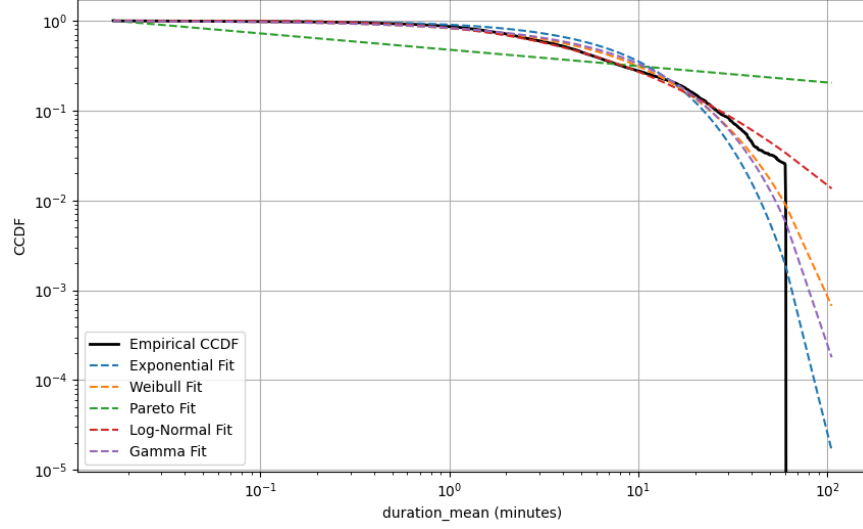


Figure 3.5: CCDF Saturn VC duration

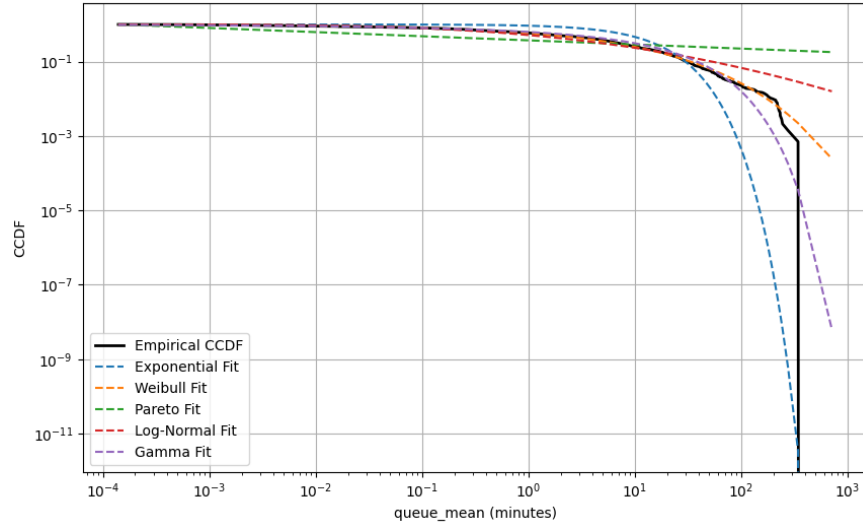


Figure 3.6: CCDF Saturn VC queue

The results of the CCDF analysis for the virtual cluster (VC) closely resemble those observed for the physical GPU cluster. As shown in the duration time series (Figure 3.5), the empirical data aligns well with the Log-Normal and k-Weibull distributions. This indicates the presence of a heavy-tailed distribution, as both distributions exhibit this characteristic.

Specifically, the k-Weibull distribution confirms heavy-tailed behavior due to the parameter k being less than one, and the Log-Normal distribution is inherently heavy-tailed.

For the queue time series (Figure 3.6), the empirical data is best described by the k-Weibull distribution, with the Gamma and Log-Normal distributions providing a close second. The ability of the k-Weibull distribution to capture the empirical tail behavior highlights its suitability for modeling queue times in this VC.

The heavy-tailed nature of the CCDF timeseries suggests that extreme values occur with non-negligible probability, leading to higher system variability. This means that jobs with high duration or high queue can happen, which mean that our scheduler might not do a very good job. This means that adaptive mechanisms need to be designed to prevent those events from happening.

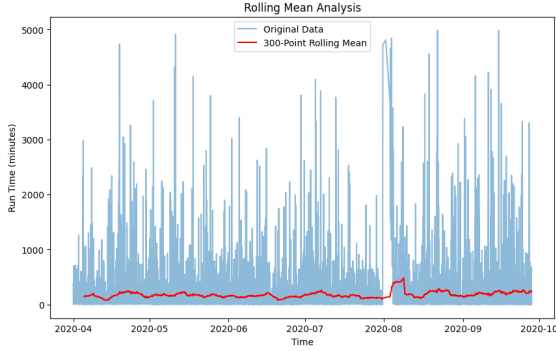
Stationarity Analysis in GPU clusters

In time series analysis, stationarity is an essential characteristic, indicating that the statistical properties of the series—such as mean, variance, and autocorrelation—remain stable over time, as discussed in Section [3.1.1]. To determine whether a time series is stationary, we applied the Augmented Dickey-Fuller (ADF) test, detailed in Section [3.1.6] of the Theoretical Background for Time Series analysis. Additionally, we used the autocorrelation function (ACF) plot to examine the temporal dependencies within the data.

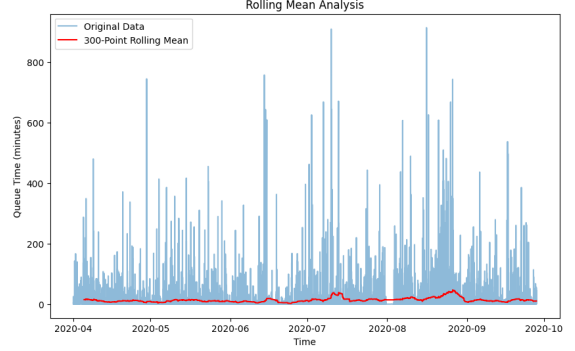
The first step in determining the stationarity of a timeseries is a visual inspection of the data. In Figures 3.1 and 3.2 the 2 time series analyzed are plotted (duration and queue). One very important characteristic of stationary timeseries is the constant mean over time. To assess the stationarity of a time series, we can utilize rolling mean analysis, which provides insights into whether the mean value of the series remains stable over time. This analysis involves calculating a moving average (rolling mean) over a specified window size, which helps identify trends or patterns that might suggest non-stationarity if they are present.

It is important to note that rolling mean analysis is particularly significant for the cloud computing datasets, as they contain hundreds of thousands of entries. This high volume of data allows the rolling mean to capture subtle trends and fluctuations more effectively, making it a valuable tool for assessing stationarity and detecting periods of volatility. With smaller datasets, the rolling mean analysis has limited impact, as shorter time series are less likely to reveal meaningful rolling trends or patterns.

In the following figures, the original data for Queue Time and Run Time from Helios is shown in light blue, along with their 100-point rolling means plotted in red. In the Queue Time analysis (Figure 3.7b), despite a relatively stable rolling mean, there are significant fluctuations in the raw data, with sharp peaks indicating bursts of activity.



(a) Duration of Jobs



(b) Queue of Jobs

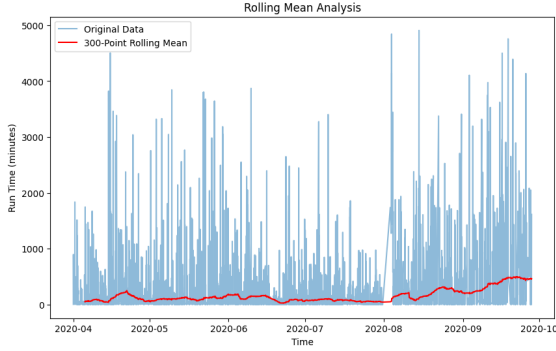
Figure 3.7: Rolling Mean Analysis Helios

These variations suggest that while the mean may appear constant, the volatility is not, indicating potential conditional heteroscedasticity. Similarly, in the Run Time analysis (Figure 3.7a), the rolling mean remains relatively stable, but the data exhibits large spikes. This pattern reinforces the idea of a non-constant variance over time, even though the mean does not exhibit a clear upward or downward trend.

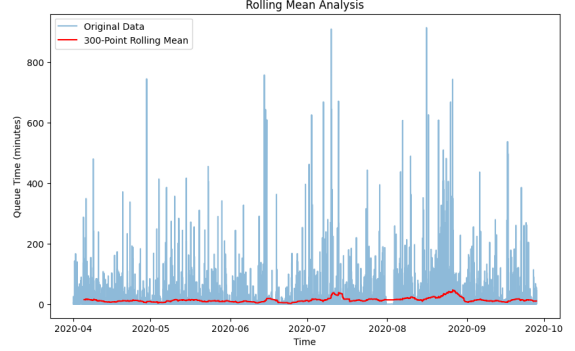
To conclude, for the physical GPU clusters, it is evident that the jobs exhibit a wide range of workload intensities, leading to varying levels of "heaviness" in job durations. This variability indicates that the data within the same GPU cluster is non-stationary, and further analysis to confirm stationarity is not required for the duration time series.

Stationarity Analysis in Virtual Clusters

Additionally, since each virtual cluster (VC) manages its own queue independently, it is reasonable to observe that the aggregated time series for the entire cluster is also non-stationary. This reflects the inherent differences in workload behaviors and queue management across individual VCs, contributing to the non-stationarity at the cluster level. We performed the same analysis but in the per VC division. The first step in our analysis is the visual inspection along with the rolling mean of 100 window for duration and queue time series.



(a) Duration of Jobs



(b) Queue of Jobs

Figure 3.8: Rolling Mean Analysis Helios VC

To sum up, for the Virtual Clusters of Helios, it is evident that the jobs exhibit a wide range of workload intensities, same as the physical GPU clusters. This variability indicates that the data within the same GPU cluster is non-stationary, and further analysis to confirm stationarity is not required for the duration time series.

Transform Timeseries into Stationary

From the initial examination of the time series, we observe that while both time series maintain a relatively constant mean over time, their variance fluctuates significantly, indicating non-stationarity. To address this, we attempted to transform the data to achieve stationarity. Since the mean is already relatively constant, our focus was on stabilizing the variance. Having stationary time series is very important in a variety of predictive models like ARMA which do not contain a preliminary differencing procedure. Models like ARIMA or SARIMA have a preliminary differencing procedure but you should know beforehand the degree of differencing that you should choose. That is why we will try to transform our timeseries to stationary.

One effective transformation for stabilizing variance is the logarithmic transformation. Another effective transformation which performs even better than the log transformation is the Box-Cox transformation. The log transformation is a specific case of the Box-Cox transformation (with $\lambda = 0$). The Box-Cox transformation is often preferred over a standard logarithmic transformation in certain scenarios because it offers greater flexibility and adaptability. The Box-Cox transformation is parameterized by a power parameter, λ , which allows it to adapt to the data better. The logarithmic transformation is most effective for data with a log-normal distribution, but it may not work well for data with other shapes or distributions. Box-Cox is designed to normalize a wide range of distributions, including skewed ones, by selecting an optimal value of λ .

The main reason that we chose the Box-Cox transformation is that the logarithmic transformation tends to overcorrect data with small values (close to zero), leading to outliers and distortions. Box-Cox, with its tunable parameter, can stabilize variance across both small and large values more effectively. One major drawback of this transformation is that it is computationally heavier than a simple logarithmic transformation. After applying the Box-Cox transformation with one order of differencing, the resulting time series is shown in (Figure 3.9). As observed, the transformed series exhibits a constant mean and stabilized variance, suggesting that the series is now stationary.

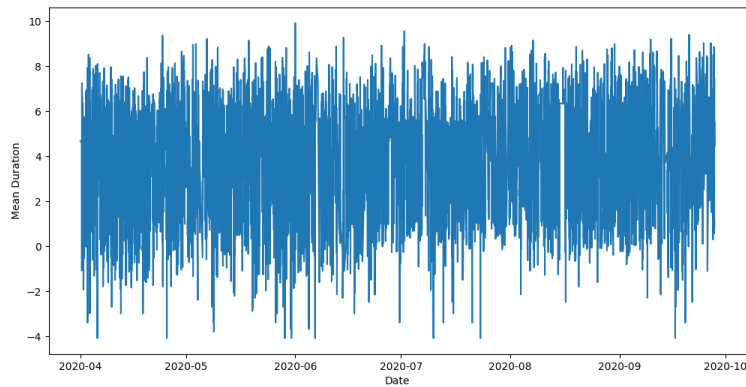


Figure 3.9: Transformed Duration

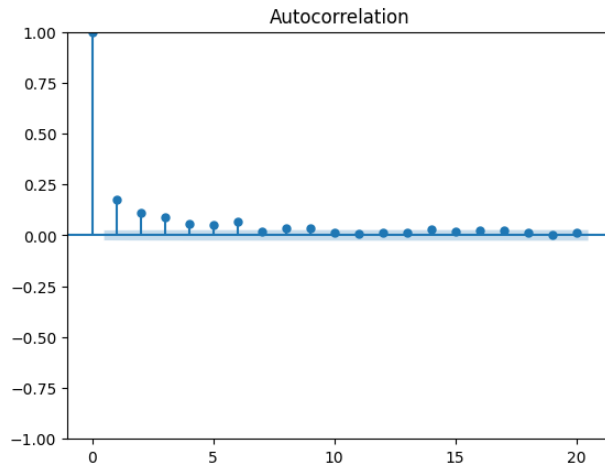


Figure 3.10: Autocorrelation of Transformed Timeseries

Time Series	ADF Test Statistic	p-value	Critical Values (1%, 5%, 10%)
run_time (VC1)	-26.79	0	(-3.431, -2.862, -2.567)
run_time (VC2)	-14.58	0	(-3.431, -2.862, -2.567)

Table 3.1: Results of ADF Test for Helios Virtual Clusters

To confirm stationarity we plotted the Autocorrelation Function (Figure 3.10), which shows no significant autocorrelation beyond the seventh lag and a big drop in lag 1, supporting stationarity. Last but not least, the ADF Test was executed to confirm stationarity. The ADF Test results are presented in Table 3.1 and as we can see the p-value is close to 0 and test statistic is significantly lower in all critical levels which means that the transformed time series are stationary. In conclusion, in this subsection, we performed a transformation in the initial timeseries and the result was a stationary timeseries, which can be used in a variety of predictive models who do not contain an integrated differencing transformation, such as ARMA. For models like ARIMA or neural networks we can use both the initial and transformed timeseries.

Correlation Analysis

The last part of the analysis for Helios dataset was to examine the relationships between key variables within the dataset. A correlation analysis was conducted for each dataset to better understand the correlations between metrics and to extract insights regarding system behavior, network performance, or user engagement patterns. These findings provide a basis for optimizing resource allocation, improving network reliability, and understanding patterns of content interaction.

The Helios dataset revealed notable interdependencies between resource-related variables, specifically between the number of GPUs, CPUs, and nodes (Figure 3.11). The strong positive correlations observed between these metrics suggest that, within this dataset, resource allocations are likely governed by a structured policy where an increase in one resource type tends to be accompanied by proportional increases in others. This finding implies that workloads within the Helios environment are highly dependent on multi-resource scaling, possibly due to the nature of applications requiring simultaneous access to multiple processing units. In contrast, the low correlations between the duration variable and other resource metrics indicate that task duration is relatively unaffected by the allocated resources, suggesting either balanced resource provisioning or workload characteristics that are independent of GPU, CPU, or node counts. Additionally, time-based metrics such as the hour of the day, day of the week, and weekend status show minimal correlations with other variables, indicating a consistent resource utilization pattern across time in the Helios environment.

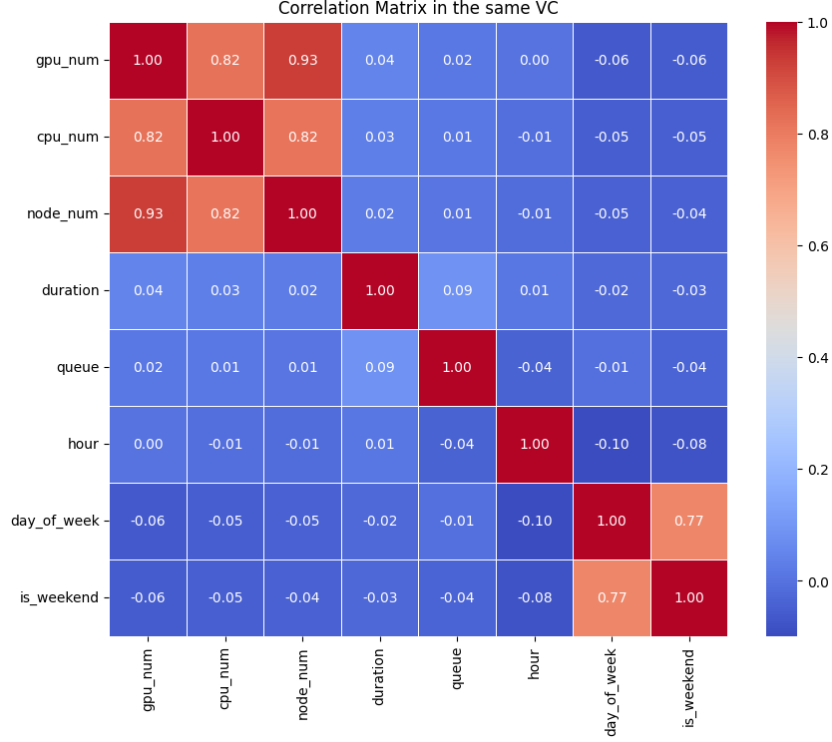


Figure 3.11: Helios Correlation

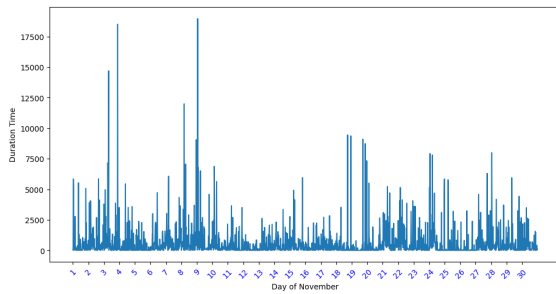
This high correlation between the resource utilization metrics might become very useful in the development of Deep Neural Networks like Deepcog, which is explained in this paper [13].

3.2.3 Philly Preprocessing

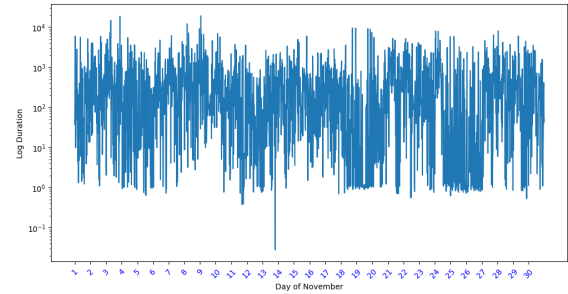
Philly traces have 2 files available where the first one contains the scheduler’s logs and the second file consists of Hardware Performance Counters. For the scheduler’s logs we divide our data per Virtual cluster and there 15 Virtual clusters available. While for the Hardware Performance Counters we divide our data per machine. In total there are 551 machines and 10 performance counters for each machine (GPU_{0-7} , CPU and Memory).

The second part of the preprocessing was to create to adjust the granularity of the samples. Philly’s dataset include time-series data like duration and queue time of a job along with the time that the job was submitted. This means that the dataset lacks a standardized temporal granularity, with job entries recorded at irregular intervals. To enable consistent time-series analysis and comparison, it was necessary to preprocess the data by aggregating it into uniform time bins.

After experimenting with the time between each bins 20 minute intervals was selected as it gave us the most consistent time bins. In this dataset as well as the previous one we have to account for extreme values. In this case a threshold of 20,000 minutes was set to filter the data for the duration timeseries and 1000 minutes was set for the queue time series. Additionally, due to the extensive volume of data, we focused on the month of November to allow for more focused and meaningful observations. The x-axis in our visualizations was adjusted to represent the days of the month numerically, facilitating a clearer temporal analysis. Notably, in November of 2017, the 1st corresponds to a Wednesday, and the sequence progresses. The final time series for duration and queue time are shown in Figures 3.12 and 3.13 respectively in both normal scale and in log scale.

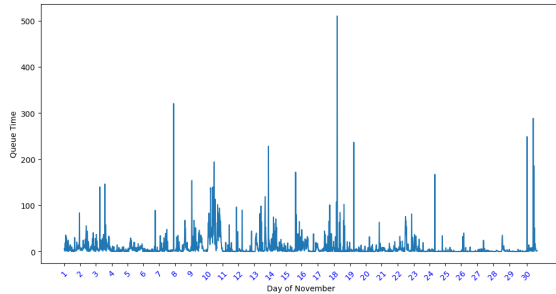


(a) Normal Scale

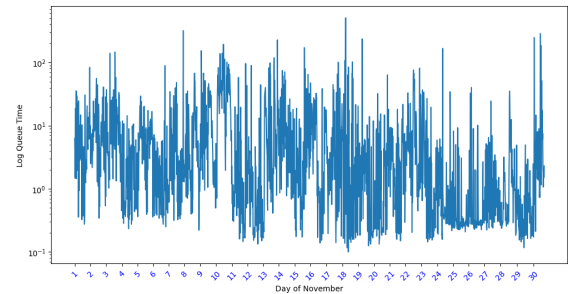


(b) Log Scale for y-axis

Figure 3.12: Binned Duration of Philly Traces



(a) Normal Scale



(b) Log Scale for y-axis

Figure 3.13: Binned Queue of Philly Traces

3.2.4 Philly Results

CCDF Analysis of Philly's Cluster

The first step in Philly's analysis was to perform a CCDF analysis in the physical GPU cluster of Philly as a whole and for some of the biggest Virtual Clusters. We plotted the CCDF of Philly's data along with the CCDF of some heavy and light-tailed distributions (like Pareto, Log-Normal, Gamma, k-Weibull and Exponential). Basically we used the same technique we used for Helios to compare our data with these known distributions. In Figures 3.14, 3.15 we plotted the CCDF of the duration and queue for the physical GPU cluster of Philly.

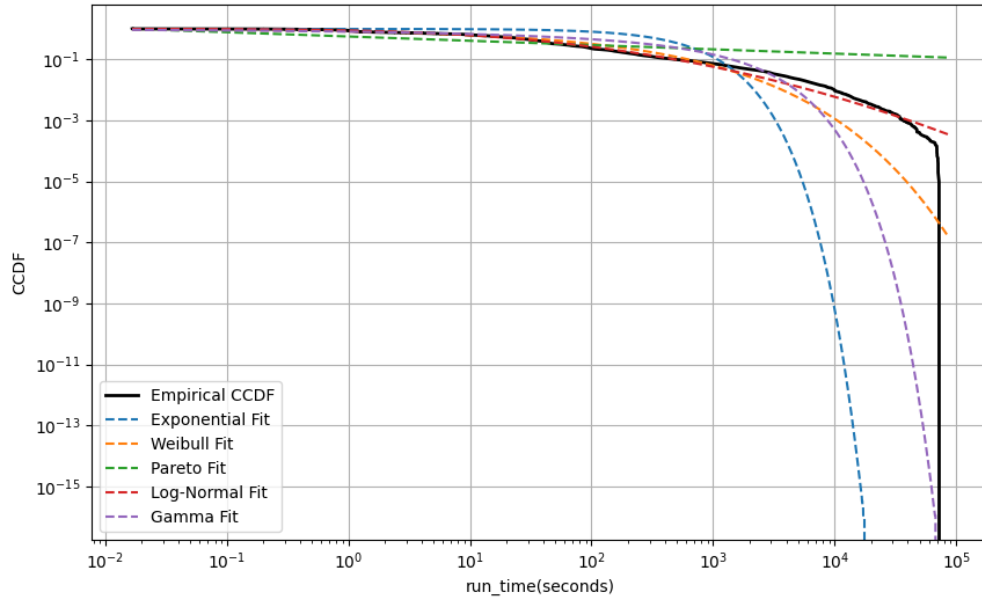


Figure 3.14: CCDF Philly GPU cluster Duration

The Complementary Cumulative Distribution Function (CCDF) analysis for queuing delay and run time reveals critical insights into the distributions governing these metrics. Both figures illustrate the alignment of empirical data with various theoretical distributions, providing a deeper understanding of the underlying workload characteristics. The empirical CCDF for both queue and duration (black line) demonstrates heavy-tailed behavior, characterized by a slower decay at higher values. Among the theoretical distributions, the Log-Normal and k-Weibull distributions align most closely with the empirical data, especially in the tail region.

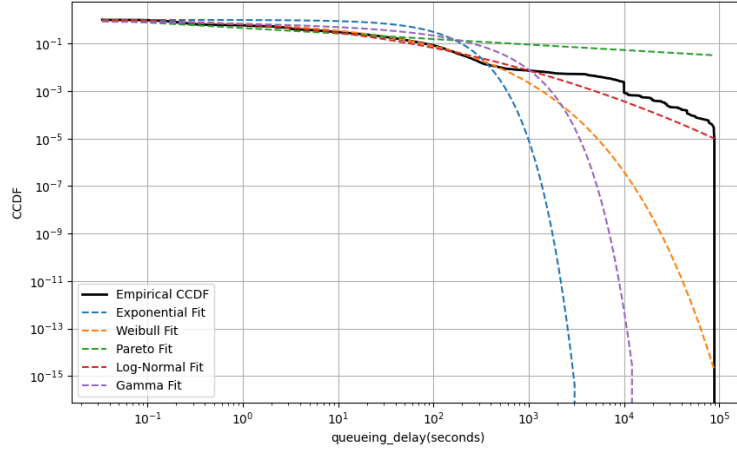


Figure 3.15: CCDF Philly GPU cluster Queue

CCDF Analysis of Virtual Clusters

We performed the same CCDF analysis but in this case we studied 2 of the busiest Virtual Clusters of Philly. This CCDF analysis was performed to identify what kind of workloads Philly's GPU cluster processes. This type of analysis is also conducted to determine whether the tail of the job distribution exhibits heavy-tailed or light-tailed characteristics, as the modeling approach and predictive strategies differ significantly between the two cases. In Figures 3.16-3.19 we plotted the CCDFs of the two biggest Virtual Clusters in Philly.

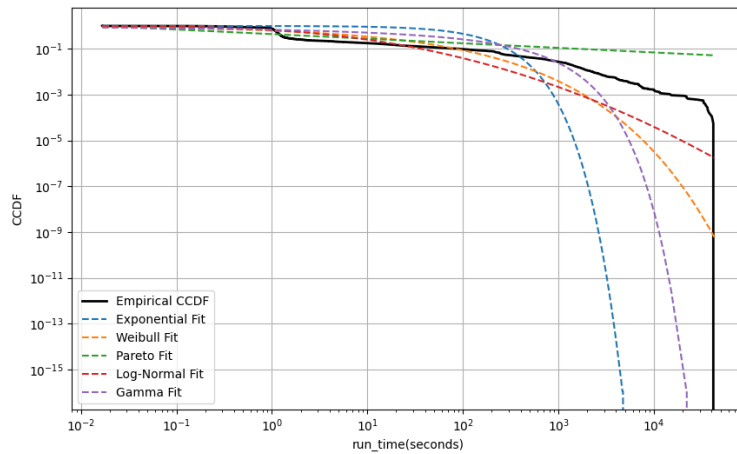


Figure 3.16: CCDF Philly VC1 Duration

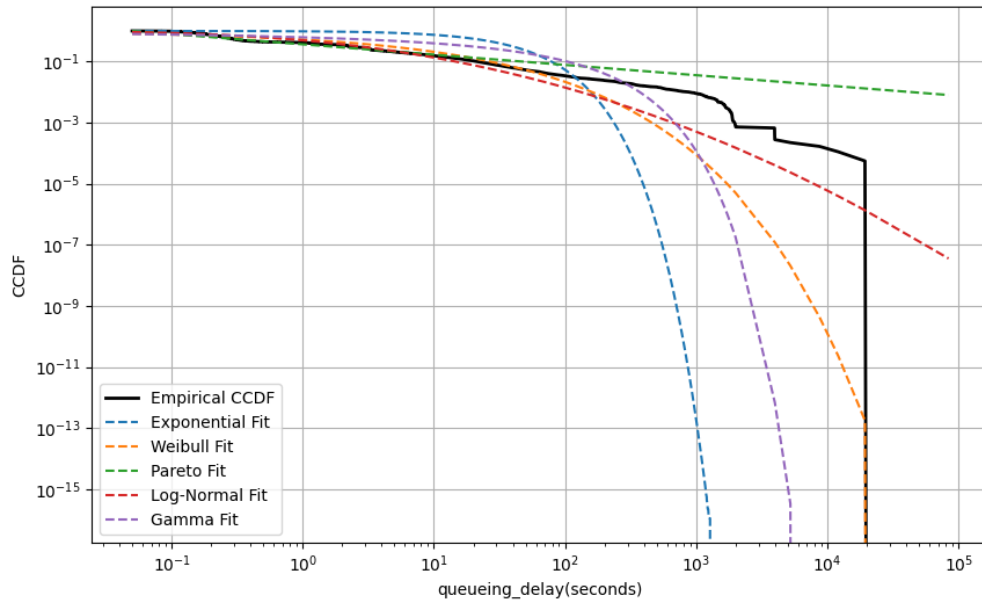


Figure 3.17: CCDF Philly VC1 Queue

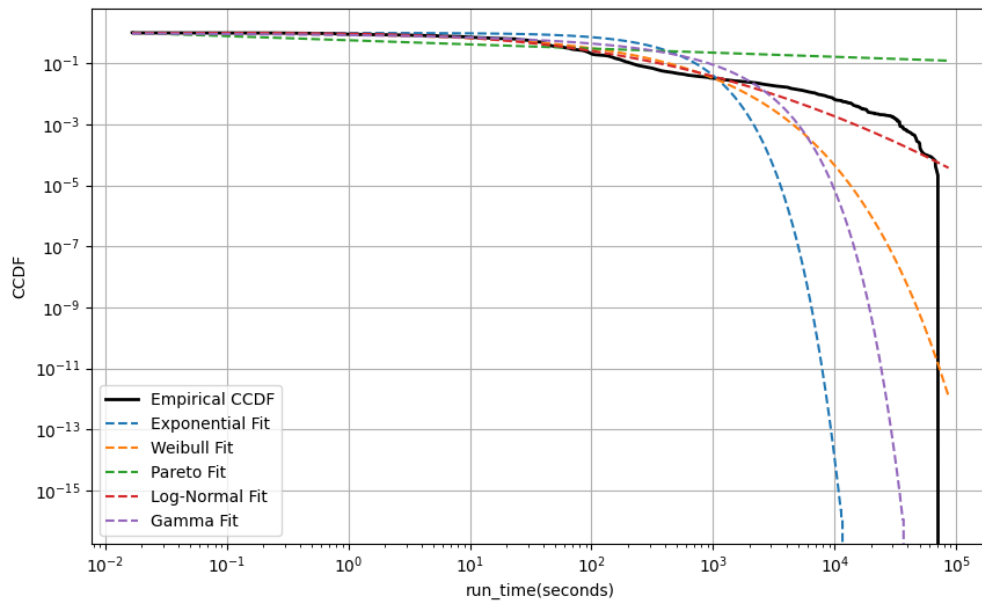


Figure 3.18: CCDF Philly VC2 Duration

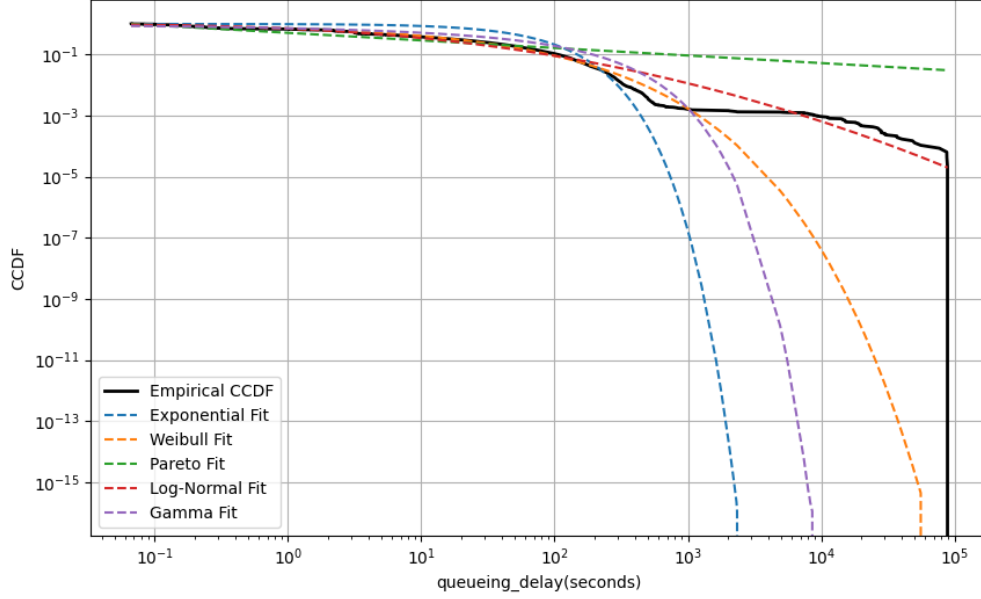


Figure 3.19: CCDF Philly VC2 Queue

In the CCDF analysis of the Virtual Clusters the same conclusion is drawn as well as for the physical GPU cluster of Philly, which is that time series that are analyzed exhibit a heavy tail. Among the theoretical distributions, the Log-Normal distributions align more closely with the empirical data. This characteristic indicates that traditional mean-based statistics may be inadequate for modeling and decision-making. More sophisticated predictive models like neural networks, which are able to capture both linear and non-linear relationships will be a better fit, when trying to predict those timeseries.

Stationarity Analysis

In analyzing the stationarity of the Philly dataset, a similar approach to that applied to the Helios dataset was employed. A visual inspection of time series data for two representative Virtual Clusters (VCs) provided initial insights into potential stationarity. Because there is only one physical GPU cluster we also inspected the cluster as whole (Figure 3.12-13). Specifically, run time and queue time metrics were examined to observe whether they fluctuated around a constant mean, an indication of stationary behavior.

To assess the stationarity of a time series, we can utilize rolling mean analysis, which provides insights into whether the mean value of the series remains stable over time. This analysis involves calculating a moving average (rolling mean) over a specified window size, which helps identify trends or patterns that might suggest non-stationarity if they are present. In the following Figures 3.20, 3.21 we plotted the queue and duration of the 2 biggest Virtual

Clusters along with the rolling mean of 100 window. Figures 3.20 and 3.21 depict the original data alongside a 100-point rolling mean. The rolling mean remains relatively stable throughout the observed period, indicating that the mean of the series does not exhibit significant changes over time. In contrast to the stable mean, the variance shows considerable fluctuations over time, with periods of high variability interspersed with more stable intervals. These fluctuations indicate that the variance is non stationary.

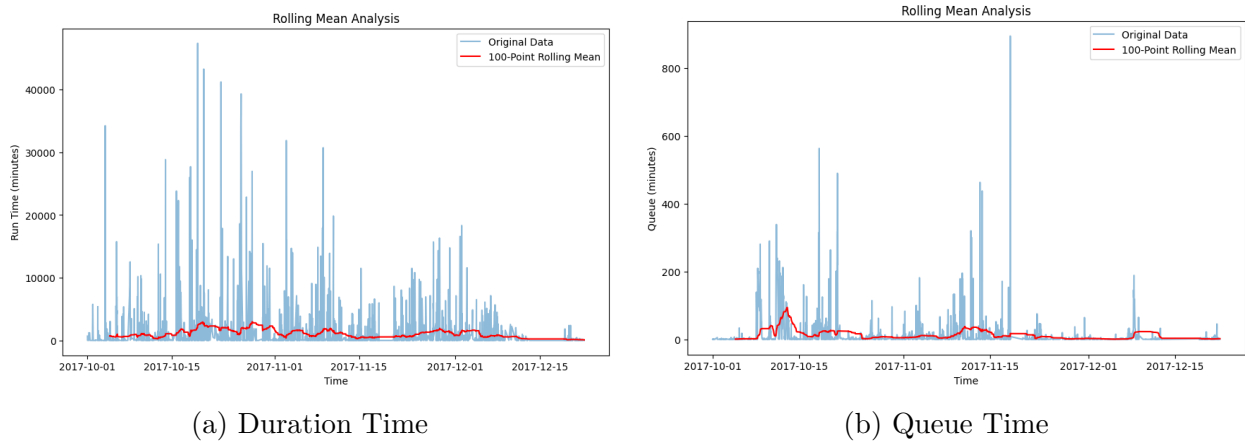


Figure 3.20: Timeseries for VC1 of Philly

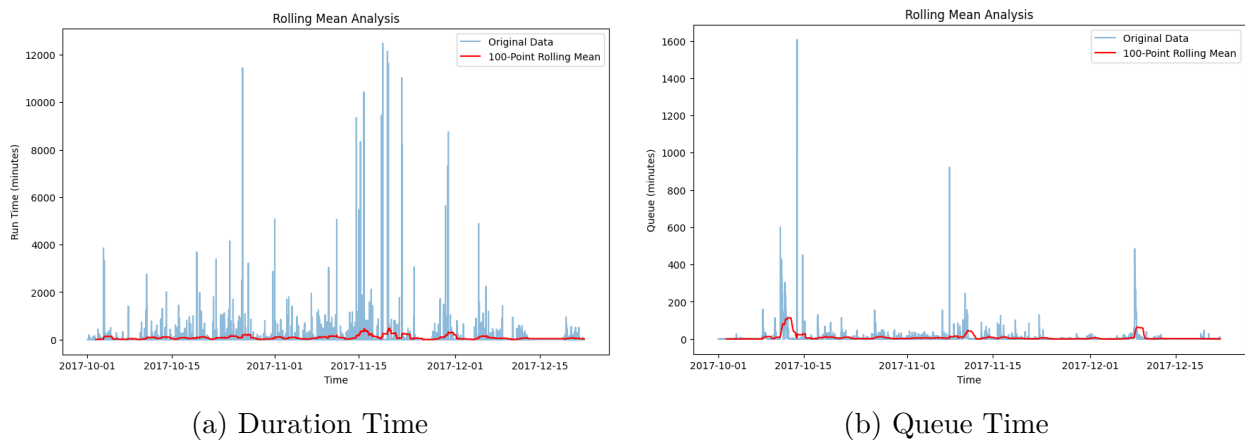


Figure 3.21: Timeseries for VC2 of Philly

Transformation into Stationary

Based on the above observations, the time series for queuing and runtime data are not stationary because stationarity requires both a constant mean and constant variance over

time. The lack of variance stationarity means that transformations are necessary to stabilize the variance before proceeding with further time series modeling. To address the non-stationarity in variance. Applying a box-cox transformation to the data can help reduce the impact of extreme values and stabilize the variance. In some rare cases where the mean is not constant over time we need to difference the data. The next plot is the differenced data (Figure 3.22)

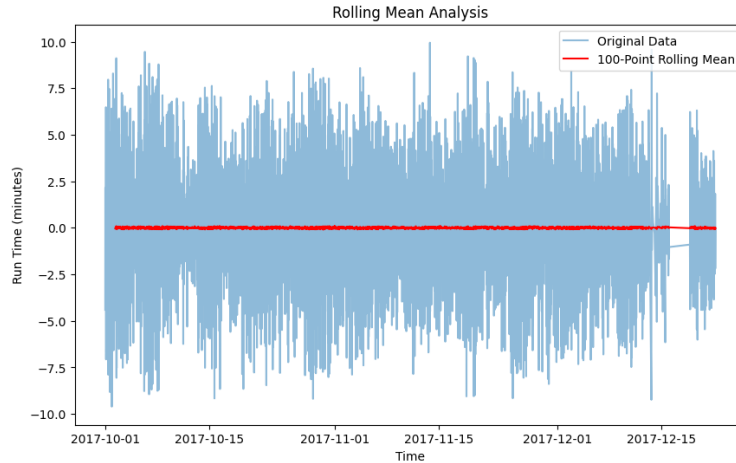


Figure 3.22: Transformed Timeseries along with Rolling mean

To verify the stationarity of the transformed time-series data, we performed the Augmented Dickey-Fuller (ADF) test and examined the Autocorrelation Function (ACF). The ADF test results, as presented in Table 3.2, indicate a test statistic significantly lower than the critical values at all levels. Additionally, the p-value is below the threshold of 0.05, allowing us to reject the null hypothesis of non-stationarity. Consequently, the transformed time series can be deemed stationary. Furthermore, the ACF plot (Figure 3.23) supports this conclusion, showing a rapid decay with significant correlations observed only at lags 0 and 1, which are characteristic of a stationary time series.

Time Series	ADF Test Statistic	p-value	Critical Values (1%, 5%, 10%)
duration (VC1)	-20.31	0	(-3.431, -2.862, -2.567)
duration (VC2)	-20.03	0	(-3.431, -2.862, -2.567)

Table 3.2: Results of ADF Test for Philly

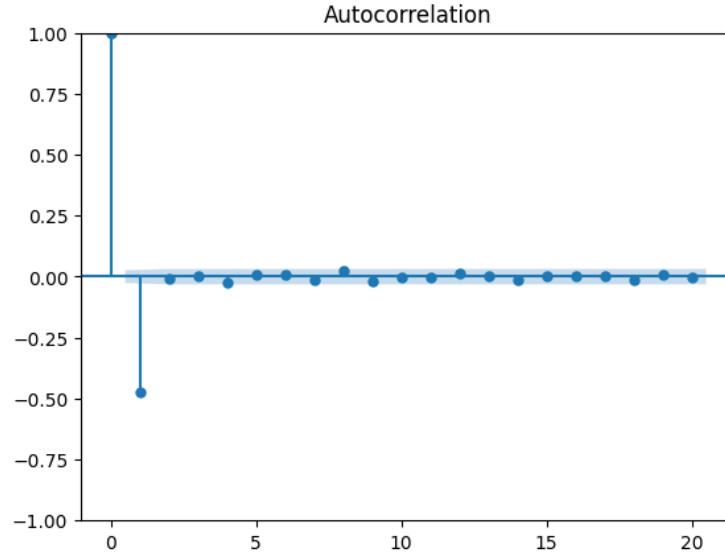
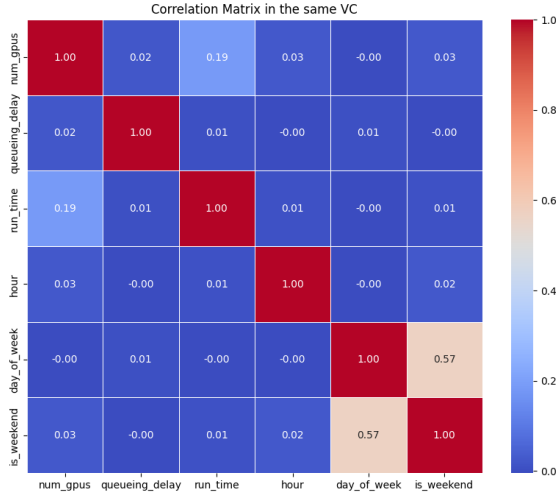


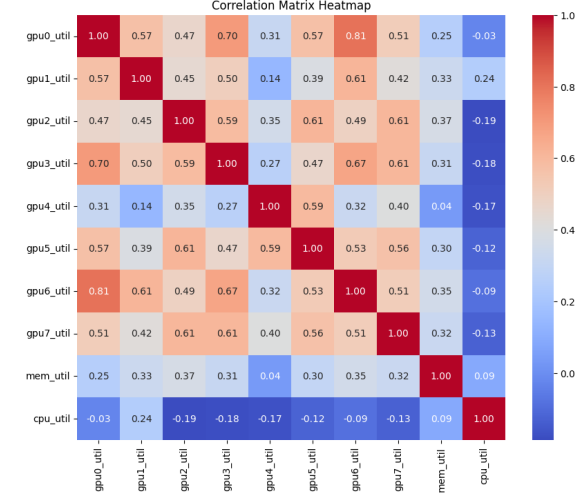
Figure 3.23: Autocorrelation Function Plot Transformed Duration Philly

Correlation Analysis

The last part of Philly’s analysis was to find the correlation between the available time-series. The Philly dataset demonstrated a distinctive pattern with respect to both general resource allocation metrics and GPU-CPU utilization metrics (Figure 3.24). In the first set of correlations, which included metrics like GPU count, queuing delay, and runtime, weak correlations were observed across the board. The results suggest that, in the Philly dataset, resource allocation and job performance indicators such as queuing delay and runtime are largely independent of GPU count, which may point to effective load-balancing mechanisms or resource management practices that mitigate congestion and runtime variability. The analysis of GPU and CPU utilization revealed strong correlations among GPU utilization metrics, suggesting parallel or concurrent usage patterns, likely in multi-GPU tasks or distributed computations. A moderate correlation between CPU and GPU utilization indicates that CPU resources play a supporting role in managing or coordinating GPU-intensive tasks, a common scenario in GPU-driven computational environments. Furthermore, memory utilization displayed weak correlations with both CPU and GPU usage, suggesting that the workloads in this dataset are more computation-bound than memory-intensive.



(a) Times Correlation of Philly



(b) Correlation of Utilization Metrics

Figure 3.24: Philly Correlation

As mentioned before the resource utilization metrics exhibit high correlation. This high correlation is the motivation of the second experiment of our practical application in section 4, where we used the research from this paper "DeepCog: Cognitive Network Management in Sliced 5G Networks with Deep Learning" [13]. The existence of these dependencies allow AI models to learn patterns effectively. This paper proposes Convolutional Neural Networks (CNNs) process time series data, treating them as images instead of traditional sequential data. CNNs excel in recognizing spatial dependencies, making them well-suited for traffic forecasting and anomaly detection in network management.

3.3 4G Dataset

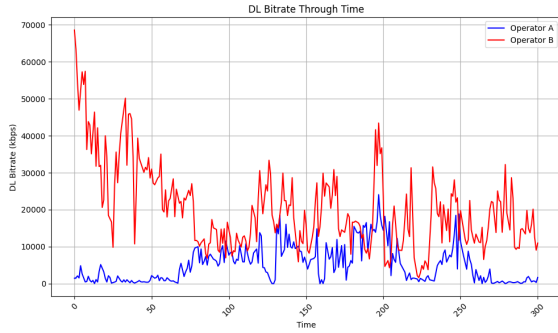
3.3.1 Preprocessing of Dataset

For the 4G trace we have 135 different traces across 5 different mobility patterns. Each trace contains the same information about the network. The following table shows how many timeseries are available for each dataset. In total for each trace there 7 timeseries available (RSSI, RSRQ, RSRP, SNR, CQI, DL_bitrate, UL_bitrate). The only task that needed to be addressed was the issue of missing values (NaN) in the dataset. Two strategies were considered filling the NaN values or dropping the columns containing them. The decision depended on the proportion of missing values and the significance of the respective columns to the analysis. We opted to drop the missing values to ensure the integrity and reliability of the dataset.

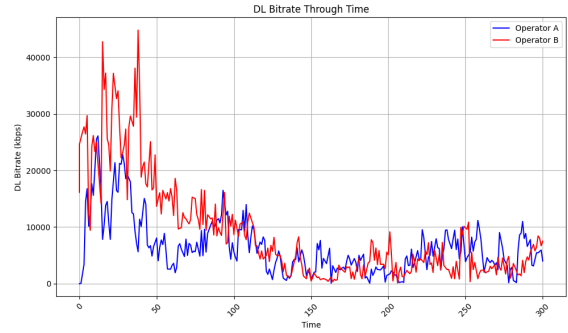
3.3.2 4G Dataset Results

Stationarity Analysis

The first step in the analysis of this dataset was to assess the stationarity of the time series that correspond to the throughput of the network. Specifically the stationarity of the throughput of the network is assessed among all the different mobility patterns. The first step in stationarity analysis is the visual inspection of time series. In Figure 3.25 and 3.26 the throughput of the network for both operators is presented across all the different mobility patterns. A visual inspection of the Downlink (DL) Bitrate (kbps) for Operator A and Operator B reveals several characteristics that suggest non-stationarity. The time series exhibits significant fluctuations over time, with no consistent mean or clear trend stabilization. Additionally, the variance of the DL bitrate appears to change dynamically, reinforcing the assumption of non-stationarity.

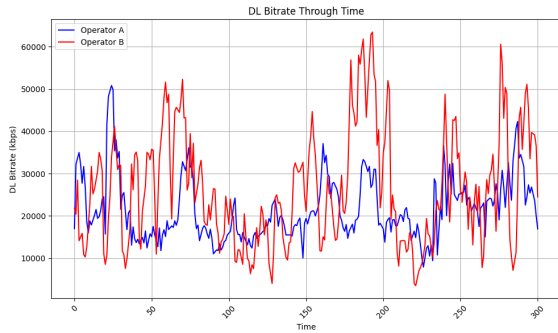


(a) Static Mobility Pattern

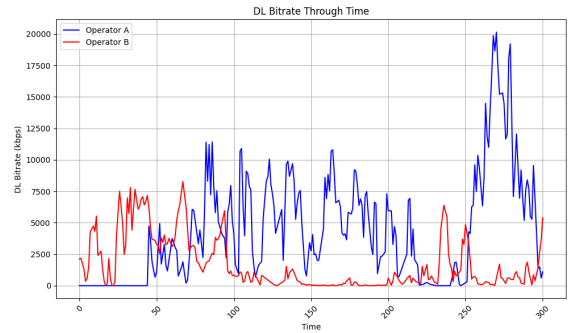


(b) Pedestrian Mobility Pattern

Figure 3.25: Throughput for Pedestrian and Static



(a) Car Mobility Pattern



(b) Train Mobility Pattern

Figure 3.26: Throughput for Car and Train

No evident periodic or repetitive patterns are discernible in the data, though the variability of fluctuations suggests potential underlying influences that merit further analysis. Overall, the lack of a stable mean and variance implies that the time series is likely non-stationary.

Transformation into Stationary

Non-stationary time series data must be transformed into stationary form for effective predictive modeling. Stationarity is essential because models trained on stationary data can identify consistent patterns over time, leading to more reliable predictions. To achieve a constant mean, we apply differencing to our data, and to stabilize variance, we take the logarithm of the series. However, since throughput values can occasionally reach zero, which is undefined for logarithmic transformation, firstly we need to shift the data by a very small value. Another solution which gives us better results for the variance is the Box-Cox Transformation. The Box-Cox transformation is often preferred over a standard logarithmic transformation in certain scenarios because it offers greater flexibility and adaptability.

This transformation results in a stationary series, as demonstrated in Figure 3.27. In the transformed timeseries some statistical test were conducted to confirm the stationarity of the new timeseries. In Table 3.3 the results of the ADF test for all different mobility patterns (except Train) is presented. In Figure 3.28 the ACF plot of the transformed timeseries is also presented.

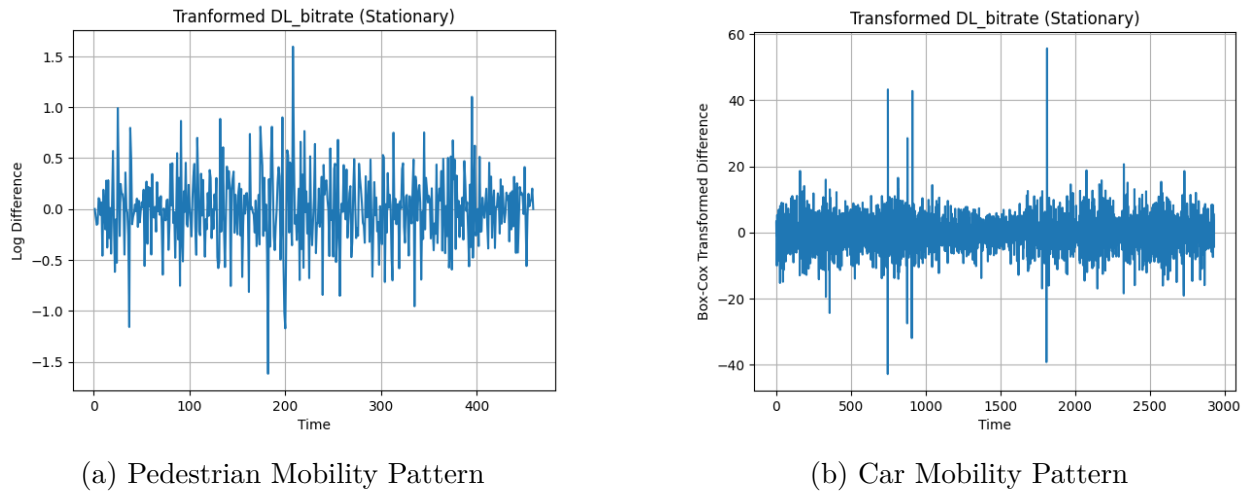


Figure 3.27: Transformed Throughput for Car and Pedestrian

Mobility Pattern	ADF Test Statistic	p-value	Critical Values (1%, 5%, 10%)
Static	-15.67	0	(-3.97, -3.41, -3.13)
Pedestrian	-10.90	0	(-3.44, -2.865, -2.568)
Car	-15.67	0	(-3.97, -3.41, -3.13)

Table 3.3: Results of ADF Test for 4G Dataset

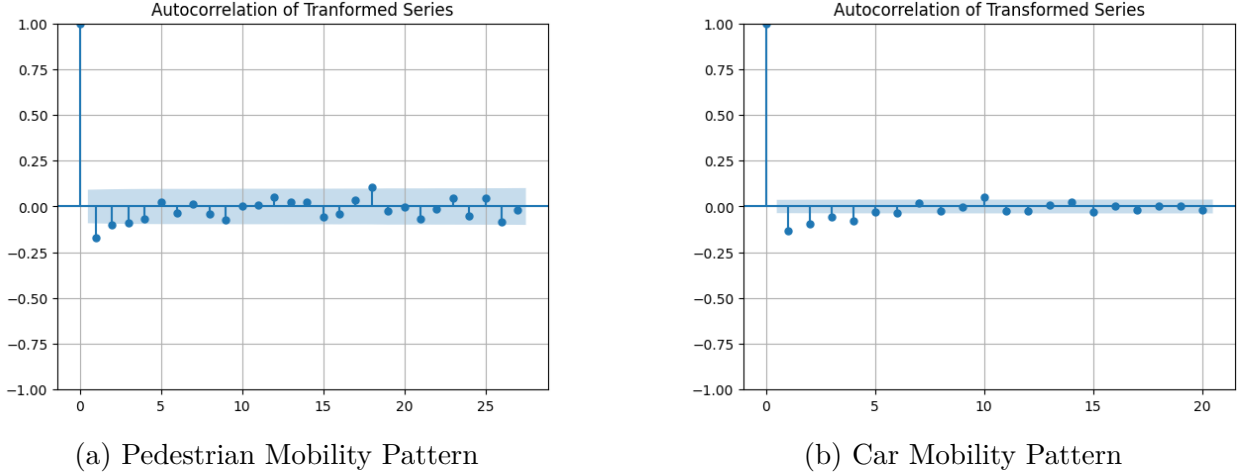


Figure 3.28: Autocorrelation Function for Car and Pedestrian

The plot of the transformed series shows fluctuations around a zero mean, with consistent variance across the time axis. There are no apparent trends or seasonality, which are hallmarks of a stationary series. The ACF plot demonstrates a rapid decay, with significant autocorrelations observed only at lag 0. This behavior aligns with stationary time series characteristics, where correlations decrease quickly as lag increases. Last but not least, the results of the ADF test confirm that the null hypothesis of non-stationarity is rejected, meaning the series is stationary.

The throughput in the train mobility pattern, exhibits non-stationary behavior even after applying transformations of Box-Cox and differencing (Figure 3.29a). While the mean of the series appears steady, the variance fluctuates significantly over time, as seen in the transformed DL bitrate plot. Additionally, the autocorrelation function (ACF) plot reveals lags that lie outside the confidence bounds, further supporting the presence of non-stationary components.

This non-stationary behavior can be attributed to the unique characteristics of train mobility patterns. Trains traverse varying geographical locations, including areas with dense urban structures, tunnels, or open rural areas, which introduce significant fluctuations in network conditions. These environmental variations affect signal quality, interference, and

network load, causing inconsistent variance in the throughput. Furthermore, the mobility of the train itself, including speed variations and handovers between base stations, contributes to irregularities in the data.

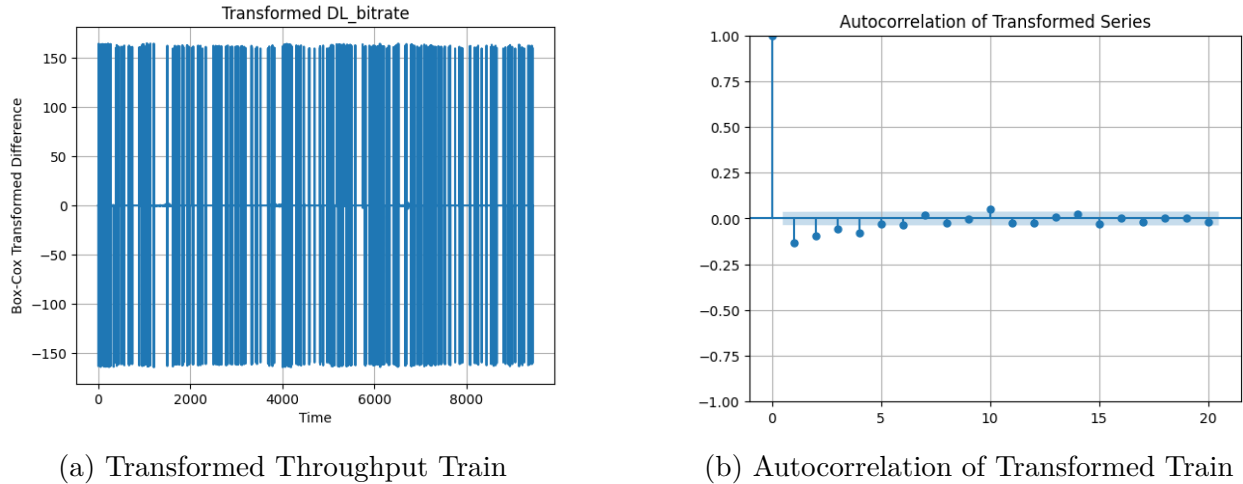


Figure 3.29: Transformed Plots in Train Mobility Pattern

Such inherent factors make it challenging to achieve stationarity in train mobility datasets, even after rigorous transformations. This observation highlights the need for robust modeling techniques capable of accommodating non-stationary dynamics when analyzing train mobility throughput patterns.

Correlation Analysis

The last part of the analysis as usual is to calculate the linear relationship between timeseries in the same file. This can be achieved using the Pearson correlation coefficient. In the LTE 4G dataset, the analysis focused on signal quality and performance metrics (Figure 3.30). Strong correlations were found among signal quality indicators such as RSRP, RSRQ, RSSI, and SNR, which are inherently interrelated due to their shared dependence on signal strength and quality. The high correlation between RSRP and RSSI, for instance, aligns with the general expectation in LTE networks where increases in power levels lead to improvements in signal strength metrics. A similarly strong correlation was observed between RSRQ and SNR, both of which are indicators of signal quality under interference conditions. This dataset also showed a high correlation between downlink and uplink bitrate, suggesting balanced usage patterns for data flow in both directions. Notably, the correlation between these signal metrics and the hour variable was low, indicating that network performance remains relatively stable across different times of the day, likely due to the LTE network's design to manage peak loads effectively.

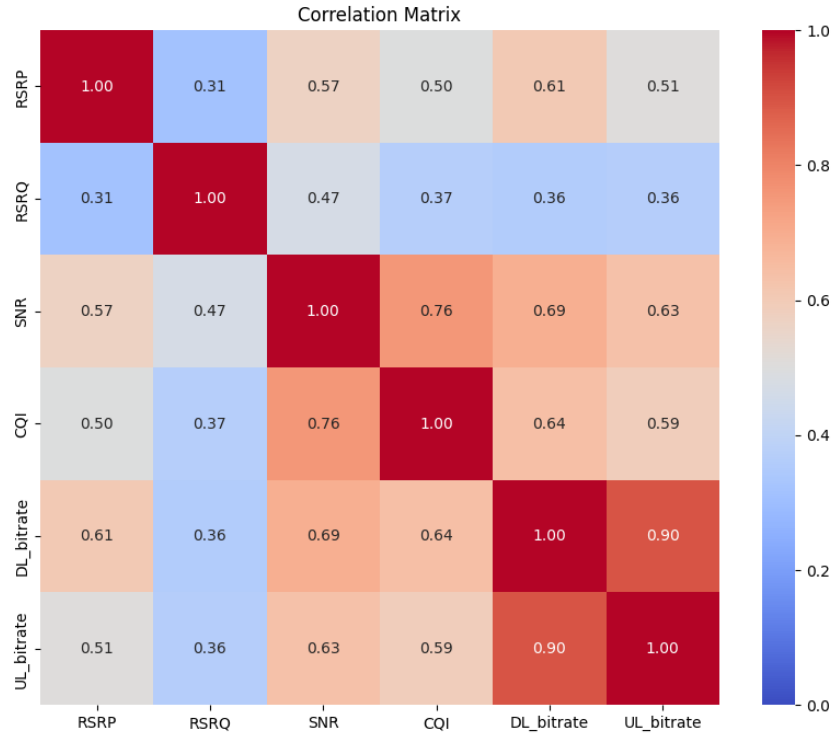


Figure 3.30: 4G LTE Correlation

3.4 Youtube Dataset

3.4.1 Youtube Preprocessing

Finally for the dataset that was collected from Youtube we have 1611 traces, where every trace represents the count observation for a video. For every video there are 8 different timeseries (views, likes, comments, dislikes and their differences per hour).

3.4.2 Youtube Dataset Results

Stationarity Analysis

During the stationarity analysis of the YouTube dataset, three distinct patterns emerged. For videos with high view counts, the increase in views was found to be steady and consistent, as illustrated in Figure 3.31. In contrast, videos with low view counts exhibited more erratic and unpredictable behavior, as we can see in Figure 3.33. Lastly, videos with average view counts tended to experience a viral period, during which the rate of increase in views was significantly higher compared to other periods (see Figure 3.32).

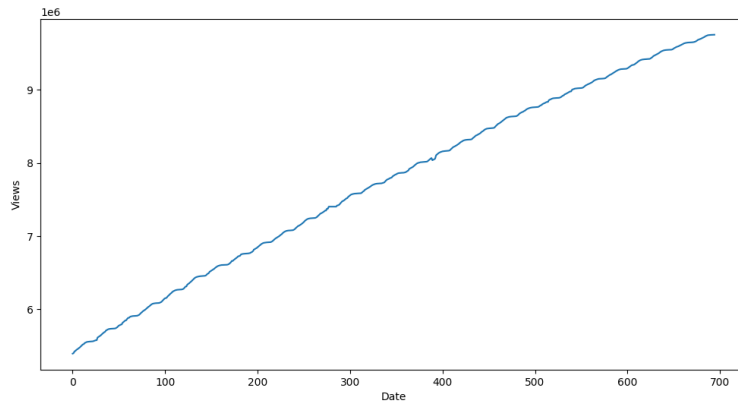


Figure 3.31: Views of Viral Video

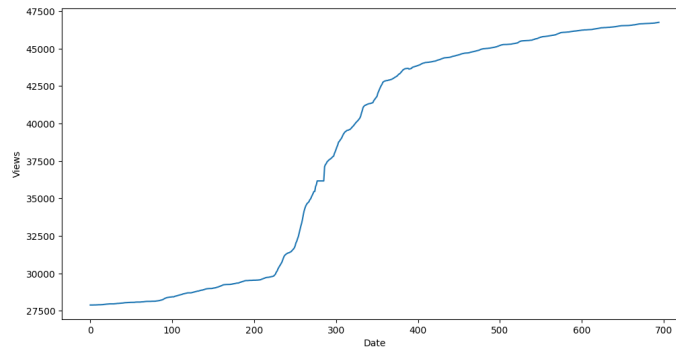


Figure 3.32: Views of Average Video

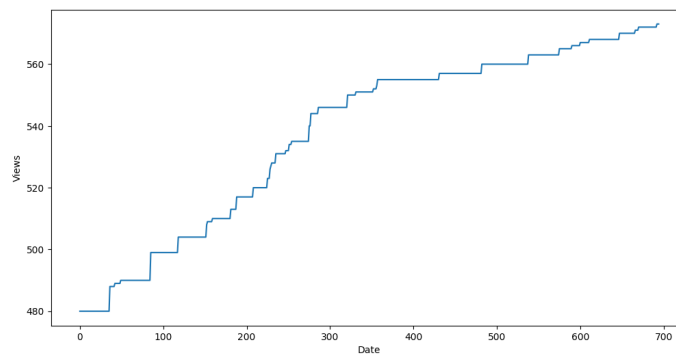


Figure 3.33: Views of Low-View Video

Like all the other datasets the ADF Test was conducted to test the stationarity or non-stationarity of the timeseries. The results from the test confirmed that the time series

exhibits non-stationary behavior, as expected, due to the clear upward trend in the data. To further support this conclusion, we also plotted the Autocorrelation Function (ACF), which reinforced the presence of strong autocorrelations in the data over time, another sign of non-stationarity.

Table 3.4 summarizes the results of the ADF test, highlighting the test statistic and critical values, which further confirm the time series is not stationary. Autocorrelation Function Plots are presented from Figure 3.36a and 3.36b. This combination of visual analysis and statistical testing provides strong evidence that the series requires further transformation, such as differencing, to achieve stationarity for more robust modeling and forecasting.

Type	ADF Test Statistic	p-value	Critical Values (1%, 5%, 10%)
High-Views	-2.65	1	(-3.9718, -3.4168, -3.1307)
Mid-Views	-0.96	0.9492	(-3.9718, -3.4168, -3.1307)
Low-Views	-0.90	0.956	(-3.9718, -3.4168, -3.1307)

Table 3.4: Results of ADF Test for Youtube Dataset

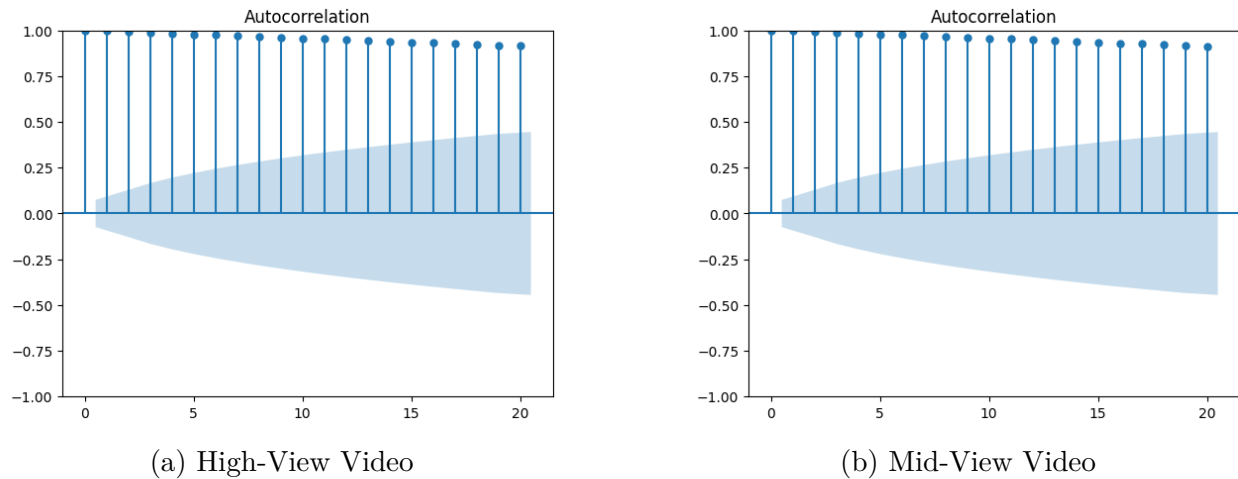


Figure 3.34: Autocorrelation Function Plot High Mid Views

Transform Timeseries to Stationary

Having non-stationary data might become a problem in the future if we want to apply predictive models. That is why we attempted to transform the time series to stationary. This can be achieved by differencing, to stabilize the mean of the series, and by logarithmic transformation or box-cox, to stabilize the variance. This transformation helps to eliminate the strong trend and reduce variability in the data, making it more stable over time.

Following the visual inspection of the transformed data, we reapplied the Augmented Dickey-Fuller (ADF) test and examined the Autocorrelation Function (ACF) to confirm if the series had achieved stationarity. Most time series showed stationary characteristics after the transformations, except for the mid-views video series. This particular series experienced a viral period, causing a persistent trend that was not fully eliminated by the initial differencing. Due to this remaining trend, additional differencing may be necessary to achieve complete stationarity.

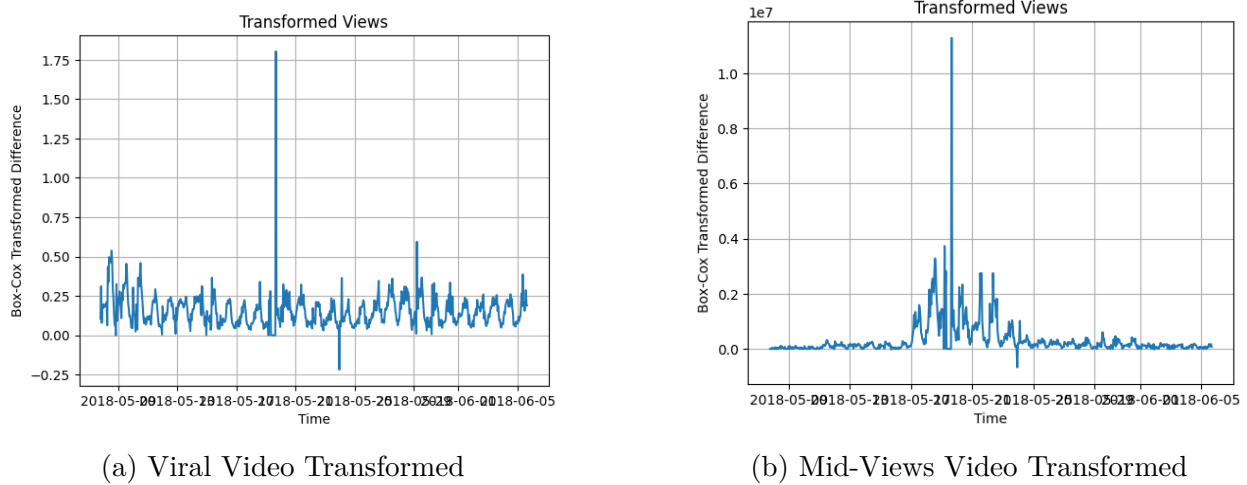


Figure 3.35: Transformed Timeseries for Views

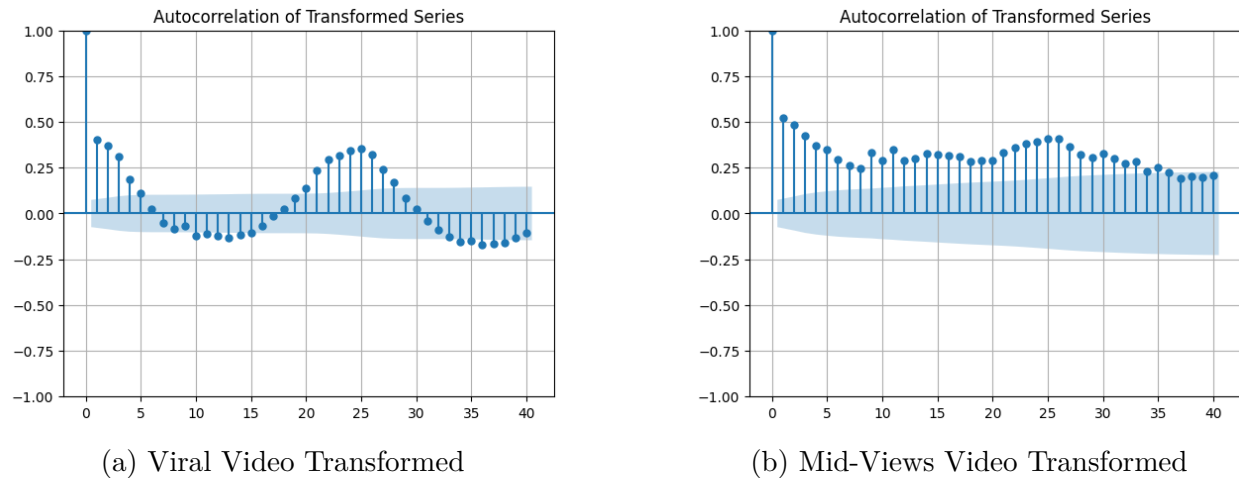


Figure 3.36: Autocorrelation of Transformed Videos

From Figure 3.33a we can conclude that the transformed timeseries shows a relatively stable mean after applying the Box-Cox transformation and differencing, which suggests some improvement toward stationarity. However, certain aspects indicate that the series might still have non-stationary components. There are noticeable spikes in the series, particularly around specific timestamps, which suggest outlier behavior or sudden events influencing the data. While the variance appears more stable compared to the raw data, it still shows fluctuations, indicating incomplete stabilization. In Figure 3.34a the ACF plot reveals several significant lags that exceed the confidence bounds. We can see a pattern in the ACF indicates the presence of cyclic or seasonal behavior in the data. This is a hallmark of time series with residual seasonality or periodic trends. Finally, the slow decay in autocorrelations suggests that the series still retains long-term dependencies, which are typically present in non-stationary data. To sum up, this timeseries is not fully stationary, as evidenced by The variance fluctuations and extreme spikes in the transformed series and by the ACF plot which shows periodic correlations and lags outside the confidence intervals.

For the mid-views video (Figure 3.33b) in the transformed series there can be observed large spikes in specific moments that disrupt the stability of the series. We can clearly state that there is a non-constant variance. In Figure 3.34b where the ACF of the mid-views video is shown there are significant positive correlations that decay very slowly, with many lags remaining above the confidence bounds. This is a strong indicator of non-stationarity. The slow decay in the ACF indicates the presence of long-term dependencies, which further suggests that the series is not fully stationary. Thus, the transformation in both cases did not result in stationary timeseries.

Correlation Analysis

Finally, the last part of our analysis was to perform a correlation analysis in the Youtube dataset. This dataset presented correlations across user engagement metrics, shedding light on content interaction patterns (Figure 3.37). Metrics such as viewCount, likeCount, and commentCount exhibited strong positive correlations, indicating that videos with higher visibility often receive more likes and comments. The relationship between dislikeCount and other engagement metrics, although weaker, was still positive, implying that popular content may attract a proportionally higher number of dislikes, possibly reflecting a broader spectrum of user responses due to increased exposure. The correlation analysis of engagement rate metrics, including the differences in view count, like count, dislike count, and comment count, revealed moderate to high correlations, especially between like and dislike rates, suggesting that engagement spikes for specific videos tend to include both positive and negative reactions. The low correlation between categoryId and the engagement metrics indicates that the video's content category does not significantly affect engagement levels, suggesting that user interest in videos may transcend categorical boundaries. Additionally, the duration variable showed negligible correlation with other metrics, implying that video length does

not play a significant role in influencing viewership or interaction rates.

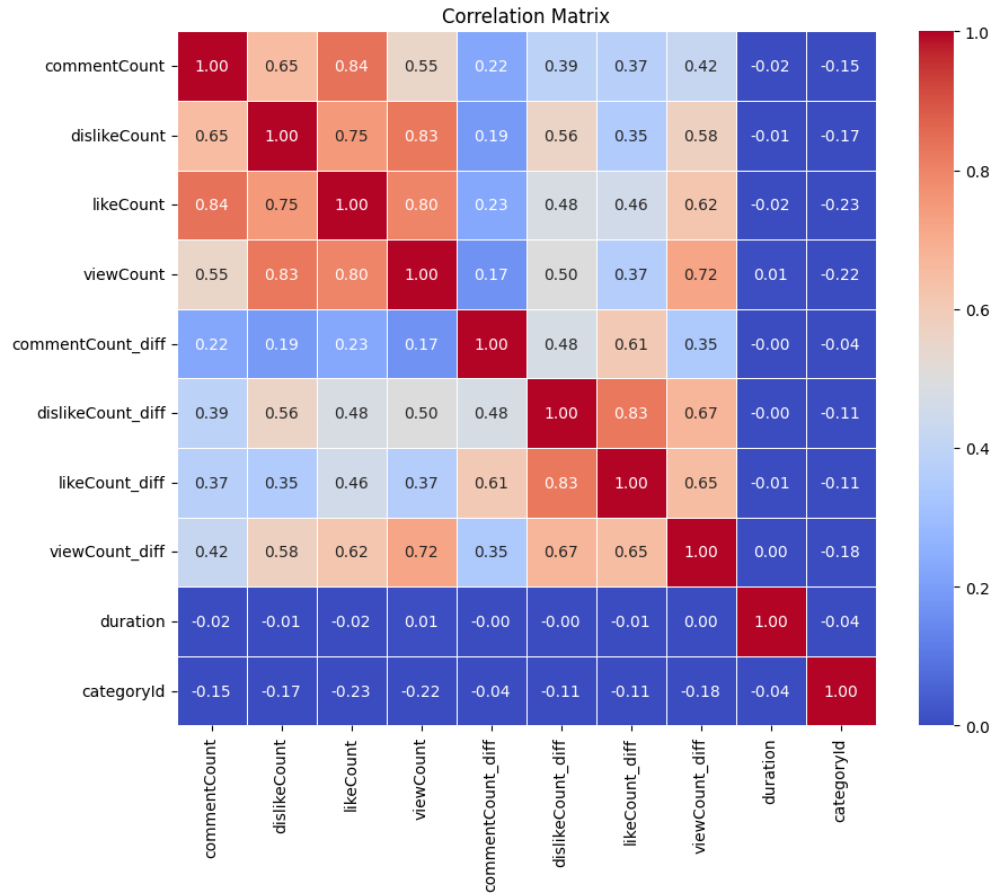


Figure 3.37: Youtube Dataset Correlation

Chapter 4

Practical Applications

In this section we are going to perform two practical application of the datasets explained in Sections 2, 3. As a first step for each experiment we are going to present a short Theoretical Background for the predictive models that were implemented. Later each experimental procedure and their results are thoroughly explained. In the first experiment we attempted to predict the 4G network throughput from the 4G LTE dataset, using various predictive models. The models that were implemented are the ARIMA model and a variety of neural networks. The performance of each model was assessed by calculating various metrics and by visual inspection. In the second experiment the idea was based in the paper of DeepCog [13], but we altered the implementation to fit our dataset. Our goal was to take advantage of the correlation that was found in the utilization metrics of the philly dataset and to identify if this correlation can aid us to make more accurate predictions. In the same spirit as the first experiment both visual inspections and metrics were calculated to assess and compare the model with a simple LSTM model that does not take into account the correlation of the time series.

4.1 Predictive Models

Predictive models play a crucial role in forecasting time series data. The main goal of these models is to capture trends, patterns and seasonality of historical data and then try to make an accurate prediction of the future values. Depending on the nature and complexity of the time series, different predictive models can be applied, from simple statistical models to more advanced machine learning models.

This section introduces two widely used predictive modeling techniques: ARIMA, a statistical approach ideal for linear and stationary time series, and neural networks, which are powerful for capturing complex and non-linear patterns. These foundational models will be further explored in the subsequent experimental analysis to evaluate their performance on

time series forecasting tasks.

4.1.1 ARIMA

ARIMA falls into the category of statistical models and is widely used for time series forecasting in areas like economics and weather forecasting. ARIMA is used for datasets that exhibit trends and seasonality patterns. ARIMA model contains 3 main components as the name announces in advance: Auto-Regressive (AR), Integrated (I) and Moving Average (MA).

The AR part models the relationship between a data point and its previous values. For example, in an AR(3) model, the current value depends linearly on the three previous values. The Integrated (I) part practically accounts for trends in the data. If activated the time series will be differenced with the aim of making the data stationary. Stationarity is a critical assumption in ARIMA, but the Integrated component makes the model suitable for non-stationary data by applying a differencing transformation d times until the data achieves stationarity. Last but not least, the MA part captures the relationship between a data point and past error terms. Specifically, this component captures the relationship between an observation and the residual errors from a moving window of previous time steps. The MA term smooths out noise in the data by modeling the dependency on past forecast errors.

When we try to implement the model, 3 parameters need to be tuned ARIMA(p, d, q). The order of the AR component is represented by the p parameter, indicating how many past values are considered. The degree of differencing required to make the series stationary is the d parameter. Finally, the q parameter is the order of the MA component, representing how many past forecast errors are considered. To find the best parameters for p and q we have to observe the Autocorrelation Function Plot (ACF) and the Partial Autocorrelation Function Plot (PACF). Moreover, the optimal d value is considered to be the times we have to difference the data, in order to become stationary.

Fortunately, Python provides the `auto_arima` function, which efficiently identifies the optimal combination of parameters for ARIMA models. This functionality significantly automates the parameter selection process, saving considerable time and effort, particularly when conducting multiple experiments, such as optimizing parameters across ten different scenarios.

In general, ARIMA is a highly effective model for short-term forecasting, particularly for time series data with linear relationships. Its ability to handle non-stationary data through the Integrated (I) component makes it versatile and applicable to a wide range of datasets. Another key advantage of ARIMA is its simplicity and relatively low computational requirements compared to more complex models like neural networks. This efficiency makes it a popular choice for time series forecasting tasks in resource-constrained environments.

Despite its strengths, ARIMA has notable limitations. It assumes linear relationships in the data, which may hinder its performance on datasets with non-linear or intricate patterns.

The model also requires extensive preprocessing, such as ensuring the data is stationary and determining the appropriate parameters, which can be time-consuming. Additionally, ARIMA is not well-suited for multivariate time series or datasets with a high level of noise, limiting its application in more complex scenarios.

4.1.2 Neural Networks

Artificial Neural Networks (ANNs) are a core component of machine learning, designed to mimic the structure and function of the human brain. Due to their adaptability, ANNs are used extensively across disciplines, with their architecture tailored to specific tasks.

A neural network typically consists of three types of layers: the input layer, where data is received; one or more hidden layers, which process the data to uncover patterns; and the output layer, which provides the final prediction. The layers are interconnected, with each neuron in one layer linked to neurons in the next via weighted connections. These connections and weights enable the model to learn complex relationships within the data.

Different neural network architectures are suited for various tasks. In this study, we utilized two primary architectures: Feedforward Neural Networks (FNNs) and Recurrent Neural Networks (RNNs). FNNs, exemplified by Fully Connected Neural Networks (FCNNs), process data in a unidirectional flow from input to output. RNNs, such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), are optimized for sequential data, as they can retain information about previous inputs to inform future predictions. In addition to these neural models, we also implemented two non-neural network techniques: Gradient Boosting Descent (GBD) and Random Forest, which are widely recognized for their robustness in predictive modeling.

4.2 Prediction in 4G Dataset

In this section, we evaluate the performance of various predictive models on the 4G network throughput. The predictive models that are implemented vary from simple statistical models like ARIMA to more advanced such neural network (LSTM, GRU, FCNN) and non-neural network (Gradient Boosting, Random Forest) approaches. The goal is to identify the most suitable models for capturing the dynamics of the dataset.

4.2.1 Prediction using ARIMA

As it is referenced in section [4.1.1], the ARIMA model (p,d,q) uses the last q error terms to make a prediction. In practice, if the model is trained only one time without updating its values (adding test error into the array), the first prediction for instance will be based on the four last known data if q equals to four, while the second one in the last 3 real values

and this goes on. This approach will not consider at all the forecast errors. The prediction of q_{th} will depend solely on the predicted values. This means that from one point on the newly predicted values will be based on the old predicted values. Each prediction will have a small error from the actual value and these deviations accumulate over time. This is why in large horizons the predicted value of ARIMA is a straight line. Having a q parameter in an ARIMA model (q represents the MA component), then you can accurately predict q steps into the future. That means your model is only able to predict the next q data points, after that the prediction will converge into a straight line.

One solution for this problem is to train the model as many times as the predictions you want to make. This solution is computationally expensive, because you have to train the model a lot of times, while in the previous solution only one. One less computationally expensive solution is to integrate the forecasted value into the model after each prediction step, which enables real-time error calculation, ensuring the model continuously refines its predictions and improves its accuracy. This approach ensures the integrity of the model and is less expensive in computational resources.

We ran 10 different experiments for the car mobility pattern. This whole experiment took approximately 2-3 minutes to complete, which confirms that this approach is less computationally expensive than the approach where we have to train the model for every forecast (which takes about 10 minutes to finish). We ran experiment for the initial dataset (not stationary so we used the ARIMA model with d value equal to 1). If we wanted to ran experiments for the transformed timeseries of chapter 3 we would need to implement an ARMA model, in which we do not need to difference the data while the data are already stationary.

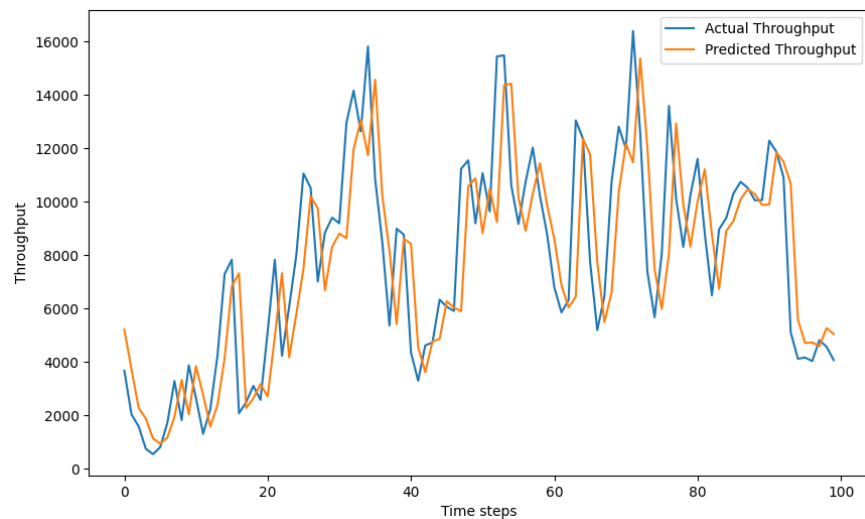


Figure 4.1: ARIMA Model (Initial Timeseries)

The R^2 value of the ARIMA model (Table 4.1 section [4.2.3]) indicates that the model explains approximately 57.81% of the variance in the target variable, which suggests moderate predictive power. While this value is not close to 1 (indicating a perfect fit), it shows that the model captures a significant portion of the variability in the data. The comparison between actual and predicted throughput is illustrated in Figure 4.1. The plot demonstrates that the model captures the overall trend of the data reasonably well. However, there are discrepancies at certain peaks and valleys, indicating areas where the model's predictions could be improved.

4.2.2 Prediction Using Neural Networks

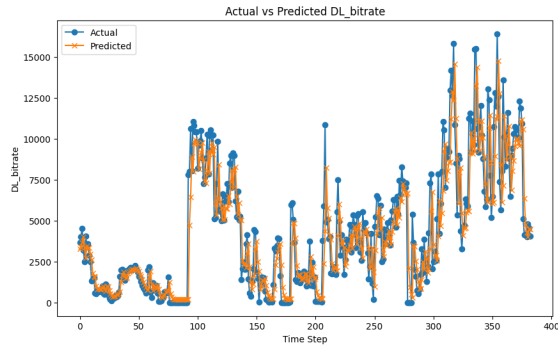
In this section, we assess the performance of three neural network architectures (LSTM, GRU, and FCNN), alongside two non-neural network models (Gradient Boosting Descent and Random Forest), for predicting throughput (DL_bitrate) in the 4G dataset. The models' effectiveness was evaluated using three key metrics: R-squared, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE). Additionally, a visual comparison of actual versus predicted values from the test dataset provided insights into model accuracy. The goal is to identify the most suitable models for capturing the non-linear dynamics of the dataset.

Preprocessing

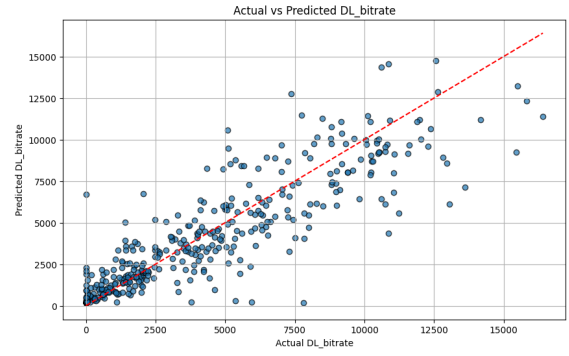
The initial step in developing time-series prediction models is preprocessing, which is crucial to enhance the performance of neural networks. In Chapter 3, outliers were identified and removed to ensure data quality. Following this, the dataset was normalized using a Min-Max Scaler, rescaling all features to values between zero and one. This step ensures that no feature dominates the learning process due to its scale. Finally, the dataset was divided into training and testing subsets in an 80:20 ratio, balancing the need for sufficient training data while retaining a representative sample for evaluation.

Visual Inspection

In the following figures the results of the following neural networks and non-neural networks are presented: Long Short Term Memory (Figure 4.2), Gated Recurrent Unit (Figure 4.3), Fully Connected Neural Network (Figure 4.4), Gradient Boosting Descent (Figure 4.5) and Random Forest (Figure 4.6)

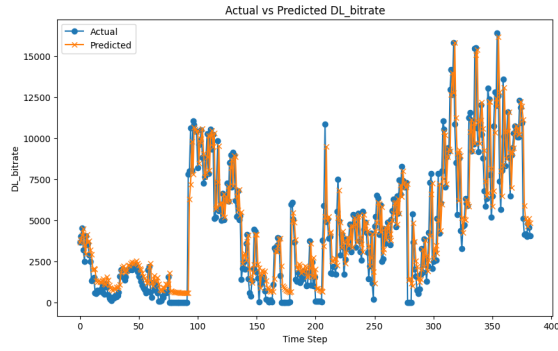


(a) LSTM Prediction vs Real

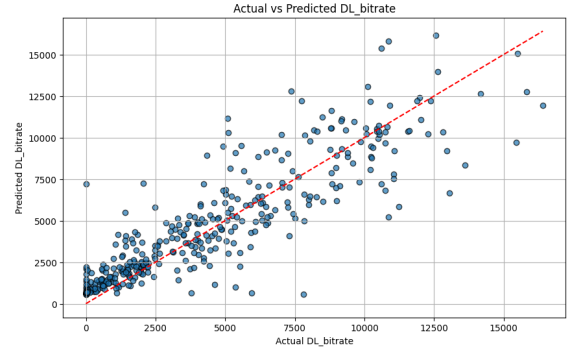


(b) LSTM Parity Plot

Figure 4.2: Results of LSTM

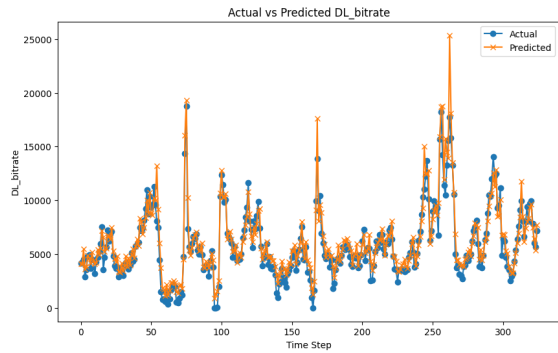


(a) GRU Prediction vs Real

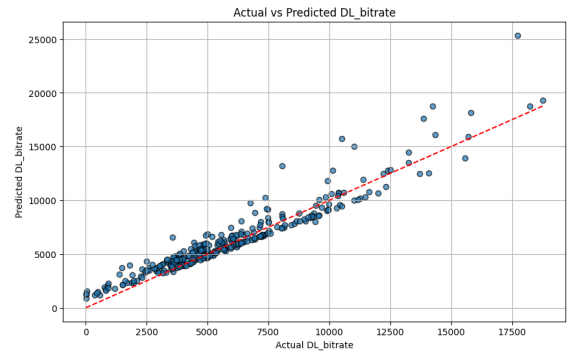


(b) GRU Parity Plot

Figure 4.3: Results of GRU

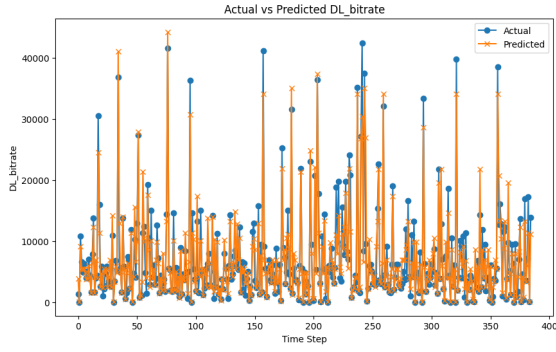


(a) FCNN Prediction vs Real

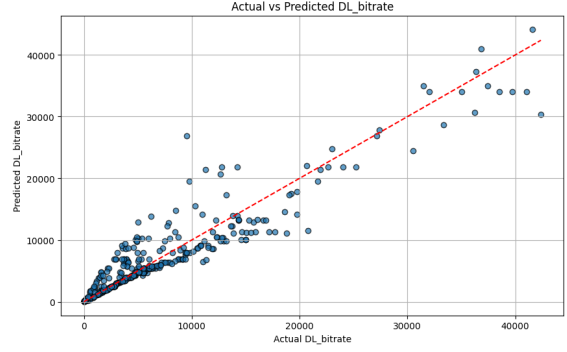


(b) FCNN Parity Plot

Figure 4.4: Results of FCNN

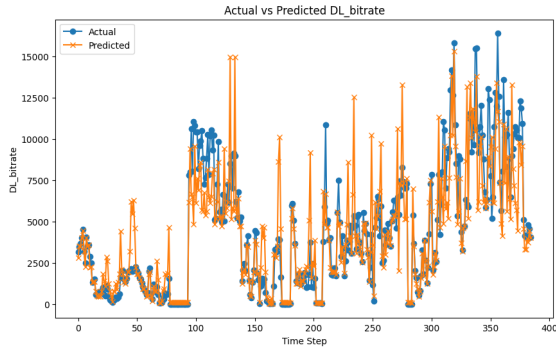


(a) GBD Prediction vs Real

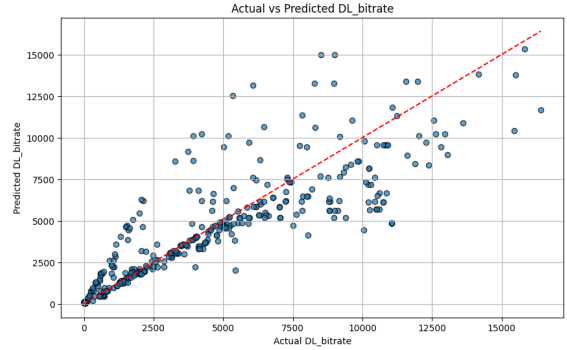


(b) GBD Parity Plot

Figure 4.5: Results of GBD



(a) Random Forrest Prediction vs Real



(b) Random Forrest Parity Plot

Figure 4.6: Results of Random Forrest

4.2.3 Results

From the evaluation, Gradient Boosting (GBD) and Fully Connected Neural Networks (FCNN) emerged as the most effective models, with R^2 scores of 0.846 and 0.841, respectively. Their ability to model non-linear relationships and adapt to the complex dynamics of 4G network throughput explains their superior performance. On the other hand, traditional models like ARIMA were less effective due to their linear nature, while LSTM and GRU, although designed for sequential data, did not outperform FCNN and GBD due to their potential sensitivity to hyperparameters and data size.

The strong correlation between the selected input features and the target variable likely contributed to the excellent performance of the Fully Connected Neural Network (FCNN), enabling it to effectively learn and predict 4G throughput. Gradient Boosting (GBD) demon-

strated excellent performance in predicting 4G throughput due to its ability to model complex non-linear relationships, automatically select important features, and robustly handle smaller datasets. The limited duration of our traces, averaging 15 minutes, results in a relatively small dataset, which restricts the ability of LSTM and GRU models to fully reach their potential.

These results suggest that non-linear machine learning models are better suited for predicting throughput in 4G networks, where complex interactions and variability are prevalent. Further enhancements, such as hyperparameter optimization and larger datasets, could improve model accuracy even further.

Type of model	R^2	MAE	RMSE
ARIMA	0.5781	1973.82	2695.16
LSTM	0.663	1890.64	2896.13
GRU	0.671	2057.82	3015.16
FCNN	0.841	1443	2075.19
GBD	0.846	1592.35	3005.56
Random Forrest	0.614	1472	2641.90

Table 4.1: Mean Evaluation Metrics for 20 Experiments

4.2.4 Key Takeaways

The results of this experiment demonstrate that non-linear machine learning models, such as Gradient Boosting and Fully Connected Neural Networks (FCNN), are highly effective in predicting throughput in 4G networks. These models excel due to their ability to handle the variability and complex interactions inherent in the data. In contrast, sequential models like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) exhibit limitations that stem primarily from the relatively small dataset size, which restricts their ability to fully leverage temporal dependencies.

From a practical perspective, the findings suggest that focusing on non-linear models and carefully optimizing their hyperparameters can lead to significant performance improvements. Furthermore, while the sequential models showed promise, their performance could likely be enhanced with access to larger datasets and more advanced architectures. This highlights the need for further research aimed at improving sequential modeling approaches, particularly in scenarios where temporal patterns play a critical role.

4.3 Parallel Prediction of Time Series

In this experiment, the primary objective was to simultaneously predict GPU, CPU, and Memory usage within the Philly cluster, leveraging state-of-the-art deep learning methodologies. To achieve this, I utilized and adapted the implementation from the paper "DeepCog: Cognitive Network Management in Sliced 5G Networks with Deep Learning" [13]. This paper presented a novel architecture designed for resource forecasting in 5G network slices, emphasizing the trade-offs between overprovisioning and underprovisioning of resources.

Using the provided code as a foundation, modifications were made to adapt the model to the Philly cluster dataset, enabling it to predict multiple resource types concurrently. The experiment aimed to evaluate the effectiveness of the deep learning architecture in managing and predicting resource usage in a large-scale computing environment, specifically GPU, CPU, and Memory demands.

The experiment involved preprocessing the dataset into an appropriate tensor format compatible with the neural network, training the model on historical data to capture temporal and spatial dependencies, and evaluating its predictive accuracy. The loss function was tailored to minimize prediction errors, ensuring a balance between resource allocation and system efficiency. The results demonstrated the feasibility of applying such advanced predictive models in real-world cluster management scenarios, offering insights into efficient resource orchestration.

4.3.1 DeepCog

Modern networking systems, such as 5G networks or cloud computing clusters, experience fluctuating resource demands due to diverse and dynamic workloads. Traditional reactive methods for resource allocation often fail to meet performance and cost requirements in such environments. DeepCog is a custom deep neural network designed for anticipatory resource management in networking systems. The architecture of this model was inspired by image processing architectures (more specifically 3D-CNNs), but fully connected layers were added as well.

The basic components of this model are four: Encoder, decoder, a customized loss function and the output layer. The encoder processes historical spatiotemporal data and extracts meaningful features. That is why 3D-CNNs are implemented. Moreover, the encoder transforms raw input tensors into a compact, high-dimensional representation, effectively compressing the data while preserving essential patterns.

The decoder takes the high-dimensional representation generated by the encoder and maps it to predictions for future resource demands. It consists of fully connected layers (dense layers) that refine the compressed information into actionable forecasts. The decoder is responsible for outputting predictions for specific resource requirements, such as bandwidth, computational capacity, or system usage metrics. To simplify the implementation, we opted

not to design a custom loss function. Instead, we used the widely recognized Mean Squared Error (MSE), a standard loss function for regression tasks, which minimizes the average squared difference between predicted and actual values. The final layer of the model produces forecasts for resource requirements. In the context of 5G network slicing, this includes metrics such as network bandwidth and computational resources for each slice. In our case this will be the CPU, GPU and Memory utilization.

DeepCog serves as the inspiration and foundation for the architecture used in this experiment. Its encoder-decoder design enables efficient extraction of spatiotemporal features and generation of accurate forecasts, which aligns with the requirements of this study. In this experiment, DeepCog’s principles were adapted to predict GPU, CPU, and memory usage in a cloud computing cluster. The next subsection details the adaptations made to DeepCog and its application in this context.

4.3.2 Experiments

In the experiments that were performed we also implemented a simple Long Short-Term Memory (LSTM) model to compare with our model. We created two different scenarios, where in the first scenario we attempted to predict the behavior of 8 GPUs simultaneously, while in the second scenario we tried to predict 2 GPUs simultaneously with the CPU and Memory utilization. In the first scenario all the timeseries that were used have a pretty strong correlation between them. In the second scenario this is not the case. The GPUs timeseries have strong correlation between themselves, but do not exhibit correlation with the CPU or memory utilization. This was done to observe how our model will react in different correlation scenarios (low or high).

Preprocessing

As outlined in the previous subsection, several preprocessing steps and adjustments were necessary to adapt the model for the Philly dataset. In both scenarios the preprocessing step is pretty much the same with little differences. First, the raw data had to be transformed into a format compatible with the model, specifically into tensors. To capture the spatiotemporal dependencies that the dataset exhibits a 3D-CNN architecture was chosen. To predict the behavior of 8 GPUs simultaneously, which exhibit high correlation, the input data needs to be transformed into a tensor format. One straightforward approach would be to structure the data as a vector with a 1x8 dimension. While this method is functional, it is not optimal for capturing spatial or temporal relationships. Drawing inspiration from the DeepCog methodology, where a 4x4 tensor was used for 16 time series, we propose arranging our 8 time series into a 2x4 grid in the first scenario where the predicted timeseries are 8. In the second scenario where the number of time series predicted is 4 we rearranged the data into a 2x2 grid. This arrangement enhances spatial representation and is well-suited for leveraging

3D-CNN architectures.

Prior to training, the position of each time series within the grid must be carefully determined. To maximize the effectiveness of the 3D-CNN, time series with high correlation should be positioned closer together within the grid. This proximity facilitates better learning of interdependencies and improves model performance. In addition, we examined and fine-tuned hyperparameters such as learning rate and dropout rate to identify the optimal ones for our dataset. By experimenting with various hyperparameters and evaluating their impact on model performance we chose the hyperparameters that gave us the best results.

Next, it was essential to determine the appropriate loss function. For this implementation, we opted not to design a custom loss function and instead utilized the widely used Mean Squared Error (MSE). This decision was made to maintain simplicity, as MSE is a standard and effective choice for regression tasks. However, in future work, exploring or designing a more specialized loss function tailored to this application could further optimize performance.

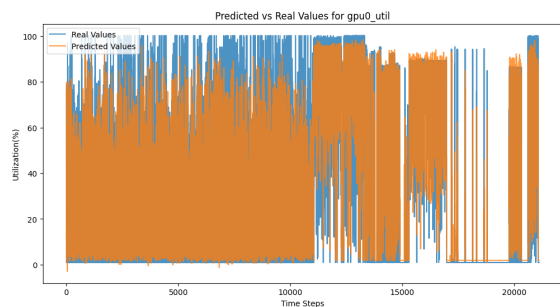
Finally, the dataset was divided into training and testing sets using the conventional 80:20 split to ensure robust evaluation. The model's performance was assessed using well-established metrics: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and the R^2 score. This process allowed us to effectively evaluate the model's ability to predict resource utilization in the Philly dataset.

4.3.3 Results

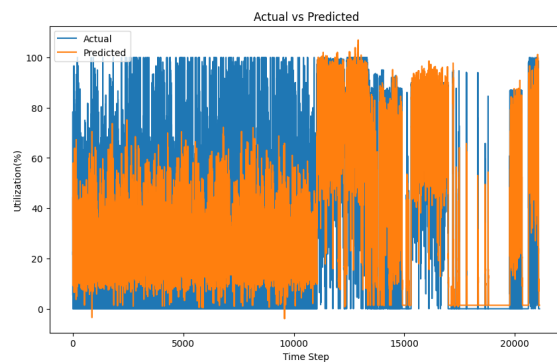
In this subsection the results of the 2 above experiments are going to be presented.

Scenario A

In the first experiment we attempted to predict simultaneously the GPU utilization of the 8 GPUs in the same machine. In Figures 4.7-4.10 we present the result of a simple LSTM model versus the model that was inspired from DeepCog. Additionally, in Table 4.2 the mean evaluation metrics for the both models and for each time series are presented by running 5 experiments in 5 different machines with 8 GPUs. Note that i presented the results of half of the GPUs as the results are very similar and it would be space-consuming to present them all.

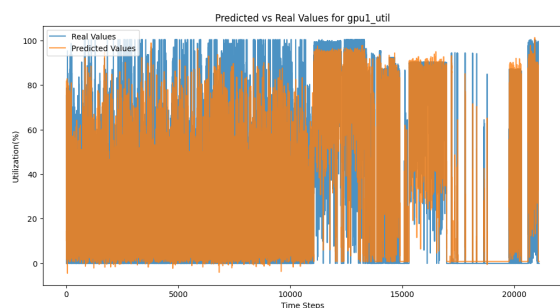


(a) DeepCog

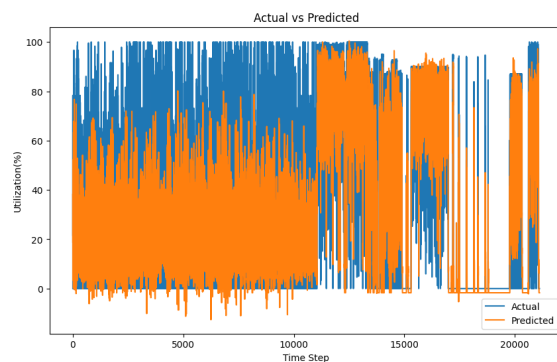


(b) LSTM

Figure 4.7: LSTM vs Deepcog GPU0

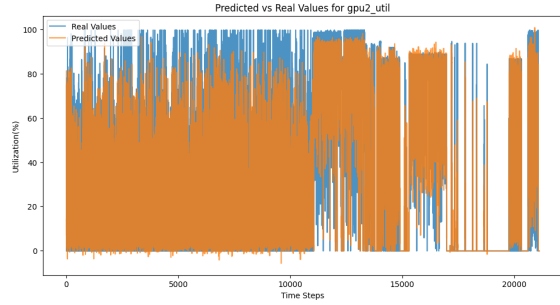


(a) Deepcog

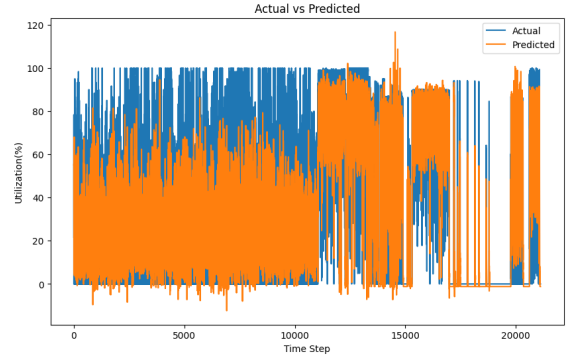


(b) LSTM

Figure 4.8: LSTM vs Deepcog GPU1

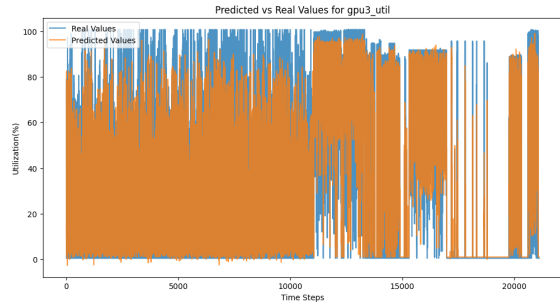


(a) Deepcog

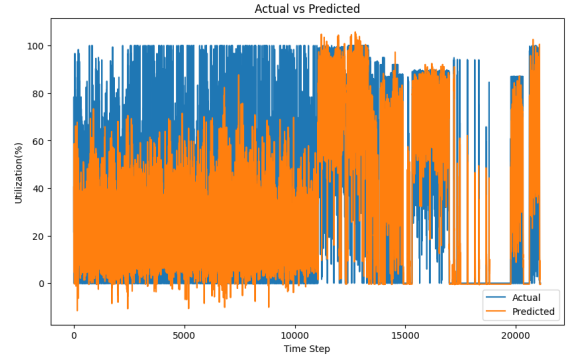


(b) LSTM

Figure 4.9: LSTM vs Deepcog GPU2



(a) Deepcog



(b) LSTM

Figure 4.10: LSTM vs Deepcog GPU3

Timeseries	DeepCog Model			LSTM Model		
	R^2	MAE	RMSE	R^2	MAE	RMSE
gpu_0	0.7305	12.73	18.90	0.6533	16.26	22.63
gpu_1	0.7125	13.91	20.33	0.6508	15.62	22.86
gpu_2	0.7194	13.00	19.25	0.6561	15.40	22.59
gpu_3	0.7293	12.85	18.95	0.6605	15.34	22.56
gpu_4	0.7086	12.37	18.53	0.6554	15.41	22.79
gpu_5	0.7058	13.70	20.22	0.6487	15.41	22.91
gpu_6	0.6995	13.65	20.03	0.6351	15.65	23.30
gpu_7	0.7013	14.59	21.15	0.6061	16.65	23.76

Table 4.2: Comparison of Mean Errors between DeepCog and LSTM Models

The results of this experiment demonstrate that the new model consistently outperforms the simple LSTM model across both visual inspection and all evaluation metrics. Upon visual inspection the predictions of the model ,which predicts the eight GPUs utilization simultaneously, align more closely with the actual data than a simple LSTM model. From Tables 4.2 and 4.3 we can see that the Deepcog model also performs better consistently in all evaluation metrics that were used.

The R^2 score for GPUs hovers around 0.7, which indicates a moderately acceptable level of accuracy. I think that the model does not perform better because of the high variance and unpredictable spikes of the GPU's utilization. However, despite the variance observed in the results, the model's performance can be considered satisfactory. Further analysis and refinements could potentially enhance its accuracy.

One significant improvement could involve designing a custom loss function tailored specifically to this model's requirements. While this approach holds the potential to substantially boost performance, it was not implemented in this study due to the time-intensive nature of developing and fine-tuning such a loss function. Future work could explore this avenue to optimize the model's capabilities further.

Scenario B

In the second experiment we tried to predict 2 GPUs along with the CPU and memory utilization to observe how our model works with less unpredictable timeseries, even though there correlation might be weaker. The following figures are some representative figures for every time series that we attempted to predict. Additionally, in Table 4.3 the mean evaluation metrics for each model are presented:

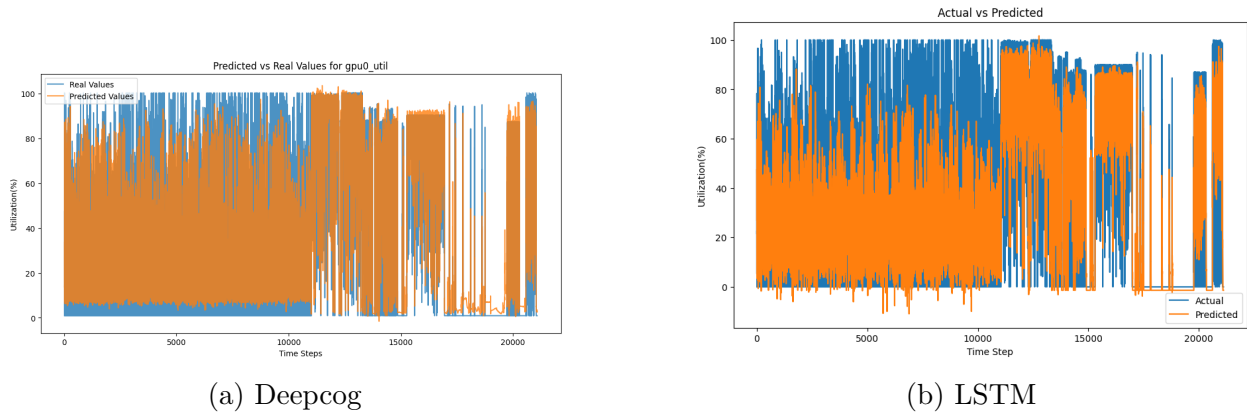
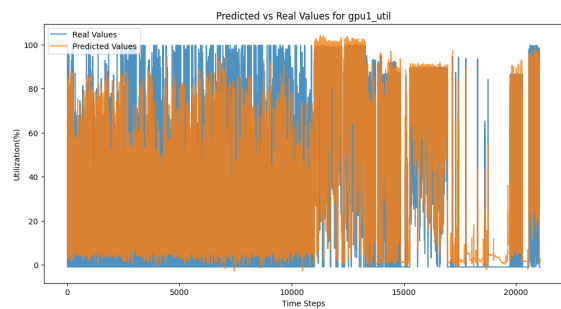
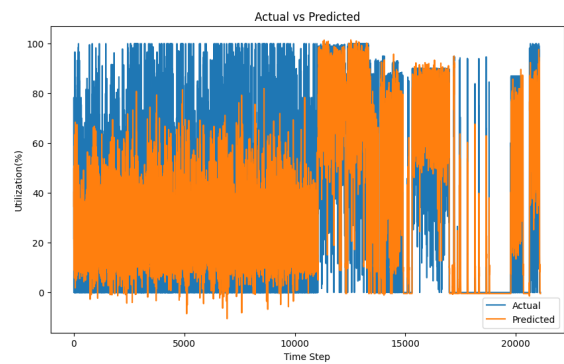


Figure 4.11: LSTM vs Deepcog GPU0

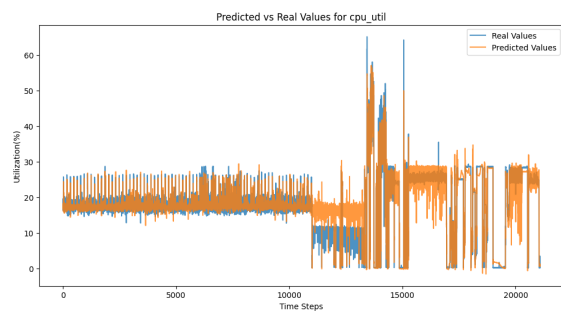


(a) Deepcog

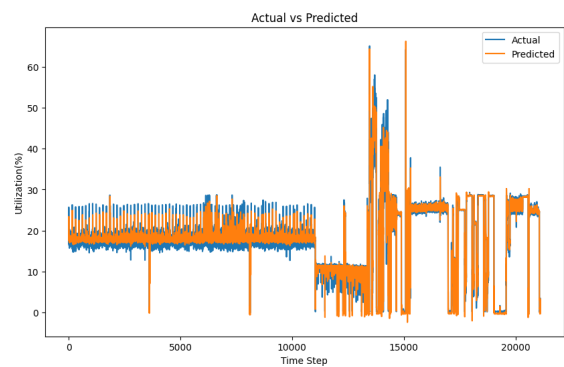


(b) LSTM

Figure 4.12: LSTM vs Deepcog GPU1



(a) Deepcog



(b) LSTM

Figure 4.13: LSTM vs Deepcog CPU

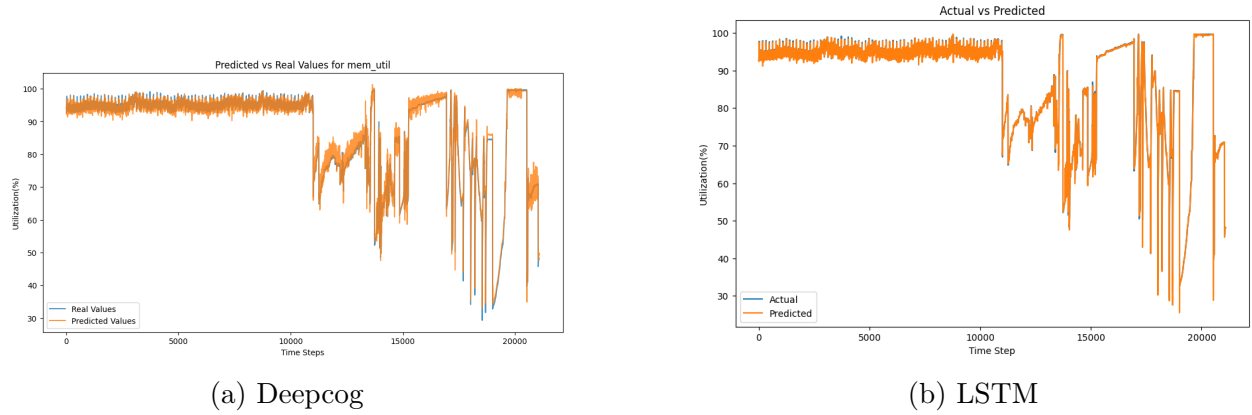


Figure 4.14: LSTM vs DeepCog Memory

Timeseries	DeepCog Model			LSTM Model		
	R^2	MAE	RMSE	R^2	MAE	RMSE
gpu_0	0.6346	15.68	23.26	0.6612	15.71	22.37
gpu_1	0.6317	15.95	23.47	0.6450	16.15	23.05
cpu	0.8558	1.95	2.98	0.9220	1.15	2.20
$memory$	0.9826	1.20	1.77	0.9934	0.44	1.09

Table 4.3: Comparison of Experiment B Mean Errors DeepCog and LSTM Models

The second experiment provides insightful findings regarding the performance of the DeepCog framework and the simple LSTM model on different types of time series. Due to the strong correlation between the GPUs, the DeepCog model demonstrates superior performance for gpu_0 and gpu_1 , as it effectively captures the spatial dependencies and interrelationships between these highly correlated time series. This highlights the strength of the 2x2 grid tensor representation and the 3D-CNN architecture in leveraging spatial information.

However, for time series with low correlation, such as cpu and $memory$, the simple LSTM outperforms the DeepCog framework. Not only does the LSTM perform better for these time series, but it does so exceptionally well, achieving near-perfect results as indicated by significantly lower MAE and RMSE values. This suggests that the LSTM model, which focuses solely on temporal patterns without spatial interactions, is particularly well-suited for time series that are more independent and less influenced by correlations with others. The DeepCog framework also delivers good predictions for cpu and $memory$, its performance does not match the level achieved by the simple LSTM for these time series, and because of its complexity we would definitely prefer a simple LSTM model when predicting low correlated timeseries (CPU and memory). This result underscores that while DeepCog excels

in scenarios where spatial correlations are critical, the simple LSTM can be more effective for independent time series.

These findings highlight the importance of selecting models based on the underlying characteristics of the data. For highly correlated time series, the DeepCog framework is preferable, whereas for more independent series, the simple LSTM demonstrates exceptional capability.

4.3.4 Key Takeaways

In this subsection a conclusion of the results of the second experiment is going to be presented. In the first scenario where high correlation is traced between the timeseries the DeepCog model has the ability to capture spatial and temporal dependencies and outperforms the LSTM model in predicting GPU utilization. Its use of the 3D-CNN architecture is critical for handling high correlations between GPUs. While the DeepCog model demonstrates clear advantages in metrics and visual inspection both models are limited by the high variance and unpredictable nature of the data. Future enhancements, such as custom loss functions tailored to GPU utilization spikes, could significantly improve predictive performance.

The results of the second scenario provide two critical insights. First and foremost, it is observed that for highly correlated time series, such as GPUs, the DeepCog model’s ability to capture spatial dependencies makes it a strong candidate for prediction tasks. The 2x2 tensor grid and 3D-CNN architecture contribute to its effectiveness. On the other hand, for less correlated or independent time series, such as CPU and memory utilization, the LSTM model demonstrates superior performance. Its simpler architecture is more effective in capturing temporal patterns without the need for spatial modeling, making it a more efficient choice in these cases.

These findings emphasize the importance of aligning model architecture with the underlying characteristics of the dataset. The DeepCog framework is ideal for tasks requiring spatial correlation, while the LSTM model is better suited for independent or weakly correlated time series. Future work should explore hybrid approaches that dynamically adapt to the dependencies present in the data.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This thesis underscored the pivotal role of datasets in shaping AI-driven networking algorithms by examining their collection, preprocessing, analysis, and application. Utilizing datasets from diverse domains, including cloud computing (Helios and Philly), 4G networks, and YouTube, this work addressed significant challenges such as non-stationary behavior, heavy-tailed data distributions, and incomplete data. These issues are critical when preparing real-world data for effective AI deployment.

Through the use of statistical and time-series techniques, such as rolling mean analysis, autocorrelation functions, and the Augmented Dickey-Fuller test, this research transformed unprocessed data into structured, actionable formats. Our analysis demonstrated the value of statistical and time-series techniques in extracting meaningful patterns and trends from complex datasets. Predictive models like ARIMA and neural networks were then applied, showcasing the potential of these datasets to predict key performance indicators, optimize resource utilization, and improve overall system performance.

The outcomes of this study highlight the necessity of tailored preprocessing methods and thorough statistical analysis to harness the potential of real-world datasets in networking contexts. Furthermore, the findings lay the groundwork for developing advanced AI-driven strategies in areas such as workload scheduling, efficient resource management, and network optimization.

5.2 Future Work

While this research makes substantial contributions to AI-driven networking, there remain numerous directions for further investigation:

- **Expanding Dataset Scope:** Future studies could explore datasets from additional

domains, such as 5G and edge computing, to address emerging challenges and leverage opportunities presented by advanced network infrastructures.

- **Further Analysis of the train mobility pattern.** In the 4G Dataset we observed a very unpredictable behavior in the network in the train mobility pattern. Further analysis can be performed in those traces to uncover why is the performance so unpredictable and what can be done to improve the performance of the network.
- **Customized Loss Functions for DeepCog:** In the second experiment, the DeepCog framework was used to analyze and predict resource usage. A promising area for improvement is the creation of a custom loss function, specifically tailored to the characteristics of the dataset. Such a function could align training goals more closely with the unique data patterns, improving predictive accuracy. However, this effort requires extensive research and experimentation, which was beyond the scope of this thesis. Nevertheless, it offers a compelling opportunity for future exploration.
- **Adopting Advanced AI Models:** Incorporating advanced AI methodologies, including transformers and reinforcement learning, could further improve predictive accuracy and adaptability. These models are particularly suited for complex scenarios like dynamic resource management and multi-agent systems, which remain challenging in AI-driven networking.

By addressing these areas, future research can extend the scope of this work, fostering innovation in AI-powered networking and enabling the development of intelligent, scalable, and reliable systems for increasingly interconnected environments.

Bibliography

- [1] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, “Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads,” presented at the 2019 USENIX Annual Technical Conference, Renton, WA, July 2019.
- [2] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, “Characterization and Prediction of Deep Learning Workloads in Large-Scale GPU Datacenters,” presented at *arXiv preprint arXiv:2009.01313v2*, Sep. 2021.
- [3] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, “Beyond Throughput: A 4G LTE Dataset with Channel and Context Metrics,” presented at the ACM Multimedia Systems Conference (MMSys 2018), Amsterdam, The Netherlands, June 2018.
- [4] D. Raca, D. Leahy, C. J. Sreenan, and J. J. Quinlan, “Beyond Throughput, The Next Generation: A 5G Dataset with Channel and Context Metrics,” in *ACM Multimedia Systems Conference (MMSys)*, Istanbul, Turkey, June 8-11, 2020.
- [5] C. Chatfield, *Time-Series Forecasting*, Chapman and Hall/CRC, 2019.
- [6] R. Bhaskar, S. R. Deepu, and B. S. Shylaja, “Dynamic Allocation Method for Efficient Load Balancing in Virtual Machines for Cloud Computing Environment,” in *Advanced Computing: An International Journal (ACIJ)*, vol. 3, no. 5, pp. 53-62, September 2012.
- [7] A. Al-Thaedan, Z. Shakir, A. Y. Mjhood, R. Alsabah, A. Al-Sabbagh, M. Salah, and J. Zec, “Downlink Throughput Prediction Using Machine Learning Models on 4G-LTE Networks,” *Bharati Vidyapeeth’s Institute of Computer Applications and Management*, published online: July 1, 2023.
- [8] M. Nazir, D. Kumar, L. A. Thebo, and S. N. A. Jaffari, “Critical Analysis of High Performance Computing (HPC),” in *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 16, no. 9, September 2018.
- [9] S. Iyer, P. Jain, and S. Pandey, “Efficient Resource Allocation in Cloud Computing Using Machine Learning Techniques,” presented at the IEEE International Conference on Cloud Computing, 2020.

- [10] R. Werner, D. Valev, and D. Danov, “The Pearson’s Correlation - A Measure for the Linear Relationships Between Time Series?,” in *Fundamental Space Research*, vol. 92, pp. 92-94, 2009.
- [11] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, “Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis,” presented at the *ACM Symposium on Cloud Computing (SoCC)*, 2012.
- [12] J. Dean and L. A. Barroso, “The Tail at Scale,” published in *Communications of the ACM*, vol. 56, no. 2, pp. 74-80, 2013.
- [13] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, “DeepCog: Cognitive Network Management in Sliced 5G Networks with Deep Learning,” in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 280-288, 2019.
- [14] R. Kumar, K. Sengupta, and S. Biswas, “A Comparative Analysis of Machine Learning Algorithms for Time Series Prediction,” published in *International Journal of Advanced Computer Science*, 2019.
- [15] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” presented at *arXiv preprint arXiv:1301.3781*, 2013.
- [16] A. Graves, A. R. Mohamed, and G. Hinton, “Speech Recognition with Deep Recurrent Neural Networks,” presented at the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2013.
- [17] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 2018.
- [18] Teerapittayanon, S., McDanel, B., Kung, H. T., “Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices,” in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, 2017.
- [19] M. C. Willems et al., “A Hybrid ARIMA-ANN Model for Network Throughput Prediction,” in *Journal of Computational Science*, vol. 45, 2020.
- [20] Manuca, R., Savit, R., “Stationarity and Nonstationarity in Time Series Analysis,” in *Physica D: Nonlinear Phenomena*, vol. 99, pp. 134-161, 1996.
- [21] Ryan, O., Haslbeck, J. M. B., Waldorp, L. J., “Non-Stationarity in Time-Series Analysis: Modeling Stochastic and Deterministic Trends,” in *Multivariate Behavioral Research*, 2025, DOI: 10.1080/00273171.2024.2436413.

- [22] Giannakas, T., Spyropoulos, T., Smid, O., “Fast and Accurate Edge Resource Scaling for 5G/6G Networks with Distributed Deep Neural Networks,” in *Proceedings of the 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Thessaloniki, Greece, 2022.
- [23] Hristopulos, D. T., Petrakis, M. P., Kaniadakis, G., “Finite-Size Effects on Return Interval Distributions for Weakest-Link-Scaling Systems,” in *arXiv preprint*, 2018, arXiv:1811.02194.