Doctoral Dissertation
博士論文

Prediction Mechanism Using Adaptively Linked Composite Ensemble Networks to Enhance Robustness and Its Application in Smart Cities

(堅牢性を高めるために適応連結する複合アンサンブルネットワークを用いた予測メカニズムとスマートシティへの応用)

March 2025
令和7年3月

The Graduate School of Sciences and Technology for Innovation,
Yamaguchi University
山口大学大学院創成科学研究科
Division of Systems and Design Engineering
システム・デザイン工学系専攻

Mohd Hafizuddin Bin Kamilin

Abstract

Machine learning is often suggested to help with the processing of huge and intricate datasets collected by sensors in smart cities, which are challenging to process or compute using conventional approaches. In addition to improving the efficiency of urban operations, it also enhances the quality of life for the residents and provides actionable insight to help with decision-making. Machine learning has various applications in smart cities, such as forecasting electricity loads to balance supply and demand.

Most of the proposed machine learning techniques for forecasting were created with the assumption that the input variables are clean with consistent data distribution and running on a stable system. However, the accuracy is compromised by problems such as missing values due to packet loss from the distributed denial-of-service attacks, concept drift that invalidates predictions due to changes in data distribution caused by seasonal or trend changes, small perturbations introduced by adversarial attacks that fool the machine learning into making erroneous predictions, and system instability that could bring down the servers or Internet of Things devices that are hosting the forecasting models.

While a unified approach has been suggested, it is limited to integrating trivial solutions, as the linear operation required to correct the input data tends to accumulate more errors and compatibility issues between the solutions. This thesis proposes a novel approach called Adaptively Linked Composite Ensemble Networks to solve this problem. To link multiple solutions into one, Composite Ensemble Networks classify solutions to problems such as missing values, adversarial attacks, and concept drift in terms of correction and enhanced forecast. Then, it sequentially links the solutions for correction and solutions for enhanced prediction in parallel based on the priority of the solutions. By linking the correction solutions for a specific problem to the improved prediction solutions, the system can effectively identify the errors from fixed missing values, adversarial attacks, and changes in trends, which helps to continually improve the prediction accuracy.

The thesis consists of five chapters.

Chapter 1 describes the optimization of essential service operations in smart cities and protection against cyberattacks. It then indicated the need to solve these problems in

a unified way and stated the purpose of this dissertation. To clarify the position of the dissertation, the previous study similar to the unified solutions was compared, and the limitations of why it is limited to simple solutions due to the errors accumulation and compatibility issues are explained. Then, the countermeasures taken to solve these issues in this thesis are defined.

Chapter 2 introduces the previous research done to unify the missing values, adversarial attacks, concept drift, and single point of failure solutions. Additionally, the shortcomings and limitations of why previous research for a unified solution are constrained to the sequential corrections and forecasting improvement are discussed.

Chapter 3 presented the implementation of Adaptively Linked Composite Ensemble Networks to improve the electric load forecast, in addition to anomaly detection, rules for linking the appropriate solutions, and system hardening against single point of failure. Furthermore, the hyperparameters to initialize and control the operations were given.

Chapter 4 demonstrated the proposed ensemble learning network implementation and application for forecasting the electricity load in New York State and its resiliency against missing values, adversarial attacks, and concept drift. Then, the accuracy was compared with the previous unified method in Chapter 2, highlighting its weaknesses.

Finally, Chapter 5 summarized the works done in this thesis.

These results highlight the effectiveness of Adaptively Linked Composite Ensemble Networks to solve missing values, adversarial attacks, and concept drift. In addition, the implementation shows its robustness against the single point of failure by distributing the essential components with redundancy. As society becomes deeply integrated with cyberspace, a unified solution is not only necessary to improve essential services but also to protect them against cyberattacks.

概要

スマートシティのセンサーで収集される大規模で複雑なデータは、従来の方法での処理が難しく、機械学習が活用されている。機械学習により、都市運営の効率を上げ、住民の生活の質を向上させ、データに基づく意思決定を支援できる。応用例として、電力消費量の予測により電力負荷のバランスを改善し、電力配布ネットワークの安定性が維持できる。

これまでに提案された機械学習モデルは、品質に優れ、傾向が変わらない入力データと安定したシステムを前提に開発されてきた。しかし、分散型サービス拒否攻撃による入力データの欠損値、季節やトレンドの変化でデータ分布が変わったことで予測が無効になる概念ドリフト、敵対的攻撃で予測を誤らせるために導入された小さな撹乱、サーバの不安定性で予測不能になる等の問題によって機械学習の予測精度が低下してしまう。

先行研究では、提案された統一的な解決策が順次処理に制約され、外乱の訂正誤差が蓄積してしまうという問題がある。それを解決するために本学位論文では、複数の解決策を統一するために、適応連結する複合アンサンブルネットワークを提案した。

複合アンサンブルネットワークは、欠損値、敵対的攻撃、概念ドリフトなどの問題に対する解決策を、訂正および予測の強化という観点で分類する。複数の解決策を連結するために、解決策の優先順位に基づき、訂正用の解決策を順次連結し、予測を強化する解決策を並列に連結した。また、特定の問題に対する訂正用の解決策の出力を、予測を強化する解決策の入力に接続することで、予測モデルは訂正された欠損値、敵対的攻撃、そして概念ドリフトを把握し、継続的に予測精度を改善できる。サーバの不安定性を防ぐために、複合アンサンブルネットワークの解決策を部品化して分散ネットワークおよびバックアップに応用した。一部のノードがダウンした場合、投票メカニズムで選ばれたノードリーダーがノードの連結を再調整し、ダウンした解決策をバックアップデータから他のノードに復元することができる。

学位論文は 5 つの章から構成される。

第 1 章では、スマートシティにおける基本的なサービス運用の最適化と、サイバー攻撃からの防御について説明した。そして、これらの問題を統一的な方法で解決する必要性を示し、本学位論文の目的を示した。続いて、学位論文の位置づけを明確にするため、これまでの研究で提案された統一的な解決策に似た例を示し、解決策の相性で先行研究が簡単

な解決策に限られていたことを示した。そのうえで、学位論文に提案された複合アンサンブルネットワークはどのようにその弱点を解決することを示した。

第 2 章では、欠損値、敵対的攻撃、概念ドリフト、単一障害点に対する解決策を統一するための先行研究を紹介した。また、先行研究の統一的な解決策が順次的な訂正や予測の改善に制約された理由について示した。

第 3 章では、提案された適応連結する複合アンサンブルネットワークが、スマートシティ管理システムから取得したデータの異常検知、適切な解決策の連結、および予測の流れを示した。さらに、システム管理者から得たハイパーパラメータが機械学習モデルの設置と検知の流れにどのように影響するかを示した。続いて、分散システムアーキテクチャと解決策連結の条件のうえで、欠損値、敵対的攻撃、概念ドリフト、そして単一障害点の解決策を示した。

第 4 章では、ニューヨーク州の電力消費量を予測するために、適応的連結による複合アンサンブルネットワークを応用した。提案手法の有効性を示すために、実際の概念ドリフトのデータやランダムな欠損、および敵対的攻撃のシミュレーションを用いて、先行研究の第 2 章の結果と比較した。そして、低い電力消費量の予測精度により、経済的にどのような悪影響が与えられるかを議論した。

最後に、第 5 章では、学位論文の全体をまとめた。

以上の成果により、適応連結する複合アンサンブルネットワークは欠損値、敵対的攻撃、そして概念ドリフトを統一的に解決できた。さらに、分散システムアーキテクチャとデータと機械学習モデルデータのバックアップにより単一障害点の問題も解決できた。統一された解決策は、都市機能の根幹を支えるサービスの品質向上と、サイバー攻撃に対する強靭なシステム構築を両立させ、市民が安心して暮らせる安全で持続可能なスマートシティの実現に大きく貢献することが期待される。

Contents

1 Introduction

1.1 Research Background

Society 5.0 is a vision proposed by the Japanese government's Cabinet Office to create a human-centric society that aims to improve economic development and solve social issues by integrating the physical space with cyberspace [1]. The digital transformation utilizes the Internet of Things (IoT) to collect complex data, process it with artificial intelligence (AI), and make data-driven choices, addressing challenges such as aging populations, climate change, urbanization, and resource management. To realize this vision, smart cities act as a forefront for the digital transformation [2], utilizing interconnected digital infrastructures to manage urban resources more efficiently, improve public services, and create sustainable living environments. Additionally, smart city implementation also opens up new market potential in providing smart city technologies in utilities, governance, transportation, and many more, which are estimated to reach 3,733 billion USD in 2030 in global market size [3], as shown in Figure 1.1.

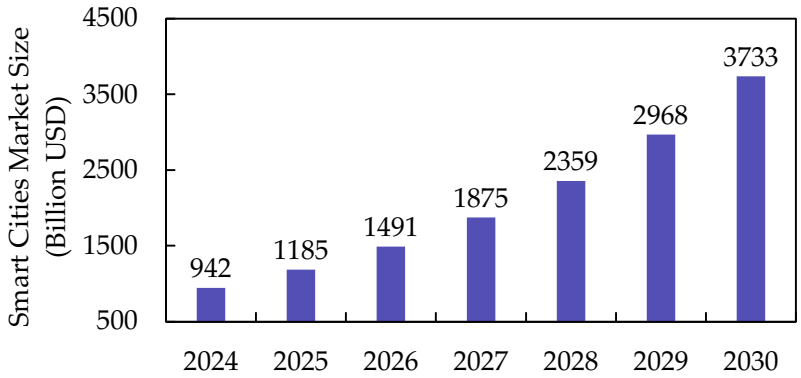


Figure 1.1: Projected global smart cities market size from 2024 to 2030.

One example of digital transformation in smart cities is to balance the electricity power distribution to meet the demand in the New York State [4], ensuring the stability of the electricity delivery network to avoid interruption. The latest electricity load demand from multiple zones was gathered by IoT devices, which were used by the AI-based forecasting model to forecast the electricity load with high accuracy [5], as shown in Figure 1.2. The forecast result assists in allocating electricity power in advance to prevent overloading the

grids [6], enhances the integration of renewable energy [7], and minimizes operational losses during peak hours [8]. These benefits enable the provision of affordable and clean energy, a feat that would be challenging to achieve with a conventional statistical model alone [9], which is facing difficulties processing intricate and multivariate data.

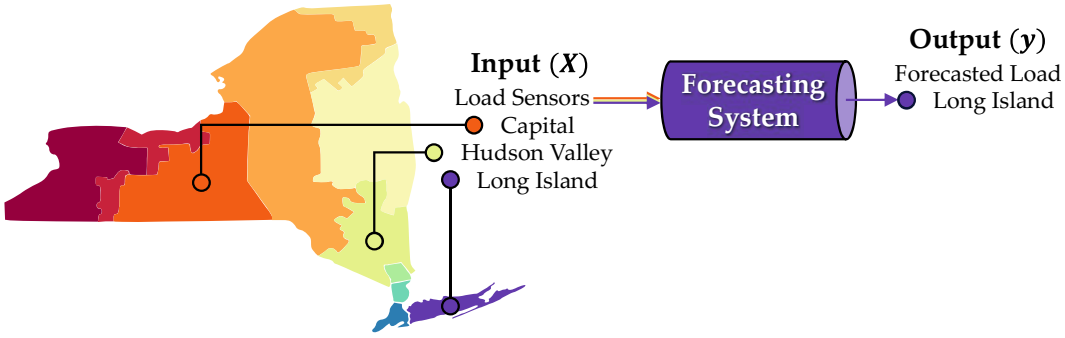


Figure 1.2: Forecasting system applications in smart cities utilize multiple data from multiple sensors to improve the accuracy.

As most of the smart city services, which include electricity production and distribution, take time for the operation adjustment to show their effects, a forecasting model to detect non-optimal operation helps the system operator or automated system management take preemptive measures to optimize the operation, as shown in Figure 1.3.

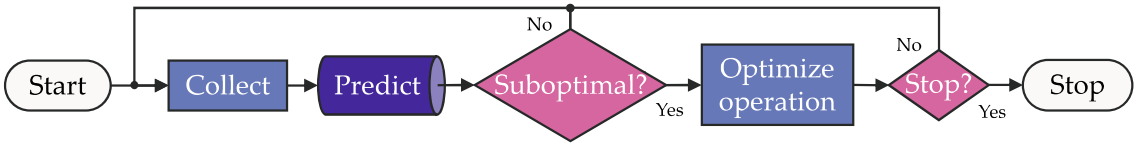


Figure 1.3: System optimization relies on prediction to anticipate suboptimal operation to perform preemptive measures.

Based on the compound annual growth rate (CAGR) shared by Juniper Research, the projected global cost savings from electricity distribution via smart grid technologies will grow from 56 billion USD in 2024 to 278 billion USD in 2030 [10], as shown in Figure 1.4. The CAGR demonstrates the advantage of smart cities in reducing wasteful electricity generation by anticipating the electricity load demands via a forecasting model, which helps the system operator to adjust the electricity generation.

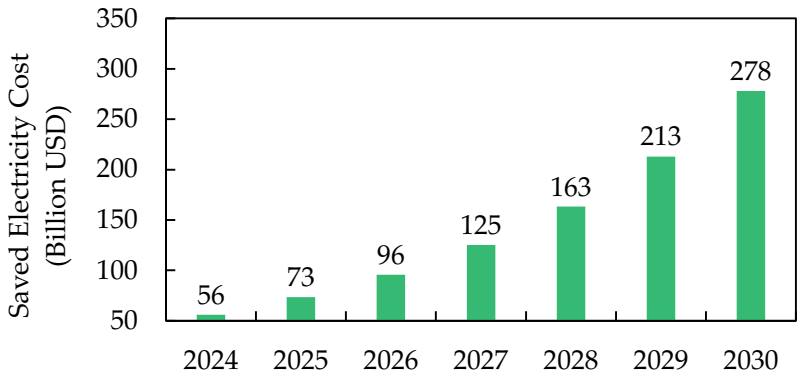


Figure 1.4: Projected global cost saving from electricity distribution from 2024 to 2030.

These examples show the benefits of integrating physical space with cyberspace with regression analysis-based AI. However, most of the proposed AI, especially those based on machine learning (ML), were made with the assumption that the input would be clean with consistent data distributions and that the centralized model architectures would be set up on a server that could run without interruption [11]. Because these conditions are not guaranteed in real-world applications, forecasting accuracy often degrades when dealing with missing values [12], inconsistent data distribution [13], adversarial data [14], and server instability [15]. Moreover, connecting physical space with cyberspace exposes the infrastructure and services in smart cities to cyberattacks [16]. Table 1.1 summarized the issues, causes, and effects that degraded the forecasting accuracy.

Table 1.1: Forecasting issues encountered in the real-world application.

Issue	**Cause**	**Effect**	
		Input Noise	*Vulnerability*
Missing Values	Sensor issues, cyberattacks	✓	
Concept Drift	Season or trend changes	✓	
Adversarial Attacks	Cyberattacks	✓	
Single Point of Failure	Server failure, cyberattacks		✓

Although the IBM X-Force Threat Intelligence Index 2024 [17] shows the cyberattacks against the energy sector relatively unchanged in recent years, with the average of 9.42% for the past five years in Figure 1.5, the analysis done by the IBM X-Force estimates that as the AI market share increases to 50%, the cybercriminals will be more incentivized to develop tools targeting the AI technologies. For example, attackers could sabotage IoT

devices measuring electricity loads or their connections, compromising data integrity by adding small perturbations using adversarial attacks [18] that are not readily apparent to trick ML models into making incorrect forecasts.

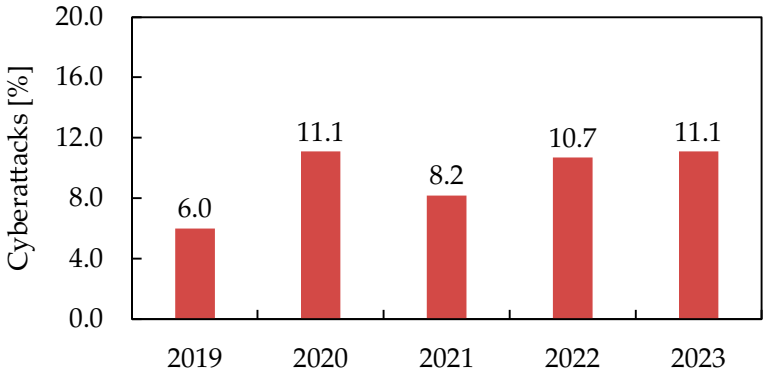


Figure 1.5: Global cyberattack trends for the past five years, targeting the energy sector, are relatively unchanged.

Besides adversarial attacks, the attacker could leverage Distributed Denial of Service (DDoS) attacks. Cloudflare's DDoS threat report for the 2024 Q2 reports a 20% increase in DDoS attacks when compared to the previous year [19]. Additionally, over 57% of application-layer DDoS attacks and over 88% of network-layer DDoS attacks last less than 10 minutes, which is too short for a system operator to manually apply mitigations to avoid missing values (MV) due to packet loss or preventing the servers from going offline [20].

In these reports, IBM X-Force and Cloudflare also mentioned the increasing number of state-sponsored attacks on specific organizations and countries and how they are utilized as a new technique of warfare. For instance, the Russo-Ukrainian War shows how vulnerable the existing infrastructures are to cyberwarfare [21]. Up until 17 December 2024, during the Russo-Ukrainian War, the CyberPeace Institute recorded a total of 125 cyberattacks on the energy sector [22], as shown in Figure 1.6. Besides causing financial losses, exposing sensitive personal information, and eroding public trust in smart cities, it also poses threats to national security. These problems highlight the necessity of having a distributed and robust forecasting model.

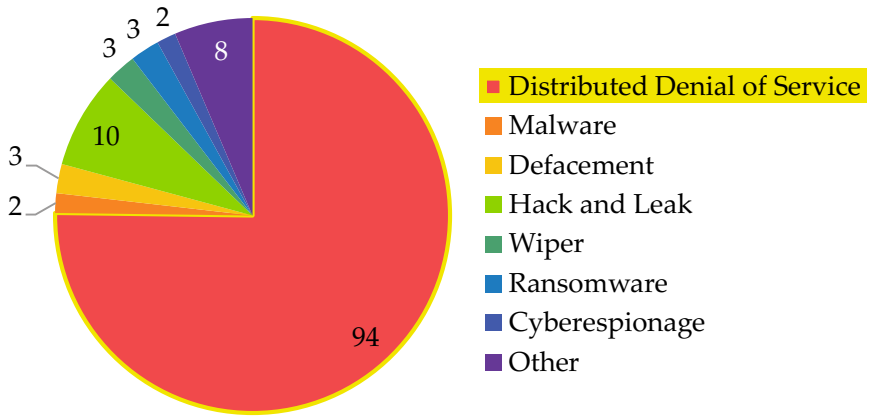


Figure 1.6: Distributed Denial of Service is the most commonly occurring cyberattack against the energy sector in the Russo-Ukrainian War.

Several solutions have been suggested. For instance, ML-based imputation mitigates missing values (MV) by complementing the MV caused by sensor issues or DDoS with high accuracy [23]. Adversarial training injects adversarial samples as training data to strengthen the ML models, allowing them to generalize against adversarial attacks [24]. Retraining mechanisms update the ML models to handle the latest data distribution when concept drift (CD) occurs [25]. Finally, federated learning helps decentralize the models [26] to avoid Single Point of Failure (SPoF) vulnerability. However, unifying these into one solution shows problems that are not effectively addressed in the previous study.

1.2 Research Objective

As briefly discussed in Chapter 1.1, MV refers to incomplete data, which can be caused by either a malfunction in the sensors or a DDoS attack that results in packet loss [27, 28, 29]. To solve MV, Zhang et al. [30] created a generative adversarial network to reconstruct values close to the correct one that is missing. Additionally, Ma et al. [31] implemented a bi-directional Long Short-Term Memory (LSTM) that uses correlation and causal data to predict the values close to the correct one that is missing.

CD explains the situation when the current data distribution diverges from data used to train, consequently invalidating the forecasting model [32, 33]. To solve CD, Bayram et al. [34] proposed a dynamic framework that could detect CD without defining the limit, which helps to retrain the MLs without external intervention. Moreover, Li et al. [35]

implement incremental updates by adding newly trained ML models to existing ensemble arrangement to help the forecast adapt to the latest trend.

Adversarial attacks are defined as methods to subject the data to small perturbations to maximize the loss of the MLs and mislead them into making incorrect decisions [36, 37]. To solve adversarial attacks, Ren et al. [38] implemented Transformer-based causal analysis to detect anomalies and prevent the data from being processed. Also, Kwon et al. [39] utilize various adversarial attacks in adversarial training to harden the MLs.

Finally, SPoF refers to the potential for the entire system to fail if cyberattacks take down the server or IoT device hosting the centralized ML model with no backup [40, 29]. Recent studies in distributed ML, such as Gupta et al. [41] and Shi et al. [42] in federated learning, show traction to maintain privacy and solve SPoF. The compatibilities between the solutions and their respective issues were summarized in Table 1.2.

Table 1.2: Compatibilities between the previous studies and their respective issues.

Author	**Citation**	**Issue**			
		Missing Values	*Concept Drift*	*Adversarial Attacks*	*Single Point of Failure*
Zhang et al.	[30]	✓			
Ma et al.	[31]	✓			
Bayram et al.	[34]		✓		
Li et al.	[35]		✓		
Ren et al.	[38]			✓	
Kwon et al.	[39]			✓	
Gupta et al.	[41]				✓
Liu et al.	[42]				✓

Although there is plenty of research found in Google Scholar dated from 1 January 2019 to 31 December 2024 that covers MV, CD, adversarial attacks and SPoF based on their specific keywords, as shown in Figure 1.7, there is only one paper that matched the "Smart City," "Forecast," "Missing Values," "Concept Drift," "Adversarial Attacks," and "Single Point of Failure" keywords, which shows the difficulties of integrating multiple solutions into one.

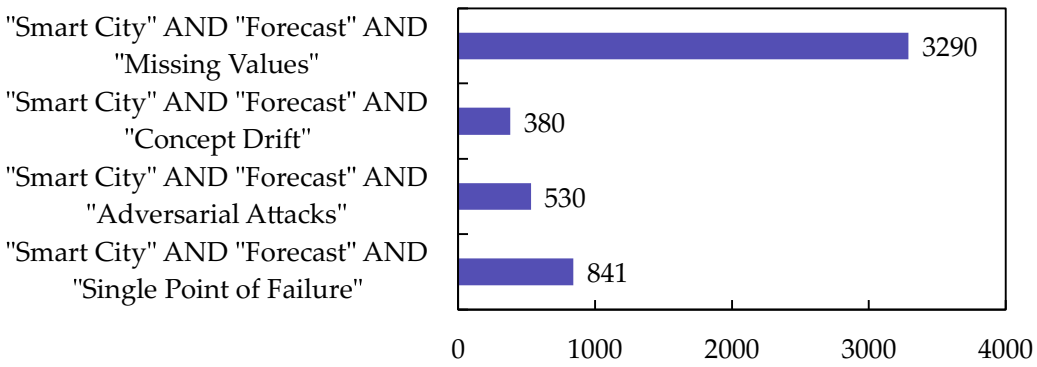


Figure 1.7: The number of research papers found using the keywords on Google Scholar from 1 January 2019 to 31 December 2024.

The research paper that has all keywords was done by Zhou et al. [43], where they utilized interpolation to impute MV, proof-of-distribution to detect adversarial attacks for contribution weight reduction, conformal scores comparison for retraining when CD was detected, and federated learning for sharing the forecasting model weights to act as a backup when one of the forecasting models is offline. Figure 1.8 shows the implementation done in this thesis, which focuses on tuning the federated learning to reduce the negative effect of adversarial attacks in training data via weighted averaging on the local models and retraining the global model to keep up with the latest trends. However, it is not a true unified solution due to weighted averaging not being applicable when facing live adversarial attacks, and the retraining effectiveness to handle CD depends on the retraining interval and data range used for retraining.

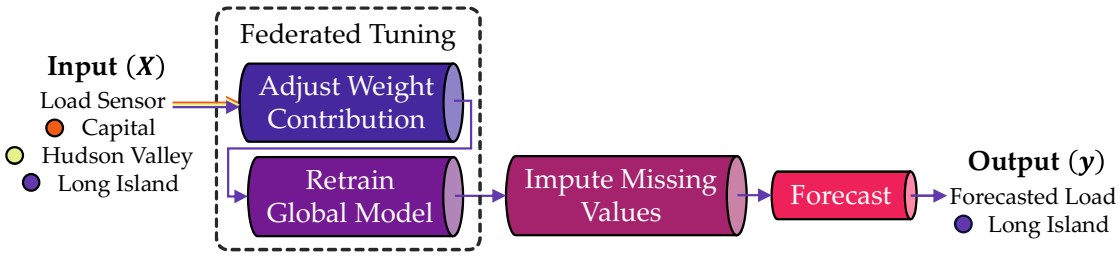


Figure 1.8: Forecasting system proposed by Zhou et al. [43] relies on detection to minimize the negative effects of these issues.

The reason why existing unified implementation relies on simple countermeasures or circumvention is due to the difficulties of implementing more complex solutions. Moreover, not all solutions are compatible with each other, as shown in the list below:

(i) **Incompatibilities with the input data:** Some solutions requires the input data to be preprocess with other solutions first to be effective. For example, MLs hardened against adversarial attacks cannot handle MV without it being imputed.

(ii) **Incompatibilities with other solutions:** Despite some solutions are aimed at the same issue, they might not be compatible. For example, a forecaster hardened against MV will not improve its accuracy with imputation, as it does not know the MV location.

(iii) **Correction errors accumulation in input data:** Input data preprocessing is commonly done in sequence to resolve the noises caused by MV, CD, and adversarial attacks. The mistakes in resolving the noises will cause the errors to accumulate, which unintentionally lowers the forecasting accuracy.

In this thesis, the proposed Adaptively Linked Composite Ensemble Networks (ALCEN) detect the issues found in the input data and dynamically link the solutions that satisfy the requirements needed. By rearranging the solutions that aim to fix the abnormalities based on their priorities in "cascading" manners, the input data are properly processed. Then, by sharing the preprocessed output with the targeted hardened forecasting models, they could effectively address the issues they were designed for. Finally, the concurrent output from the forecasting models is combined in a "stacking" manner via a meta model. Figure 1.9 demonstrates the combination used by ALCEN to address MV, adversarial attacks, and CD with correlated sensor data while SPoF was handled by sharing the model weights.

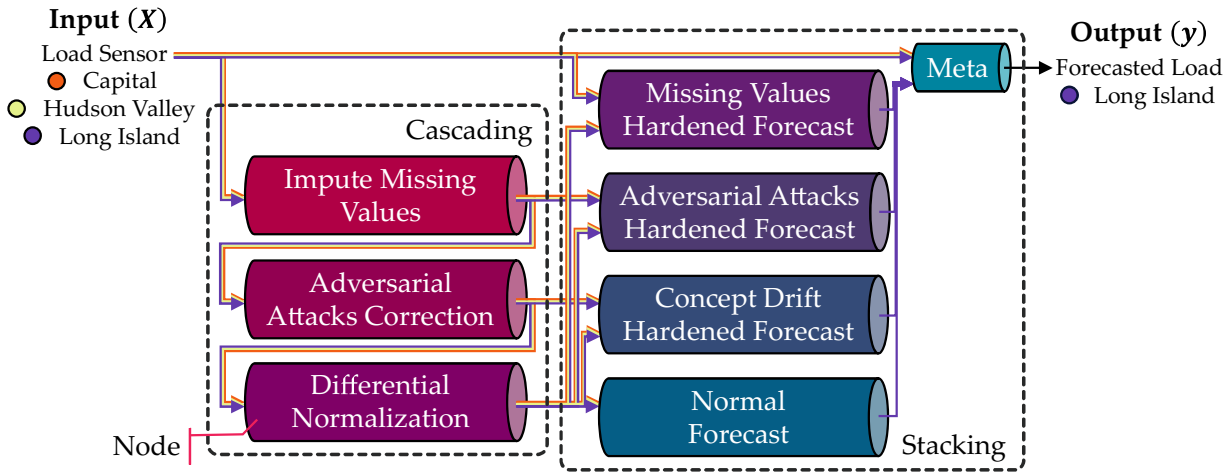


Figure 1.9: Constructed nodes connection by the proposed Adaptively Linked Composite Ensemble Network.

1.3 Contribution

This thesis demonstrates the originality of ALCEN through three unique contributions to make a robust forecasting system, which are not covered in previous research through the modularity of each ML model, wider compatibility with different solutions, and the adaptability to the external changes.

(i) **Modularity through unified template:** The model could temporarily take the place of another model that was not online by swapping the weights that were stored in ALCEN's internal database. This is possible by using the same ML model template that was optimized for different problems, allowing seamless recovery against SPoF due to DDoS attacks bringing the server offline without compromising the forecasting accuracy. This breaks away from the conventional strategy, which requires an entire forecasting system running on another server to act as a backup when the main system failed.

(ii) **Synergy through connection configuration:** The countermeasures for MV, CD, adversarial attacks and SPoF must be linked in a certain order based on the needs and priorities of each solution, as simply linking them together won't give the best result. ALCEN allows different solutions that fix the input data or solutions that harden the forecast to be unified into a single forecasting system by taking the requirements and

priorities into consideration, preventing conflicts between the solutions while ensuring the best forecasting outcome.

(iii) **Adaptability through event-driven architecture:** The event-driven approach is not only to update the ML in ALCEN when needed but also to help to detect any anomalies in the input data and respond by adjusting the connection between the ML models inside the composite ensemble networks. This approach of reconfiguring the connection between the ML models allows the ALCEN to adapt to the changes with minimal intervention from the system administrator, which is slow and reserved only for tuning the ALCEN's hyperparameters.

Through these contributions, ALCEN is capable of handling SPoF by swapping the weight of ML models taken from the backup to replicate other solutions in another nodes. Additionally, linking the ML models by considering their requirements and priorities allows different solutions that focus on fixing the input data or hardening the forecast to be integrated without compatibility issues. Finally, the event-driven architecture in ALCEN allows it to dynamically adapt to the external changes without continuous input from the system administrator. These contributions highlight the distinctions from Zhou et al. [43].

1.4 Article Structure

This thesis is divided into 6 chapters.

After introducing the research background, objective, and contribution in Chapter 1, Chapter 2 discusses the issues plaguing the forecasting model in real-world implementation and investigate the effectiveness of Zhou et al. [43] application in forecasting the electricity load in New York State against MV, CD, adversarial attacks and SPoF. Then, evaluation experiments are conducted to thoroughly assess the limitations.

Chapter 3 established the theoretical foundation for the proposed ALCEN to adapt the composite ensemble networks with external changes, addressing the network designs, compatibility problems, and selection hierarchies. The solutions for implementation and assessment under the smart city framework are also described to rectify the limitation of previous method.

Chapter 4 shows the implementation and deployment of ALCEN to forecast electrical load in New York State, as well as its resilience against MV, CD, adversarial attacks SPoF, and SPoF. Then, the accuracy was compared with the Zhou et al. [43] implementation, highlighting its weaknesses.

Finally, Chapter 6 summarized the works done in this thesis, in addition to the future work that was not addressed in this study.

2 Previous Research

2.1 Chapter Introduction

In their paper, Zhou et al. [43] proposed a robust federated forecasting system to handle MV, CD, adversarial attacks and SPoF that could be encountered when forecasting the electricity load. Their implementation was originally implemented to create a local model for each user to forecast their electricity consumption. Then, the local models' weights will be averaged to create a global model that will be retrained to forecast the aggregated electricity load for the residential area. Any local models that are suspected to be trained with falsified or adversarial data will have their contribution reduced via weighted averaging to avoid them from negatively affecting the global model. Finally, CD was resolved by retraining the global model and MV via imputation. To ensure a result for comparison with ALCEN could be obtained, their proposal was implemented to forecast the electricity load in Long Island, New York State, as shown in Figure 2.1.

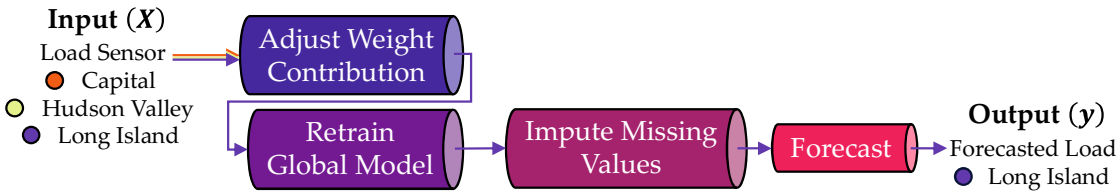


Figure 2.1: Zhou et al. [43] proposal was implemented to forecast the electricity load in Long Island.

The dataset used to train and evaluate their proposal on Long Island spans from 1 January 2018 to 31 December 2020, with 5 minutes intervals of recorded electricity load [44]. In addition, the electricity load data with a strong Kendall's τ coefficient, such as Capital and Hudson Valley, was used to enhance the forecast.

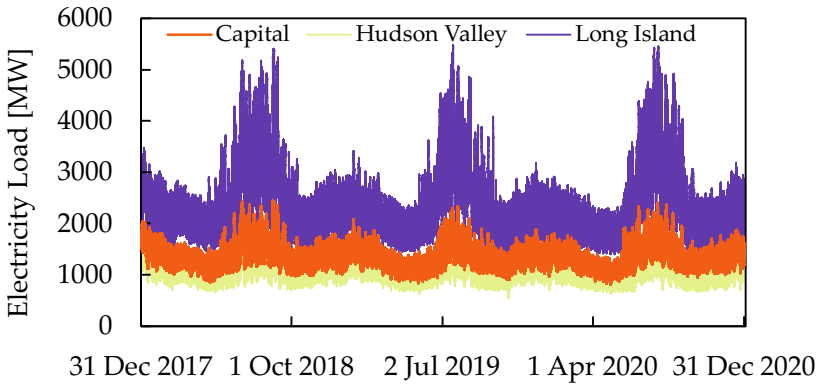
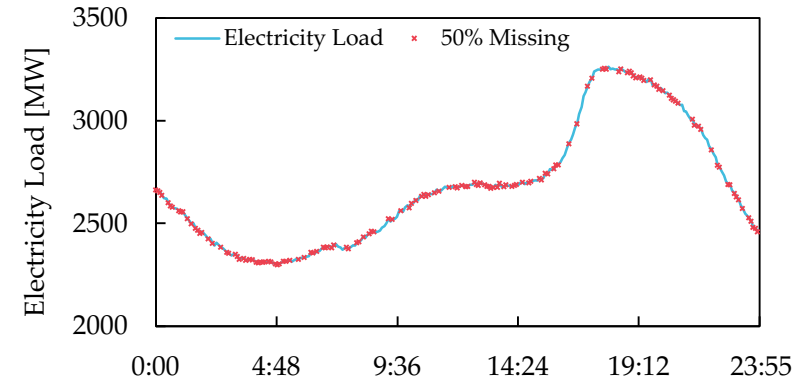
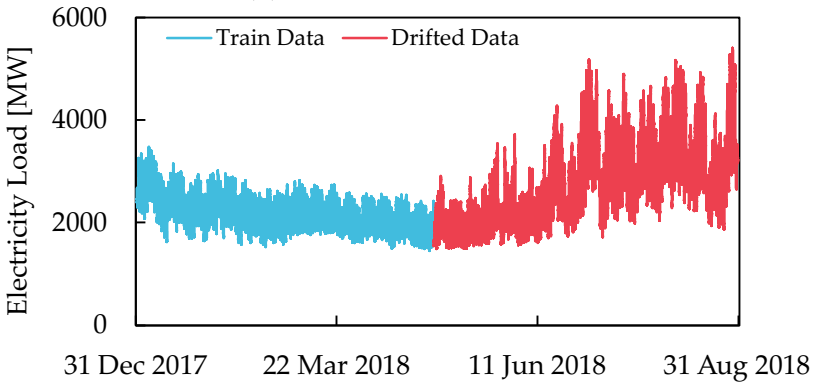


Figure 2.2: Electricity load dataset ranging from 1 January 2018 to 31 December 2020.

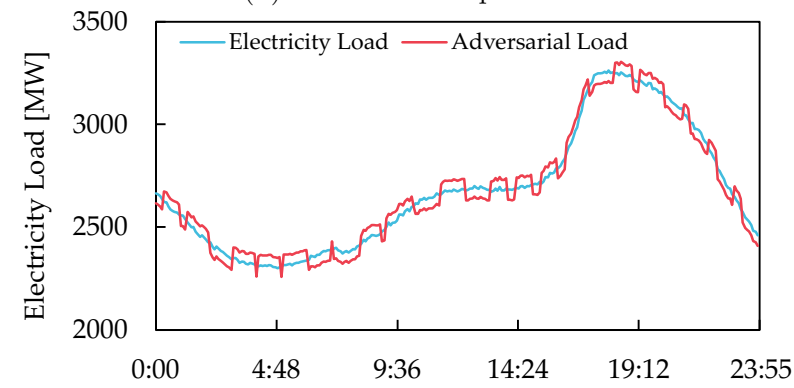
Figure 2.3 shows the disturbances that could occur in the input data and negatively affect the forecasting accuracy, such as MV due to sensor failure or DDoS attacks, CD due to seasonal changes, and adversarial attacks crafted by the attackers to trick the MLs into making incorrect predictions.



(a) Missing values of 50%



(b) Seasonal concept drift



(c) Adversarial attacks with ϵ=0.05 of perturbation

Figure 2.3: Input disturbances that will negatively affect the forecasting accuracy.

This chapter will discuss previous research on MV, CD, adversarial attacks, and SPoF, highlighting their advantages over Zhou et al. [43] unified approach. Then, the effectiveness to forecast electricity load is tested to identify weaknesses that ALCEN will address.

2.2 Countermeasure Appraisal

In countermeasure appraisal, the previous research done to solve MV, CD, adversarial attacks, and SPoF will be compared against the implementation proposed by Zhou et al. [43], highlighting the potential weakness of countermeasures deployed by their proposal to unify it in one package.

Additionally, the limitation and reasoning for why the unified solution proposed by Zhou et al. [43] does not use a more complex solution are given, highlighting the difficulties of integrating multiple solutions into one.

2.2.1 Missing Values

MV that happened when forecasting the electricity load is classified as a missing completely at random (MCAR) [45], which is shown in Figure 2.4. The reason for that is the probability of the sensor measuring the electricity load to fail and the probability for the packet data to be lost due to network issues or DDoS attacks occurred completely at random. The previously introduced imputation proposed by Zhang et al. [30] and Ma et al. [31] utilizes multivariate data with to impute the MV with high accuracy.

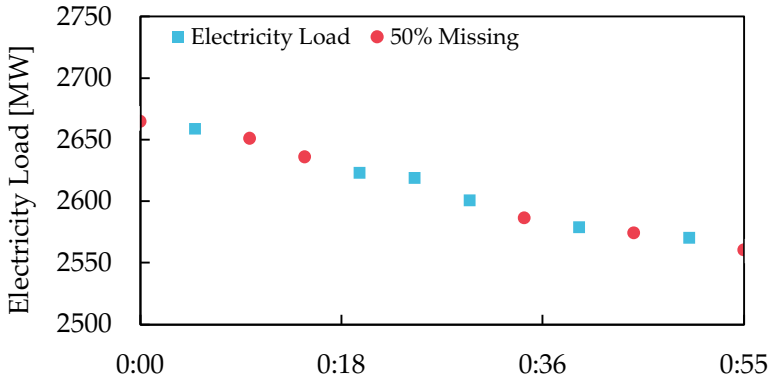


Figure 2.4: Simulated 50% missing completely at random in Long Island on 1 January 2018.

However, due to the privacy constraint from the unified method proposed by the Zhou et al. [43], the imputation use univariate interpolation method instead, which far less accurate than multivariate imputation. Furthermore, the type of interpolation used in their research were not defined. To implement their proposal, this thesis utilizes spline interpolation [46]

of order 2 and fallback imputation value of -1 if the sequence has less than 2 real values, as shown in Algorithm 2.1, which offer greater accuracy than polynomial interpolation for univariate data.

Algorithm 2.1 Spline Interpolation

Input: Input sequence v_i, Order $order$=2, Fallback value $f = -1$
Output: Scaled imputed sequence v_i^*

Function `spline_interpolation`(v_i, k)
  Create a copy of the array: $v_i^* \leftarrow$ `copy`(v_i)
  Set of indices for non-missing values: $x\_known \leftarrow$ `known_indices`(v_i^*)
  Set of values for non-missing values: $y\_known \leftarrow v_i^*[x\_known]$
  Set of indices for missing values: $x\_missing \leftarrow$ `missing_indices`(v_i^*)
  if $x\_known \geq order$ **do**
    Get the imputation: $cs \leftarrow$ `CubicSpline`(x_known, y_known)
    Match with the missing indices: $y\_missing \leftarrow cs(x\_missing)$
    Impute missing values: $v_i^*[x\_missing] \leftarrow y\_missing$
  else do
    Impute missing values: $v_i^*[x\_missing] \leftarrow f$
  return v_i^*
End Function

2.2.2 Concept Drift

CD refers to a phenomenon where the current data distribution is different from the training dataset that was used to train the ML model, which makes it less accurate [47]. While are many reason that cause CD [48], such as changing user preferences or external factors that are uncountable, this research focus on seasonality changes, which shows the normalized distribution on Figure 2.5 for Figure 2.3 (b). Additionally, the inability to generalize with CD will result in cascading effects on the entire system, which relies on the forecast to optimize its operations. The previously introduced retraining mechanism proposed by Bayram et al. [34] and Li et al. [35] resolves this issue by updating the forecasting model with new trends.

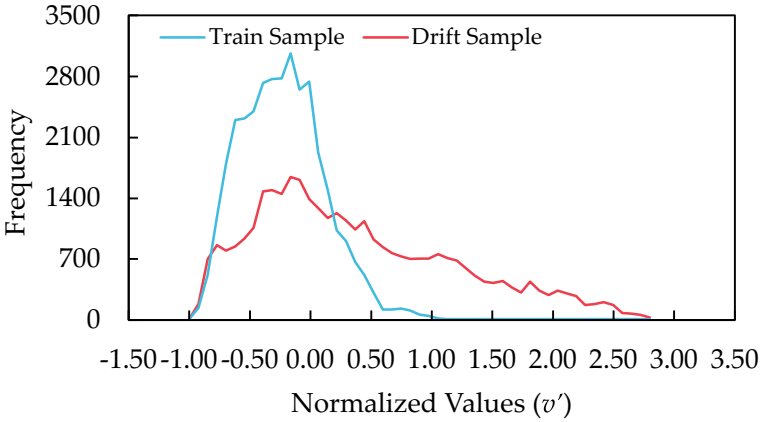


Figure 2.5: Normalized values distribution comparison on train and drifted sample due to the seasonal changes.

Zhou et al. [43] implemented a trivial retraining method on top of federated learning, with Conformal Prediction [49] being used to determine if retraining is needed. which is shown in Algorithm 2.2. Although its being a non-parametric method makes it easy to implement, its sensitivity might cause unnecessary expensive retraining.

Algorithm 2.2 Conformal Prediction

Input: Calibration c_true, c_pred, Test t_true, t_pred, Confidence c=0.95
Output: Decision to retrain model $True \vee False$

Function `conformal_prediction`$(c\_true, c\_pred, t\_true, t\_pred, c)$
  Compute absolute errors: $abs\_errors \leftarrow |c\_true - c\_pred|$
  Determine confidence level: $quantile \leftarrow$ `percentile`$(abs\_errors, 100 \times c, \text{axis=0})$
  Construct upper bounds intervals: $l\_bounds \leftarrow t\_pred - quantile$
  Construct lower bounds intervals: $u\_bounds \leftarrow t\_pred + quantile$
  Find coverage: $coverage \leftarrow$ `mean`$((t\_true \geq l\_bounds) \wedge (t\_true \leq u\_bounds), \text{axis=0})$
  if `any`$(coverage < confidence)$ **then**
    Retrain the model: **return** $True$
  else
    No retraining needed: **return** $False$
End Function

2.2.3 Adversarial Attacks

Although currently there is no recorded adversarial attacks being used to sabotage the electricity load forecasting system in real-world, recent cybersecurity surveys indicate a growing research interest in adversarial attacks over traditional attacks, such as DDoS attacks [50, 51]. Unlike DDoS attacks, which can be easily identified and trigger the

detection systems, adversarial attacks are crafted to exploit vulnerabilities in ML models, leading them to make incorrect decisions without being easily apparent [36].

Figure 2.6 shows how the adversarial attacks implementation could be done without having direct access to the forecasting model. As energy companies often share the forecast data for transparency and help consumers make data-driven decisions, the attacker could use it to craft effective adversarial attacks, as shown in the following steps:

(i) **Train surrogate model:** Use the publicly shared current and forecasted electricity load data to train the surrogate model.

(ii) **Gradient computation:** Use surrogate model to forecast and optimize the gradient needed to increase the loss score.

(iii) **Adversarial insertion:** Insert the generated adversarial samples to the compromised IoT sensors or network.

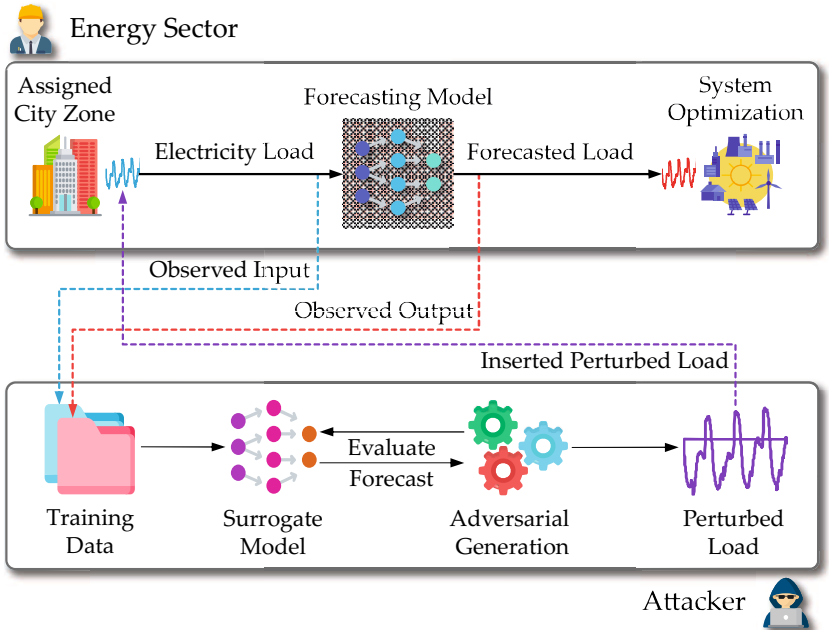


Figure 2.6: An illustration of adversarial attacks implementation against the energy sector.

Figure 2.7 shows the up-close comparison between the observed electricity load and the adversarial load, where the adversarial load tries to reduce the forecasting accuracy by shifting up the electricity load to give the impression to the forecasting model that the electricity load is going to increase. Additionally, shifting down the electricity load to

give the impression that it is going to decrease. To handle live adversarial attacks, the previously introduced countermeasures by Ren et al. [38] focus on detection to prevent adversarial data from being processed, while Kwon et al. [39] focus on model hardening to increase the reliability and help the forecasting model to generalize well with adversarial attacks.

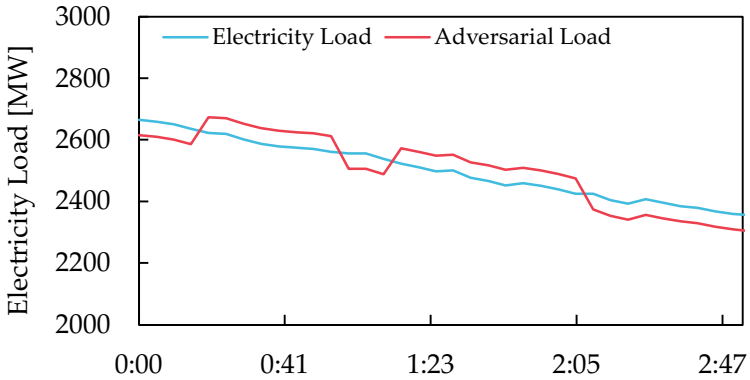


Figure 2.7: The perturbation added shifted the electricity load pattern to tricks the forecasting model.

For the unified method proposed by Zhou et al. [43], they utilize blockchain to secure the forecasting network and weighted averaging in federated learning to create a standardized internal parameters m_i without sharing the dataset. Compared to the typical federated learning, where the local models (pre-averaging) only undergo simple averaging to create a global model (post-averaging), the weighted averaging reduces the contribution w_i of the local model trained with adversarial data from negatively affecting the global model during the training phase, as shown in Figure 2.8. However, this method cannot handle live adversarial attacks.

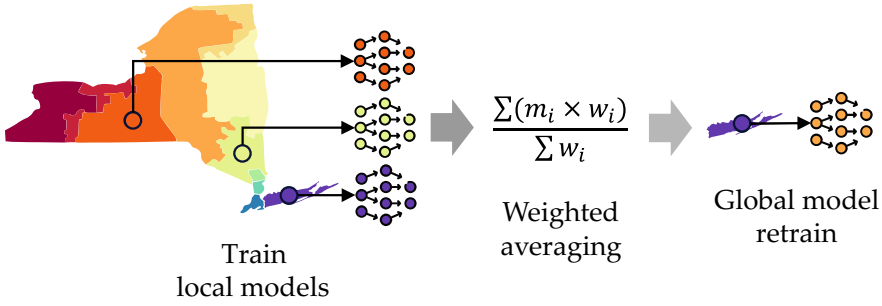


Figure 2.8: Weighted averaging was used to reduce the contribution of forecasting model trained with adversarial data.

2.2.4 Single Point of Failure

SPoF could be caused by technical issues caused the infrastructure to become unstable or DDoS attacks overwhelming the server with a massive amount of fake traffic [52]. As the server is trying to handle all the incoming traffic, the processing power and bandwidth will get exhausted, making it difficult for the legitimate users to access the server. Similarly, the inability of the server hosting the ML to forecast will negatively impact the systems that depend on it for optimal operation and decision-making.

This is due to the linear flow in system optimization, as shown in Figure 2.9 where the failure of the prediction mechanism could cause cascading failure for the next tasks that relies on the forecasting outcome to operate.

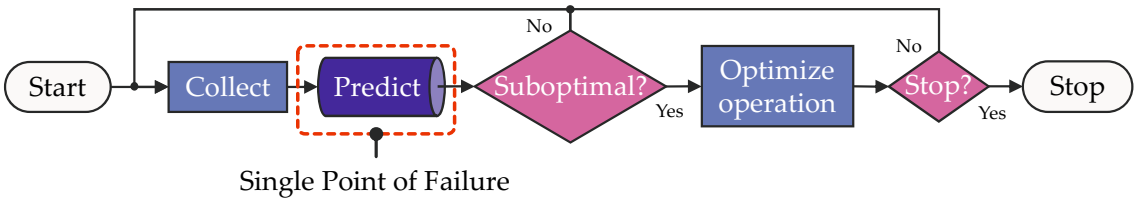


Figure 2.9: Prediction mechanism act as a single point of failure that could bring the system down due to the linear flow in optimization.

While the most trivial countermeasure against SPoF relies on creating a copy of the forecasting model in a different location to act as a backup, distributed computing also can mitigate the SPoF vulnerability by partitioning complex model architecture across multiple servers with the side effect of latency [53]. Previously introduced federated learning by Gupta et al. [41] and Liu et al. [42] is also a form of distributed computation that prioritizes data privacy.

Similarly, Zhou et al. [43] use federated learning to distribute the forecasting model to solve SPoF in their implementation. Algorithm 2.3 shows the simplified implementation used to simulate federated learning in this experiment. Differ from the typical federated learning implementation, the global model was retrained on the target forecast v'_{train_0} to help it generalize well on the target forecast without the heterogeneity issues. However, federated learning restrict the number of forecasting model architecture to one, hence the reason why the method proposed by Zhou et al. [43] does not deploy additional ML-based

data preprocessing and relies on simple statistical imputation to solve missing values.

Algorithm 2.3 Federated Learning Training Mechanism

Input: Scaled training sequences v'_{train}=[$v'_{train_0}, v'_{train_1}, v'_{train_2}, \cdots, v'_{train_j}$], Model M
Output: Global model M

Function `federated_learning`(v'_{train}, M)
  Initialize weight storage: $weights \leftarrow []$
  for j=0 **to** `len`(v'_{train}) **do**
    Train local model: $M.train(v'_{train_j})$
    Store the weight: $weights$.`append`(M.`get_weight()`)
    Reset local model weight: M.`reset_weight`()
  Average the weights stored: $averaged\_weight \leftarrow$ `average`($weights$)
  Set global model M.`set_weight`($averaged\_weight$)
    Retrain global model on the target forecast: $M.train(v'_{train_0})$
  return M
End Function

2.2.5 Summary

Compared to the previous research, the proposed method by Zhou et al. [43] relies on simple countermeasures that are less effective to create a unified solution against MV, CD, adversarial attacks, and SPoF. The reason for this is that federated learning makes it difficult to manage multiple ML-based data preprocessing across multiple global models.

Additionally, utilizing ML-based data preprocessing restricts it to linear correction, which inadvertently accumulates errors from inadequate corrections.

2.3 Method Evaluation

In method evaluation, the unified solution proposed by Zhou et al. [43] is implemented to forecast the electricity load in Long Island. Then the forecasting accuracy will be evaluated against MV, CD, and adversarial attacks.

Since the problem with SPoF was completely resolved with federated learning could create multiple identical ML models, in addition to retraining the global model on the target forecast to avoid problems with data heterogeneity, it is not evaluated.

2.3.1 Data Preparation

In Figure 2.10, Figure 2.11, and Figure 2.12, the electricity load recorded for every 5 minutes shows sporadic MV due to sensor failures or network issues, accounting for only 0.0234% of MV. Using polynomial interpolation of order 2 [54], MV was imputed to create a clean dataset for training and comparing forecasting accuracy. Outliers and seasonal drift were purposely retained to reflect real-world conditions.

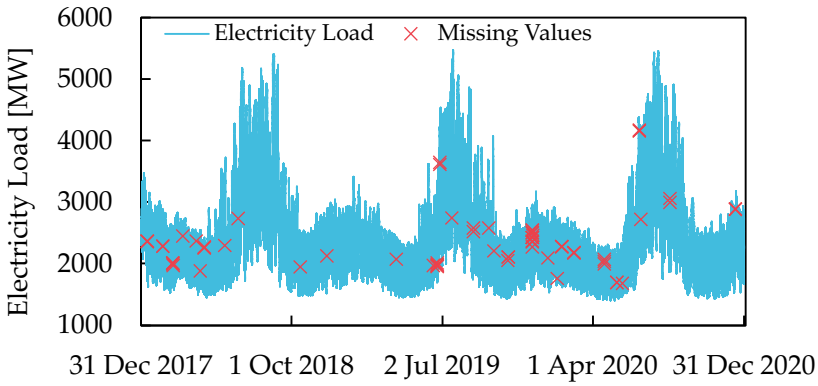


Figure 2.10: Electricity load in Long Island from 1 January 2018 to 31 December 2020.

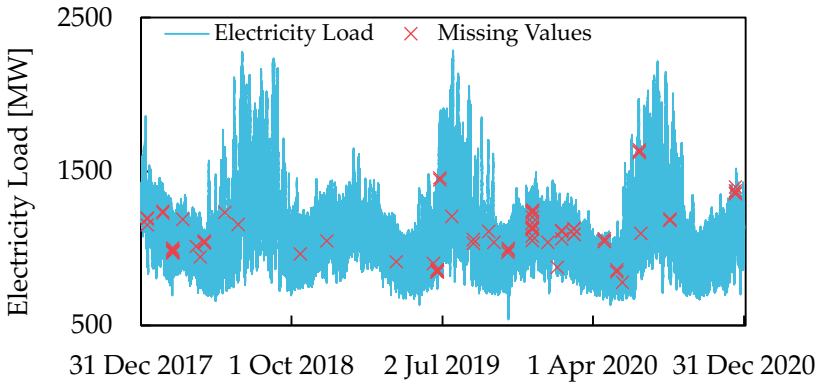


Figure 2.11: Electricity load in Hudson Valley from 1 January 2018 to 31 December 2020.

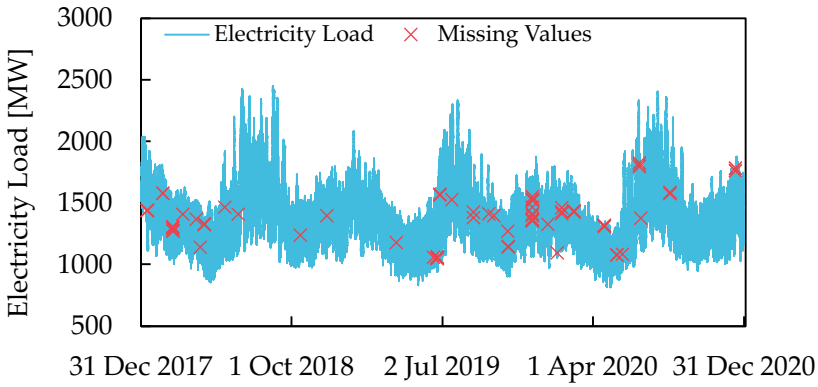


Figure 2.12: Electricity load in Capital from 1 January 2018 to 31 December 2020.

After that, min-max normalization was used to rescale the values v in the training data v_{train} and evaluation data $v_{evaluate}$ from -1 to 1. To avoid exposing the information found in $v_{evaluate}$ when training the forecasting models, the minimum and maximum values needed to normalize the dataset were solely used from the v_{train}, as shown in Equation (2.1). Not exposing the minimum and maximum values found in $v_{evaluate}$ is important to simulate the real-world scenarios, as the system administrator may not know how the trend will evolve.

$$v' = 2 \times \frac{v - \min(v_{train})}{\max(v_{train}) - \min(v_{train})} - 1 \tag{2.1}$$

Finally, the v'_{train} and $v'_{evaluate}$ were sequenced into independent X and dependent y variables to train and evaluate the forecasting models. In these experiments, the forecasting models are set to take one hour of electricity load reading to forecast the next one hour for short-term load forecasting, which is equal to 12 steps for the input length *inlen* and output length *outlen*. To ensure the forecasting models could learn the intricate patterns from v_{train}, the sliding window step s to create the sequence was set to 1, while $v_{evaluate}$ was set to 12 to avoid redundant values in the sequence. These parameters were parsed to the Algorithm 2.4 to sequence the X and y for v_{train} and $v_{evaluate}$.

Algorithm 2.4 X and y sequencer

Input: Input length $inlen$, Output length $outlen$, Scaled variables v', Step s
Output: Input array X, Output array y

Function `data_sequencer`$(inlen, outlen, v', s)$
  Initialize empty arrays: $X \leftarrow []$
  Initialize empty arrays: $y \leftarrow []$
  for $i = 0$ **to** `len`$(v') - inlen - outlen + 1$ **step** s **do**
    Append sliced sequence: X.`append`$(v'[i : i + inlen])$
    Append sliced sequence: y.`append`$(v'[i + inlen : i + inlen + outlen])$
  return X, y
End Function

2.3.2 Model Architecture

The forecasting model architecture used for the local and global model in federated learning is shown in Figure 2.13, which utilizes Seq2seq LSTM architecture to forecast 1 hour of electricity load or output 12 values that represent the load value for a 5 minutes interval ($12 \leq \hat{t} < 24$). The architecture implementation is shown in Listing 1, which utilizes the Keras 3.8.0 [55] library with TensorFlow 2.18.0 [56] as the backend.

Listing 1: Seq2seq Long Short-Term Memory architecture.

```
from keras import models, losses, layers, metrics, optimizers


def seq2seq_lstm(
    encoder_input_length: int,
    decoder_input_length: int,
    output_length: int,
    feature_number: int,
    layer_number: int,
    unit_number: int,
) -> models.Model:

    # Define the inputs
    encoder_inputs = layers.Input(shape=(encoder_input_length, feature_number))
    # Initialize the input for the encoder
    hidden_encoder_layer = encoder_inputs
    # Create the LSTM layers using a loop
    for i in range(layer_number):
        # Use return_sequences=True for all layers except the last one
        return_sequences = (i < layer_number - 1)
        # Use return_state=True only for the last LSTM layer to get the states
        return_state = (i == layer_number - 1)
        # Define the lstm layer
        lstm_layer = layers.LSTM(
            # Set the unit number
            units=unit_number,
            # Hardcoded the activation to tanh
            activation="tanh",
            # Hardcoded the activation to sigmoid
            recurrent_activation="sigmoid",
            # Hardcoded the initializer to orthogonal
            recurrent_initializer="orthogonal",
```

```
            # Hardcoded to use bias
            use_bias=True,
            # Hardcoded the initialized bias to zeros
            bias_initializer="zeros",
            # Hardcoded the initialized kernel weight using the glorot uniform distribution
            kernel_initializer="glorot_uniform",
            # Return sequence and state logic
            return_sequences=return_sequences,
            return_state=return_state
        )
        # Connect and get the output
        if return_state is False:
            hidden_encoder_layer = lstm_layer(hidden_encoder_layer)
        else:
            _, state_h, state_c = lstm_layer(hidden_encoder_layer)
    # Get the encoder output
    encoder_states = [state_h, state_c]
    # Define the decoder input
    decoder_inputs = layers.Input(shape=(decoder_input_length, feature_number))
    # Initialize the input for the decoder
    hidden_decoder_layer = decoder_inputs
    # Create the LSTM layers using a loop
    for i in range(layer_number):
        # Use return_state=True only for the last LSTM layer to get the states
        return_state = (i == layer_number - 1)
        # Define the lstm layer
        lstm_layer = layers.LSTM(
            # Set the unit number
            units=unit_number,
            # Hardcoded the activation to tanh
            activation="tanh",
            # Hardcoded the activation to sigmoid
            recurrent_activation="sigmoid",
            # Hardcoded the initializer to orthogonal
            recurrent_initializer="orthogonal",
            # Hardcoded to use bias
            use_bias=True,
            # Hardcoded the initialized bias to zeros
            bias_initializer="zeros",
            # Hardcoded the initialized kernel weight using the glorot uniform distribution
            kernel_initializer="glorot_uniform",
            # Return sequence and state logic
            return_sequences=True,
            return_state=return_state
        )
        # Connect and get the output
        if return_state is False:
            hidden_decoder_layer = lstm_layer(hidden_decoder_layer, initial_state=encoder_states)
        else:
            decoder_outputs, _, _ = lstm_layer(hidden_decoder_layer, initial_state=encoder_states)
    # Flatten the dimension
    decoder_flatten_outputs = layers.Flatten()(decoder_outputs)
    # Construct the output layer
    output_layer = layers.Dense(
        # Number of unit
        output_length,
    )(decoder_flatten_outputs)

    # Define the model
    model = models.Model(
        [encoder_inputs, decoder_inputs],
        output_layer
    )

```

```
    # Compile the model
    model.compile(
        loss=losses.MeanSquaredError(),
        metrics=[metrics.RootMeanSquaredError()],
        optimizer=optimizers.Adam(learning_rate=0.001),
    )

    return model
```

The architecture utilizes two different inputs, where the encoder input focuses on the past 2 hours of historical electricity load ($-24 \leq t < 0$), and the decoder input focuses on the latest 1 hour of electricity load ($0 \leq t < 12$). This not only helps the forecasting model to adapt to CD but also helps it not to be over-reliant on the latest data that might be compromised. Table 2.1 shows the parameters used to initialize the architecture.

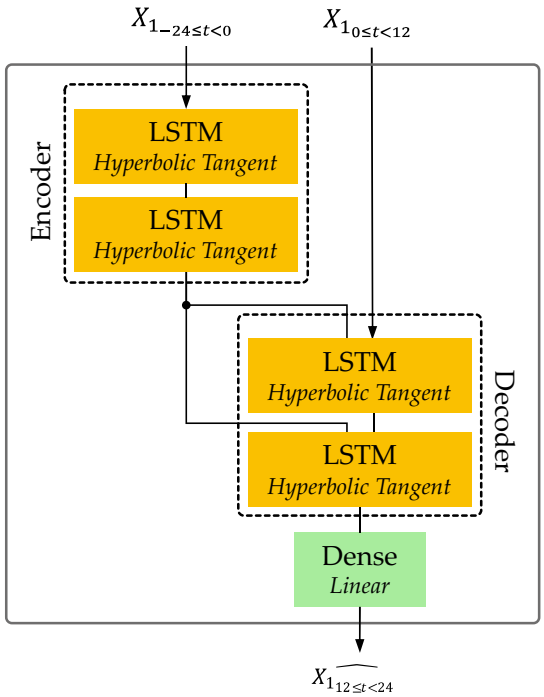


Figure 2.13: Seq2seq Long Short-Term Memory model architecture used to forecast.

Table 2.1: Parameters used to initialize Seq2seq Long Short-Term Memory model.

Parameter	**Values**
Encoder input length	24
Decoder input length	12
Output length	12
Feature number	1
Layer number	2
Unit number	96

To prevent overfitting, an early stop function was implemented, with the minimum delta in TensorFlow set to 0.0001, which represents the difference between the previous and current loss score, and the patience set to 3, which represents the allowed consecutive chances for the loss score to be less than 0.0001 before prematurely ending the training.

The reason why the forecast was set to only 1 hour of electricity load is that most of the forecasting models used to optimize the system focus on short-term forecasts to optimize the system with high accuracy, while long-term forecasts will have their accuracy worsened instead. This is the trade-off between the accuracy and the forecasting horizon, where the forecasting accuracy will fall as the forecasting horizon increases [57].

2.3.3 Missing Values

To simulate MCAR on $v'_{evaluate}$ due to sensor failure or packet loss due to the DDoS attacks, Algorithm 2.5 was used to randomly insert Not a Number (NaN) to represent MV. For reproducibility of the inserted MV, the randomization seed $seed$=2025 was used, and the percentage of MV $percentage$= $\{0\%, 10\%, 20\%, \cdots, 90\%\}$ is the MV percentage range to be randomly chosen.

Algorithm 2.5 Missing Completely at Random Generator

Input: Scaled evaluation variables $v'_{evaluate}$, Percentage $percentage$, Seed $seed$
Output: Scaled evaluation variables with missing values $v'_{evaluate_{percentage}}$

Function `mcar_generator`($v'_{evaluate}, percentage, seed$)
  Create a copy of the array: $v'_{evaluate_{percentage}} \leftarrow$ `copy`($v'_{evaluate}$)
  Set the randomization seed: `random_seed`($seed$)
  Get the total number of elements: $n \leftarrow$ `len`$v'_{evaluate}$
  Calculate the number of missing values: $missing \leftarrow percentage \times n$
  Generate missing values indices: $indices \leftarrow$`random_choice`$(n, missing)$
  Insert missing values: $v'_{evaluate_{percentage}}[indices] \leftarrow NaN$
  return $v'_{evaluate_{percentage}}$
End Function

The target forecast duration is the electricity load on Long Island, from 1 January 2019 to 30 April 2019. The MV is simulated from 0% to 90% to measure the effectiveness of the previous unified method. Although the DDoS duration reported by Cloudflare [19] last less than 10 minutes, the simulated MV duration were purposely increased to 4 months

with MV ranging from from 0% to 90% to accurately benchmark the performance.

This result should give accurate estimate on how the previously proposed unified solution by Zhou et al. [43] will perform in real-world implementation.

With scheduled federated learning denoted as T1, T2, and T3 being done every 4 months to keep the global model up-to-date, it was tested on $v'_{evaluate}$ with MV ranging from 0% to 90%. In addition, the Conformal Prediction is also used to retrain the federated learning in 2 months intervals whenever the accuracy does not satisfy the requirement. Figure 2.14 shows the visualized scheduled federated learning with the evaluation range to measure the accuracy.

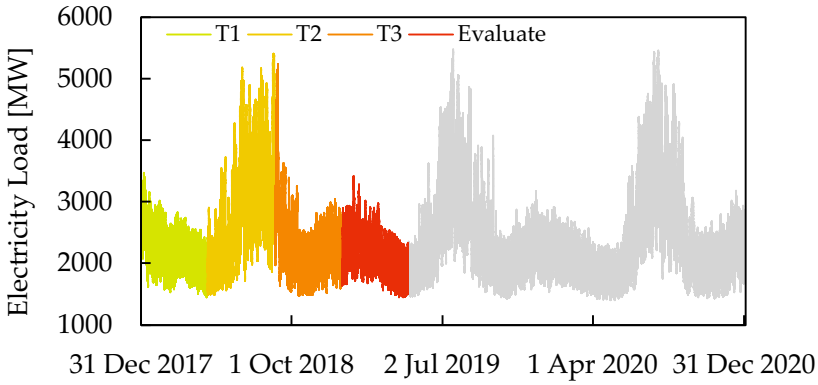


Figure 2.14: Scheduled federated learning done to update the global model and the evaluation range.

Table 2.2, Table 2.3, and Table 2.4 show the forecasting accuracies measured using the coefficient of determination (R^2), root mean squared error (RMSE), and mean absolute error (MAE) from 1 January 2019 to 30 April 2019, with MV being incremented by 10%. As ML is non-deterministic, the experiment was repeated 3 times to get the averaged score.

Table 2.2: Coefficient of determination scores against the missing values.

Missing Percentage [%]	**Coefficient of Determination (R^2)**			
	Session			*Average*
	1	2	3	
0	0.9827	0.9812	0.9804	0.9814
10	0.9779	0.9762	0.9768	0.9770
20	0.9612	0.9535	0.9527	0.9558
30	0.8605	0.8392	0.8250	0.8416
40	0.7536	0.6351	0.7029	0.6972
50	0.6549	0.4139	0.5777	0.5488
60	0.2917	-0.1707	0.2055	0.1088
70	-0.2555	-0.6823	-0.1540	-0.3639
80	-2.4321	-2.0580	-2.0229	-2.1710
90	-4.8134	-4.6257	-4.1809	-4.5400

Table 2.3: Root mean squared error scores against the missing values.

Missing Percentage [%]	**Root Mean Squared Error (RMSE)**			
	Session			*Average*
	1	2	3	
0	43.641	45.562	46.491	45.231
10	49.421	51.198	50.572	50.397
20	65.404	71.636	72.221	69.753
30	124.06	133.19	138.96	132.07
40	164.88	200.63	181.04	182.18
50	195.12	254.28	215.85	221.75
60	279.53	359.37	296.07	311.66
70	372.17	430.80	356.80	386.59
80	615.34	580.83	577.49	591.22
90	800.84	787.80	756.02	781.55

Table 2.4: Mean squared error scores against the missing values.

Missing Percentage [%]	**Mean Absolute Error (MAE)**			
	Session			*Average*
	1	2	3	
0	30.369	32.391	33.051	31.937
10	33.703	34.641	33.939	34.094
20	37.827	39.460	38.940	38.742
30	46.042	50.224	47.640	47.969
40	61.330	66.611	64.047	63.996
50	77.210	84.438	80.673	80.773
60	108.15	123.56	112.74	114.81
70	158.76	184.37	154.93	166.02
80	330.55	315.04	317.15	320.92
90	585.28	580.68	560.29	575.42

The results shows that as the percentage of MV rises, the forecasting accuracy will also worsen. In addition, as MV rise to 70%, a negative R^2 score indicates that the prediction are worse than simply using the mean of the observed data. Due to the univariate imputation limitation of spline interpolation, it could only handle up to 30% of MV before the R^2 score fell below 0.8. This observation is also confirmed from the RMSE scores, where the average score rises above 150 when handling more than 30% of MV.

The accuracy could be quantify using MAE scores, which describe the averaged absolute error of electricity load. As the MV percentage rise more than 30% the expected average absolute electricity load loss will rise more than 132.07 [MW]. Figure 2.15, Figure 2.16, and Figure 2.17 plotted the relationships the averaged scores against MV.

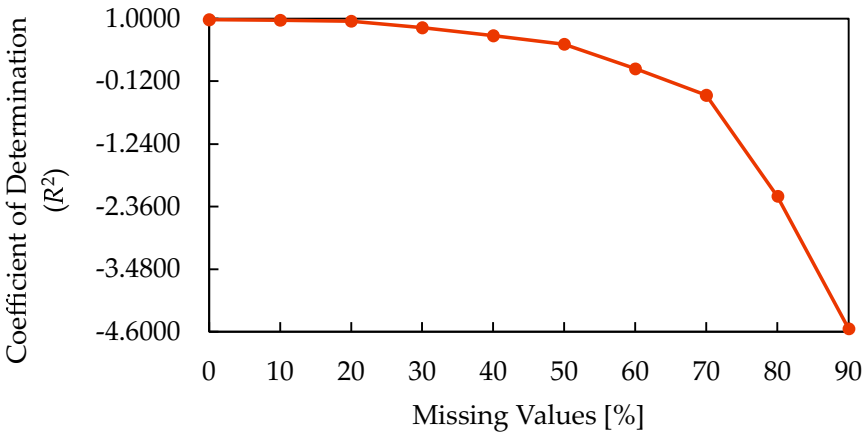


Figure 2.15: Averaged coefficient of determination scores against missing values from 1 January 2019 to 30 April 2019.

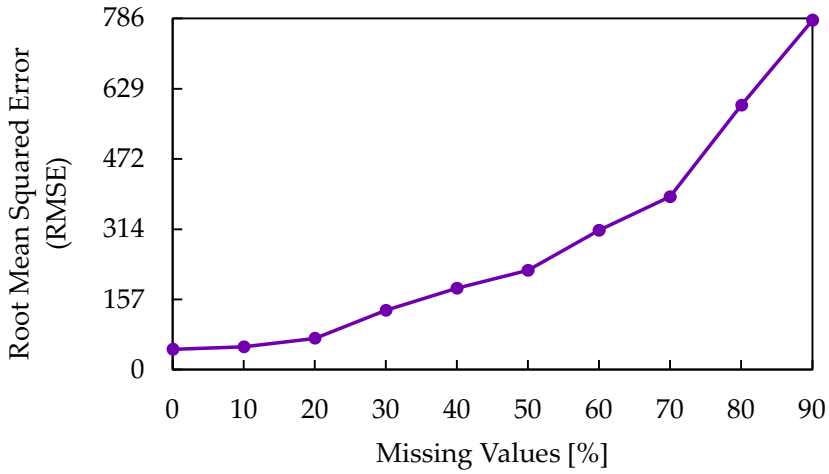


Figure 2.16: Averaged root mean squared errors score against missing values from 1 January 2019 to 30 April 2019.

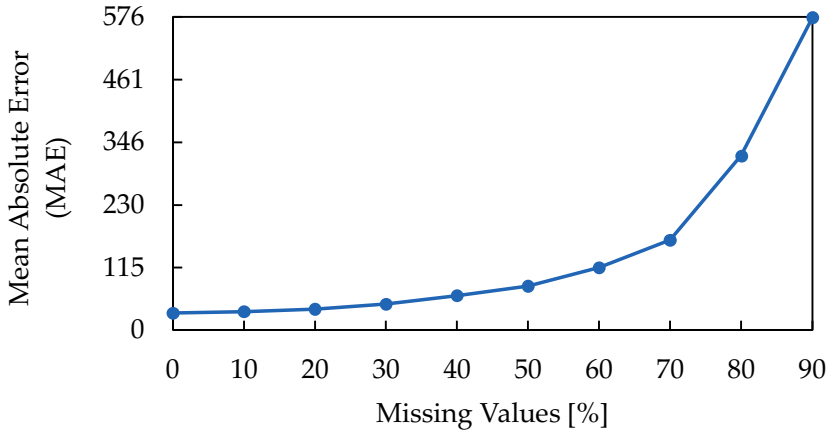


Figure 2.17: Averaged mean absolute errors score against missing values from 1 January 2019 to 30 April 2019.

2.3.4 Concept Drift

As shown in Figure 2.18, the dataset shows a noticeable drift every summer and a small drift during the winter. One of the main contributors to these was confirmed by the number of heating, ventilation, and air conditioners (AC) sold in New York State from 2018 to 2020 [58], which shows that AC is preferred during the summer compared to the electric boiler or furnace during the winter. Conversely, this also explain why the electricity load during the winter is low, as electricity are not mainly used to heat the building.

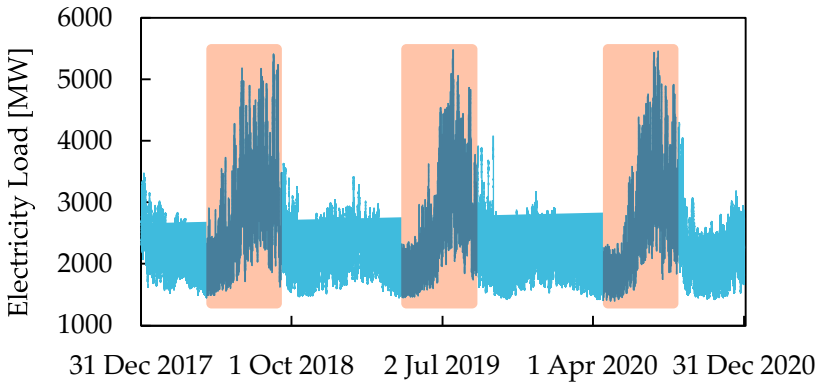


Figure 2.18: Seasonal drift that occurred from May to August in each year shows high electricity consumption.

To simulate CD, the evaluation range was set from 1 May 2020 to 31 August 2020, which is right after the scheduled federated learning (T7) was done to update the global model. Figure 2.19 shows the visualized scheduled federated learning with the evaluation range to measure the accuracy. In addition, on top of CD, this experiment also simulate MV ranging from 0% to 90% to see how the previously proposed unified solution by Zhou et al. [43] handle multiple input noise.

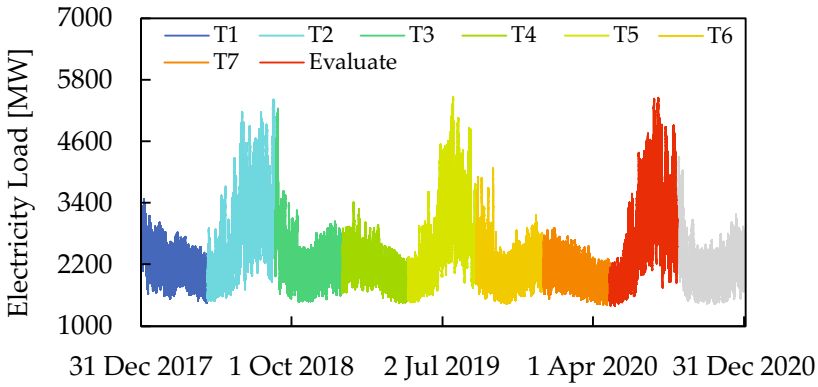


Figure 2.19: Scheduled federated learning done to update the global model and the evaluation range.

Table 2.5, Table 2.6, and Table 2.7 show the forecasting accuracies measured using the R^2, RMSE, and MAE from 1 May 2020 to 31 August 2020, which exhibit seasonal drift with MV being incremented by 10%. Similar to the previous experiment, as ML is non-deterministic, the experiment was repeated 3 times to get the averaged score.

Compared to the experiment done in Chapter 2.3.3, the R^2 scores shown in Table 2.5 seems to achieve better forecasting accuracy than the R^2 scores shown in Table 2.2, as it can handle up to 70% of MV before the R^2 score fell below 0.8. However, this observation is not supported by the RMSE and MAE scores.

Table 2.5: Coefficient of determination scores against the missing values and concept drift.

Missing Percentage [%]	**Coefficient of determination (R^2)**			
	Session			*Average*
	1	2	3	
0	0.9954	0.9949	0.9907	0.9937
10	0.9883	0.9872	0.9911	0.9889
20	0.9810	0.9813	0.9839	0.9821
30	0.9647	0.9675	0.9672	0.9665
40	0.9282	0.9350	0.9310	0.9314
50	0.8248	0.8215	0.8058	0.8174
60	0.8175	0.8160	0.8069	0.8135
70	0.6714	0.6453	0.6457	0.6541
80	0.2139	0.1426	0.1705	0.1757
90	-0.7257	-0.8565	-0.8080	-0.7967

Table 2.6: Root mean squared error scores against the missing values and concept drift.

Missing Percentage [%]	**Root Mean Squared Error (RMSE)**			
	Session			*Average*
	1	2	3	
0	60.271	63.692	85.927	69.963
10	96.471	101.03	84.192	93.90
20	123.01	122.07	113.31	119.46
30	167.84	160.92	161.80	163.52
40	239.29	227.68	234.61	233.86
50	373.87	377.35	393.55	381.59
60	381.53	383.11	392.51	385.72
70	512.01	531.93	531.62	525.19
80	791.91	827.04	813.46	810.80
90	1173.3	1217.0	1201.0	1197.1

Table 2.7: Mean absolute error scores against the missing values and concept drift.

Missing Percentage [%]	**Mean Absolute Error (MAE)**			
	Session			*Average*
	1	2	3	
0	38.546	40.540	45.196	41.427
10	56.912	59.131	51.066	55.703
20	64.370	65.576	58.629	62.858
30	79.443	78.672	74.099	77.405
40	99.220	97.335	94.042	96.87
50	133.03	131.45	130.06	131.51
60	154.53	152.61	151.95	153.03
70	221.71	224.03	224.00	223.25
80	395.98	410.87	408.00	404.95
90	784.09	815.97	813.63	804.56

These disparities between the R^2 with RMSE and MAE are due to the nature of R^2 itself, which measures how close the forecasted values matches with the real values instead of measuring the error. With 90% of MV, the RMSE score from Chapter 2.3.3 increases by 53.803%, and the MAE score increases by 97.542%. These results indicate that although the forecast closely follows the real values, it has more noise in it. This contributes to the worsened forecasting accuracy when dealing with CD.

Figure 2.20, Figure 2.21, and Figure 2.22 support these observations, even though the global model has undergone scheduled federated learning at T7 and additional retraining

triggered by Conformal Predictions to improve the forecast. The reason for this is the LSTM layers utilized by the Seq2seq LSTM are better at interpolating within the last known minimum and maximum values of normalization [59].

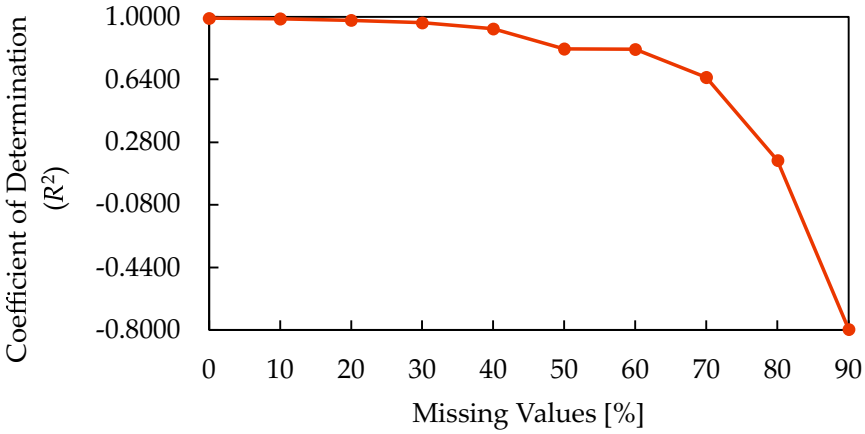


Figure 2.20: Averaged coefficient of determination scores against missing values and concept drift from 1 May 2020 to 31 August 2020.

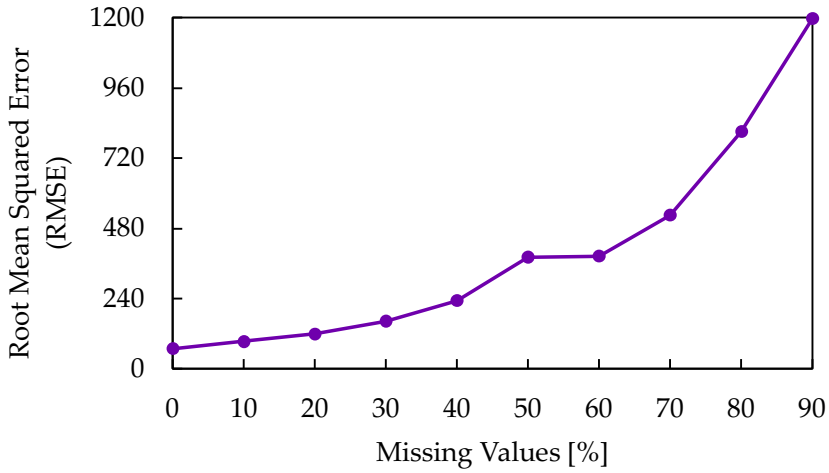


Figure 2.21: Averaged root mean squared errors score against missing values and concept drift from 1 May 2020 to 31 August 2020.

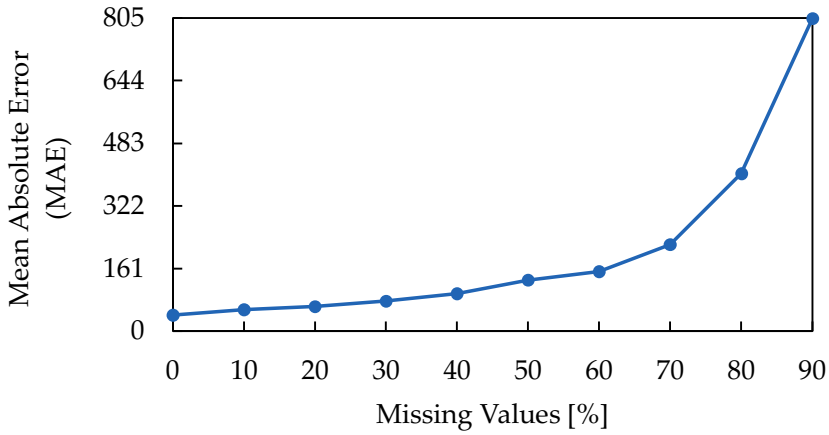


Figure 2.22: Averaged mean absolute errors score against missing values and concept drift from 1 May 2020 to 31 August 2020.

2.3.5 Adversarial Attacks

In this experiment, on top of MV and CD, Projected Gradient Descent (PGD) was used to create the adversarial attacks on the previously proposed unified solution by Zhou et al. [43]. PGD was proposed by Madry et al. [60], which uses iterative adversarial creation with random perturbation initialization to ensure the perturbation added is within the constraint using the projector operator $\Pi_{X,\epsilon}$. The difference from other commonly used adversarial attacks, such as Fast Gradient Sign Method (FGSM) and Basic Iterative Method (BIM), is not only it calculated the gradient and add perturbation iteratively to make the attack more robust, the added random perturbation initialization help PGD in exploring a wider range of effective adversarial examples.

With δ as the random perturbation taken from the uniform distribution $\mathcal{U}$ within the step ϵ, the formula to insert the random perturbation is defined in Equation (2.2). From there, with $\Pi_{X,\epsilon}$ to constrain the perturbation, the formula to obtain the adversarial sample generated with PGD is shown in Equation (2.3).

$$X_{PGD}^{(0)} = X + \delta \text{ where } \delta \in \mathcal{U}(-\epsilon, \epsilon) \tag{2.2}$$

$$X_{PGD}^{(j+1)} = \Pi_{X,\epsilon}\left(X_{PGD}^{(j)} + \alpha \cdot \text{sign}\left(\bigtriangledown_X J\left(\vartheta, X_{PGD}^{(j)}, y\right)\right)\right) \tag{2.3}$$

For this experiment, the PGD implementation is shown in Algorithm 2.6, the random perturbation initialization in $X_{PGD}^{(0)}$ was defined as $\delta \in \mathcal{U}(-\epsilon \cdot 0.01, \epsilon \cdot 0.01)$ instead, as ϵ

multiplied by 0.01 is already effective in improving the robustness of PGD.

Algorithm 2.6 Projected Gradient Descent Implementation

Input: Scaled sequences X, Surrogate model ϑ, Epsilon ϵ=0.05, $iteration$=10
Output: Scaled adversarial sequence X_{PGD}

Function `pgd_sample`$(X, \vartheta, \epsilon, iteration)$
  Get forecast: $y_{forecast} \leftarrow \vartheta$`.forecast`$(X)$
  Create a copy of the array: $X_{PGD} \leftarrow$ `copy`(X)
  Generate noise: $noise \leftarrow$ `random.uniform`$(-\epsilon \cdot 0.01, \epsilon \cdot 0.01,$ `len`$(X))$
  Add noise: $X_{PGD} \leftarrow X_{PGD}$+$noise$
  Get range: $minval$, $maxval \leftarrow$ `min`(X), `max`(X)
  Clip perturbations: $X_{PGD} \leftarrow$`clip`$(X_{PGD}, minval, maxval)$
  Calculate the alpha: $\alpha \leftarrow \epsilon / iteration$
  for j=0 to `len`$(iteration)$ **do**
    with `GradientTape()` **as** `tape` **do**
      Tape on X_{PGD}: `tape.watch`(X_{PGD})
      Get prediction: $y_{predict} \leftarrow \vartheta(X_{PGD})$
      Compute loss: $loss \leftarrow$ `mean_squared_error`$(y_{forecast}, y_{predict})$
    Compute gradient: $gradient \leftarrow$ `tape.gradient`$(loss, X_{PGD})$
    Insert perturbation: $X_{PGD} \leftarrow X_{PGD}$+$\epsilon \cdot$ `sign`$(gradient)$
    Clip perturbations: $X_{PGD} \leftarrow$`clip`$(X_{PGD}, minval, maxval)$
  return X_{PGD}
End Function

As this experiment need to emulate the black-box scenarios where the attackers does not have the access to the forecasting model, stacked LSTM architecture shown in Figure 2.23 was used to mimic the behavior of the forecasting model that is targeted for the attacks. Similar to the Seq2seq LSTM, it output 12 values that represent the load value for a 5 minutes interval and utilizes the same training parameters to simplify the process.

The architecture was chosen due to it being a commonly used architecture to forecast, with a stacking configuration to help the surrogate capture the intricate pattern.

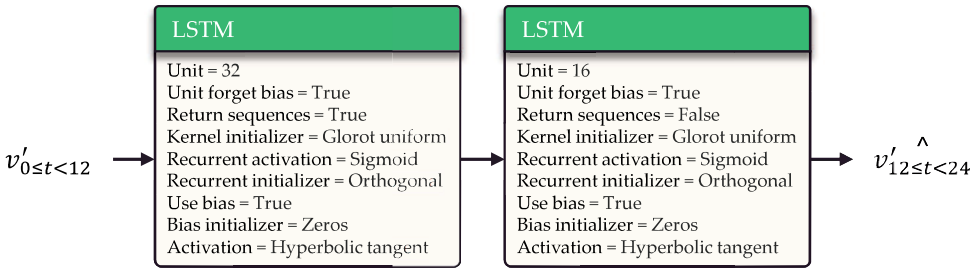


Figure 2.23: Stacked Long Short-Term Memory architecture was used to be the surrogate model.

Algorithm 2.7 shows the mechanism to train the surrogate models for each of their corresponding forecasting models, which use the dependent variable obtained $y_{forecast_{train}}$ from forecasting non-overlapped independent variable X_{train} as a training label. Table 2.8, Table 2.9, and Table 2.10 show the forecasting accuracies from 1 May 2020 to 31 August 2020 against MV, CD, and PGD-based attacks with ϵ=0.05.

Algorithm 2.7 Surrogate Model Training Mechanism

Input: Scaled train sequence v'_{train}, Forecasting Model m, ϑ
Output: Trained surrogate model ϑ

1: **Function** `surrogate_train`$(v'_{train}, m, \vartheta)$
2: Generate sequence: $X_{train}, \_ \leftarrow$ `data_sequencer`$(12, 12, v'_{train}, 12)$
3: Get forecast: $y_{forecast_{train}} \leftarrow m$`.forecast`$(X_{train})$
4: Train surrogate: ϑ`.train`$(X_{train}, y_{forecast_{train}})$
5: **return** ϑ
6: **End Function**

Table 2.8: Coefficient of determination scores against the missing values, concept drift, and adversarial attacks.

Missing Percentage [%]	**Coefficient of determination (R^2)**			
	Session			*Average*
	1	2	3	
0	0.9862	0.9836	0.9888	0.9862
10	0.9704	0.9824	0.9660	0.9729
20	0.9706	0.9702	0.9730	0.9713
30	0.9217	0.9368	0.9245	0.9277
40	0.8345	0.8209	0.8166	0.8240
50	0.6647	0.6454	0.5882	0.6327
60	0.6150	0.5732	0.5296	0.5726
70	0.4743	0.4328	0.4257	0.4443
80	0.0450	-0.0309	-0.0511	-0.0123
90	-0.7770	-0.8943	-0.8712	-0.8475

Table 2.9: Root mean squared error scores against the missing values, concept drift, and adversarial attacks.

Missing Percentage [%]	**Root Mean Squared Error (RMSE)**			
	Session			*Average*
	1	2	3	
0	105.020	114.46	94.554	104.68
10	153.703	118.56	164.64	145.64
20	153.09	154.14	146.86	151.36
30	249.86	224.57	245.45	239.96
40	363.33	377.97	382.52	374.61
50	517.22	531.90	573.18	540.76
60	554.19	583.48	612.58	583.42
70	647.58	672.70	676.85	665.71
80	872.82	906.86	915.69	898.46
90	1190.6	1229.3	1221.8	1213.9

Table 2.10: Mean absolute error scores against the missing values, concept drift, and adversarial attacks.

Missing Percentage [%]	**Mean Absolute Error (MAE)**			
	Session			*Average*
	1	2	3	
0	81.963	87.424	75.928	81.772
10	85.975	89.406	80.588	85.323
20	93.590	97.419	88.106	93.038
30	116.05	116.03	110.641	114.24
40	143.28	142.43	140.27	141.99
50	193.31	193.95	195.26	194.17
60	225.61	226.98	233.24	228.61
70	281.91	287.38	289.10	286.13
80	466.32	481.29	486.89	478.17
90	812.31	842.38	844.13	832.94

These results show that with adversarial attacks of intensity ϵ=0.05, it further worsens the forecasting accuracy when compared to the results in Chapter 2.3.4. As the unified solution proposed by Zhou et al. [43] could not handle live adversarial attacks, the attackers could use DDoS to mask the adversarial attacks, making it hard to be detected. However, as the MV rate increases, adversarial attacks effectiveness would also decrease due to the adversarial values being missing. Hence, a good balance between MV and adversarial attacks, such as 50% of MV with ϵ=0.05, could increase the MAE score by 47.647%, or by 62.663 [MW]. Figure 2.24, Figure 2.25, and Figure 2.26 support these observations.

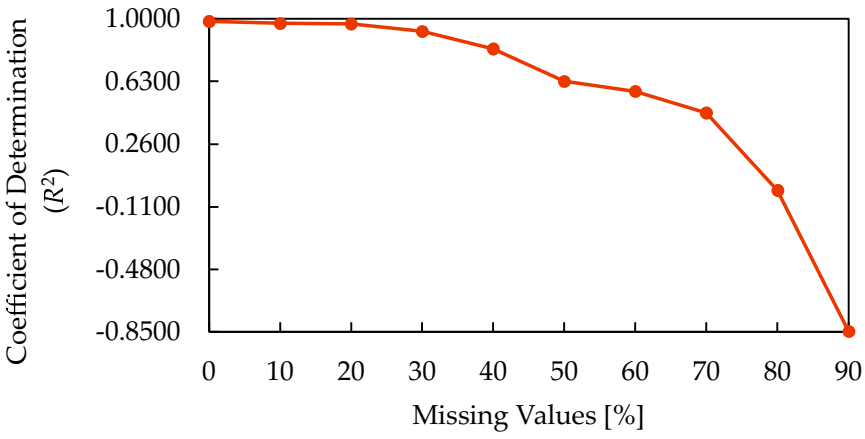


Figure 2.24: Averaged coefficient of determination scores against missing values, concept drift, and adversarial attacks from 1 May 2020 to 31 August 2020.

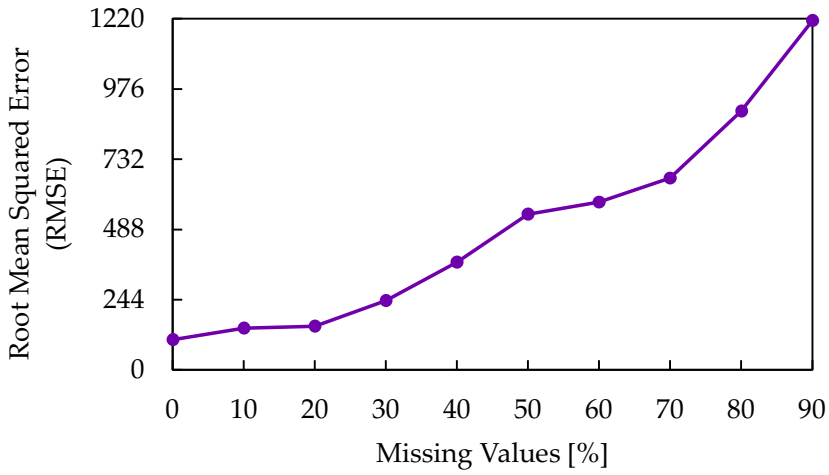


Figure 2.25: Averaged root mean squared errors score against missing values, concept drift, and adversarial attacks from 1 May 2020 to 31 August 2020.

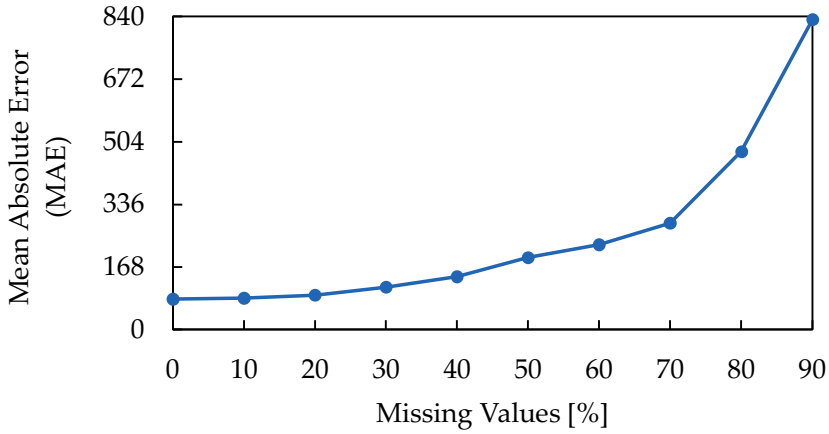


Figure 2.26: Averaged mean absolute errors score against missing values, concept drift, and adversarial attacks from 1 May 2020 to 31 August 2020.

2.3.6 Summary

These experiments show the weakness of the previously proposed solution by Zhou et al. [43]. Due to non-live adversarial attacks countermeasure and inability to address MV, the attacker could use these loopholes to hide the adversarial attacks with MV, making it hard to identify and address the issue.

Additionally, the LSTM layers used in the Seq2seq LSTM make it hard to adapt with CD, as LSTM-based models inherently perform badly when extrapolating beyond the minimum and maximum value range they were trained with.

2.4 Conclusion

Compared to previous research, the method proposed by Zhou et al. [43] employs simple countermeasures that fall short of providing a unified solution against MV, CD, adversarial attacks, and SPoF. This limitation arises because federated learning complicates the management of multiple ML-based data preprocessing across global models, restricting it to linear corrections and inadvertently accumulating errors from inadequate rectifications.

Furthermore, experiments reveal the weaknesses of Zhou et al.'s approach, particularly its non-live adversarial attack countermeasures and inability to effectively address MV. These gaps allow attackers to exploit loopholes, concealing adversarial attacks with MV, thus complicating detection and countermeasures. Additionally, the use of LSTM layers in the Seq2seq LSTM struggles with CD adaptation, as LSTM models inherently perform poorly when extrapolating beyond their trained value range.

The following chapter will address this issues, by providing the method to integrate multiple solutions into one, in addition to addressing the incompatibilities with input data, incompatibilities with other solutions, and countermeasures to address accumulation of correction errors.

3 Proposed Method

3.1 Chapter Introduction

ALCEN is a proposed distributed system housed in each server or IoT device hosting the ML model, and it is used for linking the forecasting models in a network to adapt to the external changes observed in the input data, such as MV, adversarial attacks, and CD. It also addresses the SPoF caused by some servers or IoT devices hosting the forecasting models not being online by sharing the model architecture or weights to create redundancy. This lets the other servers or IoT devices take over the forecasting work of the offline models. Figure 3.1 represents a level-0 data-flow diagram (DFD) interaction with external entities, where it interacts with the smart city management system to get and provide the forecasted data, in addition to the system administrator for requesting new training data or for the hyperparameters to be adjusted based on the operation report.

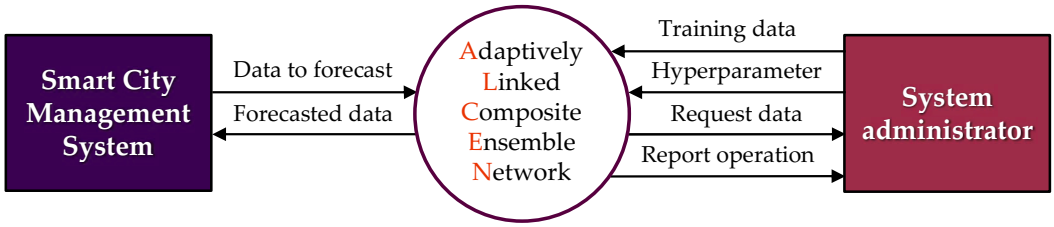


Figure 3.1: Context diagram representation to shows how Adaptively Linked Composite Ensemble Network interact with external entities.

An in-depth ALCEN's internal operations are shown in Figure 3.2, which shows a level-1 DFD to represent the logical flow inside the distributed system of ALCEN. The deployment is divided into two phases that might interchange if ALCEN requires retraining or if the system administrator updates the hyperparameters.

In the learning phase, the hyperparameters and the training data provided by the system administrator are used to train and harden the forecasting models. Once done, the abnormality detections for MV, adversarial attacks, and CD were prepared before moving to the forecasting phase. ALCEN can shift back to the learning phase if the accuracy is inadequate or if the system administrator provides new hyperparameter. In the forecasting phase, ALCEN links the necessary ML models or solutions based on the anomalies detected

in the input data and forecast. Then, it compares the current input with the previous forecast to see if retraining is necessary. If retraining is necessary, it will revert back to the learning phase, reset the weight, and request new training data. If not, it will send the logs to the system administrator for monitoring.

While event-driven architecture [61] of ALCEN handles automated training, hardening, anomaly detection, retraining, and failsafe against SPoF to ensure timely response to these issues, it relies on the system administrator to provide the hyperparameter to define how the system should behave.

Hyperparameters in ALCEN not only refer to the parameters used to set the training and define the ML model architecture but are also used to set the threshold detection for anomalies, retraining conditions, and countermeasures against overfitting to ensure optimal operation, which were summarized in Table 3.1. Although ALCEN deploys multiple ML models to fix the input data and harden the forecast, the same architecture parameters are used to simplify the ML model design across the solutions. This is to ensure transferability to other servers or IoT devices if one of them is offline.

Table 3.1: Hyperparameter used when initializing the Adaptively Linked Composite Ensemble Network.

Target	**Parameter**
Training	batch size
	number of epoch
	Adam learning rate
Detection	Drift threshold
	Adversarial threshold
	Missing percentage threshold
Retraining	Sliding window size
	Fixed training interval
	Acceptable accuracy threshold
Overfitting	Patience
	Minimum loss delta
Architecture	Kernel size
	Input length
	Output length
	Number of filter
	Number of input data

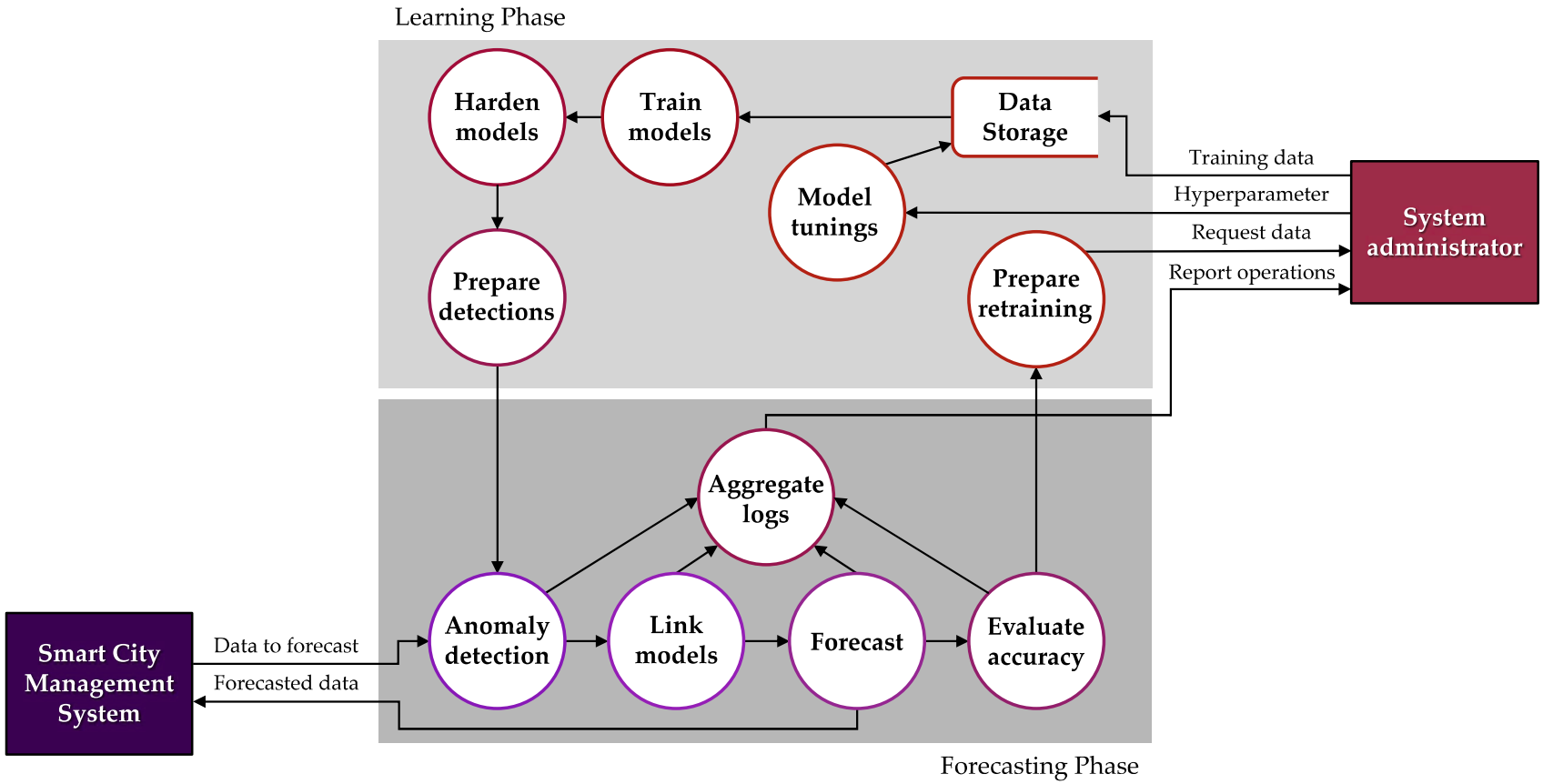


Figure 3.2: Logical level-1 data-flow diagram representation to shows the internal operations of Adaptively Linked Composite Ensemble Network.

After introducing the internal operation of ALCEN, this chapter defines the theories on leader election for selecting the server or IoT device to be the leader, the redundancy strategy to ensure multiple backups of training data, model architectures, and weights that are distributed to avoid SPoF, and composite models linking and arrangement rules were defined. Once the theories were defined, the ML model architecture and solutions for anomaly detection, fixing the input data, and hardening the forecasting models against MV, adversarial attacks, and CD were defined.

3.2 Theory

3.2.1 Leader Election

Originally proposed by Gracia [62], the bully algorithm is a method to elect a leader for coordinating the processes in the system [63]. The election works when the current leader has failed, triggering broadcasts from other servers or IoT devices, comparing who has the highest-numbered servers or IoT devices to be elected as a leader. It is called a "bully algorithm" due to higher-numbered servers or IoT devices "bullying" their way to the leadership.

In this thesis, bully algorithm is used to elect the servers or IoT devices, which are defined as a nodes to handle anomaly detections and delegating the task to fix the input data and hardening the forecast to other nodes. With the assumptions that each node has unique identifier, as shown in Figure 3.3 and they can communicate with each others, the following shows the high-level overview of how the bully algorithm works:

(i) Initialization: When a node does not receive the response from the leader or other nodes, it initiates an election process.

(ii) Response: When a node with higher numbering receives the election, it sends a "response" to declare its willingness to be the leader and starts its own election process.

(iii) Victory: If the initiating node does not receive any responses, it considers itself the leader and broadcasts "victory" message to all nodes. Else, it waits to receive a "victory" message from the new leader.

(iv) Failure: If a node fails and later restarts, it checks if it has missed any election. If it has the highest ID among available nodes, it can start a new election process to become the leader.

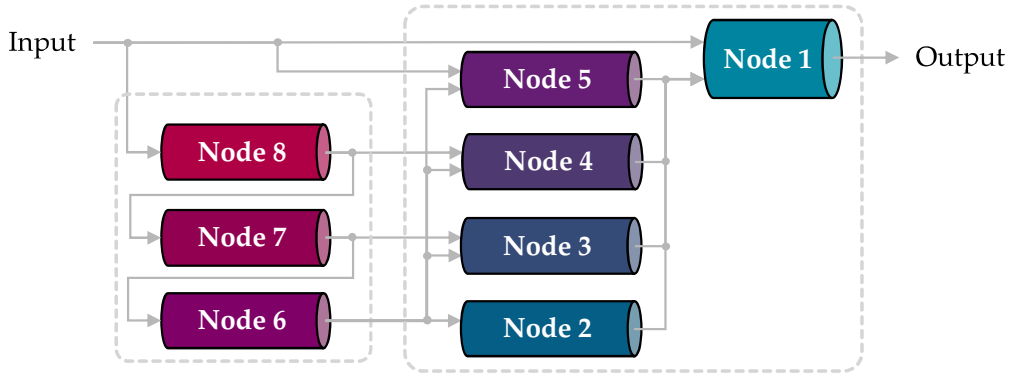
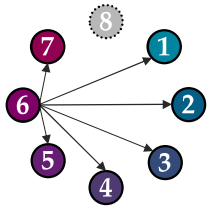
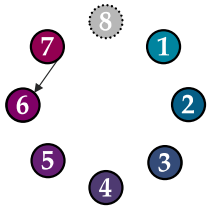


Figure 3.3: Node numbering for each server or Internet of Thing device that host the solutions to be elected as a leader, with higher-numbered node having higher priority.

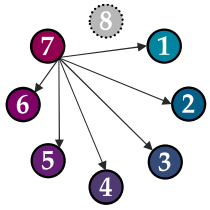
Figure 3.4 demonstrates the bully algorithm operation example when Node 8 is offline, triggering the election, in which only Node 7 will respond to its willingness to be a leader. From there, Node 7 will broadcast another election to confirm that there is no node that has a larger number than itself before announcing the victory.



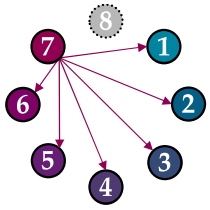
a) Node 6 broadcast election.



b) Node 7 broadcast response.



c) Node 7 broadcast election.



d) Node 7 broadcast victory.

Figure 3.4: Bully algorithm operation example when Node 8 is offline, triggering the election process.

A Python code implementation done for this thesis is in Listing 2. In addition to SPoF

prevention, this algorithm also helps identify the nodes that are offline, which guide the leader to assign the tasks that need to be done to other nodes.

Listing 2: Bully algorithm implementation.

```
class Node:

    def __init__(self, node_id, nodes):
        self.node_id = node_id
        self.nodes = nodes
        self.is_leader = False
        self.alive = True

    def start_election(self):
        print("Node {}: Starting an election.".format(self.node_id))
        higher_nodes = [
            node for node in self.nodes
            if node.node_id > self.node_id and node.alive
        ]
        if not higher_nodes:
            self.is_leader = True
            self.announce_victory()
        else:
            for node in higher_nodes:
                print(
                    "{} sends election to {}".format(
                        self.node_id,
                        node.node_id
                    )
                )
                node.receive_election(self)

    def receive_election(self, from_node):
        if self.alive:
            print(
                "{} received election from {}".format(
                    self.node_id,
                    from_node.node_id
                )
            )
            from_node.receive_response(self)
            self.start_election()

    def receive_response(self, from_node):
        print(
            "{} received response from {}".format(
                self.node_id,
                from_node.node_id
            )
        )
        self.is_leader = False

    def announce_victory(self):
        print("{} is the leader.".format(self.node_id))
        for node in self.nodes:
            if node.node_id < self.node_id and node.alive:
                print(
                    "{} sends victory to {}".format(
                        self.node_id,
                        node.node_id
                    )
                )
                node.receive_victory(self)

    def receive_victory(self, from_node):
        print(
            "{} acknowledges {} as leader.".format(
```

```
                self.node_id,
                from_node.node_id
            )
        )
        self.is_leader = False

    def simulate_crash(self):
        print("{} crashed.".format(self.node_id))
        self.alive = False

    def simulate_recovery(self):
        print("{} recovered.".format(self.node_id))
        self.alive = True
        self.start_election()


if __name__ == "__main__":

    # Simulate nodes
    nodes = [Node(i, []) for i in range(1, 8)]
    # Update nodes list for each node
    for node in nodes:
        node.nodes = nodes
    # Start an election from a node
    nodes[0].start_election()
    # Simulate a crash of the current leader and recovery
    nodes[4].simulate_crash()
    nodes[0].start_election()
    nodes[4].simulate_recovery()
```

3.2.2 Redundancy Strategy

To distribute the training data for retraining and ML model data to replicate the solutions, which include the architectures and weights, a replication with controlled distribution [64] was implemented to evenly distribute the data to create redundancy.

Data replication refers to the process of creating a copy of data in another location with the goal of creating a backup for redundancy in the event of one of the nodes becoming offline due to technical failures or cyberattacks. This ensures ALCEN can continue running even if one of the nodes is inaccessible. However, simply copying the data in another location is insufficient, as it must be done in a controlled manner that can guarantee that the system can still run in the event some of the nodes are offline. By managing the replication rate that adheres to the storage limitation of each node, true distributed data replication with redundancy can be achieved.

By representing the specific data to be replicated as j with D as set of data, and specific node in the distributed system to store the data as k with N as set of nodes, the objective function to optimize the data distribution as shown in Equation (3.1). The optimized

result must adhere to the replication rate R in Equation (3.2) and storage constraint C for each node in Equation (3.3). Also, each data j can be stored once on any specific node k, as shown in Equation (3.4) to avoid duplication on the same node.

$$\text{Minimize:} \quad \sum_{j \in D} \sum_{k \in N} D(j,k) \tag{3.1}$$

$$\text{Subject to:} \quad \sum_{k \in N} D(j,k) = R, \quad \forall j \in N \tag{3.2}$$

$$\sum_{j \in D} D(j,k) \leq C(k), \quad \forall k \in N \tag{3.3}$$

$$D(j,k) \in \{0,1\}, \quad \forall i \in D, j \in N \tag{3.4}$$

A Python code implementation done for this thesis is in Listing 3, which is based on a greedy implementation of controlled distribution replication. With four months of training data and eight ML model data, the data was distributed to the nodes, with replication of R=4 and storage constraint for each node of C=7.

Listing 3: Greedy implementation of controlled distribution replication.

```
from random import shuffle, seed
from typing import List, Set


# Node simulation
class Node:

    def __init__(
        self,
        id: int,
        capacity: int
    ) -> None:

        self.id: int = id
        self.capacity: int = capacity
        self.data: Set[str] = set()

    def can_store(
        self
    ) -> bool:

        return len(self.data) < self.capacity

    def store_data(
        self,
        data_id: str
    ) -> bool:

        if self.can_store():
            self.data.add(data_id)
            return True

        return False


# Controlled replication implementation
```

```python
def controlled_replication(
    nodes: List[Node],
    data_ids: List[str],
    num_replicas: int,
    randomization_seed: int = 2025
) -> None:

    for data_id in data_ids:

        seed(randomization_seed)
        shuffle(nodes)
        replicas_stored = 0
        # Extract the number from the data_id
        data_number = int(data_id.split('_')[-1])

        for node in nodes:
            if replicas_stored < num_replicas:
                # Avoid storing in node with the same id
                if node.id != data_number and node.store_data(data_id):
                    replicas_stored += 1

        print(f"Data {data_id} stored {replicas_stored} times.")


if __name__ == "__main__":

    # Define nodes with equal storage capacities
    nodes: List[Node] = [
        Node(id=1, capacity=7),
        Node(id=2, capacity=7),
        Node(id=3, capacity=7),
        Node(id=4, capacity=7),
        Node(id=5, capacity=7),
        Node(id=6, capacity=7),
        Node(id=7, capacity=7),
        Node(id=8, capacity=7),
    ]

    # Replicate a list of data items with controlled distribution
    model_to_replicate: List[str] = [f"model_{i}" for i in range(1, 10)]
    training_data_to_replicate: List[str] = [f"month_{i}" for i in range(1, 4)]
    data_to_replicate: List[str] = model_to_replicate + training_data_to_replicate
    replication_factor: int = 4

    controlled_replication(
        nodes=nodes,
        data_ids=data_to_replicate,
        num_replicas=replication_factor,
        randomization_seed=2025
    )
```

Table 3.2 shows the data distribution outcomes subject to the R and C limitation with no duplication on the same node. Additionally, this implementation allows the system to tolerate up to $R-1$ node failures with at least one copy of each piece of data still being accessible.

Table 3.2: Data stored in each nodes computed by greedy implementation of replication with controlled distribution.

Node	Data Storage						
	1	*2*	*3*	*4*	*5*	*6*	*7*
1	model_4,	model_5,	model_8,	model_9,	month_3		
2	model_3,	model_4,	model_6,	model_7,	model_8,	month_3	
3	model_1,	model_4,	model_5,	model_8,	model_9,	month_2	
4	model_3,	model_7,	model_8,	month_2,	month_3		
5	model_1,	model_2,	model_4,	model_6,	model_9,	month_1,	month_3
6	model_2,	model_3,	model_7,	month_1,	month_2		
7	model_1,	model_2,	model_5,	model_6,	model_9,	month_1,	month_2
8	model_1,	model_2,	model_3,	model_5,	model_6,	model_7,	month_1

This implementation, in conjunction with the bully algorithm solve the SPoF issue by distributing the essential data and leadership role, allowing decentralized system operation in addition to failsafe mechanism to keep the forecast running.

3.2.3 Composite Models Arrangement

As mentioned in Chapter 1, the reason why existing unified implementations, such as the implementation done by Zhou et al. [43], rely on simple countermeasures or circumvention is due to the incompatibilities with the input data, other solutions, and sequential operation. To make it easier for visualizing the difference between the input and output data, the data that need to be forecasted is denoted as X, the forecasted data is denoted as y, and the number the data has been corrected is denoted with an asterisk mark (*).

The first issue is the input data incompatibilities due to some solutions requiring them to be preprocessed with other solutions first to be effective. For example, a forecasting model hardened against adversarial attacks cannot handle MV without it being imputed, as shown in Figure 3.5. To solve this, the missing values imputation must be done first before the hardened forecasting model could handle the adversarial attacks, as shown in Figure 3.6. This issue shows that the solutions must be connected based on their priority to properly solve the issues.

Figure 3.5: Forecasted data will yield a bad accuracy if the missing values in the input data are not dealt with.

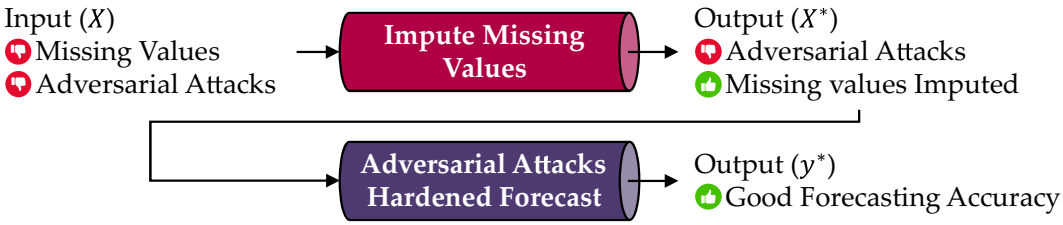


Figure 3.6: Imputing the missing values before forecasting it with adversarial attacks hardened forecast yield a good accuracy.

The second issue is the incompatibilities with other solutions targeted at the same issue, as some solutions require the input data to be unchanged for it to work. For example, a forecasting model hardened against MV will be able to forecast input data that has MV in it without much of an issue, as shown in Figure 3.7. However, when paired with the MV imputation, the accuracy will not improve as the forecasting model does not know the position of MV in the input data, as shown in Figure 3.8. To solve this, the model is to be modified to take the original and imputed data to improve upon imputation imperfection, which is the improvement technique in my previous research [65], as shown in Figure 3.9.

Figure 3.7: Forecasting model hardened against missing values work get good accuracy when forecasting the input data with missing values.

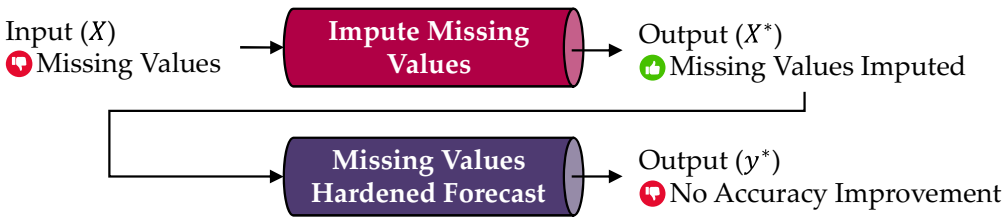


Figure 3.8: Forecasting model hardened against missing values paired with the missing values imputation will not improve the forecasting accuracy.

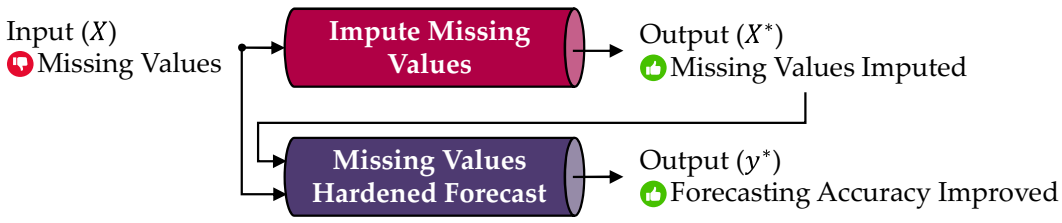


Figure 3.9: Parsing the original and imputed data to to the hardened forecasting model help it to identify the imputation imperfection.

By considering the compatibilities between the input data, the input data that has MV, adversarial attacks, and CD must be imputed first, followed by the correction to address adversarial attacks, and trend adjustment to solve CD. By processing the input data in this sequence, the data compatibility issues should be rectified. Following this, the pre-repaired input data are parsed to their corresponding hardened forecasting models to further improve the forecasting accuracies to address the imperfection done when fixing the data, solving the incompatibilities with each solution. However, this application is incomplete due to the output from these hardened forecasting models only improving the forecast for their designated problems, as shown in Figure 3.10 where there are multiple outputs that need to be combined.

While the repairs on the input data could only be done in a predefined sequential manner, the hardened forecasting models must be done in parallel, causing the hardened forecast running in parallel to be incompatible with the sequential data repair. To solve this, based on the number of issues found in the input data, a weighted average or meta model could be utilized to combine the results, as shown in Figure 3.11.

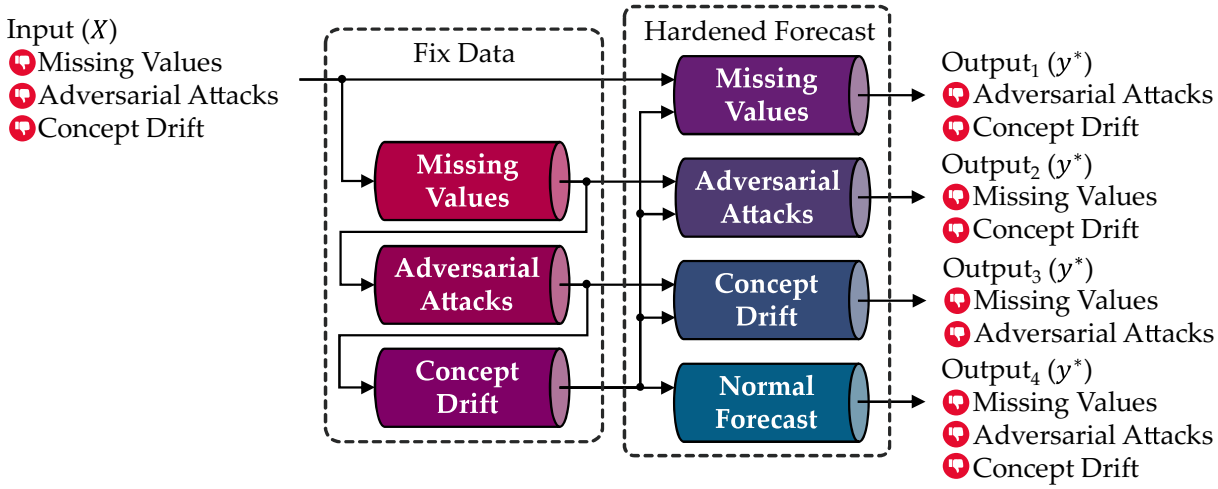


Figure 3.10: Difficulty of combining multiple hardened forecasting models running in parallel into one.

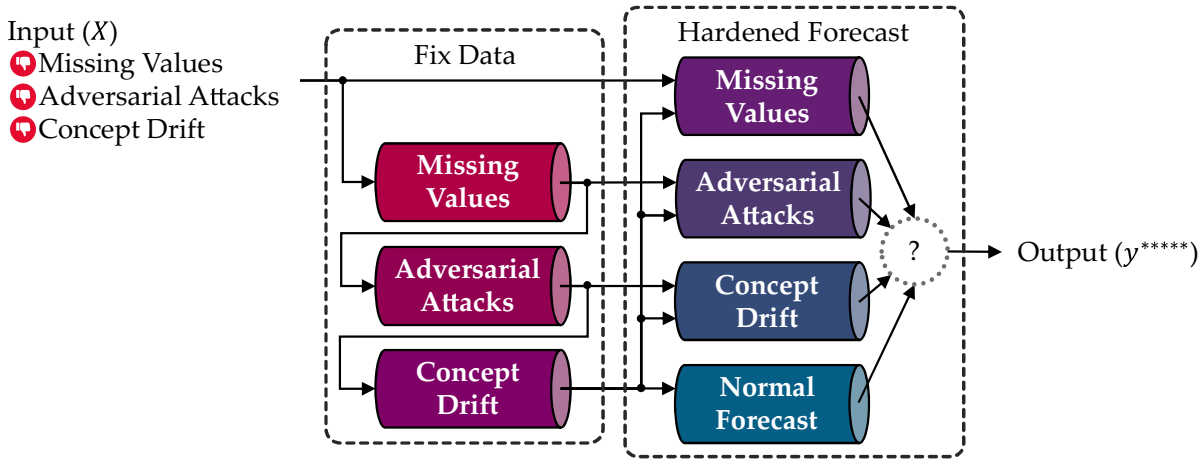


Figure 3.11: A mechanism that combines the outputs from multiple hardened forecasting models running in parallel further improves the forecasting accuracy.

These issues that I have identified explain the difficulties of implementing more complex solutions into a unified solution, hence the sequential approach taken by Zhou et al. [43] to avoid the compatibility issues with other solutions and complexities when dealing with federated learning. With the SPoF issue being addressed using the bully algorithm and replication with controlled distribution, the compatibility issues for solving MV, adversarial attacks, and CD are resolved by using the solution arrangement template I have proposed for ALCEN to follow, as shown in Figure 3.12. This template uses the concept of a cascading ensemble model to sequentially fix the input data if needed, and the stacking

ensemble ensures the output from the hardened forecasting models could be combined to improve the accuracy.

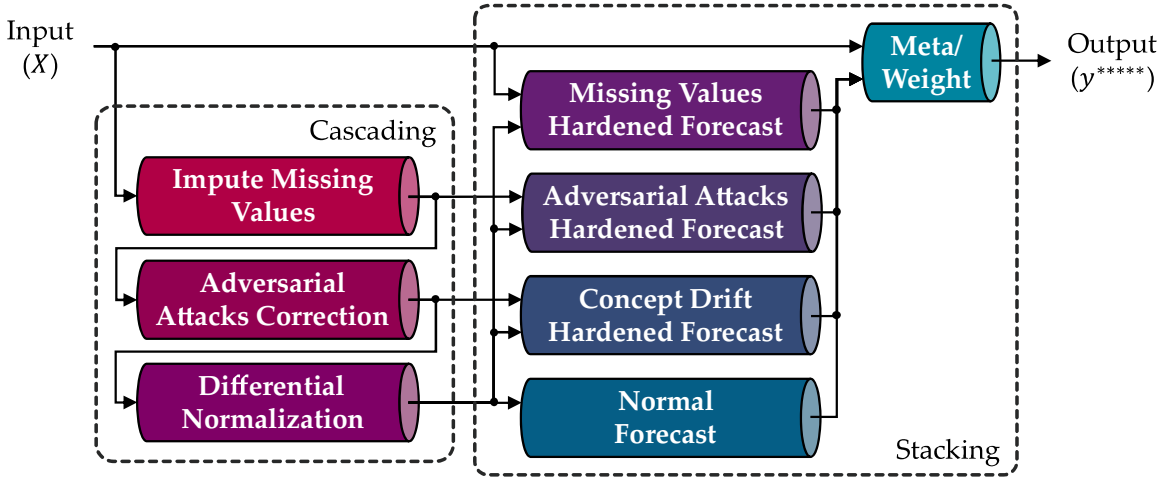


Figure 3.12: A mechanism that combines the outputs from multiple hardened forecasting models running in parallel further improves the forecasting accuracy.

The originality from the standard cascading implementation [66] is the output from each stage of correction is parsed. In addition, the originality from the standard stacking ensemble [67] is that the unprocessed input data are parsed as input data to identify the imperfection from the repairs.

3.3 Anomaly Detections

3.3.1 Missing Values

The missing rate is calculated by counting NaN in the array that needs to be forecasted X_i and dividing it with the length of X_i, as shown in the Algorithm 3.1. Although simple, this implementation not only helps to detect if there is MV in the array, where $missing\_rate > 0$ indicates MV, but also helps to intuitively know the severity of the MV due to $missing\_rate$ representing the percentage of MV in the decimal points. Once detected, X_i was imputed first before being checked for the adversarial attacks.

Algorithm 3.1 Missing Rate Measurement

Input: Scaled input sequence X_i
Output: Missing rate *missing_rate*
1: **Function** `check_missing_rate`(X_i)
2: Count the missing values: *sum_missing* $\leftarrow$ `sum`(`isnan`(X_i))
3: Count elements in the array: *sum_element* $\leftarrow$ `sum`(X_i)
4: Calculate missing rate: *missing_rate* $\leftarrow$ *sum_missing*/*sum_element*
5: **return** *missing_rate*
6: **End Function**

3.3.2 Adversarial Attacks

Adversarial attack detection is the most challenging issue to detect, as the change is not as apparent as MV or CD. One of the commonly used detection techniques relies on an autoencoder to detect subtle changes between the original and the reconstructed time series data [68, 69]. However, the first technique relies on utilizing the latent information contained in the time series itself, which might be inadequate without relying on an additional model to make the decision. In addition, the second technique combines multiple time series to create a makeshift image, which is too complex.

In this thesis, a Filter Encoder (FE) model is used to detect the subtle change, which is based on my previous research of creating a "filter" to correct the input data from the adversarial attacks [37]. Figure 3.13 shows the model implementation, where the causal convolutional network with rectified linear unit (ReLU) is repeated to reduce the dimension of concatenated m number of features with a length of 12 ($12 \times m$) to one (12×1). The reduction was achieved by setting the kernel width to 2, and the paddings are only utilized on the top and bottom of the 2-dimensional concatenated data. The template to implement the FE architecture is in Listing 4.

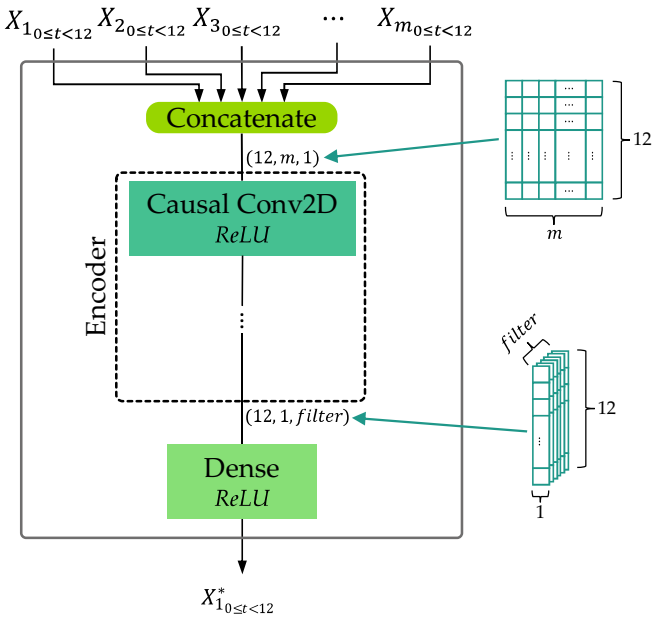


Figure 3.13: Filter Encoder model implementation that use strongly correlated time series as an additional data to improve the reconstruction accuracy.

Listing 4: Implementation template for Filter Encoder.

```python
from tensorflow.keras.layers import ZeroPadding2D, Activation, Concatenate, Flatten
from tensorflow.keras.layers import Reshape, Input, Dense, Conv2D
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model


def filter_encoder(
    kernel_size: int,
    input_length: int,
    filter_number: int,
    output_length: int,
    feature_number: int
) -> Model:

    # Create the input layer for each features
    input_layers = []
    for _ in range(feature_number):
        input_layers.append(Input(shape=(input_length,)))
    # Concatenate along the second axis to create a 2D tensor
    hidden_layer = Concatenate(axis=1)(input_layers)
    # Reshape the 2D tensor
    hidden_layer = Reshape((input_length, feature_number, 1))(hidden_layer)
    # Connect the 2D tensors to the stacked custom causal Conv2D layers
    for i in range(feature_number - 1):
        # Add 2D padding to the front of the 2D tensor for simulating causal padding
        hidden_layer = ZeroPadding2D(
            padding=(
                (
                    # Pad top
                    int((kernel_size - 1) * (i + 1) / 2),
                    # Pad bottom
                    int((kernel_size - 1) * (i + 1) / 2),
                ), (0, 0)
            ),
```

```
        )(hidden_layer)
        # Connect to the Conv2D layer
        hidden_layer = Conv2D(
            # Number of learned kernels
            filters=filter_number,
            # 2D Kernel
            kernel_size=(
                kernel_size,
                # Reduce the number of features (width) by 1
                2
            ),
            # Set the dilation_rate to increase by i for each Conv2D layer
            dilation_rate=(
                # Length
                i + 1,
                # Width
                1
            ),
            # Hardcoded the stride to (1, 1) to set the kernel's x/y-axis movement to 1
            strides=(1, 1),
            # Hardcoded Conv2D's internal padding to "valid" for disabling it
            padding="valid"
        )(hidden_layer)
        # Connect to the activation layer
        hidden_layer = Activation("relu")(hidden_layer)
    # Flatten the features extracted from the stacked causal Conv2D
    hidden_layer = Flatten()(hidden_layer)
    # Connect to the dense layer
    hidden_layer = Dense(int(input_length * filter_number / 2))(hidden_layer)
    # Connect to the activation layer
    hidden_layer = Activation("relu")(hidden_layer)
    # Connect to the output_layer
    output_layer = Dense(output_length)(hidden_layer)

    # Define the model
    model = Model(inputs=input_layers, outputs=[output_layer])
    # Compile the model
    model.compile(
        loss=MeanSquaredError(),
        metrics=[RootMeanSquaredError()],
        optimizer=Adam(learning_rate=0.001),
    )

    return model
```

The novelty of my proposed implementation is the simplicity and efficiency by reducing the number of features while maintaining the length of sequences, ensuring the correlation between the features and causality between the elements are properly utilized.

Figure 3.14 shows the FE implementation to correct the input data targeted for forecast α into α^*, before measuring the deviation with RMSE to check if it exceeds the threshold. This help to automate the detection with ML-driven decision-making [70].

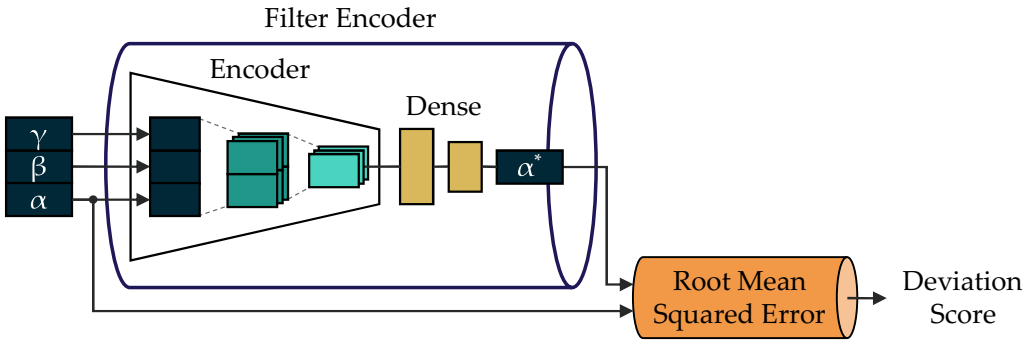


Figure 3.14: Filter Encoder is used to correct the target input forecast before measuring the root mean squared error score to determine if it exceed the threshold.

Once detected, the adversarial attacks in the input data was rectified before being checked for the CD.

3.3.3 Concept Drift

To detect CD, I have automated the overlapping histograms visualization comparison used in my previous research [59] to detect the CD by considering how many scaled current data distributions fall into bins that have low counts or are outside the scaled train data distribution. Figure 3.15 shows the example to detect drifted data, where any data that fall into the yellow area are marked as drifted data, and data that fall in the less populated train data bin will have a high chance to be classified as drifted data.

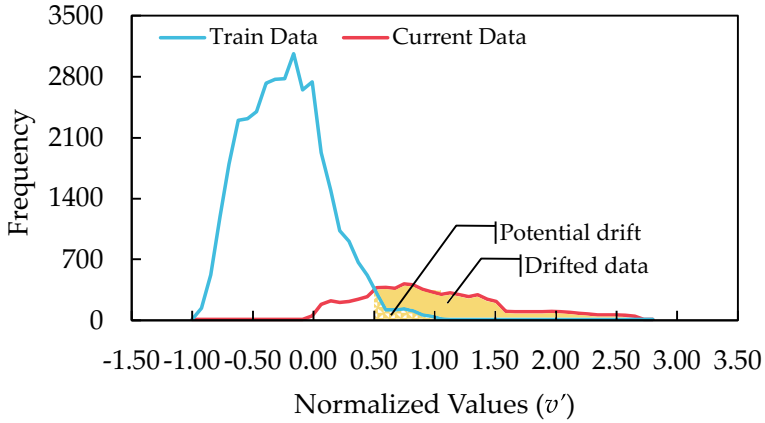


Figure 3.15: The visualization technique to detect the concept drift is automated by considering the positioning of scaled current data distributions against the train data distribution.

Although simple, it is effective in identifying the gradual and recurring drift that may

happen due to the seasonal changes. Algorithm 3.2 is the implementation for measuring the weight on the current input sequence distribution X_i, where weight lower than the threshold is recognized as potentially drifted data.

Algorithm 3.2 Drift Distribution Detection

Input: Training sequences X_{train}, Input sequence X_i, Threshold adjustment *adjust*
Output: Drift detected *detected*

Function `drift_distribution`(X_i)
  Create histogram: $counts, bin\_edges \leftarrow$ `histogram`(X_{train})
  Estimate weights for each bin: $weights \leftarrow counts/$`sum`($counts$)
  Define weights for drifted data: $out\_of\_bound \leftarrow 0$
  create list to store the current data weights: $new\_data\_weight \leftarrow []$
  for *element* in X_i **do**
    Find bin index: $bin\_index \leftarrow$ `digitize`($element, bin\_edges$) $- 1$
    if $bin\_index < 0$ **or** $bin\_index \geq len(weights)$ **do**:
      Append out-of-bound weight: new_data_weight.`append`(out_of_bound)
    else do
      Append normal weight: new_data_weight.`append`($weights[bin\_index]$)
  Get the average weight: $average\_weight \leftarrow$ `mean`(new_data_weight)
  Define threshold: $threshold \leftarrow$ `mean`($weights$) $\times$ *adjust*
    if $average\_weight < threshold$ **do**:
      Drift detected: *detected* $\leftarrow$ `True`
    else do
      Drift detected: *detected* $\leftarrow$ `False`
  return *detected*
End Function

Once detected, the CD in the input data was rectified. Based on the anomalies found in the input data, ALCEN will adjust the connection between the solutions to improve the forecasting accuracy.

3.4 Solutions

3.4.1 Missing Values

The solution for imputing the MV relies on the FE introduced in Chapter 3.3.2 to impute the MV in the sequence. To achieve this, instead of training it with clean independent variables $(X_{1_{train}}, X_{2_{train}}, X_{3_{train}}, \cdots, X_{m_{train}})$ to get $X_{1_{train}}$, Algorithm 2.5 was used to simulate MCAR in independent variables, where the MV percentage ranged from 0% to 90%, which were masked with -1 for the model to identify.

Algorithm 3.3 was used to generate independent variables that were repeated ten times, where for each repetition, the MV percentage raised by 10% and masked with -1. In addition, the dependent variable is also stacked on itself to match the length of the newly generated independent variables using the algorithm 3.4. This is the technique I developed in my previous research [29, 65] to harden the forecasting model against a high percentage of MV by purposely inducing MV masked with -1 to help the ML model to identify and work around the MV.

Algorithm 3.3 Stacked Independent Variables with Masked Missing Values

Input: Independent variables $X_{multivariate} = [X_{1_{train}}, X_{2_{train}}, X_{3_{train}}, \cdots, X_{m_{train}}]$
Output: Independent stacked variables *stacked_X*

Function `mv_x_sample`($X_{multivariate}$)
  Initialize empty array: $stacked\_X \leftarrow []$
  for *sequences* **in** $X_{multivariate}$ **do**
    Initialize empty array: $X\_temp \leftarrow []$
    for $i = 0$ **to** 0.9 **step** 0.1 **do**
      Simulate missing values: X_temp.`append`(`mcar_generator`($sequences, i$))
    Mask missing values with -1: $X\_temp \leftarrow$ `mask`($X\_temp, -1$)
    Stack and save: $stacked\_X$.`append`(`vstack`(X_temp))
  return *stacked_X*
End Function

Algorithm 3.4 Stacked Dependent Variables

Input: Dependent variables $X_{1_{train}}$
Output: Dependent stacked variables $stacked\_X_{1_{train}}$

Function `mv_y_sample`($X_{multivariate}$)
  Vertical stack the dependent variables: $stacked\_X_{1_{train}} \leftarrow$ `vstack`($[X_{1_{train}}] \times 10$)
  return $stacked\_X_{1_{train}}$
End Function

As this solution only works by reconstructing the input data for imputation, another solution for hardening the forecasting model is needed to further improve the accuracy. Figure 3.16 shows the Multivariable Convolution Encoder (MCE), which is the expanded version of the FE designed for regression analysis design from my previous research [37]. It works by leveraging the encoder capability to fix input data and TCN to forecast the corrected time series. Although the encoder block inside the MCE has similar function to FE, the causal padding is only utilized on the top of the 2-dimensional concatenated

sequence to simulate the time flow. The template to implement the MCE architecture is in Listing 5.

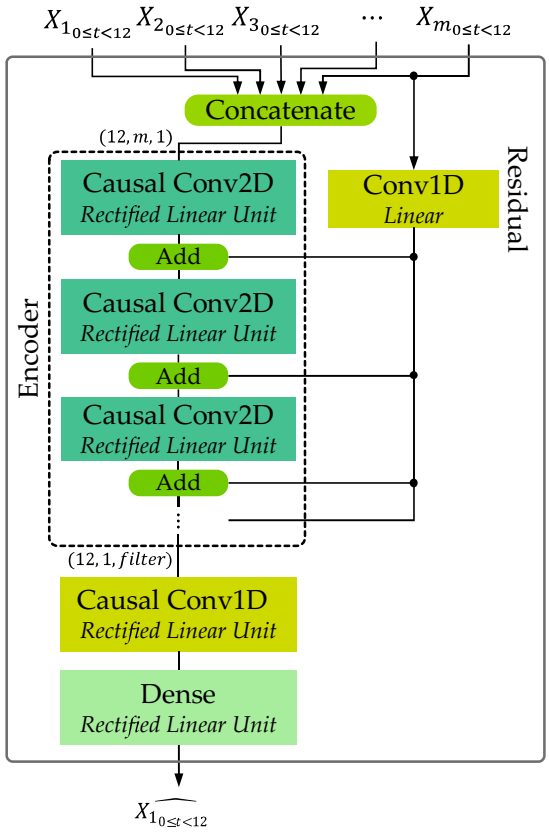


Figure 3.16: Multivariable Convolutional Encoder implementation that leverage the encoder capability to fix input data and the temporal convolutional network to forecast.

Listing 5: Implementation template for Multivariable Convolutional Encoder.

```
from keras import models, losses, utils, layers, metrics, backend, callbacks, optimizers
# import common libraries
from contextlib import redirect_stdout
from typing import Union, Tuple, List
from numpy import array, floating
from os.path import exists, isdir
from numpy.typing import NDArray
from pandas import DataFrame
from os import makedirs


def multivariable_convolutional_encoder(
    kernel_size: int,
    input_length: int,
    filter_number: int,
    output_length: int,
    feature_number: int,
    causality: bool = True,
) -> Model:

    # Create the input layers for each features
    input_layer = []
```

```
    for _ in range(feature_number):
        # Append the input with the following shape
        input_layer.append(
            layers.Input(
                # shape
                shape=(input_length,)
            )
        )
    # Concatenate along the second axis to create a 2d tensor (None, input_length *
    feature_number)
    hidden_layer = layers.Concatenate(axis=1)(input_layer)
    # Reshape the 2d tensor to (None, input_length, feature_number, 1)
    hidden_layer = layers.Reshape(
        (
            # Input length
            input_length,
            # Number of features
            feature_number,
            # Hardcoded to 1 channel only
            1
        ),
    )(hidden_layer)
    # add skip connection
    shortcut_layer = layers.Reshape(
        (
            # Input length
            input_length,
            # Number of features
            1,
            # Number of filter
            1
        ),
    )(input_layer[0])
    # Use Conv2D to reshape the shortcut_layer to match the shape of the encoder output
    shortcut_layer = layers.Conv2D(
        filters=filter_number,
        kernel_size=(1, 1),
        padding="same",
    )(shortcut_layer)
    # Connect the 2d tensors to the stacked padding and Conv2D layers
    for i in range(feature_number - 1):
        if causality is True:
            # Simulate causal padding
            hidden_layer = layers.ZeroPadding2D(
                # add 2D zeros padding
                padding=(
                    (
                        # Add 2D zeros to the front side of the 2D arrays
                        (kernel_size - 1) * (i + 1),
                        # Add no zeros to the bottom side of the 2D arrays
                        0
                    ), (
                        # Add no zeros to the left side of the 2D arrays
                        0,
                        # Add no zeros to the right side of the 2D arrays
                        0
                    )
                ),
            )(hidden_layer)
        else:
            # Simulate same padding
            hidden_layer = layers.ZeroPadding2D(
                # Add 2D zeros padding
                padding=(
                    (
                        # Add 2D zeros to the front side of the 2D arrays
                        int((kernel_size - 1) * (i + 1) / 2),
                        # Add 2D zeros to the bottom side of the 2D arrays
```

```
                        int((kernel_size - 1) * (i + 1) / 2),
                    ), (
                        # Add no zeros to the left side of the 2D arrays
                        0,
                        # Add no zeros to the right side of the 2D arrays
                        0
                    )
                ),
            )(hidden_layer)
        # Connect to the Conv2D layer
        hidden_layer = layers.Conv2D(
            # Number of learned kernels
            filters=filter_number,
            # Parsed 1d kernel_size
            kernel_size=(
                kernel_size,
                # Reduce the number of features by 1
                2
            ),
            # Set the dilation_rate to increase by i for each Conv2D layer
            dilation_rate=(
                # Length
                i + 1,
                # Width
                1
            ),
            # Hardcoded the stride to (1, 1)
            strides=(
                1,
                1
            ),
            # Hardcoded the Conv2D's internal padding to "valid" to disable it
            padding="valid",
            # Hardcoded the initialized bias to zeros
            bias_initializer="zeros",
            # Hardcoded the initialized kernel weight using the glorot uniform
    distribution
            kernel_initializer="glorot_uniform",
        )(hidden_layer)
        # Connect to the activation layer
        hidden_layer = layers.Activation(
            # Activation function
            "relu",
        )(hidden_layer)
        # Combine the shortcut connection with the hidden_layer
        hidden_layer = layers.Add()([hidden_layer, shortcut_layer])
    # Reshape layer
    hidden_layer = layers.Reshape(
        target_shape=(input_length, filter_number),
    )(hidden_layer)
    # Connect to the Conv1D layer
    hidden_layer = layers.Conv1D(
        # Number of learned kernels
        filters=int(filter_number / 2),
        # Use the parsed 1d kernel_size as it is
        kernel_size=kernel_size,
        # Set the dilation_rate to match with the feature_number
        dilation_rate=feature_number,
        # Hardcoded the stride to 1 to set the kernel's y-axis movement to 1
        strides=1,
        # Use same padding to maintain the dimension
        padding="same",
        # Hardcoded the initialized bias to zeros
        bias_initializer="zeros",
        # Hardcoded the initialized kernel weight using the glorot uniform distribution
        kernel_initializer="glorot_uniform"
    )(hidden_layer)
    # Connect to the activation layer
```

```
    hidden_layer = layers.Activation("relu")(hidden_layer)
    # Flatten the features extracted from the stacked causal Conv2D
    hidden_layer = layers.Flatten()(hidden_layer)
    # Connect to the dense layer
    hidden_layer = layers.Dense(
        # Set the number of unit to the half of the amount of the flattened layer
        int(input_length * filter_number / 4),
        # Hardcoded the initialized bias to zeros
        bias_initializer="zeros",
        # Hardcoded the initialized kernel weight using the glorot uniform distribution
        kernel_initializer="glorot_uniform",
    )(hidden_layer)
    # Connect to the activation layer
    hidden_layer = layers.Activation("relu",)(hidden_layer)
    # Connect to the output_layer
    output_layer = layers.Dense(
        # Set the number of unit to match with the output length
        output_length,
        # Hardcoded the initialized bias to zeros
        bias_initializer="zeros",
        # Hardcoded the initialized kernel weight using the glorot uniform distribution
        kernel_initializer="glorot_uniform",
    )(hidden_layer)

    # Define the model
    model = models.Model(
        inputs=input_layer,
        outputs=output_layer
    )
    # Compile the model
    model.compile(
        loss=losses.MeanSquaredError(),
        metrics=[metrics.RootMeanSquaredError()],
        optimizer=optimizers.Adam(learning_rate=0.001),
    )

    return model
```

To harden, eleven MCE models were prepared, where the first ten were trained with different MV percentages ranging from 0% to 90%, and the eleventh was trained with the input and output from the first ten to combine the outputs. This arrangement shown in Figure 3.17 is my original implementation of stacking ensemble [65].

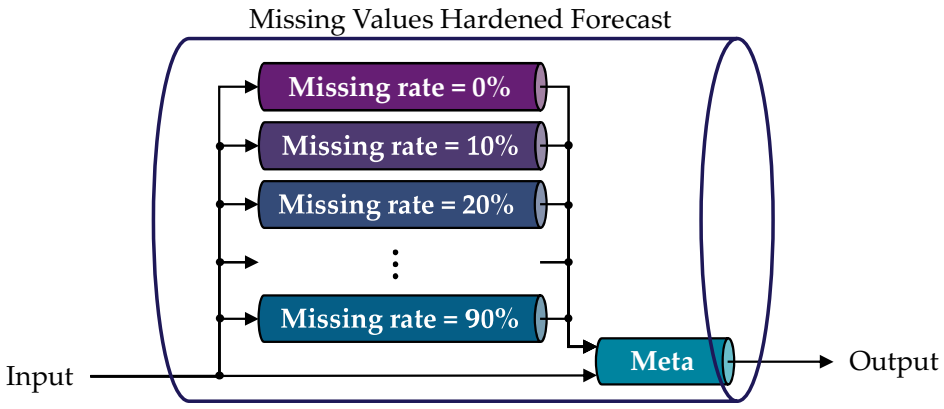


Figure 3.17: Multivariable Convolutional Encoder models trained with missing values was arranged in a stacking ensemble with input exposed to the meta.

Different from the typical stacking ensemble implementation [67], where heterogeneous ML models were used as the base models, my implementation is homogeneous, in addition to exposing the input data to the meta model. The heterogeneity that is needed to make ensemble learning more robust came from the missing rate induced to the training data when training the first ten models and by exposing the input to the eleventh meta MCE model, which helped it to notice the forecast imperfection from each of the base models, which the meta will improve upon.

3.4.2 Adversarial Attacks

Similar to the MV imputation, the adversarial attacks corrections utilize unpublicized data with strong correlation with the target forecast, along with the FE model to correct the adversarial attacks. This solution is part of my previous research [37], which uses additional unaccounted ML models and input to make it harder to create an accurate surrogate model.

Different from the previous research implementation that commonly uses adversarial training [39] which generates the adversarial sample in each batch train, which takes a long time to complete, my implementation directly uses the adversarial samples generated from the normal forecasting model as training data, which shortens the time to train. Algorithm 3.5 is the modified PGD-based adversarial samples used to create training data.

Algorithm 3.5 Multivariable Projected Gradient Descent Implementation

Input: $X_{multivariate}$, $y_{1_{train}}$, M, ϵ, $iteration = 10$
Output: Independent adversarial samples X_{PGD}

Function `pgd_multisample`($X_{multivariate}, y_{1_{train}}, M, \epsilon, iteration$)
  Create a copy of the array: $X_{PGD} \leftarrow$ `copy`($X_{multivariate}$)
  Generate noise: $noise \leftarrow$ `random.uniform`($-\epsilon \cdot 0.01$, $\epsilon \cdot 0.01$, `len`($X_{multivariate}[0]$))
  Add noise: $X_{PGD}[0] \leftarrow X_{PGD}[0]+noise$
  Get range: $minval$, $maxval \leftarrow$ `min`($X_{multivariate}[0]$), `max`($X_{multivariate}[0]$)
  Clip perturbations: $X_{PGD}[0] \leftarrow$`clip`($X_{PGD}[0]$, $minval$, $maxval$)
  Calculate the alpha: $\alpha \leftarrow \epsilon/iteration$
  for $j = 0$ to `len`($iteration$) **do**
    with `GradientTape()` **as** `tape` **do**
      Tape on X_{PGD}: `tape.watch`(X_{PGD})
      Get prediction: $y_{predict} \leftarrow M(X_{PGD})$
      Compute loss: $loss \leftarrow$ `mean_squared_error`($y_{1_{train}}, y_{predict}$)
    Compute gradient: $gradient \leftarrow$ `tape.gradient`($loss, X_{PGD}[0]$)
    Insert perturbation: $X_{PGD}[0] \leftarrow X_{PGD}[0]+\epsilon \cdot$ `sign`($gradient$)
    Clip perturbations: $X_{PGD}[0] \leftarrow$`clip`($X_{PGD}[0]$, $minval$, $maxval$)
  return X_{PGD}
End Function

The modified PGD-based adversarial sample is designed to create an adversarial sample for multivariable models that takes $X_{multivariate} = [X_{1_{train}}, X_{2_{train}}, X_{3_{train}}, \cdots, X_{m_{train}}]$ to get $X_{1_{train}}$. For simplicity, it uses the normal forecasting model M instead of using a surrogate model to compute the gradient and add small perturbation in $X_{1_{train}}$, which is the input targeted to forecast. From there, Algorithm 3.6 was used to generate stacks of independent and dependent variables to train and harden the FE model capability in filtering the adversarial attacks. A PGD-based sample was chosen to create the adversarial sample due to its robustness in negatively affecting the ML models.

Algorithm 3.6 Stacked independent and dependent variables with adversarial samples

Input: Training data $X_{multivariate}$, $y_{1_{train}}$, Normal forecasting model M, $iteration = 10$
Output: Independent and dependent stacked variables $stacked\_X, stacked\_y$

Function `adversarial_sample`($X_{multivariate}$)
  Initialize empty array: $temp \leftarrow []$
  for $\epsilon = 0$ **to** 0.9 **step** 0.1 **do**
    Sample: $temp$.`append`(`pgd_multisample`($X_{multivariate}, y_{1_{train}}, M, \epsilon, iteration$))
  Transpose the nested list: $transposed \leftarrow$ `zip`($^*temp$)
  Vertical stack: $stacked\_X \leftarrow$ [`vstack`($group$) `for` $group$ `in` $transposed$]
  Create label: $stacked\_y \leftarrow$ `vstack`($[y_{1_{train}}] \times 10$)
  return $stacked\_X, stacked\_y$
End Function

As shown in Figure 3.18, the same solution used to harden the forecast against the MV is used to harden the forecast against the adversarial attacks. The implementation uses ten MCE models hardened with different adversarial intensities ϵ ranging from 0% to 0.9%, and the eleventh MCE model was used as a meta model to combine the outputs. Exposing the original input helps the meta model to identify imperfections in the outputs.

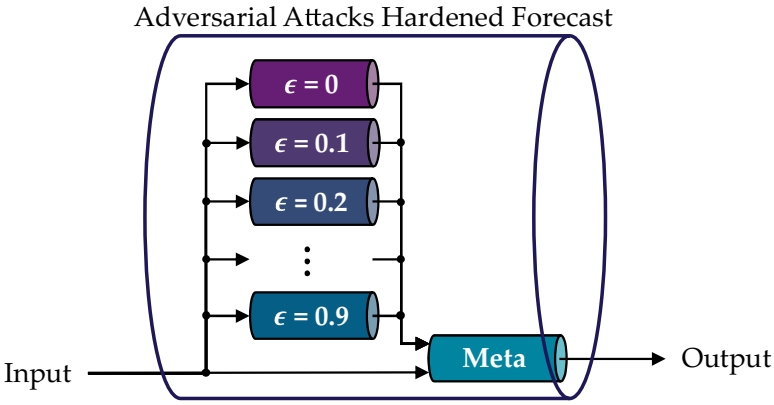


Figure 3.18: Multivariable Convolutional Encoder models trained with adversarial samples was arranged in a stacking ensemble with input exposed to the meta.

3.4.3 Concept Drift

To fix the CD in the input data, this thesis uses differential normalization, which is an extension to the radian scaling [59] to remove the trend in time series and make it stationary by performing differencing [71] on the sequence before scaling it from -1 to 1 range using min-max normalization. Equation (3.5) shows the differential normalization concept that

scales the sequence that has undergone differencing using min-max normalization.

$$v'=2 \times \frac{v-\min(\Delta v)}{\max(\Delta v)-\min(\Delta v)}-1 \tag{3.5}$$

As differencing will reduce the sequence length by 1, it is necessary to add additional value, which is called a pivot point, at the beginning of the sequence $(t-1)$ to maintain the array length via spline interpolation imputation [46]. This helps to avoid modifying the ML model architecture due to the mismatched input length that the ML model expects.

To revert the scaled sequence back to the original scale, use the inverse transformation formula for the min-max normalization shown in Equation (3.6) before adding the pivot point to the first element in Δv and performing a cumulative sum to get v.

$$v=\frac{v'+1}{2} \cdot (\max(\Delta v)-\min(\Delta v)) + \min(\Delta v) \tag{3.6}$$

Different from the previous solution that relies on ML models to impute or filter the anomalies in the sequence, this solution acts as a drop-in feature scaling for another ML model to use once it was fitted to the scale of the training data. The novelty of this technique is that it is compatible with other ML models that use the standard min-max normalization, which helps remove the trends caused by adversarial attacks without re-training the ML.

To harden the forecasting model against CD, the solution proposed uses a single MCE model trained using the data scaled with radian scaling, which differs from the solutions that use a stacking ensemble to harden the forecast against MV and adversarial attacks. Similar to the differential normalization, it also uses differencing to get the difference between two consecutive values. However, its purpose is to calculate the angle between the two consecutive values instead of making the sequence stationary. It works by leveraging the fact that the radian value θ calculated from the difference of two consecutive values Δv in a time series will never exceed the straight angle on the y-axis, as shown in Figure 3.19. This limitation is due to the linear time constraint, where the next value v_{t+1} will always be positioned on the succeeding time step, or in either quadrant I or quadrant IV of the previous value v_t.

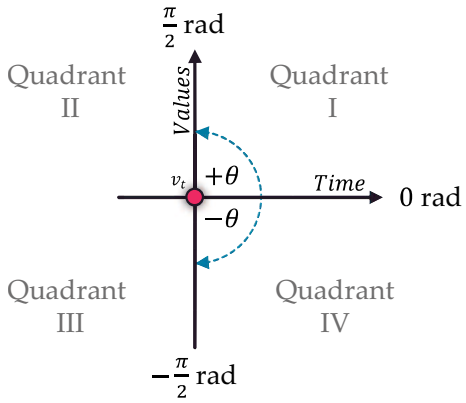


Figure 3.19: The linear time constraint ensures the radian value θ will never exceed the straight angle on the y-axis.

Additionally, the patterns in the time series are almost always the same with a different scale, mean, and median, as shown in Figure 3.20 that highlight gradual drift due to the seasonal change. Radian scaling solves these issues by scaling the values to a radian, which does not rely on scale, mean, and median found in the training data. This helps to maintain the current normalized data distribution close to what the ML models were originally trained on.

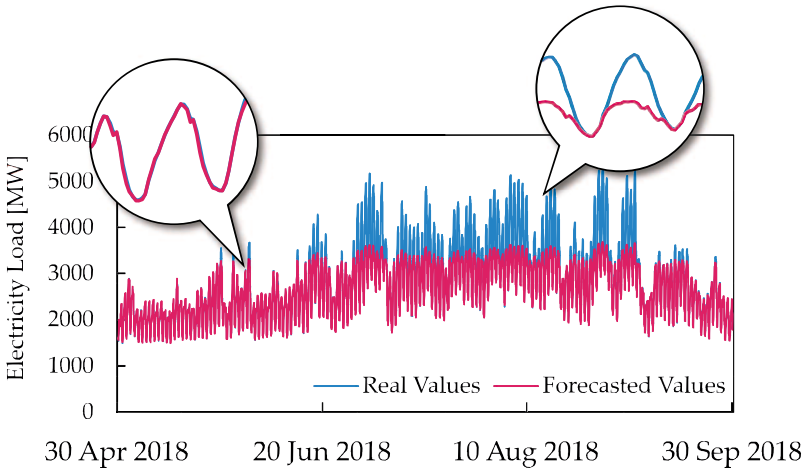


Figure 3.20: The gradual drift due to the seasonal changes shows the real values still follows the midday-to-midnight pattern, albeit on a different scale, mean, and median.

With radian scaling, the θ are guaranteed to be within the range $-\pi/2 < \theta < \pi/2$, which is approximately $-1.5708 < \theta < 1.5708$ for the ML models to process. The hard limit on the minimum and maximum values allows the model to generalize to sequences

it has not yet trained, as the values will never exceed this range, preventing normalized values with skewed distribution when handling sequences that exhibit CD.

Figure 3.21 shows the generalization example to calculate the radian value θ_1 between two consecutive values. By representing the value differences as Δv_1 and the time step difference between any consecutive values of Δt as 1, θ_1 can be calculated using the inverse trigonometric function $v_1'{=}tan^{-1}(\Delta v_1)$, which will create a bimodal distribution for the entire dataset. However, this will degrade the forecasting accuracy, necessitating in-depth data preprocessing and complex ML models [72, 73, 74] to work with it.

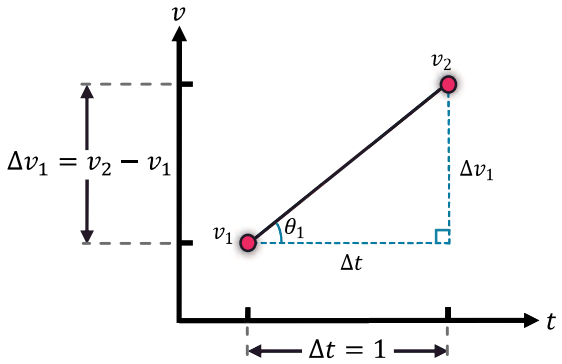


Figure 3.21: The gradual drift due to the seasonal changes shows the real values still follows the midday-to-midnight pattern, albeit on a different scale, mean, and median.

To solve this, the diminisher k was introduced to push the distribution to the center, balancing between making the unimodal distribution and avoiding the scaled values being too close to zero. Moving forward, the inverse trigonometric function can be rewritten as shown in Equation (3.7). Additionally, the comparison between the training and evaluation data samples that exhibit CD is shown in Figure 3.22 and Figure 3.23, highlighting radian scaling capability in maintaining the distribution against CD.

$$v_1'{=}tan^{-1}\left(\frac{\Delta v_1}{k}\right), \text{ where: } k{=}10^{\left\lfloor \log_{10}\left(\frac{\sum_{i=1}^{n-1}|\Delta v_i|}{n-1}+1\right)\right\rfloor} \tag{3.7}$$

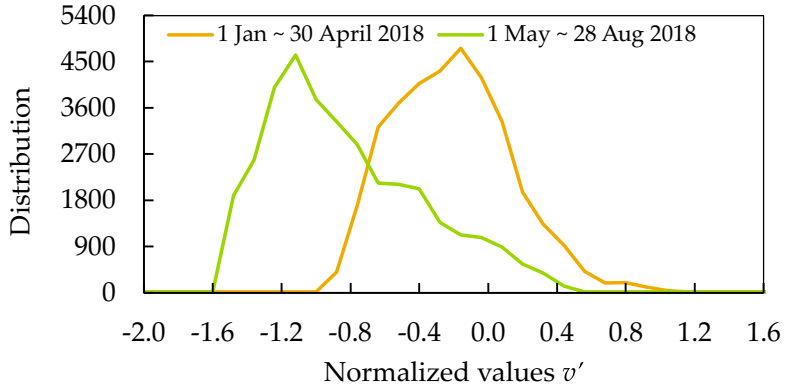


Figure 3.22: Min-max normalization distribution sample comparison between the train data (yellow) and evaluation data (green) sample.

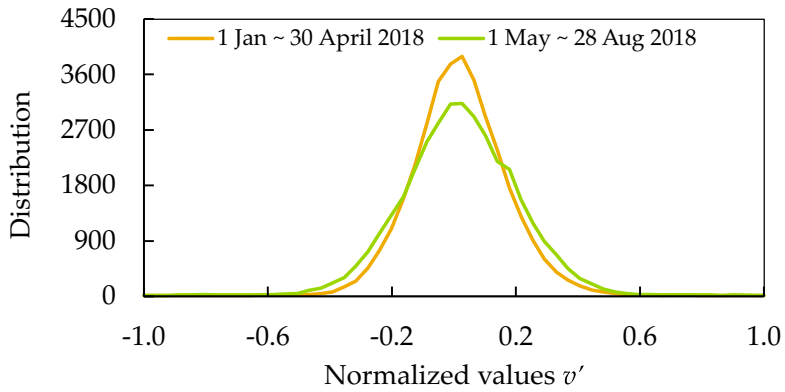


Figure 3.23: Radian scaling distribution sample comparison between the train data (yellow) and evaluation data (green) sample.

To be paired with the differential normalization that converts the time series into a stationary sequence, the forecasting model hardened against CD will accept the values scaled with differential normalization. However, the reverse transformation used to unscale the sequences back to their original scale will still rely on radian scaling, which differs from other ML models utilized in ALCEN, as the scaling and unscaling method will be replaced with differential normalization when CD was detected to fix the input sequence.

Figure 3.24 shows a dual feature scaling implementation for CD-hardened forecasting model. The baseline input representing the sequence that exhibits CD uses radian scaling, and the fixed input from CD uses differential normalization. However, the forecasted sequences only use radian scaling to change the forecast back to its original scale due to radian scaling being more superior in maintaining the drifted distribution.

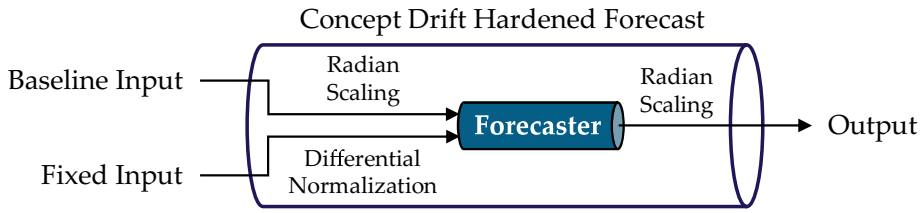


Figure 3.24: Dual feature scaling implementations are used by the forecasting model hardened against concept drift to handle baseline and corrected input.

In addition to using the convolutional layer in ML models to help it generalize with CD, the implementation of differential normalization helps other forecasting models to be more robust against CD. In addition, the CD-hardened forecasting further improves the forecasting accuracy for the primary meta model to use as a guide when combining the result from other forecasters.

3.4.4 Primary Meta Model

There are two methods to combine the output from the hardened forecasting models, which are weighted averaging and meta model. Although average weighting is the simplest solution to implement, it requires further tuning to correlate the severity of the issues with their corresponding weights in order to be effective. Additionally, the severity of an adversarial attack or CD is dependent on the context or domain knowledge needed to quantify it in the weight used to average the results.

In ALCEN implementation, a meta model was used to combine the output from the hardened forecasting models. To provide the context or domain knowledge of how each of the outputs from the hardened forecasting models differs based on the problems they are solving, the original input was parsed to the meta model as a reference. The only demerit of using a meta model is it expects the feature numbers and the sequence length to be constant. As the number of features will change depending on the number of issues found in the input data, sequence filler is used to backfill empty features with the existing ones.

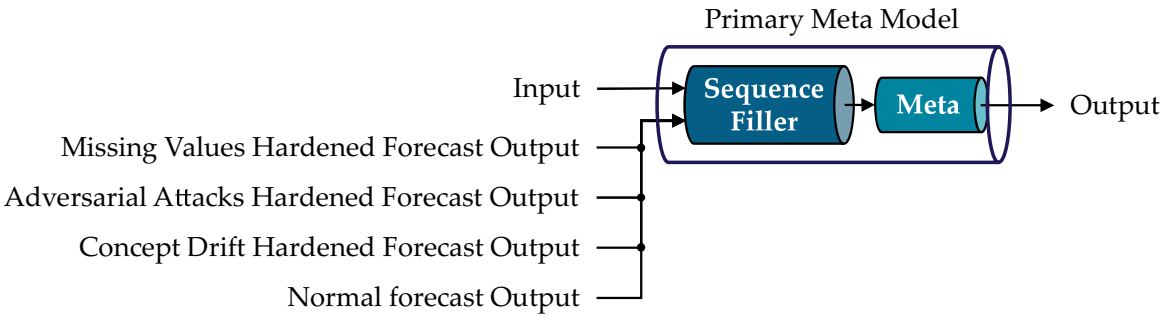


Figure 3.25: Primary meta model implementation to combine the output from multiple hardened model based on the severity of the noises found in the input.

This will not only make the hyperparameter easier to use when setting up ALCEN in Chapter 4, but it also help ALCEN make the most of the best parts of each solution without the need to compromise.

3.4.5 Training Method

To train the models or countermeasures, it must follow the numbering shown in Figure 3.26, as the hardened forecasting models require the output from the corrected input to analyze the incompleteness or correction error they must work with. This is also the same for the primary meta model, as it requires the output from the hardened forecasting model in addition to the normal forecasting model.

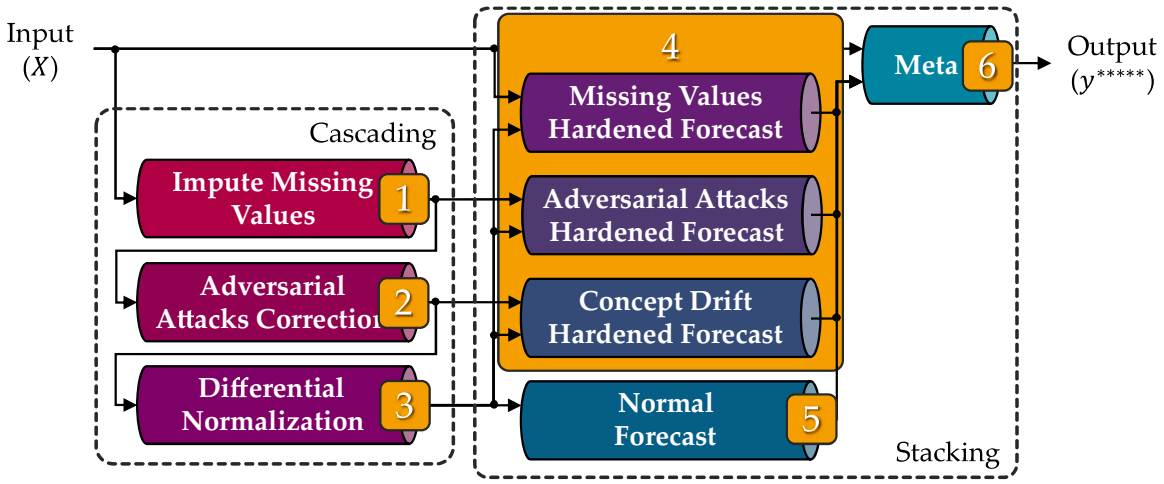


Figure 3.26: The machine learning models or countermeasures must be trained or initialized based on the denoted number, where smaller number are prioritize first.

Once the countermeasures in the cascading group were trained in sequence, the output

from these countermeasures was used to train the countermeasures in number 4. As the operation in here is in parallel, the countermeasures can be trained in any sequence. Then, the normal forecasting model was trained, and the output was used along with the hardened forecast output to train the meta model.

Additionally, each of the models and countermeasures requires the dependent variables in the training data to have noises added to it. This is necessary to expose the noisy data to the ML models during the training phase to teach them how to generalize the forecast with this issues. Table 3.3 summarize the type of training data used to train the models and countermeasures.

Table 3.3: Type of training data used to train the machine learning model or countermeasures.

Solution	**Training Data**		
	Normal	*Missing Values (0-90%)*	*Adversarial Attacks (ϵ=0 $\sim$ 0.1)*
Impute missing values		✓	
Adversarial Attacks correction			✓
Differential normalization	✓		
Missing values hardened forecast		✓	
Adversarial attacks hardened forecast			✓
Concept drift hardened forecast			✓
Normal forecast	✓		
Meta	✓	✓	✓

3.5 Conclusion

In this chapter, the theories on how to solve the SPoF problem were given, which help to maintain the forecasting accuracy as long as the number of offline nodes does not exceed more than 3. In addition, the standardized template to initialize the machine model also helps to restore the offline node functionality in a different node, simplifying the deployment and management.

Furthermore, the anomaly detection to detect MV, CD, and adversarial attacks helps identify the issues and rearrange the node configurations to tackle the issues head-on. This mechanism ensures the ALCEN is able to adapt to any situation without requiring intervention from the system operator, which might not be fast enough to rectify the issues.

Finally, the solution given resolves the MV, CD, and adversarial attacks not only by repairing the data but also by enhancing the forecast to cover any mistake done when correcting the input, which is not possible in the previous research.

The following chapter will present the ALCEN application to forecast the electricity load in Long Island City of New York State and compare the result against the method proposed by Zhou et al. [43].

4 Electricity Load Forecast

4.1 Chapter Introduction

In this section, ALCEN was applied to forecast the electricity load, and its performance against MV, CD, and adversarial data was measured. Then the forecasting accuracy was compared with previously proposed unified method by Zhou et al. [43]. For fair comparison, the same 4 months interval to update the forecaster was used, with the exception of 2 months interval for retraining, as ALCEN relies on differential normalization and radian scaling to handle CD instead of retraining.

The dataset used from the New York Independent System Operator (NYISO) [4], which consist of 11 electricity load zones shown in Figure 4.1 and Figure 4.2. Similar to the previous experiments done in Chapter 2, the target forecast is the electricity load in the Long Island, New York, which is represented as LONGIL in Figure 4.2.

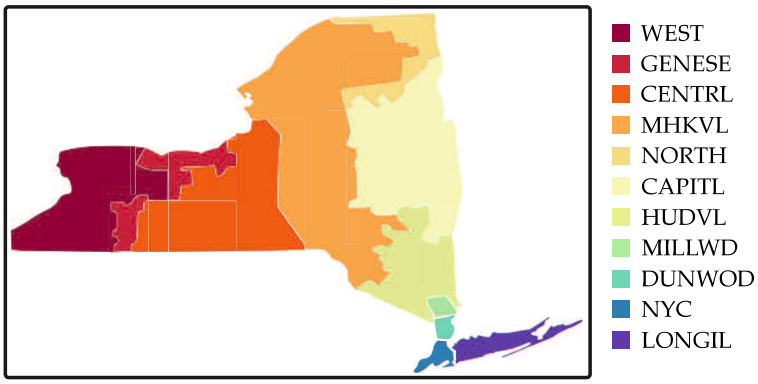


Figure 4.1: The electricity load zones and their corresponding zone codes.

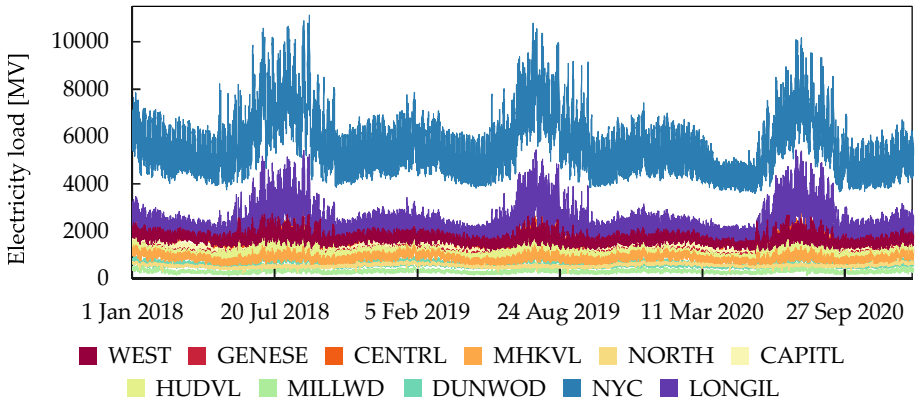


Figure 4.2: The electricity load data sampled in at 5-minute intervals in 11 zones.

4.2 Application

Table 4.1 is the hyperparameter used to initialize ALCEN. Although FE and MCE models utilizes different architecture, it use same the parameters used to initialize the models.

Table 4.1: Hyperparameter used when initializing the Adaptively Linked Composite Ensemble Network to forecast the electricity load.

Target	**Parameter**	**Value**
Training	batch size	1000
	number of epoch	300
	Adam learning rate	0.001
Detection	Drift threshold	0.5
	Adversarial threshold	r^2 <09.6
	Missing percentage threshold	100%
Retraining	Sliding window size	4 months
	Fixed training interval	6 months
	Acceptable accuracy threshold	r^2 >0.96
Overfitting	Patience	3
	Minimum loss delta	0.0001
Architecture	Kernel size	3
	Input length	12
	Input features	3
	Output length	12
	Number of filters	8

To forecast the electricity load in the Long Island zone using the ALCEN, another two zones that have a high electricity load correlation to the Long Island zone must be selected. Kendall's τ correlation was selected due to its robustness to outliers and capability of handling ordinal data [75]. From the test, Hudson Valley and Capital zones were selected due to their strong correlation with the Long Island zone. Although there is no hard limit on how much strongly correlated data can be used, the number of correlated data used is set to 2, as the accuracy gained is minuscule with a long computation time trade-off.

Then, the solutions to fix the input and harden the forecasts were initialized, which as defined in Table 4.1 with 12 input and output lengths, 3 input features that represent the input sequences of Long Island, Hudson Valley, and Capital for the ML models were prepared. Then, with training data ranging from 1 January 2018 to 30 April 2018, followed by evaluation data ranging from 1 May 2018 to 31 December 2020, the ML models were

trained based on the training and overfitting parameters before setting up the anomaly detections. Once prepared, ALCEN performance against CD, MV, and adversarial attacks was measured.

4.3 Result

4.3.1 Missing Values

Similar to the MV experiment done in Chapter 2.3.3, the scheduled learning for ALCEN is denoted as T1, T2, and T3 being done every 4 months to keep ALCEN up-to-date, it was tested on $v'_{evaluate}$ with MV ranging from 0% to 90%, on 1 January 2019 to 30 April 2019. Figure 4.3 shows the visualized scheduled updates (T1, T2, and T3) with evaluation range to measure the accuracy.

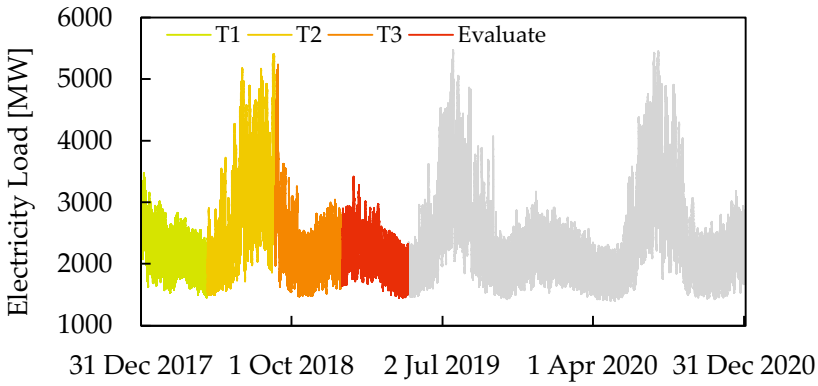


Figure 4.3: Scheduled learning done to update the adaptively linked composite ensemble model and the evaluation range.

To simulate MCAR on from 1 January 2019 to 30 April 2019 caused by sensor failures or packet loss from DDoS attacks, Algorithm 2.5 was utilized to randomly insert MV. For consistency, the randomization seed set to *seed*=2025, with MV percentage ranging from 0% to 90%. Table 4.2, Table 4.3, and Table 4.4 show the averaged forecasting accuracies.

Table 4.2: Coefficient of determination scores against the missing values.

Missing Percentage [%]	**Coefficient of Determination (R^2)**			
	Session			*Average*
	1	2	3	
0	0.9967	0.9967	0.9967	0.9967
10	0.9959	0.9960	0.9962	0.9960
20	0.9954	0.9953	0.9950	0.9952
30	0.9942	0.9940	0.9936	0.9940
40	0.9927	0.9917	0.9922	0.9922
50	0.9895	0.9895	0.9896	0.9895
60	0.9842	0.9852	0.9848	0.9847
70	0.9756	0.9744	0.9743	0.9748
80	0.9554	0.9573	0.9581	0.9569
90	0.8740	0.8727	0.8746	0.8738

Table 4.3: Root mean squared error scores against the missing values.

Missing Percentage [%]	**Root Mean Squared Error (RMSE)**			
	Session			*Average*
	1	2	3	
0	51.583	51.584	51.584	51.584
10	57.384	56.391	55.415	56.397
20	60.371	61.325	63.370	61.689
30	67.979	69.06	71.179	69.407
40	76.435	81.25	79.058	78.915
50	91.741	91.41	91.190	91.448
60	112.26	108.49	110.23	110.33
70	139.57	142.86	143.15	141.86
80	188.67	184.67	182.77	185.37
90	317.01	318.74	316.32	317.36

Table 4.4: Mean absolute error scores against the missing values.

Missing Percentage [%]	**Mean Absolute Error (MAE)**			
	Session			*Average*
	1	2	3	
0	36.540	36.541	36.541	36.540
10	39.401	39.591	39.222	39.405
20	42.647	42.596	42.991	42.745
30	46.498	46.937	46.676	46.704
40	52.125	53.806	53.390	53.107
50	61.175	60.818	60.918	60.970
60	74.369	73.263	73.417	73.683
70	93.653	93.749	92.602	93.334
80	124.52	125.20	123.35	124.36
90	207.36	202.46	203.84	204.55

These result shows huge accuracy improvement against MV. Although the average accuracies with low percentage of MV on Zhou et al. [43] is slightly better, this method really shine when the MV percentage rise more than 40%. By comparing the MAE result on 90%, ALCEN reduce the MAE score by 49.776%, which is 357.51 [MW] reduction of averaged absolute error. The reason for this is due to the multivariate nature of ALCEN, where multiple input values will inadvertently add small noises to the forecast. Figure 4.4, Figure 4.5, and Figure 4.6 confirmed these observations.

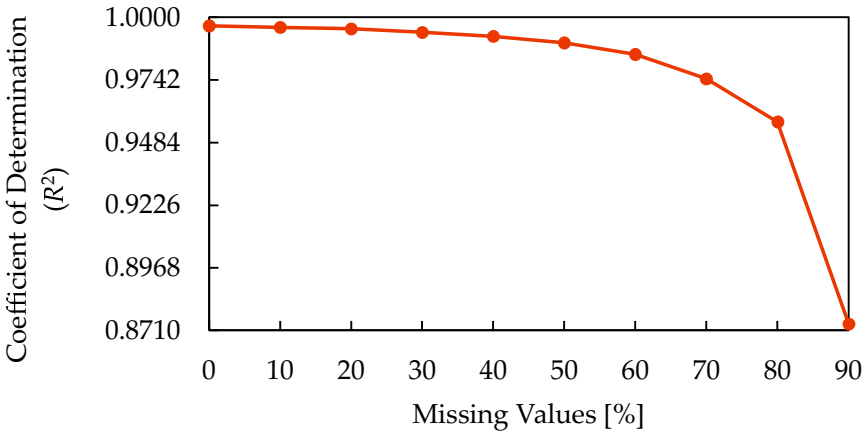


Figure 4.4: Averaged coefficient of determination scores against missing values from 1 January 2019 to 30 April 2019.

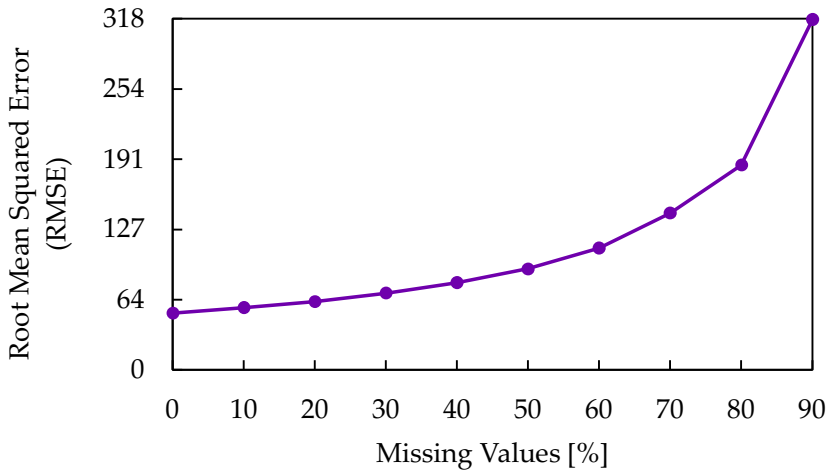


Figure 4.5: Averaged root mean squared errors score against missing values from 1 January 2019 to 30 April 2019.

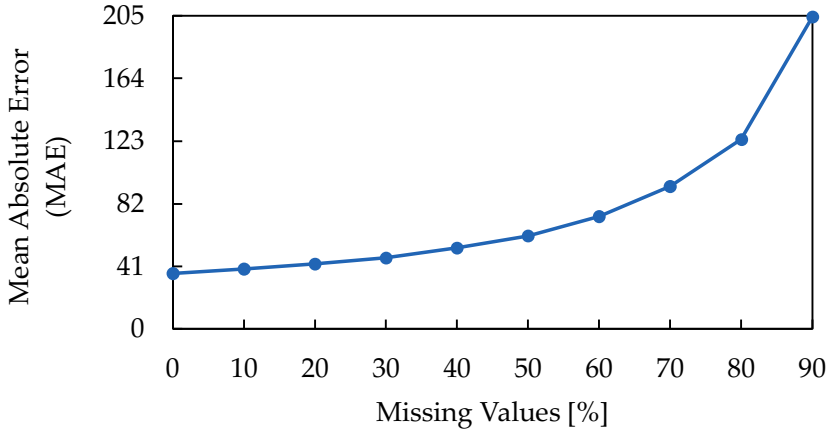


Figure 4.6: Averaged mean absolute errors score against missing values from 1 January 2019 to 30 April 2019.

4.3.2 Concept Drift

In this experiment, ALCEN is tested on CD, in addition to the MV that ranges from 0% to 90% to see how ALCEN deals with multiple input noises. Similar to the CD experiment in Chapter 2.3.4, the evaluation range was set from 1 May 2020 to 31 August 2020, which is right after the scheduled training (T7) was done to update the ALCEN. Figure 4.7 shows the visualized scheduled learning with the evaluation range to measure the accuracy.

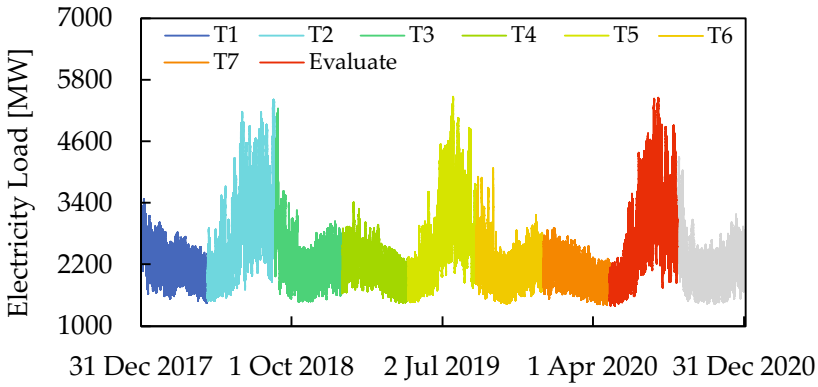


Figure 4.7: Scheduled learning done to update the adaptively linked composite ensemble model and the evaluation range.

The R^2 scores shown in Table 4.5 achieve better forecasting accuracy than the R^2 scores shown in Table 4.2, and this observation was supported by the RMSE and MAE scores, as shown in Table 4.6 and Table 4.7. This is different from the experiment done in Chapter 2.3.4, where the R^2 scores improved while the RMSE and MAE scores worsened.

In addition to the up-to-date multivariate countermeasures used by ALCEN, the sensors data is used to impute with high accuracy and rectify the error due to incomplete imputation. The forecasting model enhanced with radian scaling also serves as a guide for the primary meta model to aggregate the forecast. This explains why the RMSE and MAE scores for ALCEN do not worsen.

Table 4.5: Coefficient of determination scores against the missing values and concept drift.

Missing Percentage [%]	**Coefficient of Determination (R^2)**			
	Session			*Average*
	1	2	3	
0	0.9969	0.9969	0.9969	0.9969
10	0.9967	0.9966	0.9967	0.9966
20	0.9963	0.9962	0.9963	0.9963
30	0.9956	0.9956	0.9957	0.9956
40	0.9944	0.9945	0.9944	0.9944
50	0.9926	0.9924	0.9927	0.9926
60	0.9893	0.9885	0.9889	0.9889
70	0.9803	0.9813	0.9810	0.9809
80	0.9667	0.9632	0.9654	0.9651
90	0.9022	0.8981	0.9049	0.9017

Table 4.6: Root mean squared error scores against the missing values and concept drift.

Missing Percentage [%]	**Root Mean Squared Error (RMSE)**			
	Session			*Average*
	1	2	3	
0	49.785	49.785	49.785	49.785
10	51.366	52.249	51.674	51.763
20	54.013	55.069	54.338	54.473
30	59.354	59.007	58.504	58.955
40	66.874	66.227	66.898	66.667
50	76.835	77.772	76.113	76.907
60	92.554	95.587	94.085	94.075
70	125.30	122.05	123.00	123.45
80	162.90	171.38	166.14	166.81
90	279.40	285.17	275.45	280.01

Table 4.7: Mean absolute error scores against the missing values and concept drift.

Missing Percentage [%]	**Mean Absolute Error (MAE)**			
	Session			*Average*
	1	2	3	
0	35.939	35.939	35.940	35.939
10	36.902	37.559	37.238	37.233
20	38.717	39.606	38.926	39.083
30	42.382	42.131	41.972	42.161
40	47.136	46.767	47.147	47.017
50	53.049	53.520	52.607	53.059
60	63.759	64.904	64.191	64.285
70	81.823	81.659	82.688	82.057
80	110.00	113.24	112.52	111.92
90	180.64	183.39	177.39	180.47

Figure 4.8, Figure 4.9, and Figure 4.10 support the observation that R^2, RMSE, and MAE scores do improve. As the countermeasures updated seven times, they are becoming much better at handling MV and CD without relying on conformal predictions to preemptively start the retraining to update the models before the scheduled update.

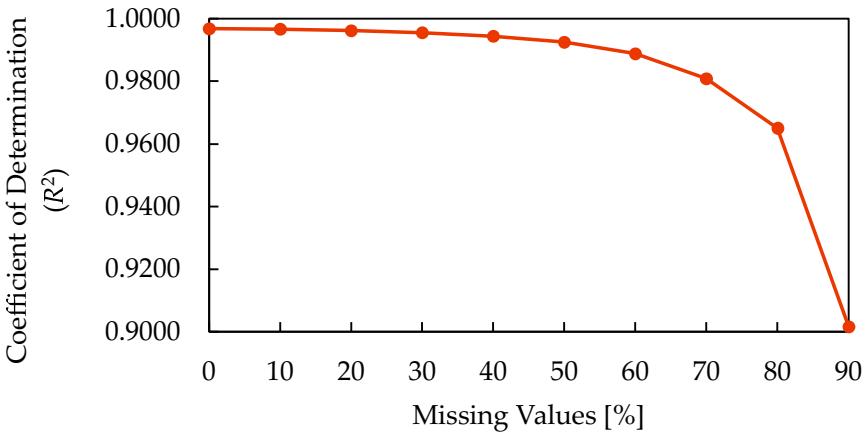


Figure 4.8: Averaged coefficient of determination scores against missing values and concept drift from 1 May 2020 to 31 August 2020.

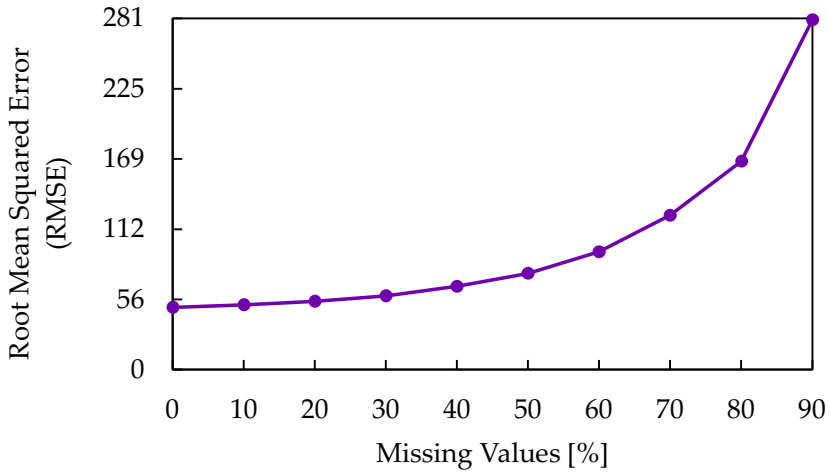


Figure 4.9: Averaged root mean squared errors score against missing values and concept drift from 1 May 2020 to 31 August 2020.

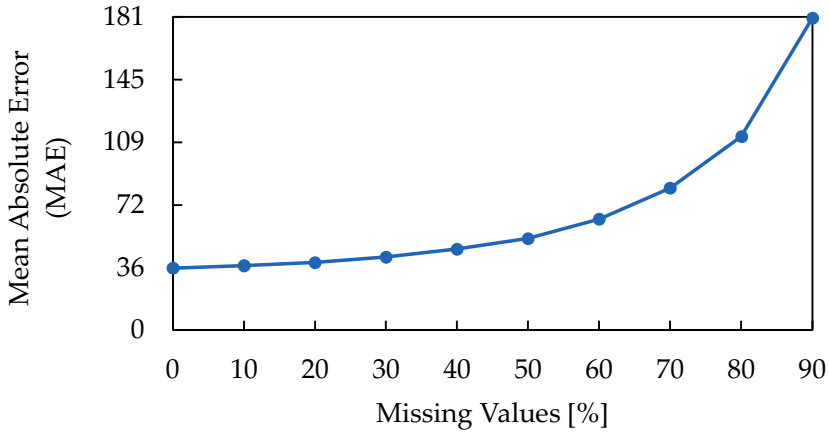


Figure 4.10: Averaged mean absolute errors score against missing values and concept drift from 1 May 2020 to 31 August 2020.

4.3.3 Adversarial Attacks

On top of MV and CD, this experiment uses the same PGD-based adversarial attacks used in Chapter 2.3.5 to measure the ALCEN resiliency. Using the same surrogate mode architecture shown in Figure 2.23 and the same training mechanism, the surrogate model copied the normal forecasting model behavior, and a PGD sample with ϵ=0.05 was created on the evaluation data that ranged from 1 May 2020 to 31 August 2020 that exhibit seasonal drift. Then, MV percentages ranging from 0% to 90% were simulated and tested on ALCEN.

Table 4.8, Table 4.9, and Table 4.10 show the forecasting accuracies from 1 May 2020 to 31 August 2020 against MV, CD, and PGD-based attacks with ϵ=0.05. Compared to the previous result obtained from Chapter 4.3.2, the R^2 suffers minimal degradation, with the exception of RMSE scores, where the averaged forecast from 0% to 90% shows RMSE scores increase by 49.674%. This is also confirmed on the MAE scores, where the averaged forecast from 0% to 90% shows MAE scores increase by 73.951%, or approximately 51.265 [MW] of averaged absolute error difference.

Although the accuracy degradation shown by RMSE and MAE is considerably noticeable, the accuracy is far better than the performance of the previously proposed method in Chapter 2.3.5, as ALCEN reduces the MAE scores by 49.645%, or approximately 118.89 [MW] of the averaged absolute error difference.

Table 4.8: Coefficient of determination scores against the missing values, concept drift, and adversarial attacks.

Missing Percentage [%]	**Coefficient of determination (R^2)**			
	Session			*Average*
	1	2	3	
0	0.9797	0.9797	0.9861	0.9818
10	0.9800	0.9802	0.9856	0.9819
20	0.9802	0.9804	0.9847	0.9817
30	0.9795	0.9793	0.9836	0.9808
40	0.9776	0.9790	0.9817	0.9794
50	0.9759	0.9766	0.9796	0.9774
60	0.9749	0.9742	0.9752	0.9748
70	0.9676	0.9681	0.9695	0.9684
80	0.9502	0.9534	0.9542	0.9526
90	0.8963	0.8895	0.8981	0.8946

Table 4.9: Root mean squared error scores against the missing values, concept drift, and adversarial attacks.

Missing Percentage [%]	**Root Mean Squared Error (RMSE)**			
	Session			*Average*
	1	2	3	
0	127.18	127.18	105.34	119.90
10	126.43	125.57	107.15	119.72
20	125.63	125.15	110.66	120.48
30	127.98	128.60	114.22	123.60
40	133.83	129.51	120.99	128.11
50	138.62	136.62	127.64	134.29
60	141.37	143.45	140.74	141.86
70	160.74	159.43	155.87	158.68
80	199.31	192.86	191.22	194.46
90	287.63	296.98	285.11	289.91

Table 4.10: Mean absolute error scores against the missing values, concept drift, and adversarial attacks.

Missing Percentage [%]	**Mean Absolute Error (MAE)**			
	Session			*Average*
	1	2	3	
0	112.04	112.04	87.33	103.80
10	109.28	108.62	87.94	101.95
20	106.59	105.97	91.01	101.19
30	107.49	108.13	93.20	102.94
40	109.93	107.86	98.10	105.30
50	113.07	111.17	102.73	108.99
60	112.98	115.06	110.94	112.99
70	124.87	123.60	121.38	123.28
80	147.26	143.15	141.10	143.83
90	201.41	204.26	199.13	201.60

In addition to supporting the observation in Table 4.8, Table 4.9, and Table 4.10, Figure 4.11, Figure 4.12, and Figure 4.13 also show interesting behavior where the accuracy degradation rate is much lower than the previously proposed unified methods. These results highlight the ALCEN's capability of handling multiple input noises while maintaining respectable accuracy.

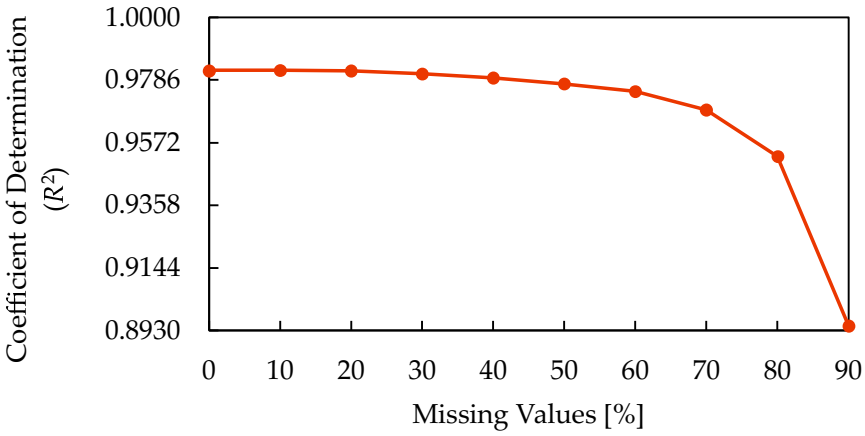


Figure 4.11: Averaged coefficient of determination scores against missing values, concept drift, and adversarial attacks from 1 May 2020 to 31 August 2020.

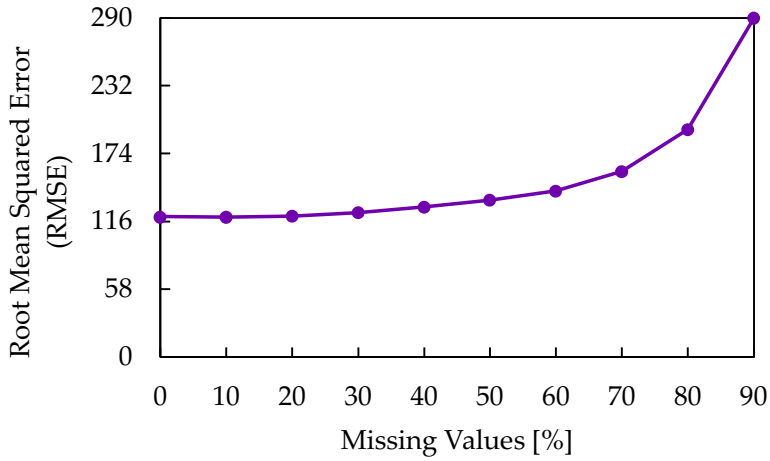


Figure 4.12: Averaged root mean squared errors score against missing values, concept drift, and adversarial attacks from 1 May 2020 to 31 August 2020.

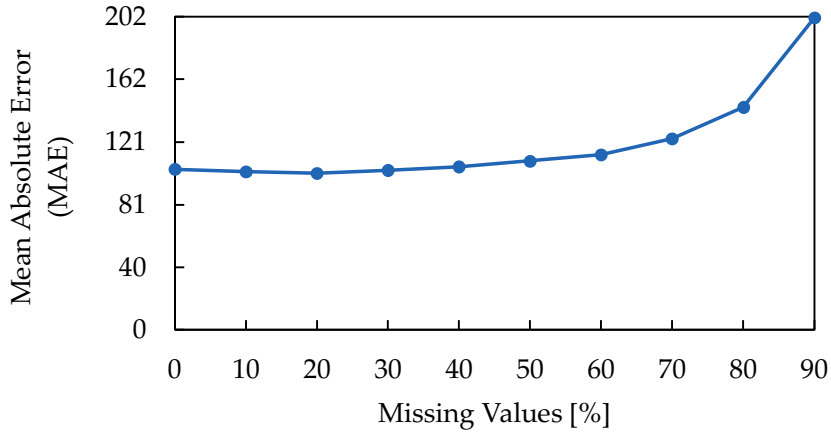


Figure 4.13: Averaged mean absolute errors score against missing values, concept drift, and adversarial attacks from 1 May 2020 to 31 August 2020.

4.4 Conclusion

In this chapter, ALCEN was tested in the same environment as the previously proposed unified solution, and it shows greater resiliency against MV, CD, and adversarial attacks.

With a high percentage of MV, ALCEN could strengthen the forecasting accuracy by considering the imputation error that could negatively impact the forecasting accuracy. Similarly, the same concept was implemented to enhance the forecast against the adversarial attacks by considering the correction error that could decrease the forecasting accuracy. Finally, the CD was resolved by using radian scaling and differential normalization to avoid computationally expensive retraining.

These results highlight the composite models arrangement method that was utilized by ALCEN to negate correction error accumulation due to sequential input corrections and solutions compatibilities, allowing multiple countermeasures to be unified.

Finally, the experiment also highlights the potential misuse that the attackers could utilize to weaken the forecasting model, which was shown by launching DDoS attacks to cause MV and hide well-crafted adversarial attacks that could potentially cause a negative impact on a forecasting system that relies only on a single countermeasure.

5 Conclusion

In Chapter 1, the necessity of accurate forecasting mechanisms to optimize smart city systems was introduced. It highlights real-world challenges such as MV, CD, adversarial attacks, and SPoF that can adversely affect forecasting accuracy. Additionally, the correction error accumulation and solution compatibility are given as the reasons why most of the proposed solutions only focus on a single problem.

In Chapter 2, the unified method proposed by Zhou et al. [43] attempts to address forecasting challenges but falls short due to its trivial solutions, which fail to provide a comprehensive solution for MV, CD, adversarial attacks, and SPoF. This is due to the federated learning complicates the management of multiple machine learning-based data preprocessing, restricting solutions to linear corrections and accumulating errors from inadequate rectifications. ZThe experiments reveal weaknesses in their approach, especially its ineffective adversarial attack and MV countermeasures. Furthermore, the Seq2seq LSTM model struggles with CD adaptation, as it performs poorly when extrapolating beyond trained value ranges.

In Chapter 3, the theories to solve the SPoF problem were detailed using distributed computing and standardized templates to facilitate restoring offline node functionality, simplifying deployment and management. Anomaly detection mechanisms identify MV, CD, and adversarial attacks, allowing node configuration rearrangements without operator intervention, which may be too slow to resolve issues. The proposed solution addresses MV, CD, and adversarial attacks by not only repairing data but also enhancing forecast accuracy, overcoming limitations of previous research.

In Chapter 4, ALCEN was tested in the same environment tested on Zhou et al. [43], demonstrating improved resiliency against MV, CD, and adversarial attacks. ALCEN enhances forecasting accuracy by considering imputation and correction errors that could negatively impact results. CD issues are addressed using radian scaling and differential normalization, avoiding costly retraining. The results showcase ALCEN's composite model arrangement, negating correction error accumulation and unifying multiple countermeasures. The experiment also highlights potential vulnerabilities, such as DDoS attacks

causing MV and concealing sophisticated adversarial attacks, which could undermine a system relying solely on one countermeasure.

For the future works, a more in-depth investigation is needed to confirm the effectiveness of MV or CD to hide well-crafted adversarial attacks, and their impact on the forecasting model is necessary to further improve the resiliency and reliability of the current forecasting mechanism.

Additionally, the current application of ALCEN relies on a meta model to perform an adjustment to combine multiple prediction results into one. Although adversarial attack intensity could be estimated via an autoencoder model, it is not possible to mathematically show how much the adjustment is required. As a future challenge, we aim to develop an explainable ML model to make the decision-making done by ML more trustworthy.

References

[1] A. Deguchi et al., "What Is Society 5.0?," in Society 5.0: A People-centric Super-smart Society. Singapore: Springer Singapore, 2020, pp. 1–23.

[2] A. Deguchi, "From Smart City to Society 5.0," in Society 5.0: A People-centric Super-smart Society. Singapore: Springer Singapore, 2020, pp. 43–65.

[3] Grand View Research. "Smart Cities Market Size, Share And Growth Report." www.grandviewresearch.com. Accessed: 6 October 2024. [Online.] Available: https://www.grandviewresearch.com/industry-analysis/smart-cities-market.

[4] New York Independent System Operator. "New York Control Area Load Zones." www.nyiso.com Accessed: 22 April 2024. [Online]. www.nyiso.com/documents/20142/1397960/nyca_zonemaps.pdf.

[5] I. K. Nti, M. Teimeh, O. Nyarko-Boateng, and A. F. Adekoya, "Electricity load forecasting: a systematic review," Journal of Electrical Systems and Information Technology, vol. 7, no. 1, 13, 2020.

[6] J. Kruse, B. Schäfer, and D. Witthaut, "Predictability of Power Grid Frequency," IEEE access, vol. 8, pp. 149435–149446, 2020.

[7] C. Sweeney, R. J. Bessa, J. Browell, and P. Pinson, "The Future of Forecasting for Renewable Energy," Wiley Interdisciplinary Reviews: Energy and Environment, vol. 9, no. 2, e365, 2020.

[8] R. V. Klyuev et al., "Methods of Forecasting Electric Energy Consumption: A Literature Review," Energies, vol. 15, no. 23, 8919, 2022.

[9] E. Vivas, H. Allende-Cid, and R. Salas, "A Systematic Review of Statistical and Machine Learning Methods for Electrical Power Forecasting with Reported MAPE Score," Entropy, vol. 22, no. 12, 1412, 2020.

[10] Juniper Research. "Smart Grid Cost Savings to Exceed $125 Billion by 2027" www.juniperresearch.com Accessed: Oct. 7, 2024. [Online.] Available: https://www.juniperresearch.com/press/smart-grid-cost-savings-to-exceed-125bn.

[11] L. Myllyaho, M. Raatikainen, T. Männistö, J. K. Nurminen, and T. Mikkonen, "On Misbehaviour and Fault Tolerance in Machine Learning Systems," Journal of Systems and Software, vol. 183, 111096, 2022.

[12] R. Tawn, J. Browell, and I. Dinwoodie, "Missing Data in Wind Farm Time Series: Properties and Effect on Forecasts," Electric Power Systems Research, vol. 189, 06640, 2020.

[13] R. K. Jagait, M. N. Fekri, K. Grolinger, and S. Mir, "Load Forecasting Under Concept Drift: Online Ensemble Learning With Recurrent Neural Network and ARIMA," IEEE Access, vol. 9, pp. 98992–99008, 2021.

[14] R. Heinrich, C. Scholz, S. Vogt, and M. Lehna, "Targeted Adversarial Attacks on Wind Power Forecasts," Machine Learning, vol. 113, no. 2, pp. 863–889, 2024.

[15] Z. Xu and J. H. Saleh, "Machine Learning for Reliability Engineering and Safety Applications: Review of Current Status and Future Opportunities," Reliability Engineering & System Safety, vol. 211, 107530, 2021.

[16] V. Demertzi, S. Demertzis, and K. Demertzis, "An Overview of Cyber Threats, Attacks and Countermeasures on the Primary Domains of Smart Cities," Applied Sciences, vol. 13, no. 2, 2023.

[17] IBM X-Force. "X-Force Threat Intelligence Index 2024." www.ibm.com. Accessed: 20 October 2024. [Online.] https://www.ibm.com/reports/threat-intelligence.

[18] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "A Survey on Adversarial Attacks and Defences," CAAI Transactions on Intelligence Technology, vol. 6, no. 1, pp. 25–45, 2021.

[19] O. Yoachimik and J. Pacheco. "DDoS Threat Report for 2024 Q2." www.cloudflare.com. Accessed: 20 October 2024. [Online.] https://blog.cloudflare.com/ddos-threat-report-for-2024-q2/.

[20] N. Agrawal and S. Tapaswi, "Defense Mechanisms Against DDoS Attacks in a Cloud Computing Environment: State-of-the-Art and Research Challenges," IEEE Communications Surveys & Tutorials, vol. 21, no. 4, pp. 3769–3795, 2019.

[21] L. Gjesvik and K. Szulecki, "Interpreting Cyber-Energy-Security Events: Experts, Social Imaginaries, and Policy Discourses Around the 2016 Ukraine Blackout," European Security, vol. 32, no. 1, pp. 104–124, 2023.

[22] CyberPeace Institute. "Cyberattacks Impact and Harm on the Energy Sector." cyberpeaceinstitute.org. Accessed: 11 December 2024. [Online.] https://cyberconflicts.cyberpeaceinstitute.org/impact/sectors/energy.

[23] H. Alkabbani, A. Ramadan, Q. Zhu, and A. Elkamel, "An Improved Air Quality Index Machine Learning-Based Forecasting with Multivariate Data Imputation Approach," Atmosphere, vol. 13, no. 7, 1144, 2022.

[24] W. Zhao, S. Alwidian, and Q. H. Mahmoud, "Adversarial Training Methods for Deep Learning: A Systematic Review," Algorithms, vol. 15, no. 8, 283, 2022.

[25] R. K. Jagait, M. N. Fekri, K. Grolinger, and S. Mir, "Load Forecasting Under Concept Drift: Online Ensemble Learning With Recurrent Neural Network and ARIMA," IEEE Access, vol. 9, pp. 98992–99008, 2021.

[26] V. Venkataramanan, S. Kaza, and A. M. Annaswamy, "DER Forecast Using Privacy-Preserving Federated Learning," IEEE Internet of Things Journal, vol. 10, no. 3, pp. 204–2055, 2023.

[27] T. H. Lin, X. R. Zhang, C. P. Chen, J. H. Chen, and H. H. Chen, "Learning to Identify Malfunctioning Sensors in a Large-Scale Sensor Network," IEEE Sensors Journal, vol. 22, no. 3, pp. 2582–2590, 2022.

[28] B. Tushir, Y. Dalal, B. Dezfouli, and Y. Liu, "A Quantitative Study of DDoS and E-DDoS Attacks on WiFi Smart Home Devices," IEEE Internet of Things Journal, vol. 8, no. 8, pp. 6282–6292, 2021.

[29] M. H. Bin Kamilin, S. Yamaguchi, and M. A. Bin Ahmadon, "Fault-Tolerance and Zero-Downtime Electricity Forecasting in Smart City," presented at the 2023 IEEE 12th Global Conference on Consumer Electronics (GCCE), 10-13 October, 2023.

[30] Y. Zhang, B. Zhou, X. Cai, W. Guo, X. Ding, and X. Yuan, "Missing value imputation in multivariate time series with end-to-end generative adversarial networks," Information Sciences, vol. 551, pp. 67–82, 2021.

[31] J. Ma, J. C. P. Cheng, F. Jiang, W. Chen, M. Wang, and C. Zhai, "A bi-directional missing data imputation scheme based on LSTM and transfer learning for building energy data," Energy and Buildings, vol. 216, 109941, 2020.

[32] F. Bayram, B. S. Ahmed, and A. Kassler, "From Concept Drift To Model Degradation: An Overview on Performance-Aware Drift Detectors," Knowledge-Based Systems, vol. 245, 108632, 2022.

[33] M. H. Bin Kamilin, S. Yamaguchi, and M. A. Bin Ahmadon, "Radian Scaling: A Novel Approach to Preventing Concept Drift in Electricity Load Prediction," presented at the 2023 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia), 23-25 Oct. 2023, 2023.

[34] F. Bayram, P. Aupke, B. S. Ahmed, A. Kassler, A. Theocharis, and J. Forsman, "DA-LSTM: A dynamic drift-adaptive learning framework for interval load forecasting with LSTM networks," Engineering Applications of Artificial Intelligence, vol. 123, 106480, 2023.

[35] S. Li, Y. Zhong, and J. Lin, "AWS-DAIE: Incremental ensemble short-term electricity load forecasting based on sample domain adaptation," Sustainability, vol. 14, no. 21, 14205, 2022.

[36] Rauber, R. Zimmermann, M. Bethge, and W. Brendel, "Foolbox Native: Fast Adversarial Attacks to Benchmark the Robustness of Machine Learning Models in PyTorch, TensorFlow, and JAX," Journal of Open Source Software, vol. 5, no. 53, 2020.

[37] M. H. Bin Kamilin, S. Yamaguchi, and M. A. Bin Ahmadon, "Leveraging Trusted Input Framework to Correct and Forecast the Electricity Load in Smart City Zones Against Adversarial Attacks," presented at the 2024 International Conference on Future Technologies for Smart Society (ICFTSS), Kuala Lumpur, Malaysia, 7-8 August 2024, 2024.

[38] M. Ren, Y.-L. Wang, and Z.-F. He, "Towards Interpretable Defense Against Adversarial Attacks via Causal Inference," Machine Intelligence Research, vol. 19, no. 3, pp. 209—226, 2022.

[39] H. Kwon and J. Lee, "Diversity Adversarial Training against Adversarial Attack on Deep Neural Networks," Symmetry, vol. 13, no. 3, 2021.

[40] H. H. Dreany and R. Roncace, "A Cognitive Architecture Safety Design for Safety Critical Systems," Reliability Engineering & System Safety, vol. 191, 106555, 2019.

[41] H. Gupta, P. Agarwal, K. Gupta, S. Baliarsingh, O. Vyas, and A. Puliafito, "FedGrid: A Secure Framework with Federated Learning for Energy Optimization in the Smart Grid," Energies, vol. 16, no. 24, 8097, 2023.

[42] B. Shi, X. Zhou, P. Li, W. Ma, and N. Pan, "An IHPO-WNN-Based Federated Learning System for Area-Wide Power Load Forecasting Considering Data Security Protection," Energies, vol. 16, no. 19, 6921, 2023.

[43] Y. Zhou, Y. Ge, and L. Jia, "Double Robust Federated Digital Twin Modeling in Smart Grid," IEEE Internet of Things Journal, vol. 11, no. 24, pp. 39913–39931, 2024.

[44] New York Independent System Operator. "Load Data." www.nyiso.com Accessed: 22 April 2024. [Online]. https://www.nyiso.com/load-data.

[45] K. J. Lee, J. B. Carlin, J. A. Simpson, and M. Moreno-Betancur, "Assumptions and analysis planning in studies with missing data in multiple variables: moving beyond the MCAR/-MAR/MNAR classification," International Journal of Epidemiology, vol. 52, no. 4, pp. 1268–1275, 2023.

[46] M. Sun, L. Lan, C.-G. Zhu, and F. Lei, "Cubic spline interpolation with optimal end conditions," Journal of Computational and Applied Mathematics, vol. 425, 115039, 2023.

[47] M. Lima, M. Neto, T. Silva Filho, and R. A. d. A. Fagundes, "Learning under concept drift for regression—a systematic literature review," IEEE Access, vol. 10, pp. 45410–45429, 2022.

[48] F. Hinder, V. Vaquet, J. Brinkrolf, and B. Hammer, "Model-based explanations of concept drift," Neurocomputing, vol. 555, 126640, 2023.

[49] F. Matteo, Z. Gianluca, and V. Simone, "Conformal prediction: A unified review of theory and new challenges," Bernoulli, vol. 29, no. 1, pp. 1–23, 2023.

[50] B. T. Familoni, "Cybersecurity Challenges in the Age of AI: Theoretical Approaches and Practical Solutions," Computer Science & IT Research Journal, vol. 5, no. 3, pp. 703–724, 2024.

[51] M. Macas, C. Wu, and W. Fuertes, "Adversarial Examples: A Survey of Attacks and Defenses in Deep Learning-Enabled Cybersecurity Systems," Expert Systems with Applications, vol. 238, 2024.

[52] A. Makuvaza, D. S. Jat, and A. M. Gamundani, "Deep Neural Network (DNN) Solution for Real-time Detection of Distributed Denial of Service (DDoS) Attacks in Software Defined Networks (SDNs)," SN Computer Science, vol. 2, no. 2, 107, 2021.

[53] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A Survey on Distributed Machine Learning," ACM Computing Survey, vol. 53, no. 2, 30, 2020.

[54] S. M. Ribeiro and C. Castro, "Missing data in time series: A review of imputation methods and case study," Learning and Nonlinear Models, vol. 20, no. 1, pp. 31–46, 2022.

[55] Chollet, F. et al. "Keras" www.keras.io. Accessed: 30 January 2025. [Online] https://keras.io/getting_started/.

[56] M. Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning," In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2-4 November 2016, pp. 265–283.

[57] X. Fan, C. Tao, and J. Zhao, "Advanced Stock Price Prediction with xLSTM-Based Models: Improving Long-Term Forecasting," in 2024 11th International Conference on Soft Computing & Machine Intelligence (ISCMI), 2024.

[58] New York State Government Data "HVAC Market Share by Efficiency and Capacity: Beginning 2017." https://data.ny.gov/Energy-Environment/HVAC-Market-Share-by-Efficiency-and-Capacity-Begin/tf22-v9nz (accessed on 21 November 2024).

[59] M. H. Bin Kamilin, S. Yamaguchi, and M. A. Bin Ahmadon, "Radian Scaling and Its Application to Enhance Electricity Load Forecasting in Smart Cities Against Concept Drift," Smart Cities, vol. 7, no. 6, pp. 3412-3436, 2024.

[60] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," stat, vol. 1050, no. 9, 2017.

[61] M. H. Bin Kamilin, M. A. Bin Ahmadon, and S. Yamaguchi, "An Auto-Scheduling Framework for the Internet of Things Based on Process and Optimizer Modules," presented at the 2019 IEEE 8th Global Conference on Consumer Electronics (GCCE), 15-18 October, 2019.

[62] M. Garcia, "Elections in a Distributed Computing System," IEEE Transactions on Computers, vol. C-31, no. 1, pp. 48–59, 1982.

[63] V. K. Madisetti and S. Panda, "A dynamic leader election algorithm for decentralized networks," Journal of Transportation Technologies, vol. 11, no. 3, pp. 404–411, 2021.

[64] S. Liu, N. Gupta, and N. H. Vaidya, "Approximate Byzantine Fault-Tolerance in Distributed Optimization," presented at the Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, 2021

[65] M. H. Bin Kamilin and S. Yamaguchi, "Resilient Electricity Load Forecasting Network with Collective Intelligence Predictor for Smart Cities," Electronics, vol. 13, no. 4, 718, 2024.

[66] I. de Zarzà, J. de Curtò, E. Hernández-Orallo, and C. T. Calafate, "Cascading and Ensemble Techniques in Deep Learning," Electronics, vol. 12, no. 15, p. 3354, 2023.

[67] M. H. D. M. Ribeiro and L. dos Santos Coelho, "Ensemble approach based on bagging, boosting and stacking for short-term prediction in agribusiness time series," Applied Soft Computing, vol. 86, 105837, 2020.

[68] A. Sarıkaya, B. G. Kılıç, and M. Demirci, "RAIDS: Robust autoencoder-based intrusion detection system model against adversarial attacks," Computers & Security, vol. 135, 103483, 2023.

[69] J. Kang, M. Kim, J. Park, and S. Park, "Time-Series to Image-Transformed Adversarial Autoencoder for Anomaly Detection," IEEE Access, vol. 12, pp. 119671–119684, 2024.

[70] M. H. Bin Kamilin, M. A. Bin Ahmadon, and S. Yamaguchi, "Multi-Task Learning-Based Task Scheduling Switcher for a Resource-Constrained IoT System," Information, vol. 12, no. 4, 150, 2021.

[71] U. M. Sirisha, M. C. Belavagi, and G. Attigeri, "Profit Prediction Using ARIMA, SARIMA and LSTM Models in Time Series Forecasting: A Comparison," IEEE Access, vol. 10, pp. 124715–124727, 2022.

[72] S. Ghodratnama and R. Boostani, "An efficient strategy to handle complex datasets having multimodal distribution," in ISCS 2014: Interdisciplinary Symposium on Complex Systems, 2015: Springer, pp. 153–163.

[73] M. Koosha, G. Khodabandelou, and M. M. Ebadzadeh, "A hierarchical estimation of multimodal distribution programming for regression problems," Knowledge-Based Systems, vol. 260, 110129, 2023.

[74] D. Stefanovski, M. Schulze, and G. Althouse, "Multimodal distribution and its impact on the accurate assessment of spermatozoa morphological data: Lessons from machine learning," Animal Reproduction Science, 107564, 2024.

[75] R. H. A. Shiekh and E. F. El-Hashash, "A comparison of the pearson, spearman rank and kendall tau correlation coefficients using quantitative variables," Asian Journal of Probability and Statistics, pp. 36–48, 2022.

Acknowledgement

I am very grateful to Professor Shingo Yamaguchi for his support and encouragement to help me explore new ideas and to not be afraid of new challenges throughout this work. In addition, I wish to express my gratitude to Associate Professor Mohd Anuaruddin Bin Ahmadon (Universiti Teknologi PETRONAS) for his advice and insight.

I would like to express my appreciation to the members of the examination committee for this thesis: Professor Shingo Mabu, Professor Hideaki Nakamura, Associate Professor Toshikazu Samura, and Associate Professor Yuanyuan Wang for their careful reading and precious comment on this thesis.

Finally, I express my sincere gratitude to my parents for their invaluable support and encouragement.

List of Publications

[1] M. H. Bin Kamilin, M. A. Bin Ahmadon, and S. Yamaguchi, "Multi-Task Learning-Based Task Scheduling Switcher for a Resource-Constrained IoT System," Information, vol. 12, no. 4, 150, 2021.

[2] M. H. Bin Kamilin and S. Yamaguchi, "Resilient Electricity Load Forecasting Network with Collective Intelligence Predictor for Smart Cities," Electronics, vol. 13, no. 4, 718, 2024.

[3] M. H. Bin Kamilin, S. Yamaguchi, and M. A. Bin Ahmadon, "Radian Scaling and Its Application to Enhance Electricity Load Forecasting in Smart Cities Against Concept Drift," Smart Cities, vol. 7, no. 6, pp. 3412-3436, 2024.

[4] M. H. Bin Kamilin, M. A. Bin Ahmadon, and S. Yamaguchi, "An Auto-Scheduling Framework for the Internet of Things Based on Process and Optimizer Modules," presented at the 2019 IEEE 8th Global Conference on Consumer Electronics (GCCE), 15-18 October, 2019.

[5] M. H. Bin Kamilin, M. A. Bin Ahmadon, and S. Yamaguchi, "Evaluation of Process Arrangement Methods Based on Resource Constraint for IoT System," presented at the 2020 8th International Conference on Information and Education Technology, Okayama, Japan, 23 May, 2020.

[6] M. H. Bin Kamilin, S. Yamaguchi, and M. A. Bin Ahmadon, "Fault-Tolerance and Zero-Downtime Electricity Forecasting in Smart City," presented at the 2023 IEEE 12th Global Conference on Consumer Electronics (GCCE), 10-13 October, 2023.

[7] M. H. Bin Kamilin, S. Yamaguchi, and M. A. Bin Ahmadon, "Radian Scaling: A Novel Approach to Preventing Concept Drift in Electricity Load Prediction," presented at the 2023 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia), 23-25 Oct. 2023, 2023.

[8] M. H. Bin Kamilin, S. Yamaguchi, and M. A. Bin Ahmadon, "Leveraging Trusted Input Framework to Correct and Forecast the Electricity Load in Smart City Zones Against Adversarial Attacks," presented at the 2024 International Conference on Future Technologies for Smart Society (ICFTSS), Kuala Lumpur, Malaysia, 7-8 August, 2024.