Statistical Transformations and Normalization Strategies in Transformer Models for Time Series Forecasting

by

Subas Rana

(Under the Direction of John A Miller)

Abstract

Time series forecasting plays an important role in predicting future outcomes from past observations, especially in applications such as infectious diseases, where accurate and timely forecasts can guide public health decisions. However, forecasting indicators such as COVID-19 or influenza-like illness (ILI) cases remains challenging due to the limited, non-stationary, and skewed nature of infectious disease data. Deep learning models, particularly Transformers, typically require large datasets to generalize effectively, making them less reliable during the early stages of an outbreak when only small datasets are available. Moreover, common global preprocessing methods used in Transformers, such as z-normalization, are sensitive to extreme values and assume data to be roughly normal, which does not hold for infectious diseases data characterized by distributional shifts, extreme values, and skewness. This dissertation aims to address these challenges through a two-fold approach. First, it evaluates sixteen forecasting models, ranging from statistical and recurrent models to Transformers, under both small and large data conditions. A retraining strategy is integrated into deep learning models for limited datasets to allow the model to relearn from newly available data, enabling it to remain adaptive and accurate when forecasting future points. Second, the dissertation applies statistical transformations (Logarithmic, Square Root, Yeo–Johnson, Box–Cox, and Differencing) before z-normalization to better handle the challenges inherent in infectious disease time series. The univariate Box–Cox transformation is extended to a multivariate form that jointly estimates parameters using feature covariances. Previous studies have remained largely theoretical or implemented only in R, whereas this work provides the first practical Python implementation. Finally, a new instance normalization technique called Context-aware Instance Normalization (CoIN) is proposed to overcome the limitations of Reversible Instance Normalization (RevIN), which does not account for differing statistical characteristics between input and forecast horizons. Overall, this work contributes toward improving the generalization and adaptability of Transformer-based forecasting models for real-world data. Extensive experiments have been conducted on viral disease datasets to demonstrate the effectiveness of the proposed methods.

Index words: Multivariate Time Series Forecasting, Infectious Disease Forecasting, Transformers, Statistical Transformations, Data Preprocessing, Standardization,

Multivariate Box–Cox, Instance Normalization, Data Non-Stationarity, Data Scarcity, Data Skewness, Heteroskedasticity, Deep Learning

Statistical Transformations and Normalization Strategies in Transformer Models for Time Series Forecasting

by

Subas Rana

B.S.(CS), COMSATS University Islamabad, Pakistan, January 2019

A Dissertation Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the Degree.

Doctor of Philosophy

Athens, Georgia

2025

Statistical Transformations and Normalization Strategies in Transformer Models for Time Series Forecasting

by

Subas Rana

Major Professor: John A Miller

Committee: Ismailcem B Arpinar
Yuan Ke
Ninghao Liu

Electronic Version Approved:

Ron Walcott
Dean of the Graduate School
The University of Georgia
December 2025

Dedication

To my beloved father,
whose unwavering belief in me made this journey possible.
Coming from a world where women's education was often discouraged,
he chose courage over convention and stood beside me at every step.
His endless support, encouragement, and sacrifices have been my greatest strength.
Every milestone I achieve is a reflection of his faith, love, and determination.
This work is and will always be for him.

Acknowledgments

I am profoundly grateful to Almighty Allah for His countless blessings, guidance, and mercy throughout this journey. It is through His will and grace that I was able to persevere and accomplish this milestone.

I would first like to express my deepest gratitude to my advisor, Dr. John A. Miller, for his invaluable guidance, wisdom, and support throughout this journey. His vast knowledge, insightful feedback, and constant encouragement have shaped my research and helped me grow as both a scholar and a person. I am truly honored to have worked under his mentorship.

I would also like to extend my sincere appreciation to my committee members, Dr. Ismailcem B. Arpinar, Dr. Ninghao Liu, and Dr. Yuan Ke, for their time, thoughtful input, and constructive suggestions, which have greatly enriched this work.

My heartfelt thanks go to the Higher Education Commission (HEC) of Pakistan for awarding me the scholarship that made this Ph.D. possible. Their support provided me with the opportunity to pursue my studies at the University of Georgia and dedicate myself fully to my research.

I owe immense gratitude to my parents, whose love and prayers have been my greatest strength. My mother and father have been my pillars of support, and everything I have achieved is because of their belief in me. To my elder siblings, Reesham and Ferhad, who always believed in me, and to my younger siblings, Yumna and Zakyas, whose playful demands reminded me why I must keep striving — thank you for keeping me grounded and motivated.

A special thanks to my husband, Danish, who may have witnessed only the last three years of my Ph.D., but they were the most crucial ones. His constant encouragement, patience, and belief in my abilities gave me the strength to keep going, even on the hardest days. I am also deeply thankful to my parents-in-law for their heartfelt prayers and to my sisters-in-law for their kindness and constant love throughout this journey.

Lastly, I want to thank my dear friends, Saba, Fereshteh, Nazish, Samantha, and Nasid, for their love, laughter, and companionship. You have been there through every high and low, and I am forever grateful for your presence in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Time Series Forecasting

A time series represents data collected at regular time intervals. Time Series Forecasting (TSF) leverages these historical, time-ordered data points to build models that can anticipate future trends or behaviors. Over the years, TSF has become a major area of research, finding applications in diverse fields such as weather H. Wu et al., 2023; Y. Zhang et al., 2023, traffic flow Lippi et al., 2013; Lv et al., 2014, infectious disease N. Wu et al., 2020; Zeroual et al., 2020, and electricity Weron, 2014, where it plays a key role in guiding decisions and shaping strategic planning. In the context of infectious diseases[1], forecasts of indicators such as influenza-like illness (ILI) and COVID-19 cases or deaths inform resource allocation, public-health advisories, and policy interventions. In recent years, particularly during the COVID-19 pandemic, TSF has played a crucial role in helping organizations formulate policies to manage and control the spread of the virus CDC, 2022. Its ability to find patterns over time and predict what might happen next has made it an important tool for understanding complex and changing situations.

Forecasting tasks generally fall into two categories: univariate and multivariate. The **univariate** approach involves predicting a single variable based solely on its past values, without considering the influence of other factors. In contrast, **multivariate** forecasting incorporates multiple interrelated variables to improve prediction accuracy. These variables are often highly correlated and may exhibit linear or nonlinear dependencies. For instance, in the case of COVID-19, the number of deaths is strongly influenced by the number of hospitalizations and ICU patients, as illustrated in Figure 1.1. Beyond cross-feature dependencies, time series data exhibit **temporal dependencies**, meaning each observation is influenced by its previous values. This sequential nature distinguishes TSF from traditional regression problems, as forecasting models must learn to capture both short-term fluctuations and long-term components such as trends and seasonality. The dissertation focuses on **Multivariate Time Series Forecasting (MTSF)** of pandemic indicators under data conditions that are small, skewed, and non-stationary.

[1] In this work, we use the terms infectious disease, viral disease, disease outbreak, epidemic, and pandemic interchangeably to broadly refer to time series related to viral disease forecasting. For context, an epidemic refers to a sudden and unexpected increase in cases within a region, whereas a pandemic is an epidemic that spreads across countries or continents.

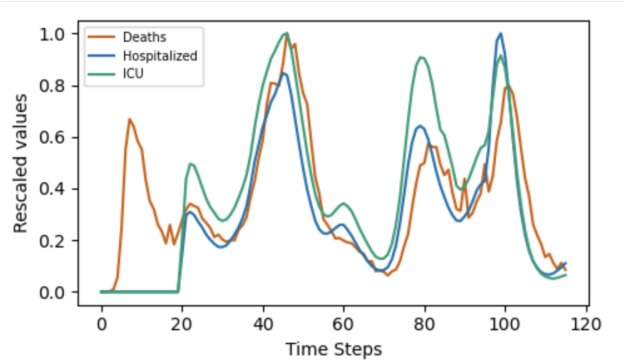


Figure 1.1: COVID-19 deaths, hospitalized, and ICU patients, indicating a strong linear relationship.

1.2 Challenges in Infectious Disease Time Series

Forecasting infectious disease time series (e.g. COVID-19 cases, ILI rates, Respiratory syncytial virus (RSV) detections) presents numerous challenges. These challenges stem from the nature of epidemiological data and the dynamic processes generating them. In what follows, we outline the major issues that make infectious disease forecasting difficult.

1.2.1 Data Scarcity in Early Disease Outbreaks

Deep learning models typically require large and diverse datasets to capture complex temporal dependencies and achieve stable generalization. However, during the early stages of emerging pandemics such as COVID-19, available data are limited. This scarcity makes it difficult for models to learn meaningful temporal patterns, leading to overfitting or unstable forecasts. The limited and inconsistent nature of pandemic datasets underscores the need to develop strategies that ensure forecasting reliability even when only limited data are available and to investigate how different models perform under small-data conditions, such as the early COVID-19 period.

1.2.2 Non-Stationarity and Structural Changes

A stationary time series is one whose statistical properties do not depend on the time at which the series is measured Hyndman, 2018, while a non-stationary series has statistical properties that change over time. A time series is said to be weakly stationary if its mean and variance remain constant over time, and the autocovariance between observations depends only on the time lag, not on the specific time at which they occur. Pandemic time series are inherently non-stationary, meaning their statistical properties, such as mean and variance, change over time. These changes arise from evolving epidemic conditions and external

interventions. Factors such as public health interventions (lockdowns, mask mandates, vaccination drives), changes in human behavior, and viral mutations (like new COVID-19 variants) can all create sudden shifts in how a disease spreads. As a result, a model that performs well on past data may quickly lose accuracy when conditions change. A related challenge, called **distributional shift**, happens when the statistical distribution of new unseen data differs from the data used to train the model. In time series forecasting, this often means that future values show different levels, variances, or patterns compared to earlier observations. For example, Figure 1.2 shows a clear distributional shift in the ILI dataset: the training data (blue) are concentrated around smaller values, while the testing data (red) are shifted toward larger values with a longer tail. This indicates that the mean and variance of the test set differ substantially from those of the training set. The test data also contains very large values, suggesting that a model trained on earlier, lower-value data may struggle to generalize effectively to these later, higher-value samples. Such behavior exemplifies a distributional shift in time series data, often resulting from changes in disease dynamics. These non-stationary and uneven patterns break the assumption that training and testing data come from

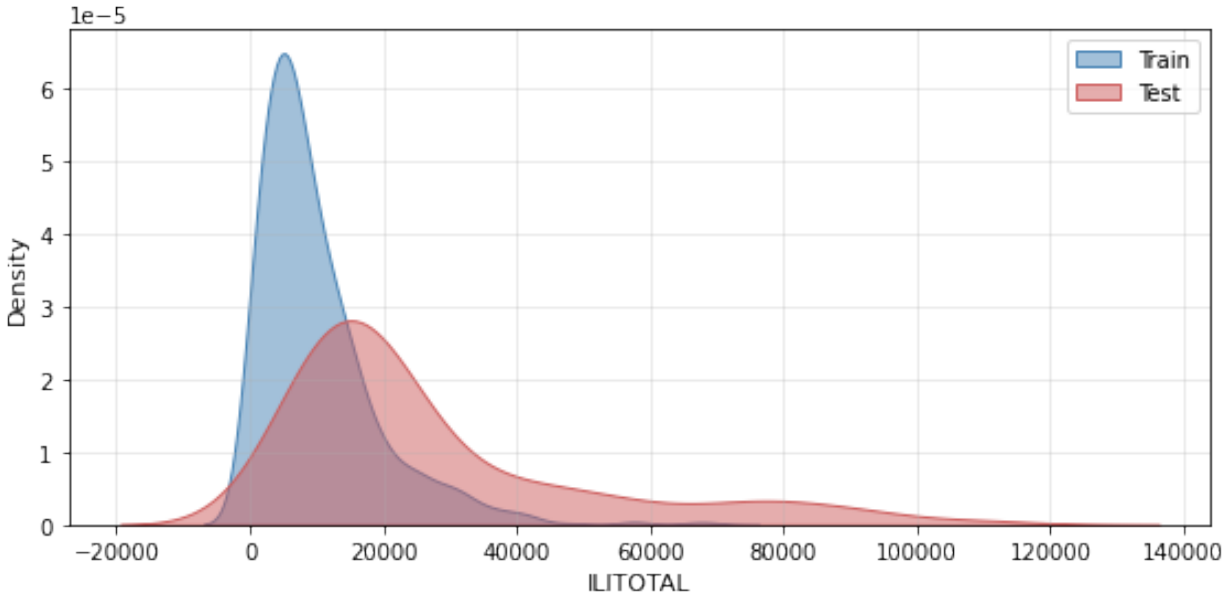


Figure 1.2: Distributional shift between the training and testing sets of the ILI dataset. The training data (blue) is concentrated around lower ILITOTAL values, while the testing data (red) is shifted toward higher values.

the same distribution. Therefore, identifying and addressing distributional shifts is crucial for developing forecasting models that can remain reliable under changing epidemic conditions.

The stationarity of a time series can be assessed using tests available in Statsmodels Seabold and Perktold, 2010, such as the Augmented Dickey–Fuller (ADF) and Kwiatkowski–Phillips–Schmidt–Shin (KPSS) tests. The ADF test examines whether a unit root is present in a time series. A *unit root* is a property of certain stochastic processes in which the effects of random shocks persist indefinitely rather than dissipating over time. A unit root can be illustrated using a simple autoregressive (AR) model. For

an AR(1) process,

$$x_t = \phi x_{t-1} + \varepsilon_t,$$

the process contains a unit root when $\phi = 1$. In this case, shocks accumulate instead of decaying, causing the variance to grow without bound and preventing the series from reverting to a stable long-term mean. As a result, the process is non-stationary. The hypothesis of ADF test are:

$$H_0 : \text{the series has a unit root (non-stationary)},$$
$$H_1 : \text{the series is stationary}.$$

The test reports both a test statistic and a corresponding p-value. The p-value represents the probability of obtaining a statistic as extreme as the observed one under the assumption that the null hypothesis H_0 is true. The decision is made by comparing the p-value with a chosen significance level α, commonly $\alpha = 0.05$, which corresponds to a 5% tolerance for Type I error (committing a false positive by rejecting a true null hypothesis). If $p < \alpha$, we reject H_0 and conclude that the series is stationary.

The KPSS test complements the ADF test by reversing the hypothesis:

$$H_0 : \text{the series is stationary},$$
$$H_1 : \text{the series is non-stationary}.$$

As with the ADF test, the KPSS statistic also comes with a p-value, and the decision is made in the same way: if $p < \alpha$, we reject the null hypothesis of stationarity and infer that the series is non-stationary.

Using both tests together provides a more reliable assessment of stationarity. The ADF test looks for evidence of a unit root, while the KPSS test looks for evidence against stationarity. When both tests lead to the same conclusion, the classification of the series is strengthened. When they disagree, the series may be borderline or trend-stationary. The test results for the time series datasets are provided in Appendix B.2.

1.2.3 Skewness and Heteroskedasticity

Right Skewed Counts. Pandemic data such as new cases, hospitalizations, and deaths usually show a right-skewed distribution. This means that values are low most of the time but occasionally rise sharply during major outbreaks. These sudden increases create a long right tail in the distribution. As a result, models that assume normally distributed (symmetric) data are not suitable. In diseases such as COVID-19, daily case numbers remain low for many weeks and then suddenly surge to a peak before declining again. The right skewness indicates the presence of very large values. Figure 1.3 shows the histograms of seven ILI features, where most of the features exhibit right skewness, which can significantly affect the forecasting performance of the model.

Normality can be tested using the Shapiro–Wilk statistical test, which evaluates whether a time series is normally distributed. In this context, *normality* refers to whether the underlying distribution of the series matches a Gaussian distribution in terms of shape, symmetry, and tail behavior. The hypothesis of

the Shapiro–Wilk test are:

$$H_0 : \text{the data are drawn from a normal distribution,}$$
$$H_1 : \text{the data are not drawn from a normal distribution.}$$

The test produces a statistic and a corresponding p-value. The p-value represents the probability of observing a test statistic as extreme as the one computed, assuming that the null hypothesis H_0 is true. At a chosen significance level α, the decision rule is:

$$\text{Reject } H_0 \quad \text{if} \quad p < \alpha.$$

Thus, a small p-value provides evidence against normality, whereas a p-value larger than α indicates that the data do not significantly deviate from a normal distribution. The test results for the time series datasets are provided in Appendix B.2.

Heteroscedastic Variance. The variance of errors in epidemiological forecasts is not constant over time;

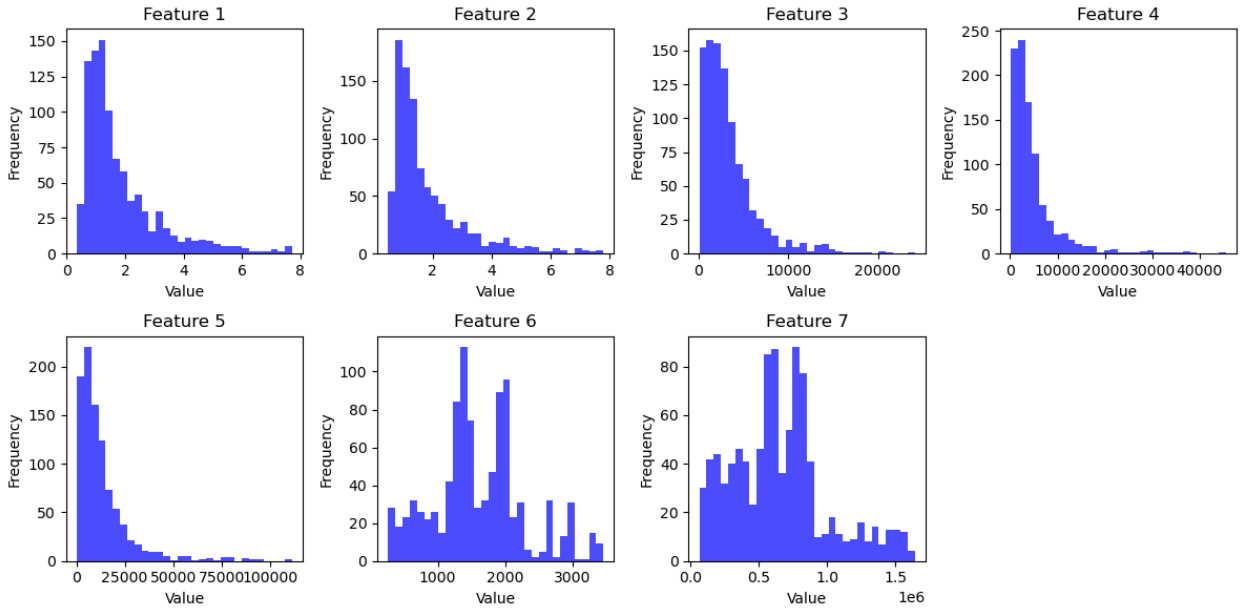


Figure 1.3: Histograms of the ILI dataset with no transformation or normalization.

it grows and shrinks with the level of epidemic activity. During major outbreaks, the errors and uncertainty in forecasts are much larger than during stable or low-incidence periods. This behavior, known as heteroscedasticity, violates the common assumption that residual errors have constant variance. In practice, a model may perform well during periods of low disease activity but underestimate uncertainty when cases or deaths surge rapidly during an outbreak. Managing heteroscedasticity remains challenging, as infectious disease data naturally fluctuates with changing disease dynamics.

Heteroscedasticity can be evaluated using the Breusch–Pagan test, which examines whether the variance of the residuals remains constant across observations. In this context, *homoscedasticity* refers to the assumption that the error variance is constant, whereas *heteroscedasticity* indicates that the variance is not constant. The hypothesis of the Breusch–Pagan test are:

$$H_0 : \text{the residuals are homoscedastic},$$
$$H_1 : \text{the residuals are heteroscedastic}.$$

The test computes a statistic and an associated p-value. At a chosen significance level α, the decision rule is:

$$\text{Reject } H_0 \quad \text{if} \quad p < \alpha.$$

A small p-value therefore provides evidence against the null hypothesis of homoscedasticity and indicates that the residuals exhibit heteroscedasticity. The test results for the time series datasets are provided in Appendix B.2.

Extreme Peaks and Long Tails: The combination of skewness and heteroscedasticity leads to highly non-normal error distributions. Forecast errors tend to be small during routine periods but can be enormous around exponential growth or a peak. A single large peak can dominate average error metrics. Models might systematically under-predict the height of peaks (since those are rare extremes) and then over-predict during the declining tail of an epidemic curve. This skewed error behavior complicates both modeling and evaluation. Forecasters have noted that their models often "overshoot at the peak" after having performed well for the rising phase, reflecting the difficulty of capturing the very top of a wave accurately Ray et al., 2021.

1.3 Limitations of Normalization in Deep Learning

Deep learning approaches commonly employ normalization techniques such as Min–Max scaling and z-normalization as a global preprocessing technique. The latter, z-normalization, is the most widely used method in recent state-of-the-art (SOTA) Transformer models and is referred to as **StandardScaler** throughout this work. It transforms each feature by subtracting its mean and dividing by its standard deviation, effectively centering the data around zero with a unit variance. In mathematical terms, for each value y, the transformation is given by:

$$y_{scaled} = \frac{y - \mu}{\sigma} \tag{1.1}$$

where y is the original value, μ is the mean, and σ is the standard deviation of the feature. The result, y_{scaled}, is the standardized value. The mean and standard deviation are obtained from only the training set instead of the entire dataset to avoid data leakage during training Kuhn and Johnson, 2019. It is typically applied globally to place variables on a comparable scale and improve numerical conditioning during training. However, this global procedure is *(i)* sensitive to extreme values, because a few large observations can

distort the estimated mean and standard deviation; *(ii)* it preserves the *shape* of the distribution, if your data is skewed (lots of small numbers and a few very big ones), it stays skewed after standardization. *(iii)* it enforces global zero mean and unit variance, improving overall scale consistency but failing to stabilize local heteroscedastic variance, and *(iv)* it does not align train and test if their distributions differ: the same global scaling just rescales the mismatch. Moreover, equation 1.1 must satisfy that the distribution is Gaussian Lirio et al., 1989; Pedregosa et al., 2011. These issues are often overlooked in deep learning models and while using normalization methods, which can degrade model generalization, especially in the case of pandemic datasets that exhibit these challenges and cause losses such as Mean Squared Error (MSE) to be dominated by high-variance regions, leading to unstable training, slower convergence, and degraded generalization.

Additionally, SOTA Transformer models Y. Liu et al., 2023; Nie, 2022; Y. Wang et al., 2024 employ instance-level normalization in addition to global preprocessing to handle non-stationarity or distribution shifts between train and test data at the instance level. The instance normalization method used by these models is Reversible Instance Normalization (RevIN) Kim et al., 2021, which mitigates the distributional shift by standardizing each input sequence using its instance-wise mean and standard deviation and reverses the predictions to the original scale by using the same statistics. It has several limitations. RevIN uses the same statistics for input and forecast points without considering the distribution discrepancy between the inputs and the forecast horizons. Second, it assumes that all time steps in a sequence share the same mean and variance, even though recent values and earlier ones may behave very differently. Third, RevIN applies a single, fixed reference, either the mean or the last value, for centering the entire sequence, ignoring that short-term forecasts depend more on recent behavior while long-term forecasts should reflect the broader trend. Consequently, RevIN applies instance normalization uniformly across all input and forecast sequences, using identical statistics (mean and variance) for all time steps within an instance. This uniformity overlooks the temporal heterogeneity inherent in many real-world time series, where recent values often carry more predictive relevance than earlier ones, and fails to adapt normalization to the differing statistical properties of various temporal segments or forecast horizons.

1.4 Proposed Solutions

To address the limited data availability observed during pandemics such as COVID-19, this dissertation first evaluates time-series models under both large and small datasets and then proposes a retraining strategy for deep learning models. Building on these findings, the dissertation further investigates data-related challenges within pandemic time series, particularly those linked to normalization practices in deep learning. These approaches are discussed below:

1.4.1 Effectiveness of Time Series Models for Infectious Diseases Forecasting

With the emergence of COVID-19, transformer-based models show limited effectiveness due to the limited data H. Wu et al., 2021, as they are inherently data-hungry and require large datasets to capture long-range

dependencies and generalize well. They fail to capture the extreme values in the dataset, which is crucial for disease control and prevention. To investigate this limitation, we evaluate sixteen time-series forecasting models across both early and later stage pandemic data to assess their performance under varying data availability conditions.

We also implement a retraining strategy for deep learning models. Each model is iteratively retrained after producing the first h-step (future time steps) forecast, allowing it to adapt continuously as new data becomes available. This strategy helps the model remain relevant to the evolving time series, as performance typically deteriorates at later steps when the temporal distance from the training data increases Ray et al., 2021. By retraining on the most recent observations, the model can update its parameters and maintain forecasting accuracy over time.

1.4.2 Transformations and Normalization in Transformers

The proposed solutions for the limitations in section 1.3 are listed below:

Statistical Transformations

This dissertation investigates classical statistical transformations, including Logarithmic, Square Root, Box-Cox, Yeo-Johnson, and Differencing, applied before standardization in transformer-based architectures to handle statistical challenges such as skewness, heteroscedastic variance, extreme peaks, and distributional shifts present in the pandemic datasets. We also extend univariate Box-Cox to a multivariate setting.

Multivariate Box-Cox

The Box–Cox transformation is a commonly used method to stabilize variance and achieve normality. It applies a power transformation to each feature individually. In its classical form, the transformation parameter for each feature is estimated independently by maximizing a univariate log-likelihood, which depends only on that feature's variance Box and Cox, 1964. However, this independent treatment ignores correlations between features, which is problematic for multivariate time series where variables are often interdependent, for example, mortality counts and hospital admissions in COVID-19 data. To address this, we extend the Box–Cox formulation to a multivariate setting by jointly estimating all transformation parameters using a covariance-based log-likelihood. This joint estimation accounts for both individual feature variances and cross-feature covariances, producing transformation parameters that better capture the dependency structure among correlated variables and improve overall variance stabilization.

Previous works Andrews et al., 1971; Rahman and Pearson, 2009; Velilla, 1992 on multivariate Box–Cox transformations have largely remained theoretical, with limited practical development. Existing implementations have primarily been available in **R**, and most studies have focused only on simulation-based analyses . In this study, we bridge this gap by developing the first complete **Python implementation** of the multivariate Box–Cox transformation, formulated consistently with Scikit-learn's univariate Box–Cox implementation. We further extend its scope by conducting comprehensive experiments on real-world

datasets, demonstrating its practical effectiveness and integration potential within modern machine learning workflows.

Context-Aware Instance Normalization

To address RevIN's limitation, we propose Context-aware Instance Normalization (CoIN), a model-agnostic extension of RevIN that explicitly considers both the look-back window and the forecasting horizon during normalization and denormalization. CoIN adaptively shifts recent time steps using the last observed value to preserve continuity near the forecast point, while earlier parts of the sequence are normalized using the instance mean to maintain historical stability. During denormalization, short-term forecasts are rescaled relative to the last observed value, as these predictions are temporally close to the most recent data point and still influenced by its level. In contrast, longer-term forecasts are rescaled using the mean, since the effect of the last observation gradually fades over time, and relying on it can cause unrealistic drift in later horizons. Overall, CoIN provides a reversible, horizon-aware normalization framework that adapts to the structure of both the input and the forecast. CoIN is architecture-independent and can be seamlessly integrated into any neural forecasting model.

1.5 Contributions

The main contributions of this dissertation are summarized as follows:

1. A comprehensive comparison of sixteen statistical and machine learning models, ranging from simple to advanced architectures, is presented. This study highlights which model categories are most effective across different stages of infectious disease progression, an aspect often neglected in prior time series research. The results show that PatchTST and SARIMA are the most suitable models for large datasets and long-term forecasting, effectively capturing long-range temporal dependencies, whereas Seq2Seq with attention performs best for short-term forecasting and rapidly changing conditions.

2. An empirical evaluation of models submitted to the Centers for Disease Control and Prevention (CDC) for forecasting U.S. incident weekly deaths at the national level, demonstrating that Sequence-to-Sequence models with RNN variants and vanilla Transformer outperform the CDC-submitted models.

3. A detailed analysis showing that optimizing hyperparameters of existing Transformer-based models yields substantial performance improvements.

4. A retraining framework demonstrating that iterative retraining enhances model adaptability and accuracy, particularly for datasets with limited samples, by continuously integrating recent observations into model updates. This strategy improved the performance of ten (out of sixteen) models, including both Transformer-based and other deep learning architectures.

5. A systematic investigation of classical statistical transformations, including Logarithmic, Square Root, Box–Cox, Yeo–Johnson, and Differencing, applied prior to standardization in Transformers, revealing their effectiveness in correcting skewness, stabilizing variance, handling extremely large values, and mitigating distributional shifts beyond what standard normalization achieves.

6. A multivariate extension of the univariate Box–Cox transformation is introduced, where transformation parameters are jointly estimated by leveraging feature covariances to preserve inter-feature dependencies. While earlier studies and implementations were mainly limited to theoretical analyses and R-based environments, this work presents the first practical implementation in Python and provides a comprehensive empirical evaluation within deep learning–based time series forecasting.

7. The development of a novel *Context-aware Instance Normalization (CoIN)* technique that dynamically balances mean and last-value normalization according to input characteristics and forecast horizons, offering a simple yet effective normalization approach for deep neural networks.

8. Empirical results in this dissertation demonstrate that integrating statistical transformations and improved normalization techniques into Transformer-based time series forecasting models consistently enhances performance, achieving up to 39.4% improvement in sMAPE and 19.9% in MAE on the ILI dataset, 15.2% in sMAPE and 16.9% in MAE on the COVID-19 dataset, and 44% in sMAPE and 6.89% in MAE on the RSV dataset compared to standard preprocessing, as well as up to 24.6% improvement over the existing instance normalization method.

1.5.1 List of Publications

Below is a list of publications that have resulted from this dissertation.

1. Rana, S., Barna, N.H. and Miller, J.A., 2023, September. Exploring the predictive power of correlation and mutual information in attention temporal graph convolutional network for COVID-19 forecasting. In International Conference on Big Data (pp. 18-33). Cham: Springer Nature Switzerland.

2. Rana, S., Miller, J.A., Nesbit, J., Barna, N.H., Aldosari, M. and Arpinar, I.B., 2024, November. How Effective are Time Series Models for Pandemic Forecasting?. In International Conference on Big Data (pp. 3-17). Cham: Springer Nature Switzerland.

3. Rana, S., Miller, J.A. (2025, September). Normalization in Transformers. Submitted to the European Conference on Computer Systems (EuroSys 2026).

1.6 Additional Work: Graph-based Modeling for COVID-19 Forecasting

In addition to transformer-based forecasting models, this dissertation also explored a Graph Neural Network (GNN) approach for modeling the spatial and temporal spread of COVID-19 across U.S. states. Specifically, we implemented the Attention Temporal Graph Convolutional Network (A3T-GCN) model from PyTorch Geometric Temporal to forecast state-level COVID-19 mortality as presented in our earlier work Rana et al., 2023. The model combines temporal attention mechanisms with graph convolution to capture temporal and spatial dependencies among states. To establish connections between highly related regions, we constructed state graphs using Pearson's correlation and Mutual Information (MI), enabling the model to capture both linear and nonlinear spatial relationships. Experimental results showed that the MI-based graph structure led to the best performance, outperforming baselines and adjacency-based graphs, and achieving the second-highest forecasting accuracy among all models submitted to CDC. All implementation details, experimental settings, and results for this work are provided in Appendix A.

Chapter 2

Effectiveness of Time Series Models for Infectious Disease Forecasting

Rana, S. Accepted by International Conference on Big Data. Reprinted here with permission of publisher, 11/14/2024

The increasing frequency and impact of pandemics highlight the need for better preparedness and accurate prediction of infectious disease outbreaks. While traditional tools remain valuable, the focus on predicting infectious diseases has shifted towards time series analysis using statistical and machine learning models. However, the emergence of simple linear models has recently challenged the dominance of complex Transformer-based architectures, demonstrating that model effectiveness does not always scale with complexity. This paradigm shift, combined with the persistent challenge of limited data availability during early pandemic stages, necessitates a critical reevaluation of time series forecasting approaches. This research Rana et al., 2024 analyzes the performance of sixteen time series models, including simple and complex architectures, in forecasting pandemic data. We evaluate the models on the United States weekly ILI cases, RSV Detections, and COVID-19 incident deaths. By evaluating models on datasets of different scales, we aim to identify effective forecasting methods for early-stage and later-stage pandemics. To further address the limited data problem, we employ a retraining strategy, where models are periodically updated as new data becomes available, improving their temporal adaptability and stability. In line with our broader reevaluation of time series models, we also examine the accuracy of the Centers for Disease Control and Prevention models predicting COVID-19 weekly incident deaths on a national level.

2.1 Introduction

In 2024, the global community finds itself in the aftermath of a global health emergency caused by the COVID-19 pandemic. Historical pandemics, from the Russian Flu to influenza, have shaped our response strategies, including quarantine, social distancing, and vaccination. Despite these efforts, COVID-19 caused immense loss, with over 1.15 million deaths in the U.S. by December 2023. This highlights the urgent need for improved preparedness for future pandemics. A significant shift has occurred towards using time series analysis for predicting and managing infectious diseases. Historical data is crucial in predicting infectious diseases. Analyzing this data with predictive techniques, including statistical methods and machine learning (ML), improves outbreak forecasting. These models help public health authorities, like the Centers for Disease Control and Prevention (CDC), effectively communicate forecasts to the public.

Studies Kondo et al., 2019; L. Liu et al., 2018; Q. Liu et al., 2021 highlight the effectiveness of ML models in accurately predicting the progression of infectious diseases. Recurrent neural networks (RNNs) and their variants, including Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), have gained attention for time series pandemic forecasting. Following the success of RNNs, encoder-decoder models have demonstrated promising outcomes across various tasks. Statistical methods such as Auto-Regressive, Integrated, and Moving Averages (ARIMA) have proven quite competitive. Additionally, Transformers, known for their success in natural language processing Vaswani et al., 2017, have achieved state-of-the-art (SOTA) performance in influenza forecasting N. Wu et al., 2020. Surprisingly, a simple linear model Zeng et al., 2023 has recently outperformed complex Transformer models for influenza-like illness (ILI). This research encourages a reevaluation of model design principles in time series forecasting (TSF). It calls for further exploration of simpler, more efficient model structures that can deliver strong performance without sacrificing interpretability.

In this research, we conduct a comprehensive examination of various models using pandemic time series data, encompassing weekly ILI, Respiratory Syncytial Virus (RSV) and the COVID-19 dataset. The distinctive saw-tooth pattern observed in daily COVID-19 deaths has led some modeling studies to prefer working with weekly data. As the COVID-19 reporting transitioned to weekly during its later stages, the available data for training models diminished significantly. In contrast, we have an ample amount of data for ILI and RSV, enabling thorough model training. Recognizing the critical importance of accurate forecasts, especially in the early stages of a pandemic, we assess the performance of these models on both small and large datasets. While some studies have previously compared COVID-19 forecasting models, our current study stands out in several ways.

1 Following the unexpected success of DLinear Zeng et al., 2023, our study performs a comprehensive comparison of sixteen, both statistical and ML models, covering simple to complex architectures. We identify which models or strategies are suitable for the early and later stages of pandemics, while existing time series research often neglects to conduct experiments in both domains.

2 To conduct a thorough assessment of time series models, our study evaluates the predictive accuracy of models submitted to the CDC for US incident weekly deaths on a national level. We aim to find which model is the most accurate in predicting COVID-19 incident deaths and whether any other models outperform the models submitted to the CDC.

3 The research findings highlight that optimizing hyperparameters for existing Transformer-based models can significantly improve their performance, as demonstrated in our study.

4 Our study underscores the importance of retraining data, highlighting how it enhances model accuracy, particularly in scenarios with limited data. By incorporating new data at each iteration, retraining ensures the model adapts to recent information, improving accuracy over time.

2.2 Existing Methods and Related Work

Many studies in infectious disease prediction focus on time series analysis, exploring various methods, including statistical, ML, and Deep Learning (DL). We evaluate all the models discussed in this section.

2.2.1 Statistical and Machine Learning Models

The **Random Walk** Miller, 2022 model suggests that at each timestep, the predicted variable takes a random step away from its previous value. Because values are equally likely to be higher or lower, the future value y_{t+k} can be represented as a past value y_t. The **SARIMA** (seasonal ARIMA) model Box and Jenkins, 1976, combines several elements: Auto-Regressive (AR(p)), which uses the previous p values; Moving-Average (MA(q)), which uses the previous q errors; and Integrated (I(d)), allows differencing d times to achieve stationarity. It also accounts for seasonality by utilizing relevant past seasonal values.

Perone, 2022 found that SARIMA outperformed ARIMA in predicting cumulative COVID-19 deaths across 12 countries, demonstrating the importance of seasonality in daily data.

Neural models, including **FeedForward Neural Networks** (FFNN) and Recurrent Neural Networks (RNNs), are widely used for univariate and multivariate time series forecasting (MTS). FFNNs rely only on current inputs, lacking the ability to capture temporal dependencies. RNNs, however, utilize the current value at time t and the preceding hidden state from time $t-1$ with three main variants: vanilla RNNs, **GRU** Cho et al., 2014, and **LSTM** Hochreiter and Schmidhuber, 1997. In this study, we only consider the GRU and LSTM models as they are improved variants designed to keep or ignore information while modeling a sequence of observations.

RNNs handle sequence problems with aligned input and output sequences but struggle with differing input/output lengths unless the last hidden activations are passed through a linear layer. The **Sequence-to-Sequence** (Seq2Seq) approach, using two RNNs (encoder and decoder), addresses this by encoding the input into a fixed-length representation and decoding it into a variable-length output Sutskever et al., 2014. **Seq2Seq models, with Attention** (Seq2Seq-Att), have shown superior performance, as demonstrated in a 2019 study Kondo et al., 2019 where Seq2Seq-Att outperformed ARIMA and LSTM for influenza prediction using Teacher Forcing. Our research experiments with LSTM and GRU in Seq2Seq models.

2.2.2 Transformers and DLinear

The **Transformer**, introduced by Vaswani et al., 2017 for language modeling, was later adapted for TSF by N. Wu et al., 2020, outperforming ARIMA, LSTM, and Seq2Seq-Att for Influenza univariate time series only. N. Wu et al., 2020 used Transformers for autoregressive predictions, while this paper implements a Transformer with direct multi-step (DMS) forecasting, predicting future values in one pass. Zeng et al., 2023 proved DMS, introduced in H. Zhou et al., 2021, to provide significant performance boosts and has attributed much of the success of later time series models to this improvement. Recent work focuses on sparse Transformer variants for Long Sequence TSF, like the **Informer** model H. Zhou et al., 2021, which uses ProbSparse self-attention and convolution-based distilling. Zeng et al., 2023 concluded that the Informer's performance gains were due to DMS, not ProbSparse attention. These forecasting models relied on sparser self-attention mechanisms, focusing on point-wise dependency. **Autoformer** H. Wu et al., 2021 introduces series-wise dependency by substituting self-attention with auto-correlation and introduces the decomposition of time series into a "trend-seasonal" component. T. Zhou et al., 2022 introduces **FEDformer** with Fourier or Wavelet enhanced attention blocks, which substitute for multi-headed attention blocks, in addition to the decomposition of time series, further improving forecasting accuracy.

Zeng Zeng et al., 2023 questioned the necessity of memory-saving measures for Transformers and hypothesized that Transformers were not effective in TSF. They proposed a simple class of linear models, **DLinear** and **NLinear**, which outperformed Transformers. DLinear decomposes the input into seasonal and trend components, applying linear layers to each before summing the features. NLinear first subtracts the input by the last value of the sequence, then passes the sequence through a linear layer, and finally, adds the initial subtracted value back to the sequence to form the prediction. However, these models

typically use 336 data points, which are not readily available in emerging disease forecasting datasets. The COVID-19 Weekly data used in this study has only 116 points, raising questions about the impact of shorter sequences on performance.

PatchTST Nie, 2022 is a Transformer-based model that differs from FEDformer, Autoformer, and Informer by using patches instead of individual points. It tokenizes the normalized time series into fixed-length, overlapped, or nonoverlapped patches and employs a channel-independent structure where each input token contains data from a single channel. This model outperformed other Transformer models, showing that longer sequence lengths enhance performance.

2.3 Experiments

2.3.1 Data and Code

Here is the description of the three datasets that we have used:

1. **COVID-19 Weekly:** The daily data was extracted from the COVID-19 Dataset by Our World in Data (OWID)[1] Ritchie et al., 2020. We included the dataset from the first death till the weekly reporting of deaths. The daily data collected was converted to the weekly data and is available from 29^{th} February 2020 to 14^{th} May 2022 on GitHub[2]. The reason for choosing these dates is to utilize the peak periods during model training and to avoid the flat trend that appears after May 2022, as shown in Fig. 2.3. Each data time point has variables, such as deaths, tests, cases, vaccinations, hospitalization, and patients admitted to the Intensive Care Units (ICU).

2. **CDC COVID-19 Weekly Deaths Forecasts:** It is collected from the COVID-19 Forecast Hub Cramer, Huang, et al., 2022. The data has been archived now and is available on GitHub[3]. It includes the 4-weeks ahead deaths forecasts for the models submitted to the CDC.

3. **ILI Weekly:** ILI[4] dataset is collected from the CDC from 1^{st} October 2002 to 30^{th} June 2020, has the ILI patients data recorded every week for the US. The data consists of variables such as weighted and unweighted ILI cases, data by age group, number of providers, and total number of affected patients. It is available on GitHub[5].

4. **RSV Weekly:** The weekly RSV data are obtained from the CDC Respiratory Syncytial Virus Laboratory Data (NREVSS) [6]. This dataset is regional and contains two types of diagnostic test

[1] https://github.com/owid/covid-19-data

[2] https://github.com/scalation/scalation_py/blob/dev_sr/PandemicForecasting/RollingWindow/datasets/covid_till14May22.csv

[3] https://github.com/scalation/data/blob/master/CDC-COVID-Data/concatenated_CDC_20_21_22_23.csv

[4] https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html

[5] https://github.com/scalation/data/blob/master/Influenza/national_illness.csv

[6] https://data.cdc.gov/Laboratory-Surveillance/Respiratory-Syncytial-Virus-Laboratory-Data-NREVSS/52kb-ccu2/about_data?utm_source=chatgpt.com

methods. We aggregate the data from both test types across all regions to obtain national-level data. The dataset includes RSV positive detections, total tests, surveillance year, and week-ending code. We further merge lagged weather variables (temperature and humidity) extracted using the METEOSTAT library, as these factors exhibit a nonlinear relationship with the virus (Correlation map given in Appendix B). The processed dataset is available on GitHub [7] and consists of 10 variables, including RSV indicators and lagged weather variables, with a total of 519 weekly samples.

Code for all the experiments is available on GitHub under Pandemic Forecasting repository[8].

2.3.2 Implementation Details

For MTS forecasting, given an input sequence $(x_1, x_2 ..., x_L)$ for each variable, where L is the sequence length, the model forecasts $(x_{L+1}, x_{L+2}, ..., x_{L+T})$, where T is the prediction length.

We implemented Neural Networks in Python using PyTorch Paszke et al., 2019 and the Transformers follow the architectural design employed by their respective papers. Hyperparameters, detailed in Tables 2.1 and 2.2, were fine-tuned via grid search. The models were trained for 20 epochs with early stopping (patience of 3) and ADAM optimizer with learning rates. Mean Squared Error (MSE) was used as the loss function, and data was standardized using StandardScaler. Experiments were repeated three times, with datasets split chronologically into training, validation, and test sets (7:1:2 for ILI and RSV; 8:2 for COVID-19, omitting validation for short datasets). The target variables were $ILITOTAL$ for ILI, $RSV Detections$ for RSV and new_deaths for COVID-19, with input and output sequences of length 7 for ILI and COVID, and 10 for RSV. Transformers utilize time embeddings due to their superior performance in Zeng et al., 2023 and a 5% dropout ratio for ILI and COVID, 4% for RSV, 8 multi-head attention heads for ILI and COVID and 4 for RSV, and a final linear layer size of 2048 for ILI and COVID, and 512 for RSV. For PatchTST, patch length and stride were 16 and 8 for ILI, 16 and 2 for RSV, and 6 and 2 for COVID-19. SARIMA was implemented using StatsModels Seabold and Perktold, 2010. Hyperparameters p, q, P, and Q ranged from 0 to 6, while d and D from 0 to 2, detailed in Table 2.3. We applied a log transformation to the variables. Models were evaluated using symmetric Mean Absolute Percentage Error (sMAPE) and Mean Absolute Error (MAE). We tested learning rates = $\{0.1, 0.01, 0.001, 0.0001, 0.2, 0.02, 0.002, 0.0004\}$, L for ILI = $\{10, 15, 24, 48, 40, 60, 72, 96, 100, 120, 144, 168, 192, 215, 220\}$, RSV = $\{36, 48, 60, 96, 104\}$, and COVID-19 = $\{6, 7, 10, 12, 15, 20, 30\}$. Layers from 1 to 5, model dimensions = $\{32, 64, 128, 256, 512, 1024, 2048\}$. T for ILI = $\{6, 12, 24, 36, 48, 60\}$, T for RSV = $\{6, 12, 24, 36, 48\}$ and COVID-19 = $\{1, 2, 3, 4, 5, 6\}$.

ReTraining (RT) the data. To address the limited COVID-19 dataset, we use retraining to update the model with the latest data in each iteration. We train on the current dataset, forecast six weeks ahead, and then roll the window forward by one week—removing the oldest and adding the newest week from the test set. This iterative process continues until the entire dataset is covered, ensuring that the model is retrained with the latest information at each step.

[7] https://github.com/scalation/scalation_py/blob/dev_sr/datasets/RSV/rsv_weather_lagged.csv
[8] https://github.com/scalation/scalation_py/tree/dev_sr/PandemicForecasting

Table 2.1: This table shows the hyperparameter details for the models based on ILI and COVID-19 datasets. In Transformer-based models, the layers consist of encoder and decoder layers.

Datasets	Hyperparameters	FFNN	LSTM	GRU	Seq2Seq-LSTM	Seq2Seq-GRU	Seq2Seq-LSTM-Att	Seq2Seq-GRU-Att	Transformer	Informer	Autoformer	FEDformer	DLinear	NLinear	PatchTST
CDC	Learning Rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.0001	0.0001	0.001	0.001	0.02	0.0001	0.0001
	Batch Size	4	16	8	8	4	8	16	8	8	8	8	16	4	32
	Model Dims	256,256	64	512	32	8	32	256	256	256	512	256	—	—	512
	Sequence Length	6	6	6	6	6	5	6	8	8	8	8	7	4	8
	Layers	—	1	3	1	3	3	2	2,1	2,1	2,1	2,1	—	—	4
COVID-19 w/ RT	Learning Rate	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.001	0.001	0.001	0.001	0.1	0.02	0.0001
	Batch Size	16	16	16	16	16	16	16	16	16	16	16	16	16	16
	Model Dims	64,64	256	128	256	256	64	64	512	128	128	128	—	—	2048
	Sequence Length	7	15	15	10	15	10	7	7	7	7	15	10	10	15
	Layers		1	2	2	1	2	2	2,1	5,4	2,1	2,1	—	—	2
COVID-19 w/o RT	Learning Rate	0.01	0.01	0.01	0.01	0.01	0.001	0.001	0.01	0.001	0.0001	0.0001	0.2	0.2	0.0001
	Batch Size	16	16	16	16	16	16	16	16	16	16	16	16	16	16
	Model Dims	128,256	256	128	128	64	1024	256	32	512	512	512	—	—	512
	Sequence Length	7	10	10	10	10	10	15	7	7	7	7	10	10	10
	Layers	—	1	2	1	2	1	1	2,1	2,1	2,1	4,3	—	—	3
ILI 6 WEEKS	Learning Rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.002	0.002	0.0001
	Batch Size	32	32	32	32	32	32	32	32	32	32	32	32	32	32
	Model Dims	64	512	512	512	512	1024	1024	512	512	512	1024	—	—	1024
	Sequence Length	10	48	15	120	60	36	48	10	24	120	96	60	215	60
	Layers	—	1	1	1	1	3	3	2,1	2,1	2,1	2,1	—	—	4
ILI 12 WEEKS	Learning Rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.002	0.002	0.0001
	Batch Size	32	32	32	32	32	32	32	32	32	32	32	32	32	32
	Model Dims	64	512	512	512	512	1024	1024	512	1024	512	512	—	—	1024
	Sequence Length	15	48	15	100	120	215	220	215	220	144	60	220	215	60
	Layers	—	1	1	1	1	2	3	2,1	2,1	2,1	2,1	—	—	2
ILI 24 WEEKS	Learning Rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.002	0.002	0.0001
	Batch Size	32	32	32	32	32	32	32	32	32	32	32	32	32	32
	Model Dims	64	512	512	1024	512	512	1024	512	128	512	512	—	—	1024
	Sequence Length	40	60	48	120	120	48	40	48	96	215	60	215	200	60
	Layers	—	1	1	1	1	1	1	2,1	4,3	2,1	2,1	—	—	4
ILI 36 WEEKS	Learning Rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.002	0.002	0.0001
	Batch Size	32	32	32	32	32	32	32	32	32	32	32	32	32	32
	Model Dims	64	512	512	1024	512	1024	1024	512	512	512	512	—	—	1024
	Sequence Length	50	36	40	100	100	40	96	96	60	40	40	215	220	40
	Layers	—	1	1	1	1	2	2	2,1	5,4	2,1	2,1	—	—	4
ILI 48 WEEKS	Learning Rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.002	0.002	0.0001
	Batch Size	32	32	32	32	32	32	32	32	32	32	32	32	32	32
	Model Dims	64	256	512	512	512	1024	1024	512	1024	512	512	—	—	1024
	Sequence Length	60	60	48	120	100	120	72	72	60	60	120	192	220	60
	Layers	—	1	1	1	1	1	3	2,1	2,1	2,1	2,1	—	—	5
ILI 60 WEEKS	Learning Rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.002	0.002	0.0002
	Batch Size	32	32	32	32	32	32	32	32	32	32	32	32	32	32
	Model Dims	64	512	512	1024	512	1024	1024	512	512	1024	128	—	—	2048
	Sequence Length	60	72	60	100	100	144	192	72	96	60	60	192	220	60
	Layers	—	1	1	1	1	2	1	2,1	4,3	2,1	2,1	—	—	4

2.4 Evaluation of ML models for Infectious Disease forecasting

2.4.1 ILI Hyperparameter Tuning

This section focuses on weeks 24, 36, 48, and 60 results. While Zeng et al., 2023 and Nie, 2022 used FEDformer-f (Fourier Transform) for ILI, T. Zhou et al., 2022 shows that the Wavelet Transform performs better for MTS, specifically for ILI. Our experiments confirm this, with an average MAE of 0.975 using the Wavelet Transform, compared to 1.012 with the Fourier Transform. These results outperform the 0.995 and 1.143 MAEs reported for FEDformer-w and FEDformer-f, respectively, in T. Zhou et al., 2022.

In our evaluation of models on the ILI dataset, we began by replicating the results reported in Zeng et al., 2023 and Nie, 2022. The hyperparameters used in these experiments were not optimal, particularly for all Transformer-based models. Through a grid search optimization of hyperparameters, as referenced in Table 2.1, we observed up to 12% reduction in MAEs for these models. Figure 2.1a illustrates the predictions for 24 weeks, with an MAE of 0.931, produced by FEDformer-w with optimized hyperparameters. In contrast, Fig. 2.1b displays predictions using the default hyperparameters with MAE 0.963 mentioned in

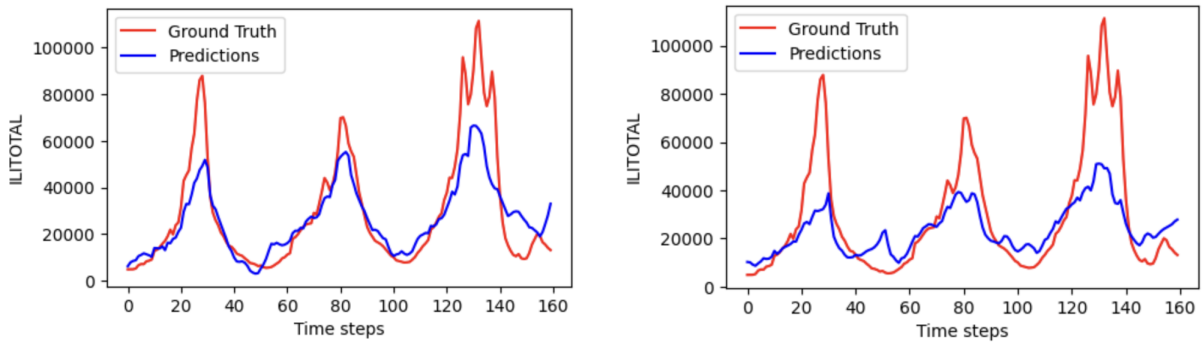
Table 2.2: This table shows the hyperparameter details for the models for RSV dataset.

Datasets	Hyperparameters	FFNN	LSTM	Seq2Seq-LSTM-Att	PatchTST
RSV 6 WEEKS	Learning Rate	0.0004	0.0004	0.0004	0.0004
	Batch Size	16	8	8	16
	Model Dims	512	256	512	256
	Sequence Length	104	48	96	96
	Layers	—	2	2	2
RSV 12 WEEKS	Learning Rate	0.0004	0.0004	0.0004	0.0004
	Batch Size	16	8	8	16
	Model Dims	256	256	256	256
	Sequence Length	48	48	104	96
	Layers	—	2	1	2
RSV 24 WEEKS	Learning Rate	0.0004	0.0004	0.0004	0.0004
	Batch Size	8	8	8	16
	Model Dims	256	256	512	256
	Sequence Length	48	104	96	96
	Layers	—	2	2	2
RSV 36 WEEKS	Learning Rate	0.0004	0.0004	0.0004	0.0004
	Batch Size	8	8	2	2
	Model Dims	256	512	512	256
	Sequence Length	48	48	104	96
	Layers	—	2	2	2
RSV 40 WEEKS	Learning Rate	0.0004	0.0004	0.0004	0.0004
	Batch Size	8	8	8	16
	Model Dims	256	512	512	256
	Sequence Length	48	48	96	96
	Layers	—	2	3	2

T. Zhou et al., 2022. The model with optimized hyperparameters more accurately captures the peaks and troughs in the dataset, demonstrating a significant enhancement in prediction accuracy over the default parameter settings. We optimized the DLinear model as well, one of the models proposed in Zeng et al., 2023, but the improvement was minimal. We assert that an L of exactly 96 weeks, as stated in Zeng et al., 2023, is unsuitable for all forecasting horizons when using Transformers. Our findings indicate that tuning L can yield better results as we achieve significantly improved performances with different L values. We achieved an average MAE of 0.783 using PatchTST, one of the best-performing models, with L of 60 weeks. This result is noteworthy as it surpasses the performance of both PatchTST/64 and PatchTST/42,

Table 2.3: SARIMA hyperparameters used for each dataset where p, d, q are non-seasonal and P, D, Q are seasonal parameters. M indicates the periodicity.

Datasets	$(p,d,q)X(P,D,Q)M$
CDC 4 wks	$(4,1,0)X(1,0,0)10$
COVID-19 6 wks (w/o RT)	$(3,1,2)X(3,0,1)10$
COVID-19 6 wks (w/ RT)	$(1,1,0)X(5,0,3)10$
ILI all wks	$(3,1,3)X(3,0,1)13$
RSV all wks	$(3,1,5)X(2,0,2)13$



(a) Using the optimized hyperparameters with MAE 0.931 (b) Using the default parameter settings with MAE 0.963

Figure 2.1: FEDformer predictions for 24 weeks compared to the ground truth using different hyperparameter settings

as reported by Nie, 2022. Specifically, PatchTST/64 employs an L of 512 weeks, while PatchTST/42 utilizes 336 weeks. This significant improvement highlights the critical role of hyperparameter tuning in optimizing model performance and underscores the efficiency of our approach in comparison to existing benchmarks.

2.4.2 Results and Discussion

Table 2.5 presents the performance of sixteen models on the ILI dataset. For comparison with existing studies, particularly for transformers and DLinear, MAEs are computed on the normalized data for all ILI variables in the MTS. In contrast, sMAPEs are calculated on the original data for the target variable only. The models' performance increases in the order of their introduction, with the exceptions of SARIMA, FFNN, and Informer. To mitigate cumulative error propagation, Informer H. Zhou et al., 2021 employs DMS for forecasting. Implementing DMS forecasting in the original Transformer led to better performance than the Informer.

The results of Zeng et al., 2023 show that NLinear outperforms DLinear for the ILI dataset, and our research also confirms this finding. NLinear addresses the distribution shifts by normalizing the input sequence. This normalization technique helps mitigate the effects of distribution shifts by making the model focus on the relative changes in the data rather than the absolute values, which may be affected by the shift. NLinear, by normalizing the input sequence, is better able to handle these shifts. Additionally, our results indicate that a simple FFNN with three linear layers outperforms most models in terms of MAE.

SARIMA also shows substantial improvement due to the combination of log transformation and differencing. The ILI dataset exhibits yearly seasonality, which SARIMA effectively captures through its seasonality parameter. Figure 2.2 illustrates the autocorrelation between the target variable and its lagged version, revealing a clear cyclical pattern in the lags every multiple of 52 weeks, indicating a clear yearly pattern. The best performance of SARIMA is due to log transformations and differencing as shown in the table 2.4.

PatchTST excels among all the models due to its innovative patching approach and channel-independent structure, which processes each channel separately, preserving unique temporal patterns for each feature. The patch embeddings in PatchTST encapsulate local semantic information, allowing the model to consider more extensive historical data compared to focusing on a single time point as an input token, leading to its superior performance.

Table 2.6 presents the performance of the COVID-19 dataset. The results for this early-stage pandemic data differ significantly from those observed for the ILI dataset. Transformers leverage their ability to model long-term dependencies and capture intricate patterns in large datasets, which unfortunately makes them perform poorly in smaller datasets. Though we implemented early stopping, dropout, and were selective with the input features, these models overfit on the limited data. For instance, Informer achieved a training loss five times lower than its testing loss, and the epoch with the lowest training loss did not align with the epoch of the lowest testing loss.

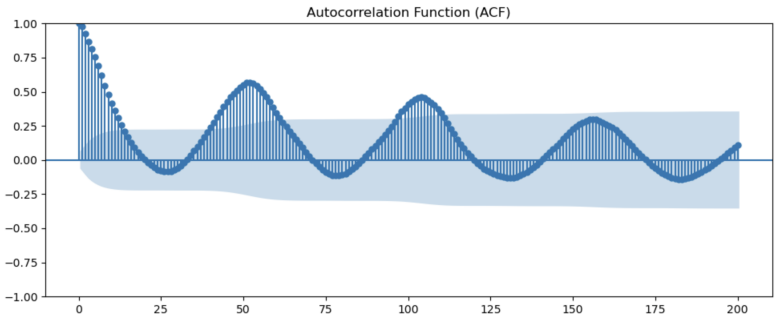


Figure 2.2: AutoCorrelation between ILI target variable and its lagged version

Table 2.4: Comparison of SARIMA model performance across forecasting horizons using sMAPE and MAE metrics on original scale for target variable, with and without logarithmic transformation and differencing.

Horizon	w/ Log + Diff	w/o Log	w/o Both
sMAPE			
6	19.22	24.65	26.89
12	28.22	38.17	43.93
24	34.74	49.48	62.80
36	36.51	51.84	69.26
48	37.57	52.73	73.34
60	39.28	53.47	81.02
Average	32.59	45.06	59.54
MAE			
6	5974.61	7559.60	7994.53
12	8675.31	11385.01	12587.22
24	11481.12	14540.75	17367.82
36	12323.43	15084.91	18306.26
48	12825.56	15280.18	18460.80
60	13426.87	15595.57	19263.84
Average	10784.48	13241.00	15663.41

DLinear, NLinear, and FFNN perform much worse than earlier recursive models when evaluated on COVID-19. An ablation study Zeng et al., 2023 indicates that the performance of simple linear models diminishes significantly as model dimensions and the size of L decrease. In our evaluation, we used $L = 10$ in contrast to the recommended $L = 336$. Although we tested DLinear and NLinear with higher L values, such as 30, $L = 10$ yielded the lowest test loss. Longer L reduces the number of training samples, which may

Table 2.5: sMAPE and MAE results for models using the ILI weekly dataset. The best results are in bold, and the second best are underlined.

	Random Walk		SARIMA		FFNN		LSTM		GRU		Seq2Seq-LSTM		Seq2Seq-GRU		Seq2Seq-LSTM-Att	
Week	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE
6	33.65	0.727	19.22	0.530	30.59	0.958	28.29	0.947	28.90	0.775	27.80	0.897	27.81	0.745	28.95	0.913
12	53.29	1.146	28.22	0.825	41.66	0.979	39.03	1.099	39.56	1.026	35.16	1.157	34.47	0.991	37.64	1.183
24	75.37	1.700	34.74	1.213	55.98	1.097	51.04	1.293	53.12	1.245	46.40	1.363	42.03	1.210	43.92	1.293
36	86.09	1.883	36.51	1.362	56.05	1.245	64.49	1.460	59.34	1.349	57.12	1.506	53.74	1.319	52.81	1.321
48	81.39	1.798	37.57	1.382	58.74	1.206	65.66	1.530	60.33	1.420	55.99	1.516	55.68	1.449	60.56	1.569
60	74.45	1.677	39.28	1.376	60.01	1.252	62.04	1.521	57.40	1.431	59.89	1.580	55.33	1.430	54.69	1.503
Avg	67.37	1.488	32.59	1.115	50.50	1.122	51.75	1.308	49.77	1.207	47.06	1.336	44.84	1.200	46.42	1.297
	Seq2Seq-GRU-Att		Transformer		Informer		Autoformer		FEDformer		DLinear		NLinear		PatchTST	
Week	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE
6	24.69	0.835	31.64	0.923	33.95	0.981	34.17	0.806	36.51	0.859	33.57	0.811	28.41	0.592	21.99	0.494
12	35.85	1.039	35.78	1.080	42.57	1.201	46.74	1.055	36.28	0.784	36.06	0.916	32.57	0.744	27.99	0.613
24	39.44	1.090	50.02	1.258	59.76	1.287	44.88	1.094	45.25	0.931	36.65	1.022	37.16	0.901	31.77	0.761
36	54.02	1.300	52.78	1.312	61.10	1.379	46.77	1.005	38.60	0.911	35.33	1.024	34.79	0.902	29.67	0.731
48	59.35	1.482	48.30	1.308	62.49	1.540	45.65	1.047	41.03	1.031	36.24	1.068	35.50	0.925	31.20	0.787
60	53.23	1.359	46.52	1.333	61.52	1.495	44.98	1.049	42.24	1.028	37.87	1.097	37.80	0.978	31.82	0.856
Avg	44.43	1.184	44.17	1.202	53.56	1.313	43.86	1.009	39.98	0.924	35.95	0.989	34.00	0.836	**29.07**	**0.707**

increase loss despite potentially lowering it for larger datasets. Autoformer, FEDformer, and PatchTST underperformed compared to Transformers and Informer. Nie, 2022 attributes the performance improvement of PatchTST to its ability to handle longer Ls and not the superiority of patch embedding. It would be surprising if PatchTST outperformed previous models without increasing L. FEDformer and Autoformer's main goal was to facilitate longer Ls as well. Autoformer, for instance, uses AutoCorrelation blocks to identify seasonality. AutoCorrelation blocks identify the periods of repeating patterns in the input sequence. From inspection, we can see that strong periodicity does not exist within the 7 data points in the COVID-19 data. The assumptions Autoformer was built seem false in the COVID-19 dataset.

Seq2Seq models, especially with attention mechanisms, perform better here because they can focus on relevant parts of the sequence and adapt to short-term trends and abrupt changes typical of early-stage pandemic data. This dataset likely has less clear seasonal patterns and more sudden shifts, which these models handle effectively.

Additionally, applying an RT mechanism improves the performance of nearly all models as shown in Table 2.6, though they follow a similar performance pattern as those without RT. RT enhances performance by enabling the model to adapt to the latest data trends. This approach ensures the model continuously learns from new information, captures emerging patterns, and adjusts predictions based on recent data, leading to improved forecast accuracy. This underscores the importance of RT, especially in scenarios with limited data, such as real-world pandemics.

Table 2.7 presents the RSV results across five models, showing performance improvements similar to those observed for the ILI dataset. Both RSV and ILI exhibit strong seasonality and long temporal ranges, from which models such as SARIMA and PatchTST particularly benefit. SARIMA outperforms PatchTST, particularly due to log transformations.

Table 2.6: sMAPE and MAE results for models using the COVID-19 weekly dataset with and without retraining (RT). The best results are in bold, and the second best are underlined.

		Random Walk		SARIMA		FFNN		LSTM		GRU		Seq2Seq-LSTM		Seq2Seq-GRU		Seq2Seq-LSTM-Att	
	Week	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE
w/o RT	1	17.19	0.305	10.36	0.484	28.77	0.658	11.61	0.191	12.48	0.231	20.15	0.399	19.75	0.347	14.33	0.241
	2	29.19	0.526	16.65	0.449	24.93	0.486	11.29	0.217	9.83	0.192	15.61	0.313	14.53	0.252	15.60	0.265
	3	41.72	0.742	19.72	0.445	38.08	0.696	15.80	0.300	14.55	0.232	16.14	0.332	15.05	0.268	15.86	0.248
	4	54.14	0.936	25.77	0.513	45.23	0.919	29.04	0.497	36.17	0.492	22.52	0.465	21.92	0.416	19.26	0.302
	5	63.47	1.092	32.35	0.591	39.29	0.821	40.35	0.673	28.57	0.493	30.48	0.630	31.75	0.608	22.48	0.388
	6	73.64	1.246	36.60	0.668	51.99	0.910	53.78	0.841	43.24	0.751	42.87	0.836	45.65	0.838	29.70	0.524
	Avg	46.56	0.808	23.57	0.525	38.05	0.748	26.98	0.453	24.14	0.399	24.63	0.496	24.77	0.455	19.54	**0.328**
w/ RT	1	17.19	0.305	15.85	0.433	12.67	0.280	22.39	0.339	18.84	0.332	24.30	0.436	20.97	0.313	11.78	0.191
	2	29.19	0.526	19.56	0.517	14.98	0.262	17.36	0.226	14.91	0.270	13.67	0.261	9.43	0.174	11.69	0.214
	3	41.72	0.742	25.83	0.635	17.29	0.353	13.81	0.217	17.40	0.270	13.62	0.220	9.94	0.201	14.30	0.260
	4	54.14	0.936	30.40	0.702	25.92	0.424	19.90	0.331	17.39	0.302	16.32	0.243	17.70	0.321	21.81	0.371
	5	63.47	1.092	35.13	0.760	38.43	0.633	32.37	0.517	28.59	0.538	23.83	0.390	23.93	0.447	26.06	0.464
	6	73.64	1.246	40.52	0.938	52.86	0.868	48.43	0.741	41.62	0.707	37.59	0.609	33.96	0.611	35.44	0.615
	Avg	46.56	0.808	27.88	0.664	27.02	0.470	25.71	0.395	23.13	0.403	21.56	0.360	19.32	0.345	20.18	0.353
		Seq2Seq-GRU-Att		Transformer		Informer		Autoformer		FEDformer		DLinear		NLinear		PatchTST	
	Week	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE
w/o RT	1	15.70	0.283	26.65	0.454	14.76	0.222	28.89	0.520	23.24	0.379	15.76	0.293	13.01	0.268	15.82	0.285
	2	12.07	0.184	23.56	0.379	24.24	0.348	14.49	0.250	20.13	0.311	21.42	0.443	18.88	0.374	18.98	0.398
	3	14.32	0.222	19.05	0.342	13.17	0.249	13.83	0.236	18.52	0.242	27.79	0.541	29.51	0.559	25.68	0.538
	4	18.98	0.329	21.53	0.393	25.59	0.442	21.55	0.378	22.99	0.346	36.06	0.652	39.69	0.725	36.95	0.716
	5	23.01	0.422	28.24	0.499	36.20	0.629	37.37	0.570	30.03	0.496	40.80	0.720	49.61	0.869	41.66	0.849
	6	28.89	0.537	47.67	0.718	35.84	0.623	64.78	0.999	66.37	1.000	48.05	0.805	58.55	1.012	55.50	1.017
	Avg	**18.83**	0.330	27.78	0.464	24.97	0.419	30.21	0.492	30.21	0.462	31.65	0.576	34.87	0.635	32.43	0.634
w/ RT	1	11.27	0.203	23.24	0.407	17.33	0.326	18.88	0.377	19.67	0.395	11.66	0.254	12.79	0.247	12.70	0.243
	2	11.00	0.194	15.73	0.270	17.80	0.313	24.95	0.350	22.99	0.385	18.77	0.413	21.86	0.425	18.91	0.414
	3	14.04	0.223	10.50	0.170	16.13	0.268	19.63	0.278	25.07	0.396	28.69	0.529	30.18	0.581	21.80	0.512
	4	19.14	0.287	24.94	0.367	21.93	0.354	21.22	0.329	32.49	0.453	31.04	0.616	37.37	0.711	35.00	0.687
	5	20.35	0.329	36.53	0.565	31.91	0.508	29.37	0.443	40.23	0.589	36.61	0.686	49.88	0.982	41.91	0.838
	6	30.15	0.527	54.14	0.780	46.76	0.715	45.35	0.671	50.14	0.834	49.02	0.827	59.91	1.074	47.01	0.981
	Avg	**17.66**	**0.294**	27.53	0.426	25.31	0.414	26.57	0.408	31.77	0.509	29.30	0.554	35.17	0.670	29.56	0.612

2.5 CDC COVID-19 Death Forecasting Models

2.5.1 Inclusion Criteria

There was significant variability in the forecast accuracy of individual models when predicting periods of rising and descending deaths; the models often underpredict the increasing deaths and overpredict the decreasing ones, as reported by Cramer, Ray, et al., 2022. Thus, we aim to capture both peaks and troughs in the data to assess the accuracy of models, particularly at these critical points. To illustrate these variations, we marked the time series's most prominent peaks and troughs using red and green points, respectively, in Fig. 2.3. Remarkably, the highest peaks occurred in January 2021 and February 2022. In contrast, the number of deaths was the lowest in July 2021 and exhibited a declining trend from April 2022, reaching its lowest in December 2022. To streamline our analysis, we divide the models into two categories. Category 1 has the models that capture the peaks and troughs, and the rest are assigned Category 2. The CDC model with the lowest sMAPE is considered the best-performing one. This raises the question: *How are the sMAPEs for the CDC models computed?* The CDC dataset available on GitHub[9] contains

[9] https://github.com/scalation/data/blob/master/CDC-COVID-Data/concatenated_CDC_20_21_22_23.csv

Table 2.7: Performance of forecasting models across different horizons (Weeks) for the RSV dataset. The best results are in bold, and the second best are underlined.

Weeks	SARIMA	FFNN	LSTM	Seq2Seq-LSTM-Att	PatchTST
sMAPE – Target Variable					
6	21.66	53.43	45.79	42.93	40.92
12	26.87	51.24	46.00	53.43	32.36
24	31.39	51.94	49.78	45.78	42.61
36	34.40	53.04	51.95	45.49	44.07
48	37.24	67.96	64.39	60.12	54.25
Average	**30.31**	55.52	51.58	49.55	<u>42.84</u>
Normalized MAE – All Variables					
6	0.137	0.238	0.405	0.459	0.186
12	0.171	0.294	0.433	0.401	0.195
24	0.194	0.318	0.451	0.502	0.214
36	0.206	0.330	0.452	0.493	0.215
48	0.216	0.384	0.509	0.567	0.229
Average	**0.185**	0.313	0.450	0.484	<u>0.207</u>

4-week-ahead forecasts of COVID-19 deaths for multiple models. We calculate the sMAPE by comparing each model's forecasts with the corresponding "observed" values provided in the same file, and evaluate the models based on these computed sMAPE values.

We adjust all models' starting and ending dates to align with the best-performing CDC model to avoid unavailable forecasts. To ensure a fair comparison, any missing dates in the chosen model are also excluded from the other models, allowing for a more accurate assessment of their performance based on matching dates. The interval starts from 5th December 2020 to 30th April 2022 as shown in Fig. 2.3. We did not end the interval in December 2022, which was the month for the 2nd lowest, because the models did not have the data for it. Thirteen CDC models fall within this interval, while the remaining models are placed in Category 2. It is important to note that any CDC model with data for this interval but missing information for peaks is also classified under Category 2. For example, the JHU-IDD model spans from July 25, 2020, to April 3, 2023, but lacks data from October 2021 to January 2022, which covers the beginning of the second peak. The description of Category 1 models is provided in the GitHub repository[10].

[10] https://github.com/scalation/scalation_py/blob/dev_sr/PandemicForecasting/CDC_Experiments/Models_description.pdf

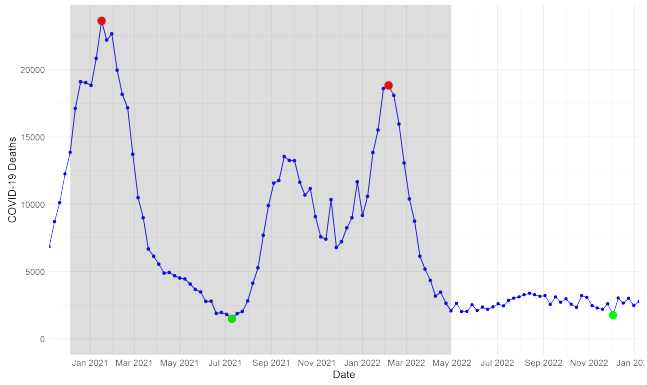


Figure 2.3: This figure illustrates the two peaks and troughs in the COVID-19 dataset. The highlighted forecast interval from December 2020 to April 2022 was initially set to choose Category 1 models.

2.5.2 Results and Discussion

Figure 2.4 shows the Category 1 models along with their sMAPEs. Among all the models in Category 1, the MIT-LCP model Sundar et al., 2021 was the most accurate, showing the lowest averaged sMAPE at 14.36 over 4 weeks. This model uses a gradient-boosted regressor Chen and Guestrin, 2016. The data is processed and features are collected from several platforms, including JHU 3-week lagged cases and deaths, COVID-19 test positivity rate, county-level socioeconomic dataset, mobility, and hospitalization data. The importance of these features is determined through Shapley Additive explanations values, and the model is trained. The model projected incident and cumulative deaths 1-4 weeks ahead at the county level across all US counties. These predictions were subsequently aggregated at the state and national levels. The forecast data was formatted for submission to the CDC Forecast Hub.

Unlike previous studies that mainly compared CDC models to CDC baselines for their evaluation, our research takes a new approach by comparing the SOTA models to compete directly with the best-performing CDC model, which is continuously needed by policymakers striving for enhanced decision-making and optimized resource allocation. To ensure a fair comparison, we align the forecasting start date and end date with those of the MIT-LCP model. Although MIT-LCP's 4-week ahead forecasts were available from September 12, 2020, with missing November 2020 data, we chose a later start date of December 5, 2020, up to the latest available forecast on June 4, 2022, making a total of 77 samples, which makes the sMAPE score 14.07 for MIT-LCP. Certain dates were identified as missing in the MIT-LCP model, namely ["2021-01-23", "2021-08-14", "2021-09-25", "2021-10-02", "2021-11-13", "2021-12-04", "2022-01-15", "2022-05-07"]. These specific dates are intentionally excluded during the evaluation of ML models to ensure an unbiased comparison.

We opted for the RT approach due to the improved results for COVID-19 predictions compared to the standard train-test split method, as shown in the tables. We take the past data from February 29,

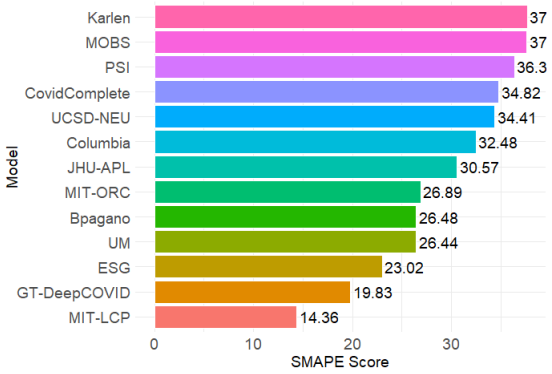


Figure 2.4: Category 1 CDC models and their sMAPE scores are presented, where lower sMAPE indicates better performance. These models are evaluated using the CDC's observed deaths data as the ground truth.

2020, to November 28, 2020, to train the models and forecast 4 weeks from December 5, 2020, to June 4, 2022. Table A.2 provides a comprehensive overview of the sMAPE and MAE results, illustrating the performance of the MIT-LCP model in comparison to other models. The ML models fairly compete against the CDC's best-performing model and outperform the model overall. This includes both the GRU and LSTM Seq2Seq model using Attention and Transformer with averaged sMAPEs of 12.94, 14.03, and 14.03, respectively, beating MIT-LCP at 14.07. As shown in the table, our models beat the MIT-LCP in the case of both averaged sMAPEs and MAEs. The use of attention in these models tends to improve the performance, but the performance degrades as the model complexity increases. Additionally, the incorporation of DMS in the Transformer model enhances its predictive capabilities. The MIT-LCP model prioritizes 2 weeks and 3 weeks prior to deaths and cases as key features. In contrast, our models leverage 6 weeks and 8 weeks of lagged deaths, hospitalized patients, patients admitted to the ICU, and the reproduction rate of COVID-19 as more influential predictors, contributing to their superior performance. These variables and hyperparameters, shown in Table 2.1, were tuned using grid search. The code to reproduce the results is provided in GitHub repository[11].

2.6 Conclusion

This paper evaluates sixteen time series models for pandemic forecasting using three U.S. weekly datasets: ILI cases, RSV detections, and COVID-19 deaths. The study compares models ranging from simple to advanced architectures to determine which perform best at different stages of a pandemic. The results show that PatchTST and SARIMA are well-suited for larger datasets and long-term forecasting, while

[11] https://github.com/scalation/scalation_py/tree/dev_sr/PandemicForecasting/CDC_Experiments

Table 2.8: sMAPE and MAE results for weekly COVID-19 deaths forecasts, comparing the CDC's MIT-LCP and our models. The models surpassing MIT-LCP are in bold. MAEs are on the original data to match MIT-LCP's evaluation. Both sMAPEs and MAEs for all models are calculated on the original data for COVID-19 deaths.

	MIT-LCP		RandomWalk		SARIMA		FFNN		LSTM		GRU		Seq2Seq-LSTM		Seq2Seq-GRU		Seq2Seq-LSTM-Att	
Week	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE
1	14.40	1278.69	14.55	924.09	12.90	1173.92	15.30	809.39	14.41	806.19	14.22	873.20	14.64	773.53	13.13	667.29	11.36	657.91
2	13.87	1276.78	21.39	1502.91	18.40	1843.05	17.12	866.07	14.83	800.70	15.08	885.56	14.67	802.99	12.98	682.78	11.43	647.87
3	12.33	1131.60	28.71	2081.03	23.90	2294.97	18.33	914.44	17.55	939.99	16.71	1034.64	15.61	905.39	13.45	762.45	12.46	729.29
4	15.66	1430.47	35.40	2579.023	34.15	3117.25	24.36	1136.57	21.57	1156.56	20.73	1277.85	19.68	1210.17	17.65	1119.59	16.50	954.23
Avg	14.07	1279.39	25.01	1771.768	22.34	2107.30	18.78	931.62	17.09	925.86	16.68	1017.81	16.15	923.02	14.30	808.03	**12.94**	**747.32**
	Seq2Seq-GRU-Att		Transformer		Informer		Autoformer		FEDformer		DLinear		NLinear		PatchTST		-	
Week	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	sMAPE	MAE	-	-
1	13.01	710.80	11.59	623.00	14.04	806.33	25.59	1136.47	19.20	1030.16	19.06	1102.14	13.49	826.73	16.18	987.23	-	-
2	12.55	685.59	12.34	657.46	14.32	822.04	24.41	1117.03	20.90	1126.54	28.96	1699.45	20.45	1291.97	20.79	1444.64	-	-
3	13.59	773.01	14.80	801.03	16.02	939.50	30.43	1385.30	24.79	1324.06	39.06	2242.33	24.11	1741.61	26.07	1881.62	-	-
4	16.97	1082.41	17.41	1055.51	17.56	908.73	33.28	1538.71	29.98	1421.86	47.70	2862.26	34.66	2507.57	34.65	2340.73	-	-
Avg	**14.03**	**812.95**	**14.03**	**784.25**	15.48	869.15	28.43	1294.38	23.72	1225.66	33.69	1976.55	23.18	1591.97	24.42	1663.55	-	-

Seq2Seq models with attention perform best on shorter datasets and short-term forecasts. We also assess the accuracy of CDC models for COVID-19 death predictions and emphasize the importance of hyperparameter optimization. The study highlights that retraining data is crucial for improving model accuracy, particularly with limited data.

Chapter 3

Transformer Architecture

In this chapter, we describe the internal working mechanism of transformer models, which form the foundation of the deep learning architectures used in our study. Transformers operate entirely through attention mechanisms rather than recurrence or convolution, allowing them to capture long-range dependencies in sequential data efficiently. We focus on explaining each of the core components: self-attention, multi-head attention, positional encoding, feed-forward networks, normalization, and residual connections. The goal of this chapter is to provide a clear understanding of how information flows through a transformer. Specifically, we explain the **Encoder-only** framework, as this architecture underpins most SOTA transformer models for time-series forecasting. In forecasting, the number of future horizons is known, and we typically prefer direct multi-step (DMS) prediction, producing all future values simultaneously to prevent error accumulation that occurs in autoregressive prediction. For this purpose, an encoder plus a prediction head is sufficient, while a decoder, which in NLP is used for token-by-token generation with cross attention, is unnecessary. Encoder-only transformers build contextual representations by stacking self-attention + position-wise feed-forward blocks with residual connections and normalization. They process the full input window in parallel and produce task-specific outputs via a lightweight prediction head. We adopt this family for direct multi-step time-series forecasting. The transformer architecture is shown in Figure 3.1.

3.1 Input and Output Notation

In a multivariate time series, let the historical sequence be denoted as

$$X = \{x_1, x_2, \ldots, x_T\} \in \mathbb{R}^{T \times N},$$

where T represents the number of time steps and N denotes the number of features or variables. The forecasting objective is to predict the next S future time steps, expressed as

$$Y = \{y_{T+1}, y_{T+2}, \ldots, y_{T+S}\} \in \mathbb{R}^{S \times N}.$$

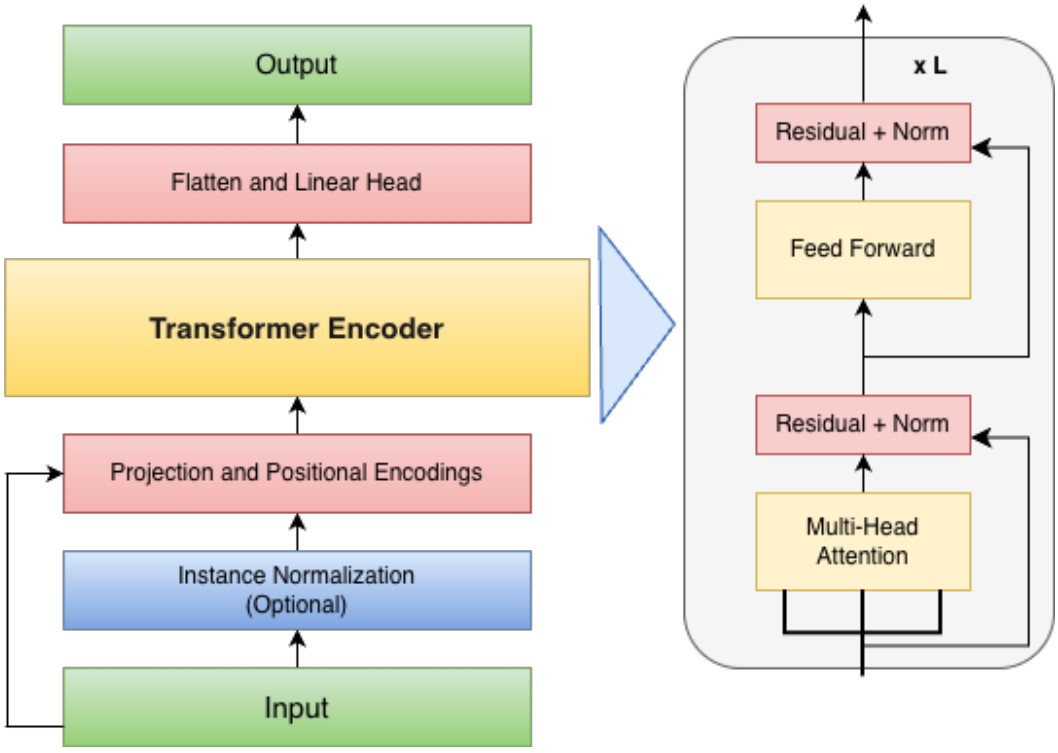


Figure 3.1: The Transformer architecture.

3.1.1 Optional: Instance Normalization

Instance normalization is optionally applied in a rolling-window setting to stabilize the statistical properties of each input sequence. Its primary goal is to mitigate distributional shifts between the training and testing phases and to reduce non-stationarity by ensuring that each instance is normalized based on its own statistics. By dynamically adjusting the mean and variance of each input instance, RevIN aligns the train and test data distributions, making it easier for the model to generalize to unseen data. Figure 3.2 shows how applying RevIN mitigates the distributional shift between the train and test data. In this technique, the input is first centered by subtracting the mean and then scaled by the standard deviation, thereby measuring how much each observation deviates from the instance mean. This technique normalizes each x_t, where t denotes the time step, and the same mean and standard deviation are later added back to the model's output during the de-normalization stage to get the predictions on the original scale.

3.2 Linear Projection / Input Embedding

Transformers employ learned embeddings, implemented as a trainable linear projection, to map each time step's input vector into an embedding of dimension d_{model}, for example 512.

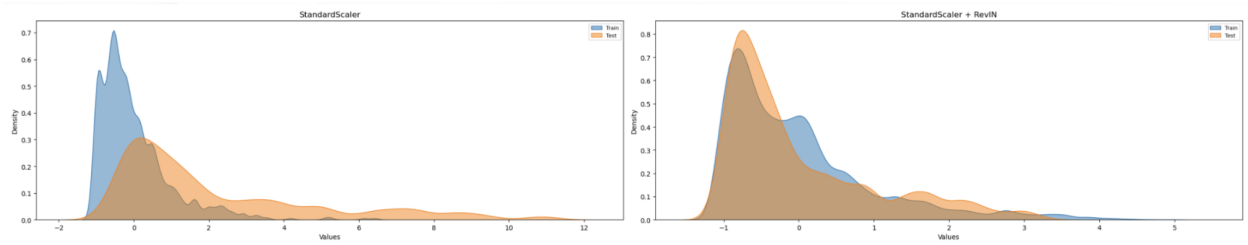


Figure 3.2: Comparison of distributional shift between the ILITOTAL train and test data using StandardScaler (left) and StandardScaler with RevIN (right).

1. Temporal embedding: This is what most Transformers like **PatchTST** do for time series: You start with data shaped like $[B, T, N]$, where B is the batch size, T is the number of time steps, and N is the number of features (variables). For example, if per iteration, the batch size is 32, and sequence length or time steps (or the look-back window) is 104, and number of features/variables is 7 (both COVID and ILI have 7 variables), then input shape $[B, T, N]$ becomes $[32, 104, 7]$. At a single time step t, your vector is

$$x_t \in \mathbb{R}^N.$$

You apply a linear layer

$$W \in \mathbb{R}^{d_{\text{model}} \times N}, \quad b \in \mathbb{R}^{d_{\text{model}}},$$
$$h_t = W x_t + b, \quad h_t \in \mathbb{R}^{d_{\text{model}}}.$$

Thus, all variables at that time step are mixed together and mapped to the Transformer model space. Shape-wise, only the last dimension changes:

$$(B, T, N) \xrightarrow{\text{linear}} (B, T, d_{\text{model}}).$$

So, now the input shape $[32, 104, 7]$ becomes $[32, 104, 512]$.

2. Variate embedding — inverted: These inverted embeddings are used in Transformer architectures such as **iTransformer**. In this setup, the model treats each variable as a token, allowing the attention mechanism to operate along the feature (variable) dimension instead of the temporal dimension. Original data is $[B, T, N]$, where N is the number of variables. The tensor is first permuted to $[B, N, T]$ so that each variable has its entire time series. A linear layer is then applied along the time dimension:

$$\text{for each variable } n: \quad x^{(n)} \in \mathbb{R}^T \longrightarrow h^{(n)} = W x^{(n)} + b,$$

where

$$W \in \mathbb{R}^{d_{\text{model}} \times T}, \quad b \in \mathbb{R}^{d_{\text{model}}}.$$

Hence, each variable is represented by a d_{model}-dimensional vector encoding all its time steps. Shape-wise:

$$(B, N, T) \xrightarrow{\text{linear over } T} (B, N, d_{\text{model}}).$$

3. Linear layer formulation A PyTorch `nn.Linear(in_dim, out_dim)` implements:

$$y = Wx + b,$$

where

$$x \in \mathbb{R}^{\text{in\_dim}}, \quad W \in \mathbb{R}^{\text{out\_dim} \times \text{in\_dim}}, \quad b \in \mathbb{R}^{\text{out\_dim}}, \quad y \in \mathbb{R}^{\text{out\_dim}}.$$

In batched form, PyTorch applies the linear layer to the last dimension, which is why both embedding strategies operate along different axes but are mathematically identical in form. All the other dimensions except the last dimension stay the same.

4. Motivation In lower-dimensional spaces, different patterns can overlap or appear too similar, making it difficult for the model to tell them apart. By projecting inputs into a higher-dimensional space, Transformers spread these patterns out, allowing attention to more easily recognize similarities and differences. This projection also ensures uniformity across all layers, since Transformers use residual connections of the form (output = x + Sublayer(x)), both terms must have the same dimensionality. Mapping all inputs to a common (d_{model}) keeps the internal architecture consistent. Additionally, a wider (d_{model}) is necessary for multi-head attention, as it splits this dimension among several heads; if the dimension were too small, each head would have too few features to learn distinct relationships effectively.

3.3 Positional Encoding

In Transformers, positional encoding is used to give the model a sense of order, since self-attention itself does not know which token or time step comes first. Each position in a sequence (denoted by t) is turned into a unique pattern of numbers across all embedding dimensions (denoted by i). These numbers are made using sine and cosine waves of different frequencies so the model can learn both short-term and long-term relationships. The formula for the encoding is Vaswani et al., 2017:

$$PE_{(t,2i)} = sin(t/10000^{2i/d_{model}}), PE_{(t,2i+1)} = cos(t/10000^{2i/d_{model}})$$

Here, smaller i values produce faster waves (high frequencies), while larger i values generate slower waves (low frequencies). Lower frequencies capture long-range dependencies, whereas higher frequencies focus on short-term or fine-grained changes. The wavelengths follow a geometric progression from roughly 2π to $10000 \times 2\pi$, giving the model coverage across both local and global time scales.

Sinusoids were chosen because they allow the model to learn relative positions easily: for any offset k, $\text{PE}(t + k)$ can be expressed as a linear function of $\text{PE}(t)$, thanks to trigonometric identities such as

$$\sin(t + k) = \sin t \cos k + \cos t \sin k, \qquad \cos(t + k) = \cos t \cos k - \sin t \sin k.$$

To integrate positional information with token embeddings, the positional encoding matrix is constructed with the same dimensionality as the embedding dimension d_{model}. The final input to the Transformer encoder is obtained through element-wise addition:

$$\tilde{H} = h + W_{\text{pos}}.$$

Here,

$$h \in \mathbb{R}^{T \times d_{\text{model}}} \quad \text{denotes the learned input embedding,}$$

$$W_{\text{pos}} \in \mathbb{R}^{T \times d_{\text{model}}} \quad \text{represents the positional encoding matrix,}$$

and

$$\tilde{H} \in \mathbb{R}^{T \times d_{\text{model}}} \quad \text{is the position-aware input passed to the encoder.}$$

In the original Transformer, W_{pos} is a *fixed*, non-trainable matrix constructed analytically using the sinusoidal formulas. Vaswani et al., 2017 has also experimented with *learnable* positional embeddings, but in both cases, the encoding must share the same dimension as the token embeddings to permit element-wise addition.

For standard time-based data embedding (used in **PatchTST** and **TimeXer**), the resulting representation has shape $T \times d_{\text{model}}$. In contrast, architectures such as **iTransformer** use variable-based embedding, producing a representation of shape $N \times d_{\text{model}}$.

3.3.1 Patch Tokenization (if Patch-based model)

In transformer architectures such as **PatchTST** and **TimeXer**, the input sequence is first divided into smaller segments, referred to as *patches*. Patches are useful because they enable the model to handle longer sequences more efficiently. Since the model views each patch as a single token instead of processing individual timesteps, the computational complexity and memory usage are reduced quadratically by a factor of stride. The input window is split into patches of length p with stride s (stride is not used in **TimeXer**, as it generates only non-overlapping patches). For a univariate series $x_{1:T}$, the resulting patch sequence is

$$X_p = \left[\, x_{1:p},\, x_{1+s:p+s},\, \dots \,\right], \quad \text{patch\_num} = \left\lfloor \frac{T-p}{s} \right\rfloor + 1.$$

Patches $X_p \in \mathbb{R}^{M \times p}$ are mapped to a latent space of dimension d_{model} through a trainable linear projection $W_p \in \mathbb{R}^{p \times d_{\text{model}}}$, where M is the number of patches, p is the patch length and d_{model} is the embedding dimension of the Transformer, so the entire patch input can be written as $X_p \in \mathbb{R}^{M \times d_{\text{model}}}$.

A learnable positional encoding $W_{\text{pos}} \in \mathbb{R}^{M \times d_{\text{model}}}$, is then added to preserve the temporal order of the M patches. The resulting embedded patch representation is

$$X_i = X_p + W_{\text{pos}},$$

where $X_i \in \mathbb{R}^{M \times d_{\text{model}}}$ is the input to the Transformer encoder at time step i. This projection step converts each local patch into a vector in the latent space while encoding its temporal position within the input window. For multivariate data, patch tokens are created for each channel, and a learned global token (used in **TimeXer**) may be appended to aggregate information across patches of the same channel. Patching enables the model to process longer sequences efficiently, often resulting in significant performance improvements.

3.4 Transformer Encoder

Transformers use self-attention Vaswani et al., 2017, which allows each position to form weighted connections to every other position in the same layer. This makes the path length between distant positions O(1) per layer and O(L) across L stacked layers, helping preserve long-range dependencies. In a Transformer encoder, all timesteps are processed in parallel within a layer; the only sequential dependency is the stacking of layers, so the sequential depth is O(L). In practice, full self-attention has $O(n^2)$ time and memory per layer. The Transformer encoder consists of two main sub-layers: a Multi-Head Attention (MHA) mechanism and a position-wise Feed-Forward Network (FFN) layer. A residual connection followed by layer normalization (or batch normalization in some implementations) is applied after each sub-layer to stabilize training. Self-attention allows every token to attend to all other tokens, which means the information can get completely mixed across the sequence. After several layers of this mixing, the model could lose track of what each token originally represented. However, because every Transformer layer adds its original input (x) back to the transformed output through a residual connection, each token always retains part of its original identity. As a result, even after many layers, the network "remembers" what each input token was before being blended with information from others. Secondly, residual connections help prevent the vanishing gradient problem by creating shortcut paths that allow gradients to flow more easily during backpropagation. In deep networks, activations like ReLU can block gradients, causing them to weaken or disappear as they pass through many layers. Residual connections address this by adding the original input back to the output of each layer, ensuring that part of the gradient always flows directly through. This uninterrupted flow helps maintain stable gradient propagation, allowing deeper models like Transformers to train effectively.

For each attention head $i \in \{1, 2, \ldots, H\}$ in the MHA module, the input X is projected into multiple smaller-dimensional subspaces using different weight matrices: the queries, keys, and values.

$$Q = XW_Q^{(i)}, \quad K = XW_K^{(i)}, \quad V = XW_V^{(i)},$$

where $W_Q^{(i)}, W_K^{(i)} \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $W_V^{(i)} \in \mathbb{R}^{d_{\text{model}} \times d_v}$ are the projection matrices for head i. Each head performs the scaled dot-product attention independently:

$$\text{head}_i = \text{Attention}(QW_Q^{(i)}, KW_K^{(i)}, VW_V^{(i)}),$$

If we use 8 heads then for each head, $d_k = d_v = d_{model}/8$, if d_{model} is 512 then $d_k = d_v = 64$, this reduces the complexity of multi-head attention to single-head attention. The attention output is computed as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V.$$

Here, the dot product $QK^\top$ measures the similarity between each query and all keys, scaled by $\sqrt{d_k}$ to maintain numerical stability. The softmax function converts these similarity scores into normalized attention weights, which are used to compute a weighted sum of the value vectors. This allows the model to assign higher importance to more relevant tokens in the sequence. The scaling parameter $\sqrt{d_k}$ is used because large values of d_k can make the magnitudes of dot product large, which can push the softmax function into regions where gradients are extremely small Vaswani et al., 2017.

This process is done H times, with different learned linear projections. Instead of applying a single attention operation over the full d_{model}-dimensional space, the Transformer uses multiple attention heads to capture diverse patterns from different representations. The input queries, keys, and values are each linearly projected H times using distinct learned projections into lower-dimensional spaces of size d_k, d_k, and d_v, respectively. The outputs from all heads are concatenated and projected back to the model dimension:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O,$$

where $W_O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ is the output projection matrix. Using multiple heads enables the model to attend to different types of dependencies and contextual information simultaneously, overcoming the limitations of a single-head mechanism.

In the case of *self-attention*, these three components are derived from the same input sequence, allowing each token to attend to every other token within the same context. In contrast, *cross-attention* uses a different source for the queries than for the keys and values. For example, in the **TimeXer** architecture, the queries originate from the global patch token, whereas the keys and values are derived from the variable-specific representations.

After the MHA module, a residual connection is applied by adding the input of the attention block to its output. The result is then normalized using Layer Normalization (or Batch Normalization in some variants) to stabilize training and maintain consistent feature scaling. Batch Normalization is applied across each mini-batch of data, whereas Layer Normalization operates independently on every individual sample. Layer Normalization performs an affine transformation using learnable scale and shift parameters. Its formulation is given by:

$$\text{LayerNorm}(x_i) = \gamma \cdot \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

$$\mu = \frac{1}{K}\sum_{j=1}^{K} x_j, \qquad \sigma^2 = \frac{1}{K}\sum_{j=1}^{K}(x_j - \mu)^2$$

where:

$$
\begin{aligned}
x_i &= \text{the } i\text{-th sample of the input,} \\
K = d_{\text{model}} &= \text{number of hidden features / model dimensions,} \\
\mu &= \text{mean of the features,} \\
\sigma^2 &= \text{variance of the features,} \\
\gamma,\ \beta &= \text{learnable scale and shift parameters,} \\
\epsilon &= \text{a small constant for numerical stability.}
\end{aligned}
$$

Following this, a position-wise FFN is applied, consisting of two linear layers with a nonlinear activation function (such as ReLU or GELU) in between. Modern Transformers use the GELU activation function instead of ReLU because ReLU outputs zero for all negative input values. The FFN in a Transformer layer refines the token representations produced by the attention mechanism. While the attention block captures relationships and dependencies between tokens, the FFN applies nonlinear transformations to each token individually. This allows the model to process and reinterpret the context-aware information gathered by attention, enhancing its ability to represent complex patterns before passing the output to the next layer. It is referred to as "position-wise" because the same fully connected network is applied independently to each position in the sequence. A common configuration sets the feed-forward dimension (d_{ff}) as $d_{ff} = 4 \times d_{\text{model}}$, following the original Transformer architecture. However, this ratio is not mandatory and is often treated as a tunable hyperparameter in modern time-series Transformer models.

Another residual connection is added after the FFN, and the output is normalized again. This sequence of operations: residual connection, normalization, and feed-forward transformation, forms a single encoder block. Stacking multiple such blocks enables the model to build progressively richer feature representations.

3.5 Flatten Head

Finally, a flattening operation followed by a linear layer maps the d_{model}-dimensional representations from the previous layer to the prediction space S. This is analogous to Convolutional Neural Networks, where the final linear layers convert high-dimensional feature maps into class scores or output values. This layer outputs the forecast sequence S for each variable N. The output is:

$$
Y = \{y_{T+1}, y_{T+2}, \ldots, y_{T+S}\} \in \mathbb{R}^{S \times N}.
$$

If instance normalization is applied, the outputs are subsequently denormalized before computing the loss between the predictions and the ground truth.

3.6 Channel-Independent vs. Channel-Dependent Architectures

Encoder-only models differ in *which axis is treated as the token axis* for self-attention and whether variables are allowed to interact within the attention mechanism.

3.6.1 Channel-Independent

These models do not explicitly model inter-channel dependencies. Each variable has its own attention map that allows it to learns its own temporal dependencies, allowing the model to adapt to different behaviors or seasonalities across series. In this architecture, each variable has its own sequence of time tokens (e.g., patches or time steps of a single channel), and self-attention is applied separately to each variable or channel. In practice, channels are folded into the batch, and the same encoder weights are shared across all variables. There are no cross-variable interactions in this setting. This is the default configuration in **PatchTST** and **TimeXer**, where each channel is treated independently in self-attention mechanism.

3.6.2 Channel-Dependent

Sometimes variables are highly correlated and can benefit from interacting with each other, especially in the case of viral diseases. Channel mixing allows the model to capture cross-variable relationships between different series, which is useful when variables influence each other (for example, hospitalizations affecting mortality rates). In this architecture, tokens from multiple variables attend to each other within the same attention map. Here, tokens correspond to variables, and each token represents its entire time window or set of patches, allowing attention to mix information across variables.

Chapter 4

Statistical Transformations and Normalization in Transformers

Rana, S. Submitted to European Conference on Computer Systems, 09/25/2025

Real-world time series often exhibit skewness, heteroscedasticity, and distributional shifts, challenges that the most commonly used default preprocessing, StandardScaler, in Transformers fails to address. These challenges can prevent accurate time series forecasting. In this work , we investigate statistical transformations, including logarithmic, square root, Box-Cox, Yeo-Johnson, and differencing, applied before StandardScaler, and demonstrate their effectiveness in stabilizing variance and reducing skewness. To mitigate instance-level distributional shifts, we introduce Context-aware Instance Normalization (CoIN), which adaptively balances mean and last value normalization based on inputs and forecast horizons. We further develop a practical Multivariate Box-Cox extension, building on the univariate implementation, in Python, that jointly estimates parameters from feature covariances to stabilize variance and more effectively preserve inter-feature dependencies for improved multivariate time series forecasting. Experiments show that our proposed methods consistently yield significant performance improvements across Transformer models on real-world infectious diseases datasets.

4.1 Introduction

Time series forecasting is fundamental in many application domains, including finance, climate, epidemiology, traffic, and bioinformatics Andersen et al., 2005; Dastjerdi et al., 2022; Mudelsee, 2019. The ability to accurately model temporal patterns has direct implications for operational efficiency and strategic decision-making across industries. Transformers, originally proposed for natural language processing Vaswani et al., 2017 and later extended to computer vision Dosovitskiy et al., 2020, have more recently demonstrated strong potential in time series forecasting Nie, 2022; H. Wu et al., 2021. By leveraging self-attention mechanisms, they are capable of capturing long-range dependencies and complex temporal behaviors while processing sequential data in parallel, giving them an advantage over traditional autoregressive and recurrent neural network models. However, their forecasting performance can be heavily influenced by the statistical properties of the input data, which are often non-ideal in real-world scenarios. Several notable variants tailored for temporal data, such as PatchTST Nie, 2022, represent some of the most effective adaptations of the Transformer framework to time series, yet important challenges remain unaddressed.

Real-world time series data present several challenges, including distributional shift, skewness, and heteroscedasticity. A distributional shift occurs when the training and testing data are drawn from different underlying distributions. For example, influenza datasets exhibit clear shifts, with virus dynamics changing abruptly in 2020 due to the COVID-19 outbreak (Figure 4.3, where the end of the green line marks the outbreak). Skewness further complicates modeling. When trained on highly skewed data (e.g., right-tailed distributions), models tend to overweight dominant patterns and yield less accurate predictions for minority cases. Heteroscedasticity introduces an additional challenge: training samples from high-variance regions produce larger residuals, and because common loss functions, such as MSE, square these residuals, high-variance regions contribute disproportionately to the total loss. As a result, gradient updates are dominated by these regions, causing the optimizer to prioritize fitting them while underweighting low-variance regions. This imbalance destabilizes training, leading to gradient oscillations and

reduced generalization Gelfand, 2015. Despite these challenges, most deep learning models rely on a simple normalization technique.

Current research typically applies the StandardScaler (zero mean, unit variance) as a preprocessing step. While this approach is effective for aligning variables with different scales, as it stabilizes training by keeping features in a similar range, it requires a Gaussian distribution and is sensitive to extreme values or potential outliers, as they influence each input when computing statistics (e.g., mean and standard deviation). Moreover, StandardScaler does not address the broader challenges of real-world data. It is only truly appropriate for stationary time series, where the mean and variance remain constant over time, a condition rarely satisfied in practice Ogasawara et al., 2010. This stands in contrast to statistical approaches, which employ the transformations that explicitly address these challenges. For instance, the logarithmic and Box–Cox Box and Cox, 1964 transformations are widely used to reduce skewness and promote normality, while the Yeo–Johnson Yeo and Johnson, 2000 transformation extends Box–Cox to handle zeros and negative values. The logarithmic transform is particularly effective for stabilizing variance in rapidly growing data, such as epidemic curves during their early exponential phase, where it linearizes exponential growth. These transformations are typically applied in statistical forecasting models, and numerous studies have demonstrated their effectiveness in improving forecast accuracy Hyndman, 2018; Salles et al., 2019.

Despite their proven value, such transformations are rarely integrated into modern deep learning frameworks. *In this work, we incorporate them into state-of-the-art Transformer models before standardization to directly tackle the aforementioned challenges*. However, a limitation of classical power transformations such as Box–Cox is that they are typically optimized feature by feature, ignoring correlations among variables. In multivariate time series, where related features often exhibit strong dependencies (e.g., hospitalizations serving as a leading indicator for deaths in COVID-19), this independent optimization can lead to suboptimal representations.

Previous work on multivariate Box–Cox transformations has received limited attention in the literature Andrews et al., 1971; Rahman and Pearson, 2009; Velilla, 1992. Velilla Velilla, 1992 laid the theoretical foundation by extending the Box–Cox framework to account for covariances across features, while most studies focused primarily on simulation-based analyses. However, this line of research has remained largely theoretical and confined to implementations in **R**, with no available implementation in Python or empirical evaluation on real-world datasets.

In this study, *we bridge this gap by developing the first complete* ***Python implementation*** *of the multivariate Box–Cox transformation, fully compatible with the design and interface of Scikit-learn's preprocessing framework* Pedregosa et al., 2011. Our implementation jointly estimates transformation parameters by leveraging the covariance structure among features, thereby stabilizing variance while explicitly modeling inter-feature dependencies. We further extend its application beyond simulations by conducting comprehensive experiments on real-world time series datasets and evaluating its effectiveness when integrated into Transformer-based forecasting models. This work thus transforms a previously theoretical concept into a practical and reproducible tool for modern machine learning workflows.

Recent research Kim et al., 2021; Z. Liu et al., 2023; Passalis et al., 2019 has addressed instance-level distributional shift by embedding normalization directly into deep learning architectures. The most

widely used approach in Transformers is Reversible Instance Normalization (RevIN), which normalizes and then restores input features dynamically to mitigate dataset shift during training and testing. However, RevIN has key limitations: (i) it uses the input series' statistics for both normalization and denormalization, ignoring discrepancies between input and forecast horizons; (ii) it assumes uniform statistics across all time points, overlooking that different steps within an input sequence may follow different distributions; and (iii) it applies a rigid mean- or last-value-based subtraction, neglecting that different horizons may require different normalization references (e.g., short-term forecasts depend more on recent changes, while long-term forecasts align with overall trends). This distinction is illustrated in Figure 4.1, where 24-week-ahead predictions on the Influenza dataset using PatchTST under both RevIN strategies show that mean-based normalization leads to higher errors at earlier horizons, whereas last-value-based normalization yields higher errors at later horizons. To address these limitations, *we introduce **Co**ntext-aware **I**nstance*

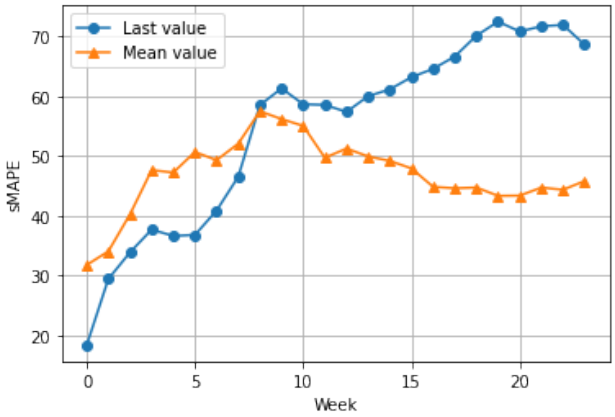


Figure 4.1: Per-horizon forecasting errors (sMAPEs) for a 24-week prediction length on the Influenza dataset using the PatchTST. Results compare RevIN with mean-based versus last-value-based normalization.

***N**ormalization (**CoIN**), which adapts normalization behavior based on the temporal context, where context refers to prediction horizon and the look-back window*. CoIN dynamically balances between last value and mean-based strategies, offering a more flexible and effective solution for mitigating distributional shifts across varying horizons and sequence lengths. The contributions of this work are summarized as follows:

1. A systematic investigation of classical statistical transformations, including Logarithmic, Square Root, Box–Cox, Yeo–Johnson, and Differencing, applied before standardization. These transformations are shown to effectively address skewness, stabilize variance, handle outliers, and mitigate distributional shifts that standard normalization alone cannot capture.

2. A multivariate extension of the univariate Box–Cox transformation is introduced, where transformation parameters are jointly estimated by leveraging feature covariances to preserve inter-feature dependencies. While earlier studies and implementations were limited to theoretical analyses or R-based environments, this work presents the first practical implementation in Python and provides a comprehensive empirical evaluation within deep learning–based time series forecasting.

3. A novel Context-aware Instance Normalization (CoIN) technique that adaptively balances mean and last value normalization based on inputs and forecast horizons. CoIN is a simple yet effective technique that can be adopted for any deep neural network.

4. We show that incorporating statistical transformations and improved normalization into Transformer-based models for time series forecasting yields consistent performance gains, achieving improvements of up to 39.4% on ILI, 16.9% on COVID-19 and 44% on RSV compared to standard preprocessing, and up to 24.6% over the existing instance normalization method.

4.2 Related Work

4.2.1 Normalization Techniques

Normalization is a crucial preprocessing step in time series forecasting, helping to stabilize variance and mitigate distributional shifts. The study by Salles et al., 2019 talks about different transformation methods, e.g., Logarithmic Transform, Box-Cox Transform, Differencing, etc., applied to the ARIMA model. The results highlight the importance of selecting the appropriate transformation method based on the dataset's characteristics. Experiments show that although the transformation methods consistently improved prediction and stationarity, no single method was universally best. Logarithmic transformations play an important role in the field of economic forecasting. The authors of the study Lütkepohl and Xu, 2012 demonstrate that these transformations stabilize variance, improving ARIMA-based forecasts. Similarly, Proietti and Luetkepohl, 2011 provides evidence that Box-Cox transformations enhance predictive performance in macroeconomic datasets. Nevertheless, these studies also warn against inappropriate usage, as transformations can introduce distortions when applied to already stable datasets. In addition, Douglas Curran-Everett Curran-Everett, 2018 highlights the importance of validating transformations using the Box-Cox method.

A study by Bandara et al., 2020 incorporates logarithmic transformations into neural networks for forecasting multiple time series datasets. Deep learning models often overlook such preprocessing steps, relying on simple normalization techniques such as StandardScaler or MinMaxScaler. The studies by de Amorim et al., 2023, Raju et al., 2020, and Ahsan et al., 2021 highlight the uses of these normalization techniques and display improved results in their respective domains.

As for deep learning, extensive research has focused on tackling the distribution shift or non-stationary time series problems on the input level. Adaptive Norm Ogasawara et al., 2010 uses a sliding-window technique from which the global statistics are calculated and used in the normalization part. DAIN Passalis et al., 2019 is a trainable, adaptive alternative to StandardScaler that learns to shift, scale, and filter each time series window based on its local distribution. Kim et al., 2021 introduced RevIN to tackle the problem of distribution shift in time series by temporarily removing instance-specific statistical properties such as mean and standard deviation and restoring them after forecasting. This normalization–denormalization process helps stabilize temporal distributions and improve forecasting accuracy across diverse real-world datasets. Recent SAN Z. Liu et al., 2023 gets the statistics from non-overlapped segments of the input

sequence for normalization and employs a statistics prediction module to estimate the distributions of future time steps. RevIN is widely used in Transformer-based models, and we adopt it as a baseline for our comparisons.

4.2.2 Transformers for Time Series Forecasting

Transformers have gained significant popularity due to their self-attention mechanism, and extensive research has focused on enhancing their attention capabilities and modeling temporal dependencies. PatchTST Nie, 2022 reformulates time series forecasting by segmenting sequences into patch tokens, enabling the model to capture local temporal patterns more effectively. Self-attention is then applied to these patches, which embed temporal information, allowing the model to attend over longer histories while maintaining efficiency. In addition, PatchTST adopts a channel-independent design, where each variate is modeled separately but shares the same embedding and Transformer weights. This combination preserves local semantics and achieves strong performance across multivariate forecasting benchmarks. iTransformer Y. Liu et al., 2023 restructures the Transformer by inverting its input dimensions, treating each variate as a token rather than each timestamp. This allows the attention mechanism to capture cross-variable dependencies. By doing so, iTransformer achieves better utilization of multivariate structure and delivers state-of-the-art results across several real-world datasets. TimeXer Y. Wang et al., 2024 addresses the challenge of forecasting with exogenous variables by extending the Transformer to jointly model endogenous and exogenous information. It introduces specialized embeddings along with patch-wise self-attention and variate-wise cross-attention to integrate external signals with the target series. This design enables TimeXer to effectively capture the dependencies between internal and external variables, achieving state-of-the-art results across a wide range of real-world forecasting benchmarks.

Other Transformer variants such as Autoformer H. Wu et al., 2021, FEDformer T. Zhou et al., 2022, and Crossformer Y. Zhang and Yan, 2023, have also been proposed, but in this work, we focus on PatchTST, iTransformer, and TimeXer as state-of-the-art baselines for multivariate time series forecasting.

4.3 Proposed Methods

This section introduces the proposed approaches. First, we extend the classical Box–Cox transformation to the multivariate setting, enabling variance stabilization while preserving dependencies among correlated features. Second, we propose a look-back window (input to the model) and horizon-aware instance normalization mechanism, designed to mitigate distributional shift at the instance level by adapting normalization to the forecasting context.

4.3.1 Multivariate Box-Cox

The Box–Cox transformation is widely used to promote *constant error variance* and *normality* of the data Box and Cox, 1964. It applies a power transform to each *positive* feature independently. In the

classical formulation, the optimal parameter λ_j for feature x_j is obtained by *maximizing the log-likelihood* of the transformed values (equivalently, minimizing the negative log-likelihood).

In our experiments, we use scikit-learn's Pedregosa et al., 2011 Box–Cox implementation, which estimates each λ_j with *Brent's method*—a scalar, derivative-free, one-dimensional optimizer. Brent's algorithm blends golden-section search with parabolic interpolation, requires only function evaluations, and is robust on unimodal objectives with fast convergence near the optimum. Scikit-learn uses a bound of (-2,2); however, Brent will expand from (-2, 2) to form a proper three-point bracket internally if needed, and the final optimum can end up outside (-2, 2). We first prove the univariate Box-Cox to better understand the multivariate setup.

Univariate Box-Cox Derivation

For feature x_j with $x_{ij} > 0$, define the Box–Cox transform

$$y_{ij}(\lambda_j) = \begin{cases} \dfrac{x_{ij}^{\lambda_j} - 1}{\lambda_j}, & \lambda_j \neq 0, \\ \log x_{ij}, & \lambda_j = 0, \end{cases} \tag{4.1}$$

The log-likelihood for λ_j is

$$\ell_j(\lambda_j) = (\lambda_j - 1) \sum_{i=1}^{N} \log x_{ij} \; - \; \frac{N}{2} \log \left(\frac{1}{N} \sum_{i=1}^{N} \left(y_{ij}(\lambda_j) - \bar{y}_j(\lambda_j) \right)^2 \right). \tag{4.2}$$

The first term in equation 1.2 is the Jacobian term, which is an adjustment/correction for the change in probability distribution of the observed variable. This Jacobian term is the derivative of the transformation applied to the observed variable x_{ij}.

Transformation (per observation i of feature j). For $x_{ij} > 0$,

$$y_{ij} = \begin{cases} \dfrac{x_{ij}^{\lambda_j} - 1}{\lambda_j}, & \lambda_j \neq 0, \\ \log x_{ij}, & \lambda_j = 0. \end{cases}$$

Derivative (1-D Jacobian) w.r.t. x_{ij}.

$$\frac{\partial y_{ij}}{\partial x_{ij}} = \begin{cases} x_{ij}^{\lambda_j - 1}, & \lambda_j \neq 0, \\ \dfrac{1}{x_{ij}}, & \lambda_j = 0, \end{cases} \qquad \text{so} \qquad \log \left| \frac{\partial y_{ij}}{\partial x_{ij}} \right| = \begin{cases} (\lambda_j - 1)\, \log x_{ij}, & \lambda_j \neq 0, \\ -\log x_{ij}, & \lambda_j = 0. \end{cases}$$

Change of variables for one observation (1-D). Preserving small probability mass, $f_X(x_{ij})\, dx_{ij} = f_Y(y_{ij})\, dy_{ij}$, gives

$$f_X(x_{ij}) \;=\; f_Y(y_{ij}) \left| \frac{\partial y_{ij}}{\partial x_{ij}} \right| \;=\; f_Y(y_{ij})\, x_{ij}^{\lambda_j - 1},$$

hence

$$\log f_X(x_{ij}) \;=\; \log f_Y(y_{ij}) \;+\; (\lambda_j - 1) \log x_{ij}.$$

Jacobian contribution for N i.i.d. observations of feature j. By independence,

$$\begin{aligned} \log L(\lambda_j \mid x_{\cdot j}) &= \sum_{i=1}^{N} \log f_Y(y_{ij}) + \sum_{i=1}^{N} \log \left| \frac{\partial y_{ij}}{\partial x_{ij}} \right| \\ &= \sum_{i=1}^{N} \log f_Y(y_{ij}) + (\lambda_j - 1) \sum_{i=1}^{N} \log x_{ij} \end{aligned} \tag{4.3}$$

Thus the *Jacobian (change–of–variables) term* in the log-likelihood is

$$\sum_{i=1}^{N} \log \left| \frac{\partial y_{ij}}{\partial x_{ij}} \right| \;=\; (\lambda_j - 1) \sum_{i=1}^{N} \log x_{ij},$$

with the $\lambda_j = 0$ case, it becomes $-\sum_i \log x_{ij}$.

Deriving the variance term in the Box–Cox log-likelihood Let $y_{ij}(\lambda_j)$ be the transformed observations of feature j. Assume i.i.d. Normal with mean μ_j and variance σ_j^2:

$$y_{ij}(\lambda_j) \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu_j, \sigma_j^2), \qquad i = 1, \ldots, N. \tag{4.4}$$

Likelihood (product of Gaussian densities):

$$L(\mu_j, \sigma_j^2 \mid y_{\cdot j}(\lambda_j)) = (2\pi\sigma_j^2)^{-N/2} \exp\left(-\frac{1}{2\sigma_j^2} \sum_{i=1}^{N} \big(y_{ij}(\lambda_j) - \mu_j \big)^2 \right). \tag{4.5}$$

Taking Log:

$$\ell(\mu_j, \sigma_j^2 \mid y_{\cdot j}(\lambda_j)) = -\frac{N}{2}\log(2\pi) - \frac{N}{2}\log(\sigma_j^2) - \frac{1}{2\sigma_j^2}\sum_{i=1}^{N}\big(y_{ij}(\lambda_j) - \mu_j\big)^2. \tag{4.6}$$

Substituting $\hat{\mu}_j(\lambda_j) = \bar{y}_j(\lambda_j)$ gives the profile log-likelihood

$$\ell_{\text{prof}}(\sigma_j^2 \mid \lambda_j) = -\frac{N}{2}\log(2\pi) - \frac{N}{2}\log(\sigma_j^2) - \frac{1}{2\sigma_j^2}\sum_{i=1}^{N}\big(y_{ij}(\lambda_j) - \bar{y}_j(\lambda_j)\big)^2.$$

MLE for σ_j^2 is:

$$\hat{\sigma}_j^2(\lambda_j) = \frac{1}{N}\sum_{i=1}^{N}\big(y_{ij}(\lambda_j) - \bar{y}_j(\lambda_j)\big)^2. \tag{4.7}$$

Plugging $\hat{\sigma}_j^2(\lambda_j)$ into ℓ_{prof},

$$\ell_{\text{prof}}(\hat{\sigma}_j^2(\lambda_j) \mid \lambda_j) = -\frac{N}{2}\log(2\pi) - \frac{N}{2} - \frac{N}{2}\log\left(\frac{1}{N}\sum_{i=1}^{N}\big(y_{ij}(\lambda_j) - \bar{y}_j(\lambda_j)\big)^2\right). \tag{4.8}$$

Dropping constants that do not depend on λ_j (i.e., $-\frac{N}{2}\log(2\pi)$ and $-\frac{N}{2}$), the part that depends on λ_j is

$$-\frac{N}{2}\log\left(\frac{1}{N}\sum_{i=1}^{N}\big(y_{ij}(\lambda_j) - \bar{y}_j(\lambda_j)\big)^2\right),$$

which is exactly the variance term used in the Box–Cox log-likelihood. Entire (profile) log-likelihood for feature j:

$$\boxed{\ell_j(\lambda_j \mid x_{\cdot j}) \;=\; \underbrace{(\lambda_j - 1)\sum_{i=1}^{N}\log x_{ij}}_{\text{Jacobian term}} \;-\; \underbrace{\frac{N}{2}\log\left(\frac{1}{N}\sum_{i=1}^{N}\big(y_{ij}(\lambda_j) - \bar{y}_j(\lambda_j)\big)^2\right)}_{\text{variance term}}} \tag{4.9}$$

This formulation, however, optimizes each λ_j independently, ignoring any correlations between features. In multivariate time series datasets with correlated variables, this independent treatment can lead to suboptimal variance stabilization. To address this limitation, we extend the log likelihood implementation of univariate Box-Cox to a multivariate setting by jointly estimating all transformation parameters $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_p)$ through a covariance matrix.

Multivariate Box-Cox Derivation

For observation i and feature j with $x_{ij} > 0$, define the componentwise Box–Cox transform

$$y_{ij}(\lambda_j) = \begin{cases} \dfrac{x_{ij}^{\lambda_j} - 1}{\lambda_j}, & \lambda_j \neq 0, \\ \log x_{ij}, & \lambda_j = 0, \end{cases} \qquad \text{and} \quad \mathbf{y}_i(\boldsymbol{\lambda}) = \big(y_{i1}(\lambda_1), \ldots, y_{ip}(\lambda_p)\big)^\top.$$

Assume $\mathbf{y}_i(\boldsymbol{\lambda}) \overset{\text{i.i.d.}}{\sim} \mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

Jacobian Term. Because we transform each feature separately, the Jacobian matrix is diagonal with entries $\partial y_{ij}/\partial x_{ij} = x_{ij}^{\lambda_j - 1}$, so

$$\big|\det J_i(\boldsymbol{\lambda})\big| = \prod_{j=1}^{p} x_{ij}^{\lambda_j - 1} \quad \Longrightarrow \quad \log\big|\det J_i(\boldsymbol{\lambda})\big| = \sum_{j=1}^{p} (\lambda_j - 1)\, \log x_{ij} \tag{4.10}$$

Summing over $i = 1, \ldots, N$ gives the total Jacobian term

$$\sum_{i=1}^{N} \log\big|\det J_i(\boldsymbol{\lambda})\big| = \sum_{j=1}^{p} \left[(\lambda_j - 1) \sum_{i=1}^{N} \log x_{ij} \right] \tag{4.11}$$

Gaussian likelihood and Trace Identity. The multivariate likelihood for $\{\mathbf{y}_i(\boldsymbol{\lambda})\}_{i=1}^N$ is:

$$L(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \mathbf{y}(\boldsymbol{\lambda})) = (2\pi)^{-\frac{Np}{2}} \, |\Sigma|^{-\frac{N}{2}} \exp\left(-\tfrac{1}{2} \sum_{j=1}^{N} (y_j - \mu)^\top \Sigma^{-1} (y_j - \mu) \right) \tag{4.12}$$

Taking log for log-likelihood:

$$\ell(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \mathbf{y}(\boldsymbol{\lambda})) = -\frac{Np}{2} \log(2\pi) - \frac{N}{2} \log\big|\boldsymbol{\Sigma}\big| - \frac{1}{2} \sum_{i=1}^{N} \big(\mathbf{y}_i - \boldsymbol{\mu}\big)^\top \boldsymbol{\Sigma}^{-1} \big(\mathbf{y}_i - \boldsymbol{\mu}\big) \tag{4.13}$$

For the third term in equation 4.13, we use the identity $\sum_i \mathbf{a}_i^\top \mathbf{M}\, \mathbf{a}_i = \mathrm{tr}\big(\mathbf{M} \sum_i \mathbf{a}_i \mathbf{a}_i^\top\big)$, with $\mathbf{a}_i = \mathbf{y}_i - \boldsymbol{\mu}$ and $\mathbf{M} = \boldsymbol{\Sigma}^{-1}$, we can rewrite third term as:

$$
\begin{aligned}
&-\frac{1}{2}\sum_{i=1}^{N} \mathrm{tr}\Big(\Sigma^{-1}(y_i-\mu)(y_i-\mu)^\top\Big) \\
&-\frac{1}{2}\,\mathrm{tr}\Big(\Sigma^{-1}\sum_{i=1}^{N}(y_i-\mu)(y_i-\mu)^\top\Big)
\end{aligned}
\tag{4.14}
$$

Profile over $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. The MLEs (given $\boldsymbol{\lambda}$) are

$$
\begin{aligned}
\hat{\boldsymbol{\mu}}(\boldsymbol{\lambda}) &= \bar{\mathbf{y}}(\boldsymbol{\lambda}) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{y}_i(\boldsymbol{\lambda}), \\
\hat{\boldsymbol{\Sigma}}(\boldsymbol{\lambda}) &= \frac{1}{N}\sum_{i=1}^{N} \big(\mathbf{y}_i(\boldsymbol{\lambda}) - \bar{\mathbf{y}}(\boldsymbol{\lambda})\big)\big(\mathbf{y}_i(\boldsymbol{\lambda}) - \bar{\mathbf{y}}(\boldsymbol{\lambda})\big)^\top.
\end{aligned}
\tag{4.15}
$$

For readability, we henceforth drop the explicit $(\boldsymbol{\lambda})$ and write

$$
\hat{\boldsymbol{\mu}} = \bar{\mathbf{y}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{y}_i, \qquad \hat{\boldsymbol{\Sigma}} = \frac{1}{N}\sum_{i=1}^{N} (\mathbf{y}_i - \bar{\mathbf{y}})(\mathbf{y}_i - \bar{\mathbf{y}})^\top,
\tag{4.16}
$$

with the understanding that $\mathbf{y}_i$, $\bar{\mathbf{y}}$, and $\hat{\boldsymbol{\Sigma}}$ still depend on $\boldsymbol{\lambda}$. Equation 4.16 can be written as:

$$
N\hat{\boldsymbol{\Sigma}} = \sum_{i=1}^{N} (\mathbf{y}_i - \bar{\mathbf{y}})(\mathbf{y}_i - \bar{\mathbf{y}})^\top,
\tag{4.17}
$$

Plugging these into ℓ 4.13 yields the profile log-likelihood in $\boldsymbol{\lambda}$.

$$
\ell_{\text{prof}}(\boldsymbol{\lambda}) = -\frac{Np}{2}\log(2\pi) - \frac{N}{2}\log\big|\hat{\boldsymbol{\Sigma}}\big| - \frac{1}{2}\,\mathrm{tr}\left(\hat{\boldsymbol{\Sigma}}^{-1}\sum_{i=1}^{N}\big(y_i - \bar{y}\big)\big(y_i - \bar{y}\big)^\top\right).
\tag{4.18}
$$

Put equation 4.17 in equation 4.18.

$$
\ell_{\text{prof}}(\boldsymbol{\lambda}) = -\frac{Np}{2}\log(2\pi) - \frac{N}{2}\log\big|\hat{\boldsymbol{\Sigma}}\big| - \frac{1}{2}\,\mathrm{tr}\Big(\hat{\boldsymbol{\Sigma}}^{-1} N\hat{\boldsymbol{\Sigma}}\Big).
\tag{4.19}
$$

Where $\hat{\boldsymbol{\Sigma}}^{-1}\hat{\boldsymbol{\Sigma}} = I_p$ and $\mathrm{tr}(I_p) = p$, hence equation 4.19 can be written as:

$$
\ell_{\text{prof}}(\boldsymbol{\lambda}) = -\frac{Np}{2}\log(2\pi) - \frac{N}{2}\log\big|\hat{\boldsymbol{\Sigma}}\big| - \frac{Np}{2}
\tag{4.20}
$$

We drop the constants that do not depend on λ (i.e. $-\frac{Np}{2}\log(2\pi), -\frac{Np}{2}$).

$$\ell_{\text{prof}}(\boldsymbol{\lambda}) = -\frac{N}{2}\log\left|\hat{\boldsymbol{\Sigma}}\right| \tag{4.21}$$

Combine with the Jacobian. Adding the Jacobian contribution from the change of variables (across all i) gives

$$\ell_{\text{prof}}(\boldsymbol{\lambda} \mid \mathbf{X}) = \underbrace{\sum_{j=1}^{p}(\lambda_j - 1)\sum_{i=1}^{N}\log x_{ij}}_{\text{Jacobian term}} - \underbrace{\frac{N}{2}\log\left|\hat{\boldsymbol{\Sigma}}(\boldsymbol{\lambda})\right|}_{\text{covariance (variance) term}}$$

Multivariate Log-Likelihood.

$$\boxed{\ell_{\text{prof}}(\boldsymbol{\lambda} \mid \mathbf{X}) = \sum_{j=1}^{p}(\lambda_j - 1)\sum_{i=1}^{N}\log x_{ij} - \frac{N}{2}\log\left|\hat{\boldsymbol{\Sigma}}(\boldsymbol{\lambda})\right|} \tag{4.22}$$

where $\hat{\boldsymbol{\Sigma}}(\boldsymbol{\lambda}) = \frac{1}{N}\sum_{i=1}^{N}\big(\mathbf{y}_i(\boldsymbol{\lambda}) - \bar{\mathbf{y}}(\boldsymbol{\lambda})\big)\big(\mathbf{y}_i(\boldsymbol{\lambda}) - \bar{\mathbf{y}}(\boldsymbol{\lambda})\big)^{\top}$. This work uses the Broyden–Fletcher–Goldfarb-Shanno (BFGS) optimizer to estimate the multivariate Box–Cox parameters λ by minimizing the negative profiled log-likelihood. It uses gradients to move toward the optimum. If we do not supply a gradient, BFGS approximates it by first differences. (If box constraints were required, we would use L-BFGS-B; here, unconstrained BFGS suffices.) Unlike the univariate formulation, which optimizes each λ_j in isolation, this multivariate extension incorporates both feature-specific variances and inter-feature covariances into the optimization process. Although the transformation is still applied feature by feature, the parameters are now estimated jointly, ensuring that the resulting powers, λ, reflect the dependency structure among correlated features.

Previous work on multivariate Box–Cox transformations has largely remained theoretical, most supported only by simulation-based analyses Rahman and Pearson, 2009; Velilla, 1992, and limited to implementation in **R**. In this study, we bridge this gap by developing the first complete **Python implementation** of the multivariate Box–Cox transformation, fully compatible with the design and interface of Scikit-learn's preprocessing framework. Moreover, we conduct comprehensive experiments on real-world datasets, demonstrating its practical effectiveness and integration potential within modern machine learning workflows.

4.3.2 Context-aware Instance Normalization

Given a batch of multivariate input sequences $X = \{x^{(i)}\}_{i=1}^{N}$ with $x^{(i)} \in \mathbb{R}^{T\times C}$, we consider C variables (channels), a look-back length T, a prediction length H and N is the number of input sequences. For

each sequence, we compute instance statistics.

$$\mu_c^{(i)} = \frac{1}{T}\sum_{t=1}^{T} x_{c,t}^{(i)}, \qquad \sigma_c^{(i)} = \sqrt{\frac{1}{T}\sum_{t=1}^{T}\left(x_{c,t}^{(i)} - \mu_c^{(i)}\right)^2 + \epsilon},$$
$$\ell_c^{(i)} = x_{c,T}^{(i)} \tag{4.23}$$

where $\mu_c^{(i)}$, $\sigma_c^{(i)}$, and $\ell_c^{(i)}$ are the mean, standard deviation, and last value of variable c in sequence i.

We extend RevIN by making both the *look-back window, T*, and the *forecasting horizon, H*, explicit in the normalization and denormalization process. In the normalization phase, we are transforming the input sequence to make it easier for the model to learn. Since the tail of the sequence (i.e., the most recent time steps) is closest to the forecast point, its behavior directly influences short-term predictions. To preserve local continuity near the forecast point, we shift the sequence by its last value, thereby recentering it around the forecast point and making short-term trends easier to capture. However, earlier in the sequence, the data might have drifted away from the current level. If we keep subtracting the last value across the entire sequence, we might distort or bias earlier signals. Therefore, for the earlier part of the sequence, we subtract the instance mean instead, which gives a more neutral, stable reference and better represents the historical level of that segment.

We construct a context-aware blending mask over the temporal dimension. Let K denote a user-defined hyperparameter that specifies how many of the last steps in the look-back window should rely on the most recent (last in sequence) value. We define a mask m of the same length as the sequence, filled with zeros everywhere except for the last K steps, where it is set to one. In other words, m looks like [0,0,…,0,1,1,…,1] with exactly K ones at the tail. The normalized z sequence at time t is then computed as:

$$z_t = m_t \cdot \ell^{(i)} + (1 - m_t) \cdot \mu^{(i)}, \tag{4.24}$$

which means that for the last K steps ($m_t = 1$), we normalize using the last value $\ell^{(i)}$, and for all earlier steps ($m_t = 0$), we normalize using the mean $\mu^{(i)}$. This design enables the model to focus on short-term variations close to the prediction point while keeping the rest of the sequence aligned to its overall statistical structure.

The model produces normalized predictions $\tilde{y}^{(i)}$, which must then be mapped back to the original scale. To achieve this, the *denormalization step* is also made horizon-aware. For each forecast horizon $h = 1, \ldots, H$, we condition the denormalization on whether the horizon falls within a short-term cutoff or a long-term regime:

$$\hat{y}_{:,h}^{(i)} = \begin{cases} \tilde{y}_{:,h}^{(i)} \cdot \sigma^{(i)} + \ell^{(i)}, & h \le H_{\text{cutoff}} \\ \tilde{y}_{:,h}^{(i)} \cdot \sigma^{(i)} + \mu^{(i)}, & h > H_{\text{cutoff}}, \end{cases} \tag{4.25}$$

where H_{cutoff} is a horizon threshold hyperparameter that divides short-term forecasts (closer to the last observation) from long-term forecasts (better stabilized by the mean). The idea is that the first few forecast steps are temporally close to the last observed input, so we assume the series hasn't shifted much yet.

Adding back the last value maintains continuity and makes the predictions look closer to the ground truth. As we move further into the future (later horizons), the influence of the last value fades, and continuing to use it can cause the predictions to drift unrealistically (especially if it was an outlier). For these longer-term horizons, it's safer to use the mean, which represents a more stable, long-term estimate of the sequence's level.

CoIN enables the normalization–denormalization layers to be reversible while adapting to both the look-back window and the forecasting horizon. As a result, the model can leverage recent information for near-term predictions while avoiding error accumulation and drift in longer-term forecasts. CoIN is a simple yet effective technique that can be adopted for any deep neural network architecture.

4.4 Experiments

Datasets. We evaluate our results mainly on two real-world time series datasets. **(i) Influenza-like illness (ILI)**[1] dataset is collected from the Centers for Disease Control and Prevention (CDC) from 1^{st} October 2002 to 30^{th} June 2020, and has the ILI patients' data recorded every week for the United States. The data[2] consists of 966 samples and seven variables, such as weighted and unweighted ILI cases, data by age group, number of providers, and total number of affected patients. **(ii) COVID-19 Weekly:** The daily data was extracted from the COVID-19 Dataset by Our World in Data (OWID)[3] Mathieu et al., 2020. We included the dataset from the first death till the weekly reporting of deaths. The daily data collected was converted to the weekly data and is available from 29^{th} February 2020 to 14^{th} May 2022 on GitHub[4]. The reason for choosing these dates is to utilize the peak periods during model training and to avoid the flat trend that appears after May 2022. Each data time point has variables, such as deaths, tests, cases, vaccinations, hospitalizations, and patients admitted to the Intensive Care Units (ICU). **(iii) RSV Weekly:** The weekly RSV data are obtained from the CDC Respiratory Syncytial Virus Laboratory Data (NREVSS) [5]. This dataset is regional and contains two types of diagnostic test methods. We aggregate the data from both test types across all regions to obtain national-level data. The dataset includes RSV positive detections, total tests, surveillance year, and week-ending code. We further merge lagged weather variables (temperature and humidity) extracted using the METEOSTAT library, as these factors exhibit a nonlinear relationship with the virus (Correlation map given in Appendix B). The processed dataset is available on GitHub [6] and consists of 10 variables, including RSV indicators and lagged weather variables, with a total of 519 weekly samples.

[1] https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html

[2] https://github.com/scalation/data/blob/master/Influenza/national_illness.csv

[3] https://github.com/owid/covid-19-data

[4] https://github.com/scalation/scalation_py/blob/dev_sr/PandemicForecasting/Train-Test%20split/datasets/covid_till14May22.csv

[5] https://data.cdc.gov/Laboratory-Surveillance/Respiratory-Syncytial-Virus-Laboratory-Data-NREVSS/52kb-ccu2/about_data?utm_source=chatgpt.com

[6] https://github.com/scalation/scalation_py/blob/dev_sr/datasets/RSV/rsv_weather_lagged.csv

Preprocessing Methods. We evaluate logarithmic, square-root, Box–Cox, Yeo–Johnson, and differencing transformations applied before standardization. `Log1p` is adopted for values near zero. Both first-order and seasonal differencing are considered, with the best results reported. Formal definitions are provided in Appendix B.

Experimental Settings. For the ILI dataset, we set the prediction length H to {6,12,24,36,48,60}, for the COVID-19, we set it to {1,2,3,4,5,6} and for RSV to {6,12,24,36,48} . For PatchTST, we used the exact hyperparameters specified in the Illness script[7], all other hyperparameters are tuned on StandardScaler using Ray-Tune Liaw et al., 2018 for a fair comparison. Each model was trained on all features of the datasets, and results are reported on the original scale for the target variables: $ILITOTAL$ (ILI), new_deaths (COVID-19) and $RSV\,Detections$ (RSV). We evaluate the models performance on symmetric mean absolute percentage error (sMAPE), and the mean absolute error (MAE). The full implementation and hyperparameter details for each model are provided in Appendix B.

Baslines Compared. We evaluate the effectiveness of our proposed methods, along with other preprocessing methods, on three state-of-the-art Transformer models, PatchTST Nie, 2022, TimeXer Y. Wang et al., 2024, and iTransformer Y. Liu et al., 2023. All these transformer models use RevIN without learnable parameters or a standard z-score instance normalization.

4.5 Results

4.5.1 Preprocessing techniques

Table 4.1 reports the averaged results of preprocessing techniques for ILI cases, COVID-19 deaths, and RSV Detections (Horizon-wise results are provided in Appendix B). Across all models, every preprocessing method outperforms StandardScaler, with improvements of up to 44%. On the ILI dataset, PatchTST benefits the most from preprocessing, with logarithmic transformations reducing its sMAPE from 46.06 to 27.9 (39%) and MAE from 10,938 to 8,751 (19.9%). On the COVID-19 dataset, the largest relative gains are observed for TimeXer with square root transformation, where sMAPE decreases from 38.12 to 32.30 (15.2%) and MAE from 3,577.07 to 2,972.23 (16.9%). For the RSV dataset, the most significant improvements are observed in sMAPE (TimeXer), with gains of up to 44% achieved through logarithmic transformations. In contrast, the lowest MAE values are obtained using the Square Root transformation, with gains up to 6.9% (PatchTST), which effectively stabilizes large variations while preserving the data's overall scale.

Despite these performance gains, iTransformer achieves the lowest overall sMAPE errors, with values of 25.90 on the ILI dataset and 29.83 on the COVID-19 dataset, while remaining competitive in MAE. In comparison, TimeXer and PatchTST record the smallest errors on the RSV dataset. We conclude that the COVID-19 dataset is short and highly correlated across variables such as cases, deaths, and hospitalizations.

[7] https://github.com/yuqinie98/PatchTST/blob/main/PatchTST_supervised/scripts/PatchTST/illness.sh

iTransformer's variate-wise attention effectively captures inter-feature dependencies, taking advantage of the strong correlations among variables. PatchTST's patching mechanism struggles in this setting because the limited sequence length in COVID-19 does not provide enough temporal context to form meaningful patches while still remaining competitive. iTransformer performs best on the COVID-19 and ILI datasets, where correlations are particularly strong (Correlation heatmaps are provided in Appendix B). Meanwhile, the RSV dataset is long and highly seasonal, offering sufficient temporal depth for PatchTST and TimeXer to extract informative patches. The ILI dataset also benefits from PatchTST's patching mechanism due to its extended and periodic structure. The detailed results explanation is given below.

The results on the ILI dataset confirm the well-established role of logarithmic transformations in reducing right skewness, stabilizing variance, and making distributions more normal Feng et al., 2013; Galli, 2024; Yeo and Johnson, 2000. This effect is visible in the ILITOTAL histograms (Figure 4.2a), where log1p produces a nearly normal distribution compared to StandardScaler, which leaves the original skewness intact. StandardScaler simply rescales values under the assumption of normality, without addressing skewness or variance heterogeneity. As a result, the distributional shape remains unchanged, leaving models exposed to outliers and uneven variance.

Although Transformers are theoretically capable of capturing nonlinearities directly, logarithmic transformations simplify optimization by stabilizing variance, reducing skewness, and linearizing multiplicative effects such as exponential growth in epidemic or financial time series. By compressing extreme values, the log transform prevents the loss from being dominated by peaks, allowing the model to capture fluctuations more accurately. After inverse transformation, these learned patterns are rescaled appropriately, enabling peaks to be reproduced with greater fidelity. Figure 4.3 illustrates this for ILITOTAL: while StandardScaler tracks the overall trend but underestimates extreme values, log1p captures the full height of peaks more precisely, enabling PatchTST to represent fluctuations with greater accuracy. More plots are shown in figure 4.4, which show the performance of Log1p in different horizon settings, both short and long-term forecasting. The final few points, where none of the models or transformations perform well, correspond to the period of the COVID-19 outbreak, an event not observed in the training data, resulting in reduced model performance during that phase.

Box–Cox and Yeo–Johnson both select an optimal λ close to 0.19 for ILITOTAL, which reshapes the distribution toward normality in a manner similar to a logarithmic transformation. While these methods yield the second-best results for PatchTST and iTransformer, they achieve the best performance for TimeXer. Log1p, along with differencing, also improves variance stabilization and non-stationarity compared to differencing alone, as illustrated in Figure B.6, which explains its superior forecasting performance for PatchTST and iTransformer. Formal tests of variance equality (Levene's test) confirming this effect are provided in Appendix B.

It is important to note that logarithmic transformations do not always reduce skewness; in some cases, they may even introduce additional skewness Feng et al., 2014. This is evident in the COVID-19 dataset (Figure 4.2b), where the distribution was already relatively close to normal. Applying log1p made the data left-skewed: the early weeks with very few deaths (1, 15, 42, …) were overstretched into small isolated

values, while the bulk of the pandemic (thousands to tens of thousands of deaths) was compressed into a narrow range (log1p of ∼10k–20k ≈ 9–10). This drawback degraded the performance of most models.

Box–Cox and Yeo–Johnson both selected optimal parameters close to 0.47, which corresponds to a transformation similar to the square root. Unlike logarithmic transformations that compress large values more aggressively, the square root transformation compresses larger values and expands smaller ones relative to the raw scale, making the distribution closer to normal. It is widely used to reduce skewness and stabilize variance in time series data when the data is not too skewed or is close to a normal distribution, as it does not overly compress large values like the logarithm. Moreover, when variance is proportional to the mean, the square root transformation is often the most appropriate choice Bartlett, 1936. Figure 4.7 illustrates this effect for COVID-19 deaths with a 10-week window: while the original and standardized series show rapidly increasing variance as the mean grows, the square root transformation produces a more stable mean–variance relationship with a few outliers. The residual variance is 0.7 under StandardScaler and decreases to 0.4 with the square root transformation, with a bit of variance instability at higher fitted values, suggesting that the square root transformation helps "reduce" heteroscedasticity.

Consistent with this statistical behavior, the square root transformation produces noticeable improvements for PatchTST and TimeXer compared to the StandardScaler. In these patch-based architectures, stabilizing variance appears to enhance the quality of patch embeddings. These results suggest that square root–type transformations align well with the inductive biases of patch-based representations and attention mechanisms. However, as discussed earlier, iTransformer performs best on the COVID-19 dataset, demonstrating that variate-wise attention captures correlations in the time series more effectively. This observation is further supported by the attention maps of the COVID-19 dataset in Figure 4.6, where the correlation heatmaps closely align with the attention maps in the iTransformer model.

The RSV data is right-skewed, containing both small and large values. Statistical transformations have a substantial impact on model performance. Applying these transformations significantly reduces skewness, making the data appear more symmetric—though not perfectly normally distributed, as illustrated in the histograms 4.8. Log1p emerges as the best technique, yielding the largest and most consistent gains in sMAPE across all models by effectively reducing skewness and balancing proportional errors between low and high values. Box-Cox and Yeo-Johnson with λ parameters of -0.007 and -0.01, respectively, also align transformations with that of the logarithm. Square Root performs as the second-best method, providing stable improvements and the lowest MAE scores due to its moderate compression of large values without distorting scale. While sMAPE benefits strongly from transformations that normalize variability, MAE shows smaller or mixed gains because it reflects absolute deviations on the original scale after inverse transformation.

Table 4.2 reports the results of the proposed multivariate Box–Cox method. We refer to multivariate Box–Cox as **Multi** and the original Box–Cox as **Classic**. For the ILI dataset, Multi outperforms Classic for all three models, with the smallest gains observed for TimeXer on sMAPE only. Multi selects $\lambda \approx 2.15 \times 10^{-6}$, effectively a logarithmic transform, yielding results close to those obtained with log1p in Table 4.1; this explains the ILI performance. In particular, Multi changes the ILITOTAL transform from a mild Box–Cox ($\lambda \approx 0.19$) to a log-like transform ($\lambda \approx 0$), which more effectively stabilizes variance. Figure

4.9 further illustrates how Multi captures the COVID-19 outbreak portion of the ILITOTAL dataset more effectively than Classic, which most preprocessing methods fail to represent. For COVID-19, the optimal λ chosen by Multi is ≈ 0.6, corresponding to a Box–Cox transform. In this case, Multi improves iTransformer and TimeXer but underperforms Classic for PatchTST (by $\approx 4\%$), which uses the $\lambda \approx 0.47$. Overall, the best results for both datasets are achieved by iTransformer, owing to its ability to capture long-range dependencies across features, which complements the effects of multivariate preprocessing. For the RSV dataset, the multi–Box–Cox transformation outperforms the classic Box-Cox, although the improvements are minimal. We apply the transformation only to RSV variables, as the weather covariates contain negative values for which Box–Cox is not applicable, but we apply z-normalization on all of them to bring them on a comparable scale after Box–Cox (More explanation is given in Appendix B). The estimated parameters are close to zero ($\lambda_{\text{classic}} \approx -0.0077$, $\lambda_{\text{multi}} \approx 0.053$), effectively exhibiting log-like behavior. Consequently, the multi–Box–Cox provides mild variance stabilization with modest gains in both sMAPE and MAE. Among the models, TimeXer achieves the lowest sMAPE, while PatchTST yields the lowest MAE, indicating that even in a channel-independent architecture, each series benefits from covariance stabilization. However, we may notice more benefits from multi–Box–Cox transformation if more numeric datasets were available.

4.5.2 CoIN

Table 4.3 summarizes the results of short and long-term forecasting with Transformer models across the datasets. Overall, CoIN surpasses RevIN in nearly all experiments, achieving improvements of up to 24.6% and demonstrating its effectiveness across both short and long horizons. The primary gains of CoIN emerge in the early horizons. Transformer models typically adopt RevIN, which normalizes using mean-based subtraction and applies the same mean for denormalization to get the predictions on the original scale. Experiments show that shorter horizons benefit more from using the most recent value in the look-back window, as it is closest to the initial forecasting point. Adding this value during denormalization produces outputs that align more closely with the ground truth, unless the recent value is an outlier. Hence, in our denormalization process, we use the most recent value for early forecast points and the mean for later horizons. As horizons increase, the performance of CoIN converges toward that of RevIN. This trend is illustrated in Figure 4.10, where the forecast errors of CoIN closely follow those of RevIN in later weeks, while CoIN clearly outperforms RevIN at earlier horizons. We also extend this procedure to the look-back window. Specifically, we assume that the most recent steps of the look-back window (the tail of the sequence length) benefit from the most recent value, while the earlier steps benefit more from the overall statistic of the window (i.e., the mean). We experimented with different values of K and H_{cutoff}, and obtained the best results when using the last value for the tail of the look-back window and for earlier forecast horizons, and the mean for the earlier steps of the look-back window and later horizons. We report the results corresponding to the best average scores. Complete hyperparameters, K and H_{cutoff}, are provided in the Table 4.4.

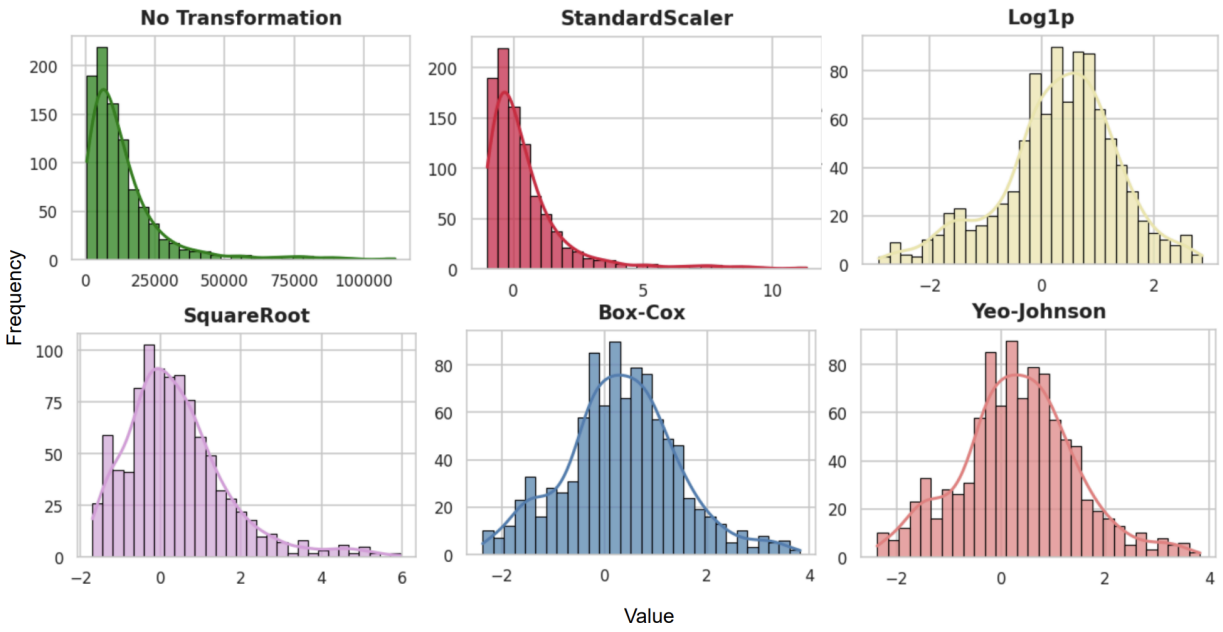
Figures 4.11 (a)–(f) illustrate the predictions at the initial horizons of a 24-week ahead forecast on ILITOTAL and 6-weeks ahead on COVID-19 deaths using transformers, comparing RevIN with CoIN.

CoIN closely tracks the ground truth, accurately capturing both peaks and troughs, whereas RevIN struggles to reproduce these critical variations.

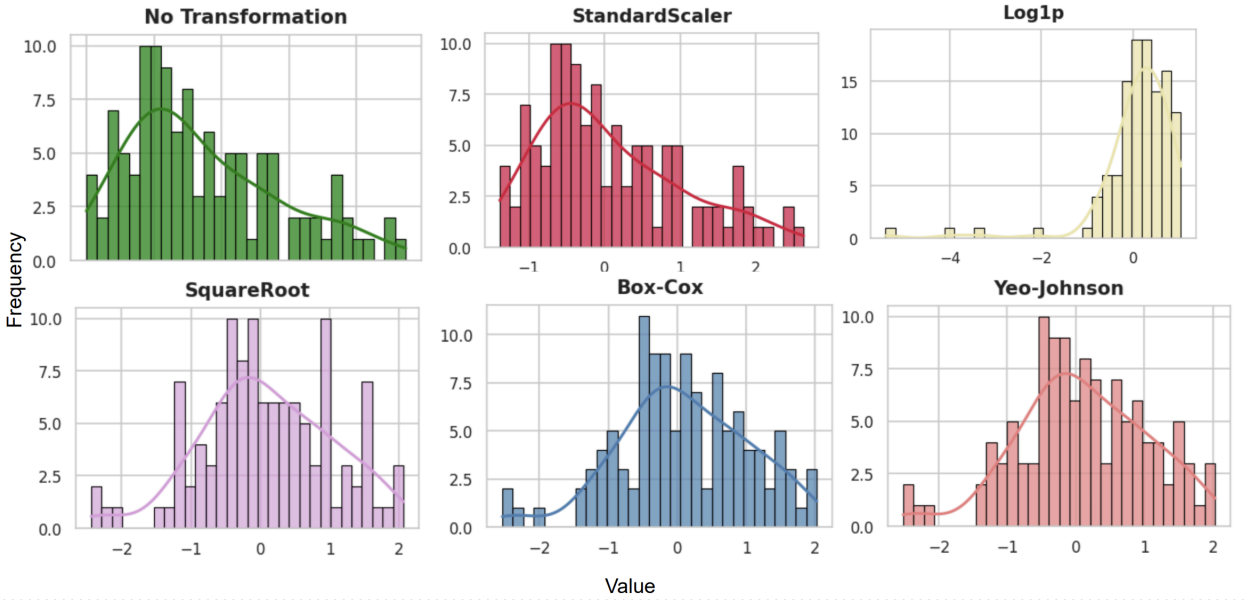
CoIN delivers its most notable relative gains with PatchTST, followed by TimeXer and then iTransformer. This effect stems from the interaction between CoIN's context-aware normalization and the models' architectural design. CoIN first operates on the look-back window: by using the last value to normalize the tail of the window and the mean for the remaining portion, the input becomes more locally aligned. When patching is applied, each segment inherits a sharper and more localized temporal structure, making the patches more informative. Temporal self-attention can then integrate these localized signals with long-range dependencies more effectively, amplifying the benefits of CoIN. Although this mechanism is especially advantageous for patch-based architectures, horizon-aware normalization further ensures that early horizons, which rely more on recent changes, are stabilized. Overall, CoIN benefits all types of models, including feature-interaction architectures such as iTransformer, demonstrating its effectiveness as an instance normalization strategy. We also experimented with moving average techniques for instance normalization, results of those techniques are shown in Appendix B.

4.6 Conclusion

This paper addresses the major challenges in real-world time series data, such as distributional shift, skewness, and heteroscedasticity, which standard normalization alone cannot handle well. We explored several statistical preprocessing techniques, including logarithmic, square root, Box–Cox, Yeo–Johnson, and differencing, and applied them to Transformer models. We also implemented a multivariate extension of the classic Box–Cox transformation in Python and tested its effectiveness on multivariate datasets. Finally, we proposed CoIN, an input and horizon-aware normalization method designed to handle distributional shift at the instance level. Our three proposed approaches outperform their respective baselines across all three Transformer models on average. On the ILI dataset, logarithmic transformations provide up to a 39.4% improvement, while on the COVID-19 dataset, square root–like transformations yield up to a 16.9% improvement. iTransformer achieves the best results on sMAPE across preprocessing methods, including the multivariate Box–Cox, while remaining competitive on MAE for ILI and COVID due to its ability to capture variate-wise attention among highly correlated variables. The RSV dataset exhibits mixed improvements, with Log1p achieving the highest gains in sMAPE (up to 44%) and Square Root performing best on MAE (up to 6.89%). Among the models, TimeXer and PatchTST show the most significant overall gains on RSV, benefiting from their ability to capture long-range dependencies through patch-based representations in large datasets. CoIN further delivers state-of-the-art results on all three Transformer models, with performance gains of up to 24.6%. Its strongest improvements are observed on PatchTST and TimeXer, on average, reflecting the benefits of context-aware normalization for patch-based architectures.



(a)



(b)

Figure 4.2: Comparison of preprocessing transformations applied before training transformer models. (a) Distributions of ILITOTAL (ILI dataset). (b) Distributions of new_deaths (COVID-19 dataset).

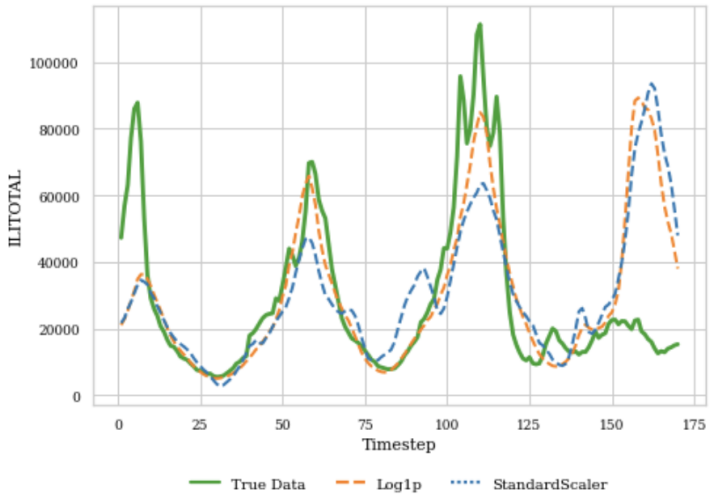


Figure 4.3: The above plots compare the ground truth ILITOTAL values with 24-weeks ahead forecasts obtained through Log1p and StandardScaler using PatchTST after inverse transformation, capturing the data's peaks and troughs.

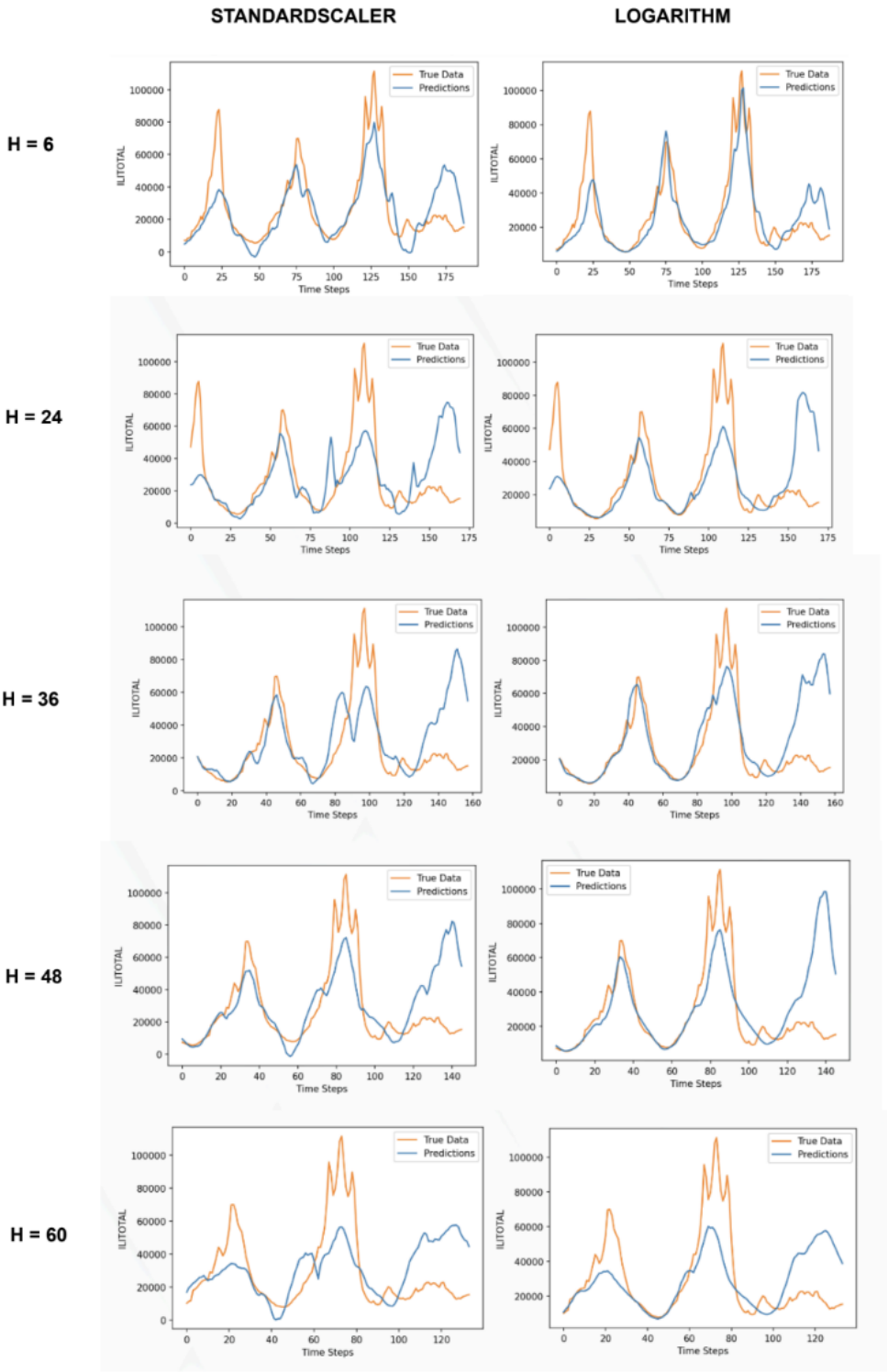


Figure 4.4: The above plots compare the ground truth ILITOTAL values with 6,24,36,48,60-weeks ahead forecasts obtained through Logıp and StandardScaler using iTransformer after inverse transformation, capturing the data's peaks and troughs.

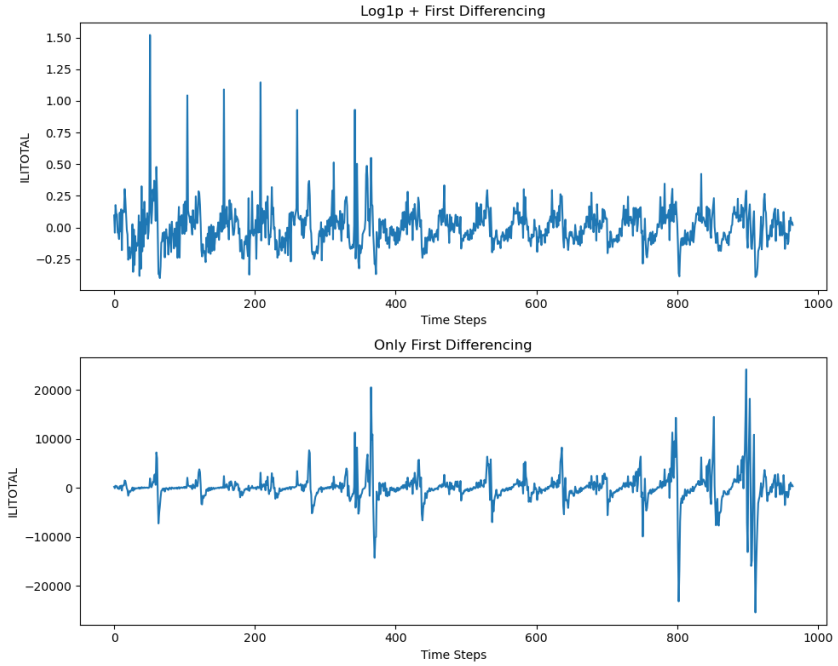


Figure 4.5: In the figure above, the top plot shows the ILITOTAL series after applying both a log1p transformation and differencing, while the bottom plot shows the series after applying only differencing.

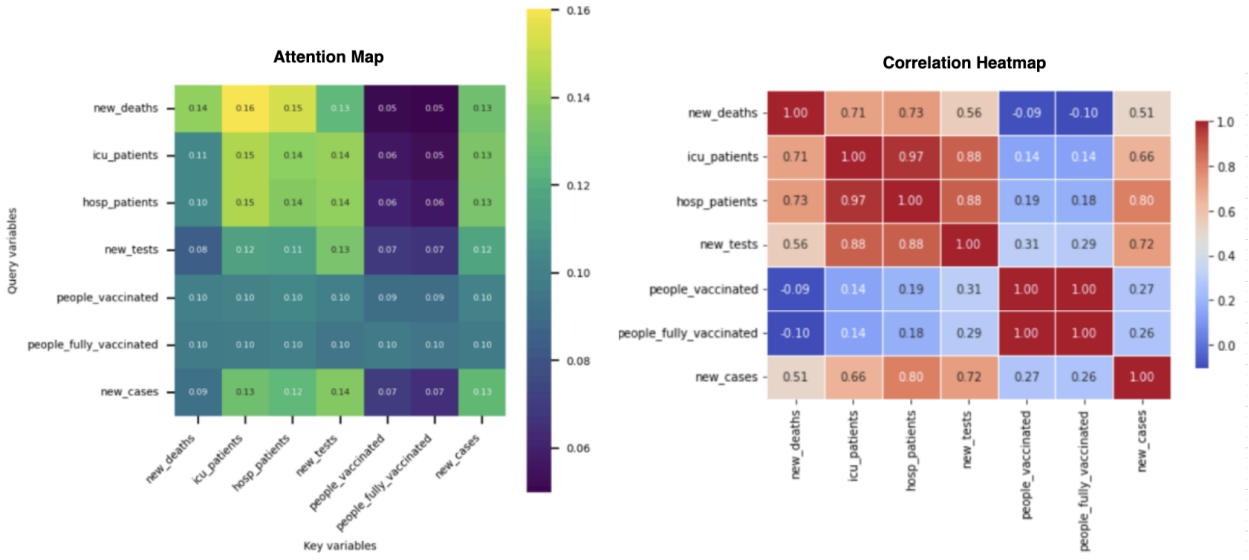


Figure 4.6: This figure shows the attention map from the iTransformer model and correlation heatmap for the COVID-19 dataset.

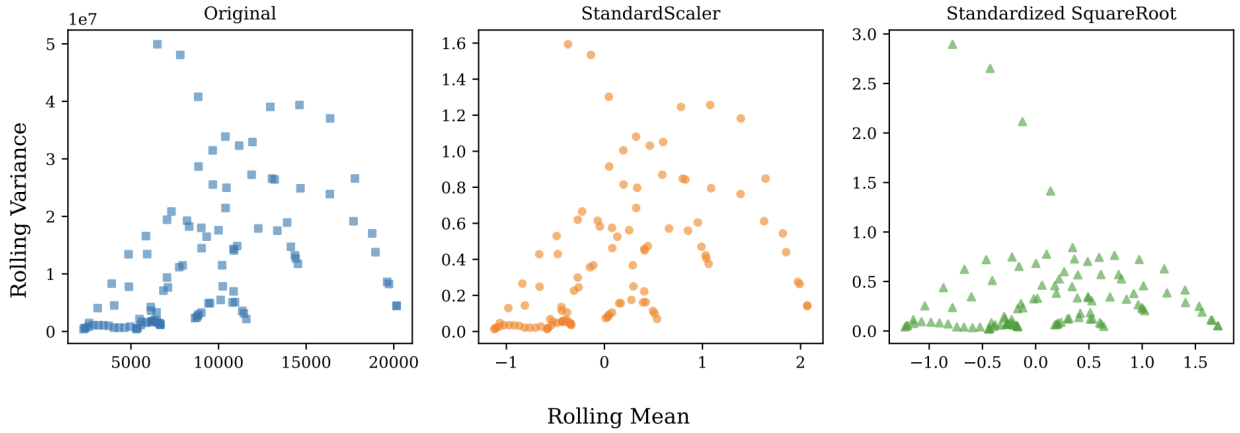


Figure 4.7: Rolling mean–variance plots of COVID-19 weekly deaths over a 10-week window under different preprocessing strategies. Left: the original series shows variance that grows rapidly with larger means. Middle: applying StandardScaler reduces the scale, but variance remains. Right: combining square root transformation with standardization compresses large values and stretches smaller ones, producing more stable variance across means.

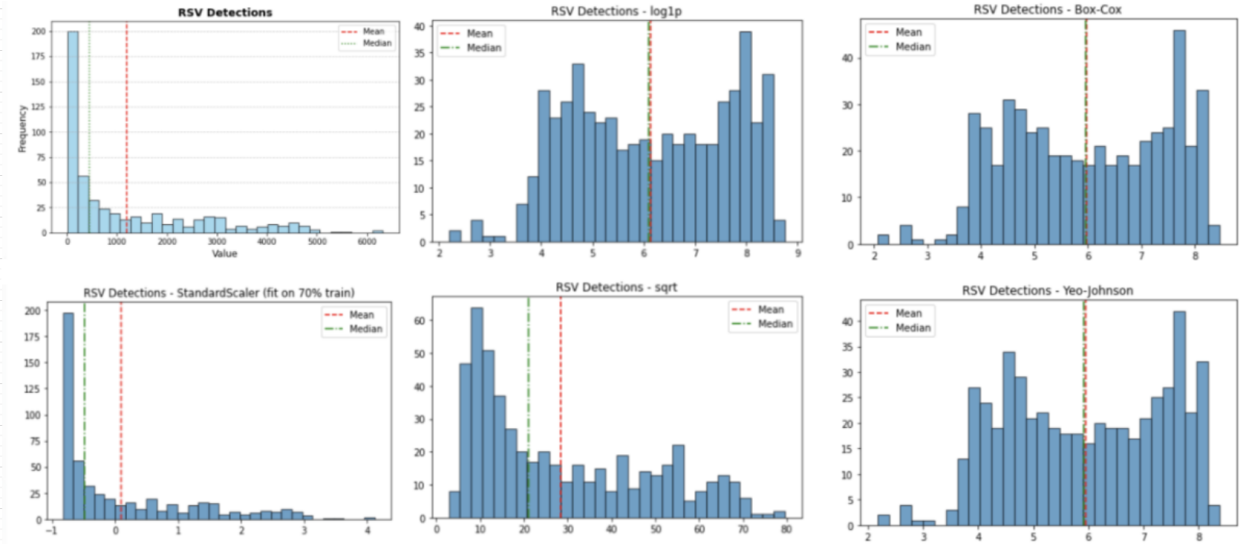


Figure 4.8: Distribution Comparison of preprocessing transformations applied before training transformer models on the RSV dataset.

Table 4.1: Multivariate time series forecasting performance and relative gains (%) of preprocessing methods on Transformers for ILI, COVID-19 and RSV datasets. Results are averaged across all horizons (weeks) H = {6,12,24,36,48,60} for ILI, {1,2,3,4,5,6} for COVID-19 and {6,12,24,36,48} for RSV. Gains are computed relative to the StandardScaler baseline. Best results are bolded in red, while second-best are bolded in blue.

Dataset (Metric)	Model	StandardScaler	Log1p	SQRT	Box–Cox	Yeo–Johnson	Log1p+Diff
ILI (sMAPE)	PatchTST	46.06	**27.90**	32.35	31.82	31.63	**27.99**
	Gain (%)	–	+39.4	+29.7	+30.9	+31.3	+39.2
	iTransformer	37.15	**25.90**	29.10	28.46	**28.26**	28.28
	Gain (%)	–	+30.3	+21.7	+23.4	+23.9	+23.9
	TimeXer	33.64	29.69	30.20	**29.36**	**29.27**	32.04
	Gain (%)	–	+11.7	+10.2	+12.7	+13.0	+4.8
ILI (MAE)	PatchTST	10938.81	**8751.02**	9838.01	9522.17	9532.62	**9356.59**
	Gain (%)	–	+19.9	+10.0	+13.0	+12.9	+14.5
	iTransformer	10499.34	**9169.59**	9600.21	9661.26	9621.16	**9542.97**
	Gain (%)	–	+12.7	+8.6	+8.0	+8.4	+9.1
	TimeXer	10263.94	10021.75	9910.38	**9862.43**	**9834.25**	10376.41
	Gain (%)	–	+2.4	+3.4	+3.9	+4.2	-1.1
COVID-19 (sMAPE)	PatchTST	35.66	37.87	**30.73**	30.95	**30.48**	34.98
	Gain (%)	–	-6.2	+13.8	+13.2	+14.6	+1.9
	iTransformer	30.17	37.00	**29.83**	32.59	31.20	**29.93**
	Gain (%)	–	-22.7	+1.1	-8.0	-3.4	+0.8
	TimeXer	38.12	35.74	**32.30**	**32.91**	36.79	42.24
	Gain (%)	–	+6.3	+15.2	+13.7	+3.5	-10.8
COVID-19 (MAE)	PatchTST	3715.11	3782.99	3252.95	**3168.61**	**3131.14**	3581.51
	Gain (%)	–	-1.8	+12.4	+14.7	+15.7	+3.6
	iTransformer	**2569.60**	3686.17	3152.80	2912.50	3039.91	**2857.06**
	Gain (%)	–	-43.4	-22.7	-13.3	-18.3	-11.2
	TimeXer	3577.07	3427.12	**2972.23**	**3292.66**	3440.56	4216.63
	Gain (%)	–	+4.2	+16.9	+7.9	+3.8	-17.9
RSV (sMAPE)	PatchTST	42.84	**26.08**	27.46	**26.15**	26.17	27.96
	Gain (%)	–	+39.12	+35.90	+38.95	+38.91	+34.73
	iTransformer	48.26	**27.07**	31.79	27.28	27.30	**27.01**
	Gain (%)	–	+43.90	+34.13	+43.47	+43.43	+44.03
	TimeXer	40.16	**25.90**	**25.84**	25.91	25.93	26.87
	Gain (%)	–	+35.50	+35.65	+35.48	+35.43	+33.09
RSV (MAE)	PatchTST	**323.22**	368.91	**301.06**	371.69	372.49	363.28
	Gain (%)	–	-14.14	+6.86	-15.00	-15.24	-12.39
	iTransformer	415.81	**400.80**	**394.92**	409.36	410.00	416.14
	Gain (%)	–	+3.61	+5.02	+1.55	+1.40	-0.08
	TimeXer	**336.40**	372.90	**334.97**	373.56	374.03	388.75
	Gain (%)	–	-10.85	+0.43	-11.05	-11.19	-15.56

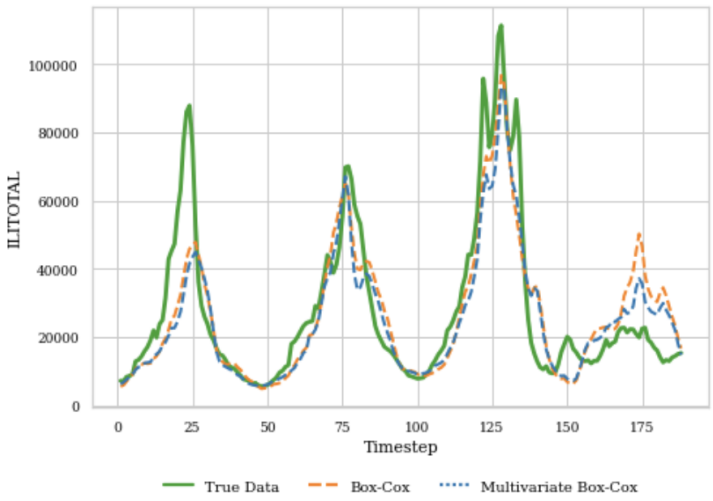


Figure 4.9: Comparison of Classic and Multi Box–Cox for short-term ILITOTAL forecasting over a 6-step horizon using PatchTST.

Table 4.2: Forecasting performance of the Classic Box–Cox vs. Multivariate Box–Cox normalization across horizons (weeks) for Transformers. Bold red numbers indicate the better (smaller) averaged results. Percentage gains are relative to Box–Cox.

ILLNESS (ILI)					COVID-19				RSV				
PatchTST													
Weeks	sMAPE		MAE		Weeks	sMAPE		MAE		Weeks	sMAPE		MAE	
	Classic	Multi (Gain%)	Classic	Multi (Gain%)		Classic	Multi (Gain%)	Classic	Multi (Gain%)		Classic	Multi (Gain%)	Classic	Multi (Gain%)
6	23.56	20.66	6554.55	5965.57	1	11.34	12.29	1184.82	1309.87	6	29.12	28.89	281.78	272.96
12	35.06	29.75	10205.94	9249.14	2	16.26	18.51	1819.25	2106.52	12	28.07	27.74	362.33	353.62
24	36.08	28.82	9403.71	9745.37	3	21.25	22.31	2381.44	2503.75	24	24.44	24.23	393.05	383.03
36	32.51	28.34	10520.07	9649.44	4	35.83	36.22	3944.62	3828.92	36	25.85	25.32	462.50	427.39
48	29.40	29.04	9267.83	9000.39	5	44.35	48.77	4528.56	5056.26	48	23.30	22.69	358.79	344.07
60	34.32	32.00	11180.91	10158.27	6	56.67	54.92	5152.98	5436.38	–	–	–	–	–
Avg	31.82	**28.10** (11.7%)	9522.17	**8961.36** (5.9%)	Avg	**30.95**	32.17	**3168.61**	3373.62	Avg	26.15	**25.77**(1.45%)	371.69	**356.21**(4.16%)
iTransformer														
Weeks	sMAPE		MAE		Weeks	sMAPE		MAE		Weeks	sMAPE		MAE	
	Classic	Multi (Gain%)	Classic	Multi (Gain%)		Classic	Multi (Gain%)	Classic	Multi (Gain%)		Classic	Multi (Gain%)	Classic	Multi (Gain%)
6	21.22	19.72	6538.90	6174.83	1	10.61	16.18	996.69	1244.53	6	32.77	32.56	397.19	380.73
12	27.79	25.88	8937.26	8522.84	2	16.91	15.91	1816.38	1680.64	12	29.94	29.78	426.47	411.71
24	32.73	29.19	11491.66	10913.69	3	24.82	23.79	2353.61	2304.90	24	26.45	25.95	472.35	448.02
36	32.26	27.39	11015.40	9941.26	4	35.51	28.25	3207.09	2882.63	36	23.13	23.61	369.74	381.76
48	28.08	25.79	9864.82	9591.77	5	50.34	38.00	4156.71	3946.68	48	24.12	23.63	381.06	366.75
60	28.66	27.67	10119.55	10089.83	6	57.38	48.52	4944.54	4641.85	–	–	–	–	–
Avg	28.45	**25.94** (8.8%)	9661.26	**9205.70** (4.7%)	Avg	32.59	**28.44** (12.7%)	2912.50	**2783.54** (4.4%)	Avg	27.28	**27.10** (0.65%)	409.36	**397.79**(2.82%)
TimeXer														
Weeks	sMAPE		MAE		Weeks	sMAPE		MAE		Weeks	sMAPE		MAE	
	Classic	Multi (Gain%)	Classic	Multi (Gain%)		Classic	Multi (Gain%)	Classic	Multi (Gain%)		Classic	Multi (Gain%)	Classic	Multi (Gain%)
6	22.95	22.39	7506.29	7307.80	1	13.25	12.87	1476.11	1440.79	6	30.51	30.39	368.85	365.83
12	29.65	29.25	9589.29	9703.64	2	20.45	23.74	2310.00	2208.74	12	29.16	29.22	395.64	394.65
24	30.56	30.33	10773.22	10793.94	3	25.80	25.52	2806.27	2555.97	24	25.90	24.88	419.27	407.92
36	28.86	28.99	10051.17	10327.72	4	35.42	30.17	3578.18	2942.23	36	22.23	22.19	357.39	353.35
48	33.17	33.20	10912.04	11108.14	5	48.08	47.12	4555.40	4150.25	48	21.75	21.52	326.63	319.42
60	30.98	31.34	10342.57	10719.40	6	54.43	50.23	5030.01	4361.50	–	–	–	–	–
Avg	29.36	**29.25** (0.4%)	**9862.43**	9993.44	Avg	32.91	**31.61** (4.0%)	3292.66	**2943.25** (10.6%)	avg	25.91	**25.64**(1.04%)	373.56	**368.23**(1.42%)

Table 4.3: Multivariate time series forecasting performance of RevIN vs. CoIN normalization across horizons for PatchTST, iTransformer, and TimeXer. Each cell shows RevIN and CoIN (Gain%). Bold red values indicate the best (smallest) averages.

ILLNESS (ILI)					COVID-19				
PatchTST									
Weeks	sMAPE		MAE		Weeks	sMAPE		MAE	
	RevIN	CoIN (Gain%)	RevIN	CoIN (Gain%)		RevIN	CoIN (Gain%)	RevIN	CoIN (Gain%)
6	35.40	22.56	8273.13	6311.72	1	12.49	14.60	1340.56	1512.70
12	49.40	30.50	11437.84	9205.96	2	20.11	18.81	2299.54	2183.72
24	46.71	41.57	12210.70	11678.96	3	29.01	25.33	3289.08	2800.98
36	47.97	39.72	11391.07	10917.75	4	40.05	37.98	4148.29	3882.33
48	58.11	37.98	11507.16	10560.14	5	50.92	45.46	5250.25	4805.20
60	38.75	35.97	10812.97	10411.99	6	61.39	55.69	5962.91	5491.21
Avg	46.06	**34.72** (24.6%)	10938.81	**9847.76** (10.0%)	Avg	35.66	**32.98** (7.5%)	3715.11	**3446.02** (7.3%)
iTransformer									
Weeks	sMAPE		MAE		Weeks	sMAPE		MAE	
	RevIN	CoIN (Gain%)	RevIN	CoIN (Gain%)		RevIN	CoIN (Gain%)	RevIN	CoIN (Gain%)
6	25.01	21.96	6771.75	6251.64	1	26.99	11.06	1867.33	1161.52
12	33.94	29.04	9200.27	8192.72	2	28.82	16.47	1862.07	1865.05
24	38.72	35.49	11252.74	11196.51	3	25.51	23.28	2037.14	2475.55
36	39.48	37.86	11713.13	11325.72	4	24.77	33.09	2523.21	3505.01
48	40.18	37.94	11420.25	11457.78	5	30.99	37.13	3208.39	4153.05
60	45.61	38.82	12637.90	11534.38	6	43.95	48.67	3919.48	4964.15
Avg	37.15	**33.51** (9.8%)	10499.34	**9993.12** (4.8%)	Avg	30.17	**28.28** (6.3%)	**2569.60**	3020.72
TimeXer									
Weeks	sMAPE		MAE		Weeks	sMAPE		MAE	
	RevIN	CoIN (Gain%)	RevIN	CoIN (Gain%)		RevIN	CoIN (Gain%)	RevIN	CoIN (Gain%)
6	27.33	21.26	7710.90	6618.94	1	18.32	12.77	1833.92	1374.31
12	35.20	31.32	10221.03	8453.30	2	23.89	20.54	2467.32	2067.17
24	35.60	32.58	11215.61	11080.34	3	36.31	26.56	3453.34	2828.78
36	34.56	32.13	10757.35	10545.55	4	46.99	40.66	4219.89	3839.28
48	33.85	33.09	10574.79	10554.20	5	47.99	44.41	4427.96	4316.17
60	35.30	34.11	11103.93	10951.03	6	55.24	53.52	5060.09	5297.45
Avg	33.64	**30.75** (8.6%)	10263.94	**9700.56** (5.5%)	Avg	38.12	**33.08** (13.2%)	3577.07	**3287.19** (8.1%)

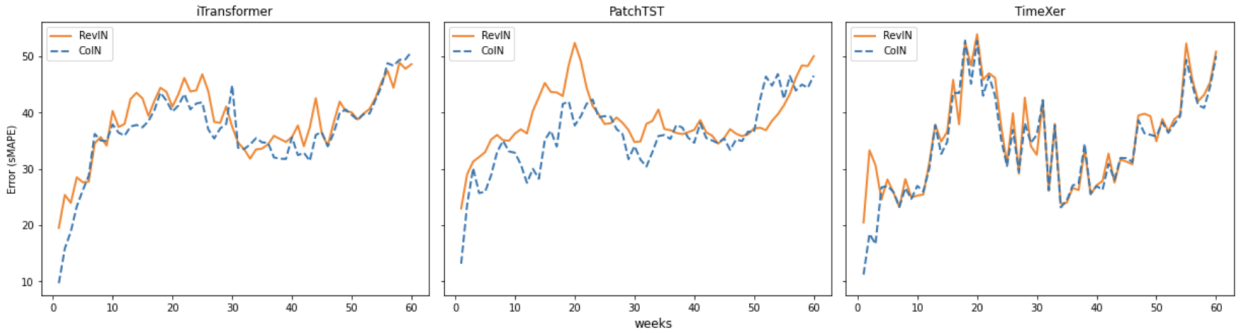
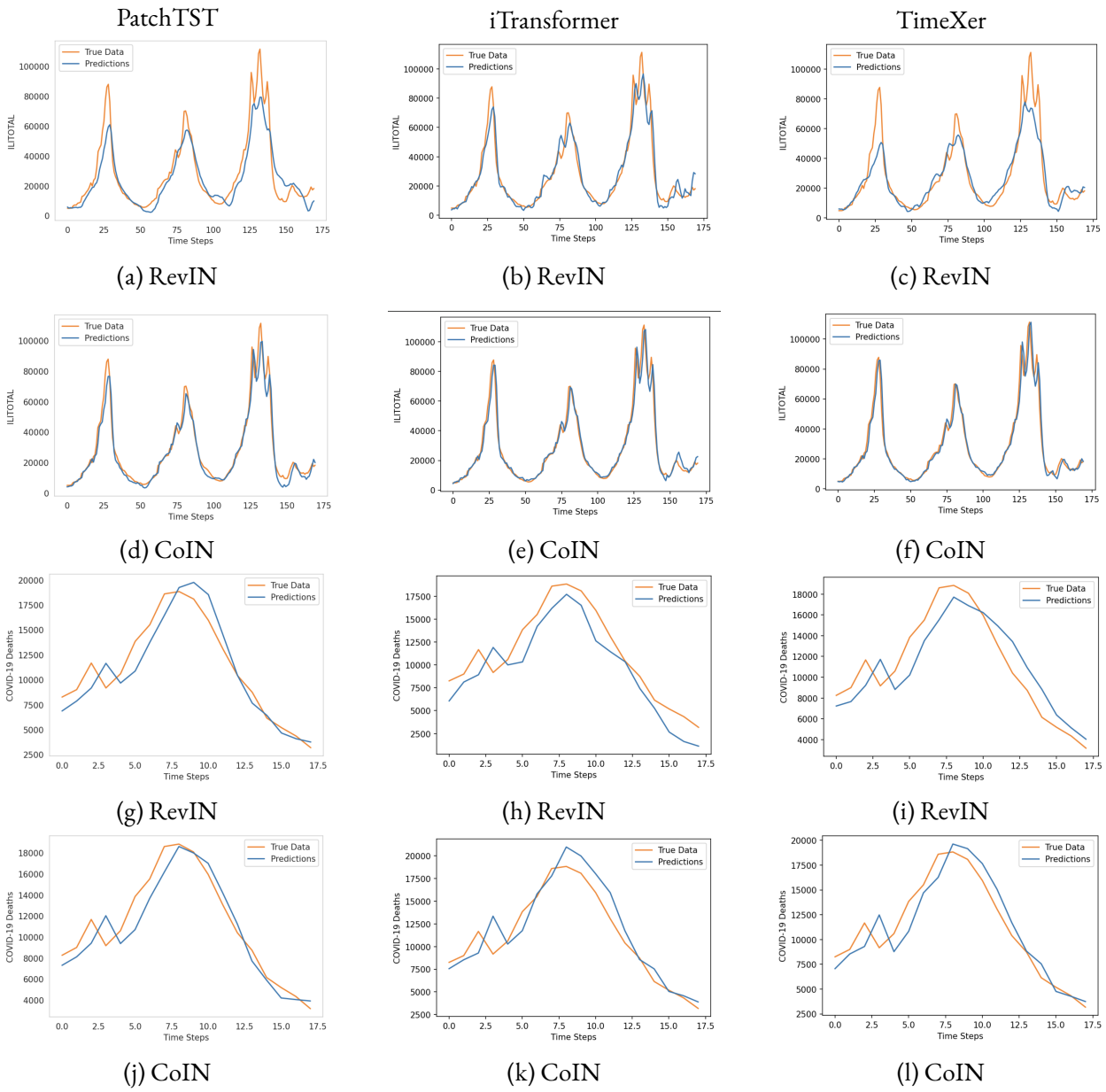


Figure 4.10: Per-horizon forecasting errors (sMAPEs) for 1–60 weeks ahead on the ILI dataset. The prediction length is fixed at 60 weeks, and results are reported for all three Transformer models under RevIN and CoIN normalization.

Table 4.4: CoIN hyperparameters H_{cutoff} and K per prediction horizon (weeks) across models.

		PatchTST	iTransformer	TimeXer
ILI				
$H = 6$	H_{cutoff}	5	1	4
	K	92	24	96
$H = 12$	H_{cutoff}	11	3	9
	K	96	24	92
$H = 24$	H_{cutoff}	8	3	3
	K	103	24	4
$H = 36$	H_{cutoff}	5	5	5
	K	5	48	3
$H = 48$	H_{cutoff}	48	2	2
	K	104	48	3
$H = 60$	H_{cutoff}	60	5	5
	K	104	60	0
COVID-19				
$H = 1$ to 6	H_{cutoff}	5	3	6
	K	2	3	3



(a) RevIN (b) RevIN (c) RevIN

(d) CoIN (e) CoIN (f) CoIN

(g) RevIN (h) RevIN (i) RevIN

(j) CoIN (k) CoIN (l) CoIN

Figure 4.11: Predictions of the initial forecast horizon for three models (PatchTST, iTransformer, and TimeXer), comparing RevIN with CoIN on both ILITOTAL and COVID-19 deaths.

Chapter 5

Summary of Research

This dissertation focuses on Multivariate Time Series Forecasting (MTSF) for infectious disease indicators such as COVID-19 deaths, ILI cases and RSV detections. Time-series forecasting leverages past observations to predict future trends, capturing both temporal dependencies and cross-variable relationships. Such models are vital for public-health decision-making, enabling accurate and timely forecasts. However, infectious disease data pose challenges: (i) limited data in early outbreaks leading to unreliable forecasts in deep learning; (ii) non-stationarity and distributional shift between train and test degrade generalization; and (iii) right-skew and heteroskedasticity cause models to fit the majority while underpredicting rare extremes, with forecast variance surging near outbreak peaks. Together, these factors lead to unstable predictions unless explicitly addressed.

Deep learning approaches commonly employ normalization techniques such as Min–Max scaling and z-normalization as a global preprocessing technique. The latter, z-normalization, is the most widely used method in recent SOTA Transformer models without considering the underlying shape of the data. Z-normalization assumes a Gaussian distribution and rescales features using the global mean and variance. While this improves training stability, it remains sensitive to extreme values or potential outliers, preserves skewed distributions, and fails to handle train–test distribution shifts. To address the challenges posed by limited, non-stationary, and skewed pandemic data, this dissertation proposes a two-fold approach. First, it evaluates sixteen forecasting models, ranging from statistical and recurrent architectures to Transformers, under both small and large data conditions, then incorporates a retraining strategy that updates models as new data becomes available. Experimental findings show that PatchTST and SARIMA perform best on the ILI and RSV datasets, effectively capturing long-range dependencies in larger and more stable datasets. In contrast, Seq2Seq models with attention outperform other methods on the COVID-19 dataset, where limited data benefit from their ability to capture short-term dynamics and local dependencies. The retraining strategy further enhances performance in the COVID-19 setting for ten models by enabling continuous model adaptation as new observations emerge, mitigating degradation in forecasting accuracy over time.

Second, it addresses normalization-related limitations in deep learning by introducing statistical transformations (Log, Square Root, Box–Cox, Yeo–Johnson, and Differencing) in Transformers to mitigate

skewness and variance instability. Furthermore, this work extends the Box–Cox transformation to a multivariate setting by jointly optimizing parameters based on feature covariances. This direction has been explored mostly theoretically and implemented previously in R. The dissertation presents full derivations, a Python implementation, and empirical evaluations for both univariate and multivariate cases.

As discussed in Chapter 4, recent Transformer variants commonly incorporate instance normalization techniques. Most employ Reversible Instance Normalization (RevIN), which mitigates distributional shifts by normalizing each input sequence individually. However, RevIN still assumes a uniform mean and variance across all time steps. This uniform approach overlooks the temporal heterogeneity present in real-world time series, where recent values often carry greater predictive importance than earlier ones, and by using the same statistics during denormalization, it fails to adapt to changing data characteristics across input and forecast horizons. To address this limitation, this dissertation introduces Context-aware Instance Normalization (CoIN), a reversible normalization method that considers both the input context and the forecasting horizon. CoIN normalizes recent time steps using the last observed value to preserve continuity near the forecast point and earlier time steps using the instance mean for historical stability. During denormalization, short-term forecasts are rescaled by the last observed value, while long-term forecasts are rescaled by the mean, thereby preventing drift as the influence of the most recent observation diminishes over time.

Empirical results in this dissertation demonstrate that integrating statistical transformations and improved normalization techniques into Transformer-based time series forecasting models consistently enhances performance, achieving up to 39.4% improvement in sMAPE and 19.9% in MAE on the ILI dataset, 15.2% in sMAPE and 16.9% in MAE on the COVID-19 dataset, and 44% in sMAPE and 6.89% in MAE on the RSV dataset compared to standard preprocessing, as well as up to 24.6% improvement over the existing instance normalization method. Among all models, the RSV dataset achieves the best performance with TimeXer and PatchTST, both of which effectively capture long-range dependencies through patch-based representations well suited for large datasets. For the COVID-19 dataset, iTransformer shows the most significant improvements by leveraging strong inter-variable correlations. The ILI dataset performs best with both PatchTST and iTransformer, benefiting simultaneously from its strong feature correlations and long temporal span.

Appendix A

Exploring the Predictive Power of Correlation and Mutual Information in Attention Temporal Graph Convolutional Network for COVID-19 Forecasting

Rana, S. Accepted by International Conference on Big Data. Reprinted here with permission of publisher, 09/23/2023

Accurately forecasting the COVID-19 spread across all states is crucial for implementing effective measures to control its transmission and minimize its impact. Since the virus's spread in one state can significantly affect other states over time through connections between them, a graph structure with temporal data is needed to capture the interdependence of COVID-19 spread among the states in the United States. In forecasting tasks that involve complex spatial and temporal dependencies, it is crucial to ensure that the model captures these dependencies accurately. In this study, we implemented an Attention Temporal Graph Convolutional Network-based model for COVID-19 mortality long-term prediction, which can effectively capture these dependencies. This model incorporates attention that enables us to weigh the significance of different time points and focus on the most informative data, including both adjacent and distant time points that capture the temporal dynamics accurately. For capturing spatial dependencies, we assessed the impact of using Pearson's correlation and Mutual Information to establish connections between highly dependent states. Our experiments showed that our model, particularly when utilizing mutual information, outperformed the existing baselines and the models that only consider neighboring states, resulting in lower sMAPE and MAE values. This emphasizes the importance of selecting the appropriate technique for accurate COVID-19 forecasting in each state. Furthermore, our model achieved the second-highest performance among the forecasting models submitted to the Centers for Disease Control and Prevention.

A.1 Introduction

The first case of COVID-19 in the United States (US) was reported in the state of Washington on January 20, 2020. Since then, the virus has spread rapidly across all 50 states, resulting in over 104 million confirmed cases and 1 million deaths as of May 4, 2023 "CDC Covid Tracker", n.d. The availability of daily and weekly COVID-19 data from various countries and states has provided opportunities to develop improved time-series models. Several statistical and machine learning (ML) models Chimmula and Zhang, 2020; P. Kumar et al., 2020; S. Kumar et al., 2021 have been used for COVID-19 forecasting. These methods either use the epidemic data of a single region to predict the future trend of the epidemic situation or establish a generalized model to predict the trend of all regions. Extensive research Chinazzi et al., 2020; Ferguson et al., 2006 has confirmed the highly contagious nature of the virus, and these studies have identified factors that contribute to its rapid spread across regions. Consequently, the spread of COVID-19 within a particular state can exert a substantial influence on neighboring states over time. This influence arises from various interconnected factors such as travel, trade, and social connections. To accurately forecast the spread of the virus in a particular state, it is important to consider the graph structure to capture these interconnections between states.

Recent studies Kapoor et al., 2020; Mahmud et al., 2021; Panagopoulos et al., 2021 have developed spatiotemporal Graph Neural Networks (GNNs) that operate on graphs, representing regions as nodes and capturing spatial and temporal dependencies between them. In forecasting tasks that involve complex spatial and temporal dependencies, it is crucial to ensure that the model captures both types of relationships accurately. These GNNs combine Graph Convolutional Networks (GCNs) to capture spatial dependen-

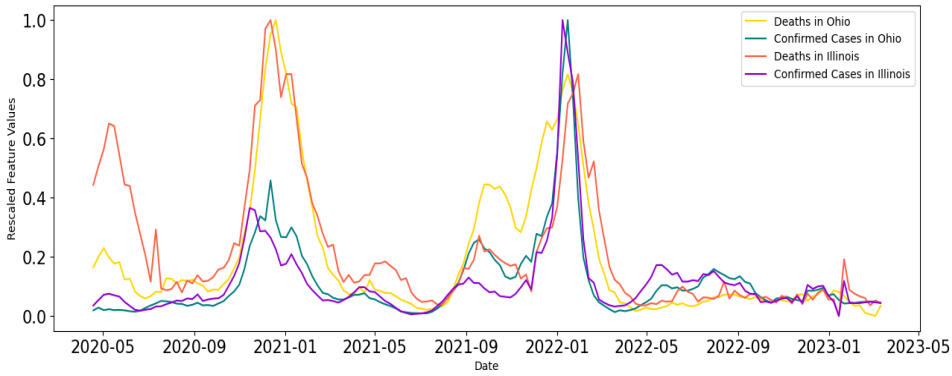


Figure A.1: Rescaled Deaths and Confirmed Cases in Ohio and Illinois indicating a strong linear relationship

cies and Recurrent Neural Networks (RNNs) or their variants to model temporal dependencies. RNNs process sequential data over time, and their hidden states carry the latest information from the past. However, this sequential processing can restrict the model's access to global information present throughout the input sequence. To address this issue, Attention mechanisms offer a solution by allowing the model to focus on relevant parts of the sequence and assign different weights, providing a means to learn and leverage global correlations. We utilize an attention-based model from PyTorch Geometric Temporal library (PyGT) Rozemberczki et al., 2021 to capture global information for COVID-19 weekly mortality prediction.

Another crucial aspect of GNN-based models is to accurately capture strong connections between different regions. Many existing GNN-based models Panagopoulos et al., 2021; L. Wang et al., 2020; Xue et al., 2022 for epidemic prediction rely on data such as mobility or social connections to establish connections between regions and capture spatial dependencies. However, acquiring and utilizing such data can pose challenges related to data availability, privacy, and accuracy. In contrast, we utilize both correlation and mutual information (MI) to capture both linear and nonlinear dependencies between the state's features. For instance, our approach shows a high correlation between Ohio and Illinois, which suggests there is a strong linear relationship between the deaths or confirmed cases in these two states. Specifically, it suggests that as deaths or confirmed cases increase in one state, they tend to increase in the other state as well. This relationship can be seen in Fig. A.1. The contributions of our proposed method are as follows:

1. We use the Attention Temporal Graph Convolutional Network (A3T-GCN) model from PyGT on the state graph for COVID-19 forecasting. The complete model allowed us to capture both local and global information about all the regions in the model, allowing us to capture temporal dependencies effectively.

2. We use correlation and MI to capture both linear and nonlinear dependencies and establish connections between highly dependent states in a graph, allowing us to effectively capture spatial dependencies. We use these graphs in the A3T-GCN model and compare these techniques with A3T-GCN based on adjacent states (states that share a border). We also compare our best models with the baselines.

3. We compared and achieved the second-highest performance among the forecasting models submitted to the Centers for Disease Control and Prevention (CDC).

A.2 Related work

Several studies have applied statistical, ML, and deep learning methods to predict COVID-19 forecasting based on various clinical, laboratory, and epidemiological data. Infectious disease prediction is often modeled as a time-series prediction problem in many studies, so many time-series methods have been studied S. Kumar et al., 2021. Chimmula et al. Chimmula and Zhang, 2020 analyzed the important factors and applied a Long Short Term Memory (LSTM) model to forecast the patterns and probable end date of the COVID-19 pandemic in Canada and also around the world. Cramer et al. Cramer et al., 2022 conducted a study that evaluates the effectiveness of both individual and ensemble probabilistic forecasts for predicting COVID-19 mortality in the US. The study focuses on evaluating the accuracy and reliability of these forecasts to provide insights into the effectiveness of different forecasting methods. Their work has been used by CDC for COVID-19 cases and death forecasts.

Kapoor et al. Kapoor et al., 2020 presented a forecasting method for COVID-19 cases using a spatiotemporal GNN that aims to capture the intricate dynamics involved in disease modeling by incorporating mobility data. In their proposed model, nodes in the graph represent county-level human mobility, spatial edges depict inter-regional interactions, and temporal edges account for the evolution of node features over time. Panagopoulos et al. Panagopoulos et al., 2021 introduced MPNN-TL, a GNN for COVID-19 dynamics across four European countries. By integrating mobility data and reported cases, the model aims to comprehend complex transmission patterns. It recognizes the vital role of mobility patterns in the disease's spread, emphasizing the significance of a graph-based representation for studying transmission dynamics and its impacts. Fritz et al. Fritz et al., 2021 proposed a fusion approach that combines GNN with epidemiological models to forecast the infection rate of COVID-19. The study utilized data from Facebook, including mobility and association information, as well as structural and spatial details of cities and districts in Germany. Cao et al. Cao et al., 2020 developed StemGNN, a model for multivariate time series that captures inter and intra-temporal correlations using the Graph Fourier Transform (GFT) and Discrete Fourier Transform (DFT). With a graph structure representing different countries, the study evaluated the model's performance in forecasting confirmed cases across multiple horizons.

A.3 Methodology

In this work, we utilized the A3T-GCN model from PyGT on the state graph to predict COVID-19 outcomes. This model allowed us to effectively consider both local and global information from all regions, capturing temporal dependencies. To capture both linear and nonlinear dependencies and establish connections between highly dependent states, we incorporated correlation and MI. By integrating these graphs into the A3T-GCN model, we compared its performance with the version that only considered adjacent states. Our evaluation showed a significant improvement when using different association techniques beyond adjacency. In this section, we will discuss the PyGT library, the model architecture with a customized dataset, and the association techniques used for the model.

A.3.1 PyGT

PyGT is an extension library for PyTorch Geometric. It is specifically designed for handling spatiotemporal data, such as dynamic graphs where edges and nodes change over time. PyGT includes various tools for creating, manipulating, and visualizing temporal graphs, as well as implementing GNN architectures for spatiotemporal data. This library provides several data iterators. We use **StaticGraphTemporalSignal** which is used when the underlying graph is fixed, but the features on each node or edge change over time. This data iterator provides an efficient way to iterate over temporal snapshots of a graph in batches. The library also comes with a train-test splitter that creates temporal splits of the data using a fixed split ratio and some benchmark datasets. In addition to that, it includes several types of existing Neural network models that operate on graphs. We use the model A3T-GCN which is based on the paper "A3T-GCN: Attention Temporal Graph Convolutional Network for Traffic Forecasting" Bai et al., 2021.

A.3.2 A3T-GCN

We implement an A3T-GCN (the second version of A3T-GCN) based model which is capable of handling batches. Our objective with this model is to use past values of COVID-19 deaths and confirmed cases to predict the weekly death counts for each state. This model, shown in Fig. A.2, is a combination of GCN and Gated Recurrent Unit (GRU) and features an attention mechanism.

The library requires a customized PyTorch Geometric-based dataset that is tailored to the problem of predicting COVID-19 deaths for the states. To achieve this, we define an adjacency matrix (based on adjacent states as an example in this figure) and a feature matrix with information on timesteps, number of states, and features. These matrices are passed to the model expanded from the A3T-GCN Bai et al., 2021 paper as shown in Fig. A.2.

1. **Adjacency Matrix:** A graph $G = (V, E)$ defines the graph structure of the US states where each node $v \in V$ represents a state in the US and each edge $e \in E$ represents the edge between two states. This whole information is defined by an adjacency matrix $A \in R^{N \times N}$ with N vertices (states in our case) and all the rows i and columns j are indexed by the states. The entry in i and

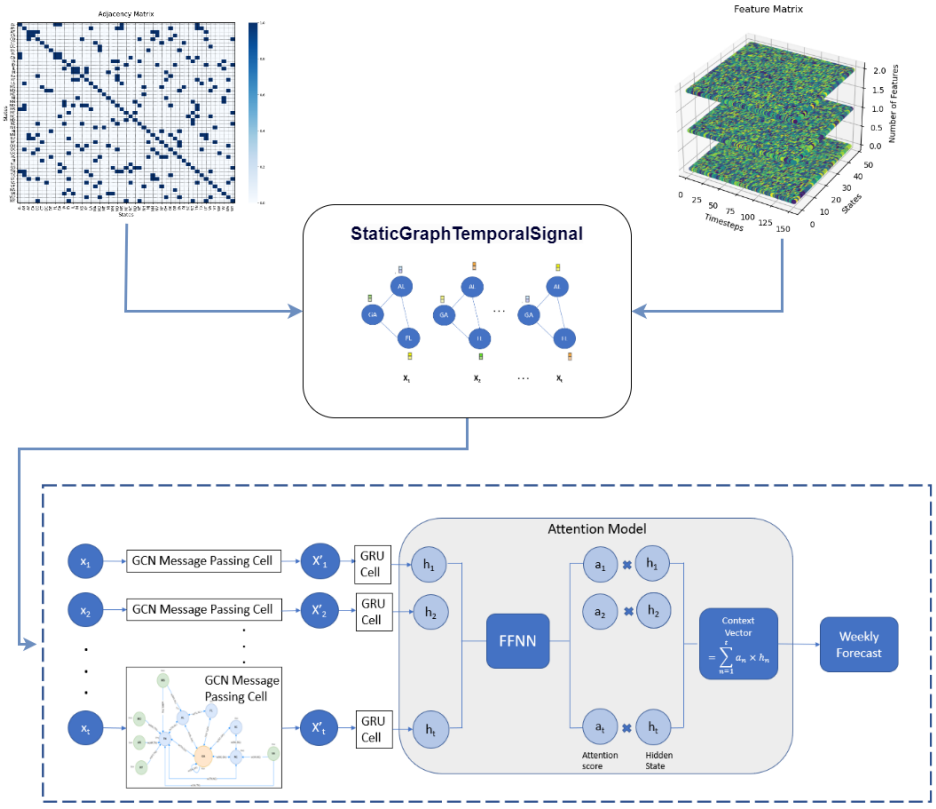


Figure A.2: Customized PyGT dataset based A3T-GCN architecture

j of the matrix, denoted by $A[i, j]$, represents the weight or degree of interdependence between state i and state j.

We use correlation and MI described in the next section to construct the matrices based on the degree of dependency between pairs of states. The procedure involves calculating the correlation and MI scores between the variables. We consider all pairs of states with a score greater than a certain threshold to be highly dependent and connect them in the adjacency matrix. Conversely, we set the weight of the edge to zero for all pairs of states with the scores below the threshold, indicating that they are not strongly interconnected.

2. **Feature Matrix:** COVID-19 deaths and confirmed cases are shown as the features of the nodes represented by a feature matrix $X \in R^{N \times F}$, where N is the number of states and F is the number of features. These features change over time, X_t is the features matrix at time t. We passed the historical values of all the features $X_{t-n}, \cdot, X_{t-1}, X_t$ along with the adjacency matrix A through the **StaticGraphTemporalSignal** iterator which returns PyTorch Geometric Data object for a single time period i.e., a week, which represents a snapshot of the graph for that time period. These temporal snapshots represent different feature values for the same underlying graph structure and

are passed as historical inputs to the GCN layers of the A3T-GCN model, which perform computations on the graph structure and associated features to produce output representations of the hidden state.

GCN.

In a GCN, each node in the graph is associated with a feature vector, and the goal is to learn a new set of feature vectors that capture the relationships in the graph through a message-passing process. Specifically, at each layer of the GCN, the feature vector of each node is updated by taking a weighted sum of the feature vectors of its neighbors, where the weights are learned by the network. This weighted sum is then combined with the feature vector of the node itself to produce a new feature vector. This enables the model to capture spatial dependencies. A two-layer GCN model Kipf and Welling, 2016 can be defined in (1) below:

$$f(X, A) = \sigma(\overline{A}\, ReLU\, (\overline{A}\, X\, W_0)\, W_1) \tag{A.1}$$

$$\overline{A} = D'^{-.5} A' D'^{-.5} \tag{A.2}$$

$$D'_{ii} = \sum_j A'_{ij} \tag{A.3}$$

$$A' = A + I_N \tag{A.4}$$

Here X is a feature matrix and A is an adjacency matrix defined above, $\overline{A}$ defined in (2) is a preprocessing step, where A' is an adjacency matrix with self connectivity defined in (4), I_N is an identity matrix, D' is a degree matrix defined in (3). $W_0 \in R^{K \times H}$ is a weight matrix from the input layer to the hidden unit layer where K is the length of time and H is hidden unit numbers, and $W_1 \in R^{H \times T}$ is a weight matrix from the hidden layer to the output layer, where T is the length of forecasted points. $f(X, A) \in R^{N \times T}$ is the output of the GCN model of the length T. The updated vectors are then passed to GRU.

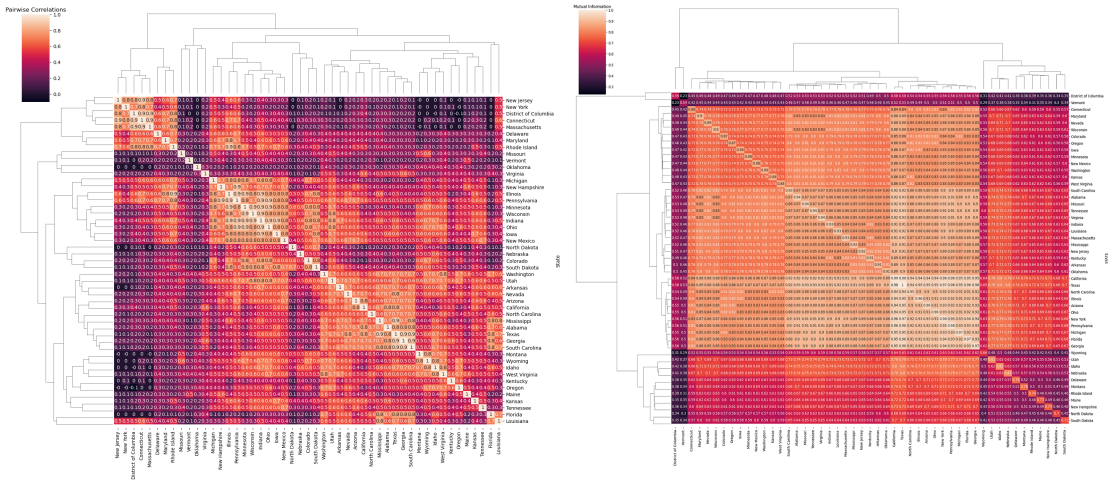
GRU.

GRUs have two gates, namely an update gate and a reset gate. The update gate determines how much of the previous hidden state should be retained for the current time step, and the reset gate controls how much of the new input should be incorporated into the new hidden state.

In our case, the gated mechanism of GRU allows them to capture the mortality information at the current moment while retaining the variation trends of historical COVID information. As a result, this model can effectively capture the dynamic temporal variation features of COVID data.

Attention.

The Attention model, which is a modified version of the Encoder-Decoder model, was initially developed for use in neural machine translation tasks Vaswani et al., 2017. In this particular study, a soft Attention model was employed to determine the importance of COVID information at each moment. This information was then used to calculate a context vector that captured the overall trends in the COVID mortality state, which could be utilized for predicting future mortality conditions.



(a) Correlation Matrix (b) MI Matrix

Figure A.3: Association matrices

In order to calculate the weight of each hidden state, a scoring function is designed, followed by an attention function that computes the context vector to capture the global COVID data variation information.

$$e_i = W_2(W_1 H + d_1) + d_2 \tag{A.5}$$

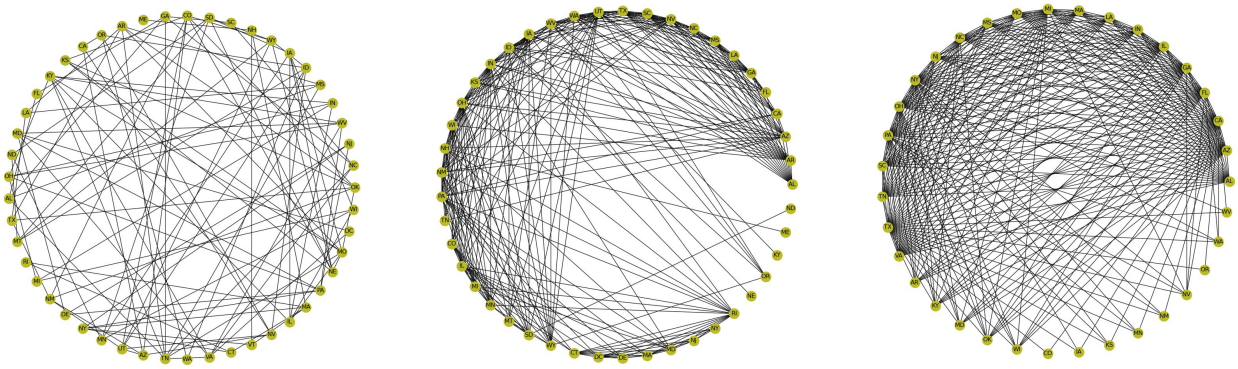
$$a_i = \frac{exp(e_i)}{\sum_{k=1}^{n} exp(e_k)} \tag{A.6}$$

$$C_t = \sum_{i=1}^{n} a_i \times h_i \tag{A.7}$$

Here H is the set of hidden states $\{h_1, h_2, \ldots, h_n\}$, W_1 and d_1 are the weight and deviation of the first layer and W_2 and d_2 are the weight and deviation of the second layer. The context vector $C_t \in R^{N \times T}$ captures global COVID data information and lastly, a linear layer is used to produce the final output results. In this study, a Feed Forward Neural Network (FFNN) was used instead of a scoring function to determine the weight of each hidden state resulting from the GRU.

A.3.3 Correlation

Correlation is used to measure the linear relationships between the variables. However, it does not measure the strength of a nonlinear relationship between variables. Fig. A.3a shows the adjacency matrix based on correlation. To connect the states/nodes, we only selected those with high correlation coefficients.



(a) Adjacency Graph (b) Correlation Graph (c) MI Graph

Figure A.4: Connection between states with different association techniques

A.3.4 MI

MI Learned-Miller, 2013 measures the mutual dependence between two random variables. It measures the amount of information that one random variable provides about the other random variable. It can measure both linear and nonlinear relationships between two random variables. MI is derived from the definition of entropy and is defined as:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \tag{A.8}$$

Let X and Y be two random variables over the space $X \times Y$. $I(X;Y)$ is the MI calculated between the two variables, $P(X,Y)$ is their joint distribution and $P(X)$ and $P(Y)$ is the marginal distribution of X and Y, respectively. The MI score is greater than or equal to zero. If it is zero, it means that the two variables are independent, i.e., knowing the value of one variable does not provide any information about the other variable. If the MI score is greater than zero, the two variables are dependent, and the larger the MI score, the stronger the dependence.

Fig. A.3b shows the MI-based matrix. To create edges between the states, we only selected those with high MI scores. Additionally, we rescaled the MI values by dividing each value by its maximum.

For both techniques, we experimented with different thresholds and selected the ones that yielded the best results. We used 0.6 for correlation and 0.85 for MI which preserved the strong connections between states while discarding weaker ones that fell below the threshold. For creating a weighted graph, we assigned the scores as edge weights for connected edges and assigned 0 for disconnected ones. Fig. A.4 shows how the graph changes based on the association techniques.

A.4 Experiments and Results

A.4.1 Data and Code

The weekly dataset used for this study is available on GitHub https://github.com/scalation/data/blob/master/COVID-State/2023-05-02-17-53-19-State_Weekly.csv. It contains 19 columns and 8819 rows corresponding to weekly reporting for each region in the US from April 18^{th}, 2020, to March 11^{th}, 2023. We only consider the data for states and drop other regions. For features, we use $Confirmed$ and $Deaths$ because other features have missing values.

The data was extracted from the COVID-19 Dataset by COVID-19 Data Repository by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University Dong et al., 2020 (JHU) https://github.com/CSSEGISandData/COVID-19. Daily data were cumulative and produced some negative values when converted to non-cumulative, for instance, Maryland's dataset showed 1140 deaths on 04/30/2020 and 1080 on 05/01/2020 which resulted in -60 deaths on May 1^{st}, 2020. To resolve this issue, we opted to aggregate the data on a weekly basis instead. Once converted, we removed the initial missing values and divided the dataset into an 80/20 split using the PyGT Temporal Signal Train-Test Split for training and testing, which returns train and test data iterators. We employed the sliding window approach for model training, where each window consisted of 114 weeks in the training set and forecasted 4 weeks in the testing set. We then shifted the window by 1 week and repeated the process until the end of the dataset. Moreover, for CDC comparison, the data was extracted from https://covid.cdc.gov/COVID-DATA-TRACKER/?submenu-select=national-lab#datatracker-home. Our code is available on: https://www.github.com/Subasranaa/COVID-19-A3T-GCN2.

A.4.2 Implementation details

We implemented our model in Python using PyTorch Paszke et al., 2019 and PyGT. The hyperparameters shown in Table A.1 are fine-tuned using grid search. The activation function is ReLU, 100 epochs, and a 1e-4 learning rate using the ADAM optimizer. We used the mean squared error (MSE) as the loss function. The data were standardized (subtracting the mean and dividing by the standard deviation) and transformed back to the original data scale for evaluation. The model was evaluated on the symmetric mean absolute percentage error (sMAPE) and the mean absolute error (MAE). sMAPE measures the absolute difference between the actual and forecast values normalized by absolute values of both Smyl et al., 2018. It is a bounded metric where 0% indicates a perfect model with no errors, while 200% indicates an erroneous model of opposite actual and forecast values Chicco et al., 2021. The sMAPE is calculated using (8)

$$\text{sMAPE} = \frac{200}{n} \sum_{t=1}^{n} \frac{|y_t - \hat{y}_t|}{|y_t| + |\hat{y}_t|} \tag{A.9}$$

where y_t is the true value, and $\hat{y}_t$ is the forecast value. MAE measures the mean absolute error between the actual and forecast values.

Table A.1: Optimal hyperparameters for A3T-GCN and all the baselines. A3T-GCN, LSTM, GRU, and FFNN hyperparameters are listed in the sequence of [past values, batch size, hidden dimensions, learning rate]. FFNN utilized ELU and Tanh activation functions and four linear layers.

	Virginia	Georgia	Illinois	Pennsylvania	Kentucky
A3T-GCN	[6,16,4,0.0001]	[6,16,2,0.0001]	[6,16,32,0.0001]	[6,16,2,0.0001]	[6,16,16,0.0001]
SARIMA	$(6,0,1)\times(0,1,1)_{10}$	$(3,1,2)\times(1,0,1)_{10}$	$(0,1,0)\times(1,1,1)_{10}$	$(4,0,0)\times(5,1,1)_{10}$	$(3,1,1)\times(1,1,1)_{10}$
LSTM	[6,16,256,0.0006]	[6,16,32,0.0006]	[6,16,128,0.0006]	[6,16,256,0.0006]	[6,16,512,0.0006]
GRU	[6,16,128,0.0007]	[6,16,512,0.0007]	[6,16,64,0.0007]	[6,16,256,0.0007]	[6,16,128,0.0007]
FFNN	[4,16,256,0.0004]	[4,16,32,0.0004]	[4,16,32,0.0004]	[4,16,16,0.0004]	[4,16,512,0.0004]
	Maryland	Massachusetts	Minnesota	Ohio	Washington
A3T-GCN	[6,16,8,0.0001]	[6,16,32,0.0001]	[6,4,32,0.0001]	[6,16,16,0.0001]	[6,16,32,0.0001]
SARIMA	$(0,1,2)\times(1,0,1)_{10}$	$(1,1,0)\times(0,0,2)_{10}$	$(1,1,1)\times(0,0,1)_{10}$	$(2,0,0)\times(0,1,2)_{10}$	$(4,0,1)\times(0,0,1)_{10}$
LSTM	[6,16,8,0.0006]	[6,16,128,0.0006]	[6,16,64,0.0006]	[6,16,256,0.0006]	[6,16,512,0.0006]
GRU	[6,16,512,0.0007]	[6,16,32,0.0007]	[6,16,4,0.0007]	[6,16,512,0.0007]	[6,16,256,0.0007]
FFNN	[4,16,16,0.0004]	[4,16,32,0.0004]	[4,16,32,0.0004]	[4,16,16,0.0004]	[4,16,16,0.0004]

Statistical and ML Baselines.

To compare our A3T-GCN model, we employed several baselines. Auto-Regressive(1) (AR) model is sufficient when the future is mainly dependent only on the most recent value. SARIMA is a time series forecasting model that combines AR, integrated (I), and moving average (MA) components to capture the patterns and seasonality in data. LSTM is designed to process and retain information over long sequences, enabling it to capture and learn from long-term dependencies in data. It uses a gating mechanism to selectively remember or forget information based on relevance. GRU is similar to LSTM but has fewer gates, making it faster to compute and easier to train. FFNN is where information flows in one direction, from the input layer through hidden layer(s) to the output layer, without any loops. We experimented with these models using various hyperparameters as shown in Table A.1.

A.4.3 Results Analysis

Comparison of A3T-GCN Models using Adjacent, Correlation, and MI Matrices.

We implemented five A3T-GCN models using Adjacent, Correlation, MI, Correlation with adjacent (Corr_Adj), and MI with adjacent (MI_Adj) matrices. The reason behind this is each state was showing different trends when they were connected to different states based on their relation. We are using the adjacent-based A3T-GCN model as the baseline here to compare it with other association technique-based models. Tables A.2 and A.3 show sMAPE and MAE results respectively, of 1 week, 2 weeks, 3 weeks, and 4 weeks ahead forecast for our best models and the baseline in this case.

Virginia, Ohio, Pennsylvania, and Washington demonstrate improved outcomes when using the MI-based A3T-GCN model, suggesting that quantifying the interdependence among these states through MI yields favorable results. The MI_Adj-based model gives us the best results for Georgia, Kentucky, Massachusetts, and Minnesota indicating that the involvement of neighboring states is significant for these

Table A.2: sMAPE results for 1-week, 2-weeks, 3-weeks, and 4-weeks ahead forecast for the states using Adjacent states-based A3T-GCN (Adjacent) and Correlation/MI-based A3T-GCN models (A3T-GCN). The lower the sMAPEs, the better the forecasting performance.

	Virginia		Georgia		Illinois		Pennsylvania		Kentucky	
Weeks	Adjacent	A3T-GCN	Adjacent	A3T-GCN	Adjacent	A3T-GCN	Adjacent	A3T-GCN	Adjacent	A3T-GCN
1	39.96	13.02	22.82	15.07	23.12	11.38	88.93	17.14	20.74	14.28
2	45.36	12.58	27.50	13.77	21.57	9.37	90.10	28.46	19.14	12.37
3	48.41	11.14	33.24	17.77	16.98	10.81	91.32	15.89	16.53	11.23
4	53.60	13.55	22.27	26.38	19.93	14.66	95.40	23.07	14.75	11.83
Average	18.36	**12.57**	26.46	**18.25**	20.40	**11.56**	47.24	**21.14**	17.79	**12.43**
	Maryland		Massachusetts		Minnesota		Ohio		Washington	
Weeks	Adjacent	A3T-GCN	Adjacent	A3T-GCN	Adjacent	A3T-GCN	Adjacent	A3T-GCN	Adjacent	A3T-GCN
1	23.47	14.11	15.44	13.65	18.38	12.31	16.16	14.81	28.27	12.90
2	18.87	14.34	17.16	13.91	14.10	12.04	18.16	6.52	27.31	13.95
3	14.45	18.04	18.09	13.33	21.43	14.35	16.26	7.28	30.05	13.90
4	15.83	23.34	20.84	13.45	19.45	13.94	16.59	14.63	31.79	12.77
Average	18.16	**17.46**	17.88	**13.58**	18.34	**13.16**	16.79	**10.81**	29.35	**13.38**

particular states. For instance, in the case of Georgia, 21 more states got added along with adjacent states using MI which improved the predictions, showing a strong impact of linear and nonlinear dependencies with other states.

In the case of Maryland and Illinois, the Corr_Adj-based model shows superior performance than all the other models indicating that correlation plays an important role in them. Here adding correlated states led to superior results instead of just using adjacent states, our models give a slightly higher MAE score for Maryland even though it has a lower sMAPE score when we compare it with the adjacent-based model. Moreover, we also get competitive scores for Arizona, Iowa, Michigan, New York, Texas, and Indiana with sMAPEs as 25.52, 28.74, 26.87, 26.90, 25.25, and 27.53, respectively by using MI. Overall, using these association techniques over using only adjacent states shows significantly better performance in terms of forecasting.

Baselines Comparison.

In this work, we compare our best-performing A3T-GCN models against the baselines. Fig. A.5 displays the average results of sMAPE and MAE for our best A3T-GCN models and the baselines across the states. A3T-GCN achieves a 52.96% reduction in sMAPE and a 46.84% reduction in MAE compared to AR and a 37.99% reduction in sMAPE and a 36.32% reduction in MAE compared to SARIMA. Although LSTM performs slightly better than A3T-GCN for Maryland, when considering the average results across all states, A3T-GCN demonstrates a 15.86% lower sMAPE and a 16.26% lower MAE. Compared to GRU, A3T-GCN achieves a 22.55% lower sMAPE and a 26.72% lower MAE, despite GRU outperforming in the case of Massachusetts and Minnesota (for MAE only). Finally, when compared to FFNN, A3T-GCN shows a 35.03% lower sMAPE and a 38.51% lower MAE.

Table A.3: MAE results for 1-week, 2-weeks, 3-weeks, and 4-weeks ahead forecast for the states using Adjacent states-based A3T-GCN (Adjacent) and Correlation/MI-based A3T-GCN models (A3T-GCN). The lower the MAEs, the better the forecasting performance.

	Virginia		Georgia		Illinois		Pennsylvania		Kentucky	
Weeks	Adjacent	A3T-GCN	Adjacent	A3T-GCN	Adjacent	A3T-GCN	Adjacent	A3T-GCN	Adjacent	A3T-GCN
1	32.76	12.36	27.10	17.38	18.55	9.11	46.32	19.92	12.94	10.35
2	36.37	11.84	34.41	15.76	16.05	7.27	47.59	30.70	11.78	8.65
3	38.81	10.61	43.93	20.69	12.78	8.52	48.63	18.90	10.91	7.86
4	42.57	12.81	26.31	32.69	15.49	11.64	52.66	29.60	10.68	8.27
Average	37.63	**11.90**	32.94	**21.63**	15.72	**9.13**	48.80	**24.78**	11.58	**8.78**
	Maryland		Massachusetts		Minnesota		Ohio		Washington	
Weeks	Adjacent	A3T-GCN	Adjacent	A3T-GCN	Adjacent	A3T-GCN	Adjacent	A3T-GCN	Adjacent	A3T-GCN
1	9.53	5.75	14.46	8.70	7.70	5.99	13.99	11.34	19.11	8.62
2	7.75	5.90	15.65	8.87	6.14	5.83	15.63	5.14	18.33	9.32
3	5.81	7.90	15.90	8.47	9.09	7.00	14.20	5.85	20.69	9.22
4	6.62	10.71	19.25	8.56	9.20	6.83	14.48	12.14	22.20	8.52
Average	**7.43**	7.57	16.32	**8.65**	8.03	**6.41**	14.57	**8.62**	20.08	**8.92**

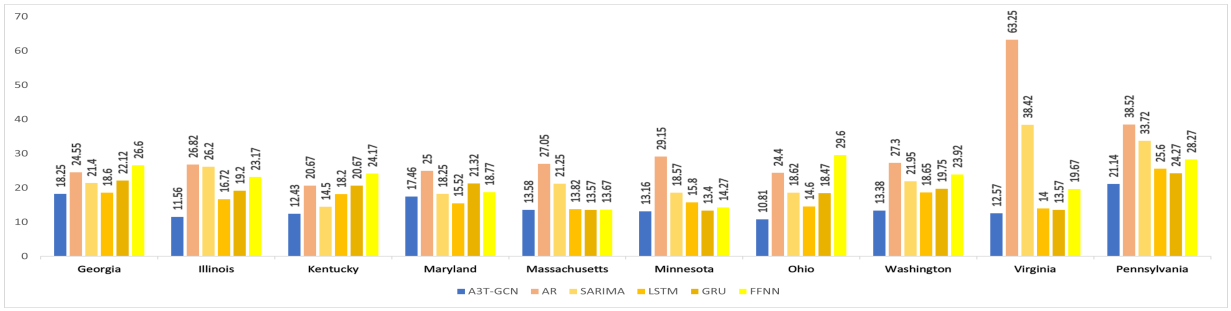
This is likely due to the model's use of Attention, GCN, and GRU, which can capture non-linear dependencies and temporal relationships between distant data points. In contrast, AR and SARIMA may not be able to capture these non-linear dependencies and relationships, leading to lower performance. The findings also highlight that relying solely on historical data from a single state may not be sufficient for accurately predicting deaths, as the interactions and relationships between different states can impact the results. Therefore, it is crucial to consider these relationships in the model, which A3T-GCN can capture. This becomes particularly significant when dealing with time series data. While LSTM, FFNN, and GRU performed well, their limitations in capturing these relationships might have hindered their performance for some states.

CDC analysis.

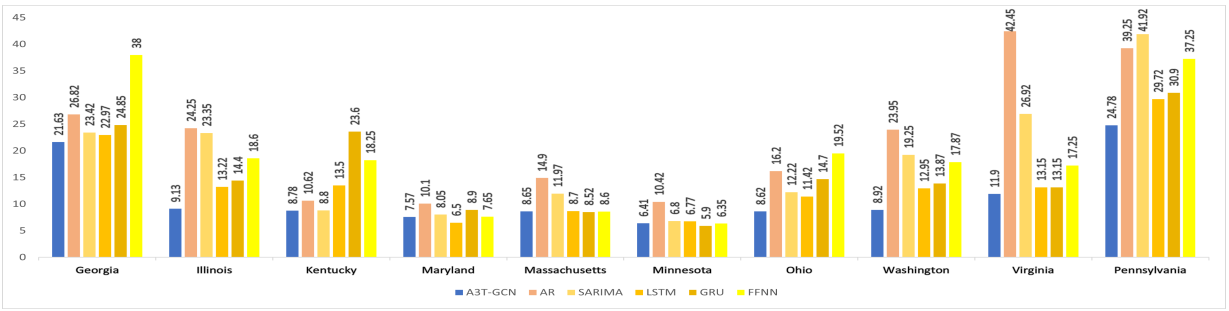
We compared our best A3T-GCN models with the models submitted to CDC for the states. The evaluation of these models was based on the sMAPE definition, and Table A.4 shows the average sMAPE score for all states, excluding Virginia and Pennsylvania because the data for those two states was unavailable. We limited our selection to only those models with matching forecasting dates. A3T-GCN ranked second overall, outperforming most other models. In addition, our model demonstrated superior performance for Ohio, as depicted in Fig. A.6.

A.5 Conclusion and Future Work

Our work highlights the importance of accurately capturing both spatial and temporal dependencies in COVID-19 forecasting. The MI and correlation-based A3T-GCN model proposed in this work addresses



(a) sMAPE results



(b) MAE results

Figure A.5: Average of prediction results of the best A3T-GCN and the baselines

this challenge by leveraging the attention mechanism to focus on the most informative data that includes both recent and distant data points to capture temporal dependencies and correlation and MI to capture the spatial dependencies effectively. We found utilizing MI to be the most effective technique to create the graph that the model uses for the US state's weekly mortality predictions. Our model outperforms the existing baselines and fairly competes with the models adopted by CDC.

Future work can explore various association techniques for COVID-19 forecasting and extend the study to other diseases with complex spatiotemporal dynamics. Further research can focus on separate state-specific models, considering inherent differences and inter-state impacts. Overall, this work has the potential to provide accurate COVID-19 forecasting for spatiotemporal data, aiding in implementing effective measures to control the virus's spread.

Table A.4: Comparison of A3T-GCN with the models forecast submitted to CDC using sMAPE

Model	sMAPE	Model Method
JHU-IDD "JHU-IDD by Johns Hopkins University, Infectious Disease Dynamic Lab", n.d.	82.67	Metapopulation Susceptible-Exposed-Infected-Recovered (SEIR) model
LUcompUncertLab "Computational Uncertainty Lab by Prof. Thomas McAndrew", n.d.	66.33	A Bayesian Vector Auto Regression model
Microsoft "Microsoft by Microsoft AI", n.d.	53.1	SEIR model on a spatiotemporal network
USC Srivastava, 2022	48.82	Discrete heterogeneous rate model
ESG "ESG by Robert Walraven", n.d.	36.04	Fitting reported data to multiple skewed gaussian distributions
Masaryk	35.59	ARIMA models with outlier detection applied to transformed series
UCSD-NEU "UCSD-NEU University of California, San Diego and Northeastern University", n.d.	26.54	Age-structured metapopulation model with deep learning
JHU-APL "JHU-APL by Johns Hopkins University, Applied Physics Lab", n.d.	25.73	Metapopulation SEIR model
Karlen "Karlen by Karlen Working Group", n.d.	23.33	Discrete time difference equations
MOBS "MOBS by Northeastern University, Laboratory for the Modeling of Biological and Socio-technical Systems", n.d.	21.3	Metapopulation, age-structured Susceptible-Latent-Infected-Removed (SLIR) model
GT-DeepCOVID "GT-DeepCOVID by Georgia Institute of Technology, College of Computing", n.d.	20.22	Deep Learning
BPagano "BPagano by Bob Pagano", n.d.	15.97	Susceptible-Infected-Recovered (SIR) model
Ensemble Ray et al., 2020	14.28	Combination of several forecasts
A3T-GCN	**13.82**	Our Model
Columbia "Columbia by Columbia University", n.d.	10.51	Metapopulation SEIR model

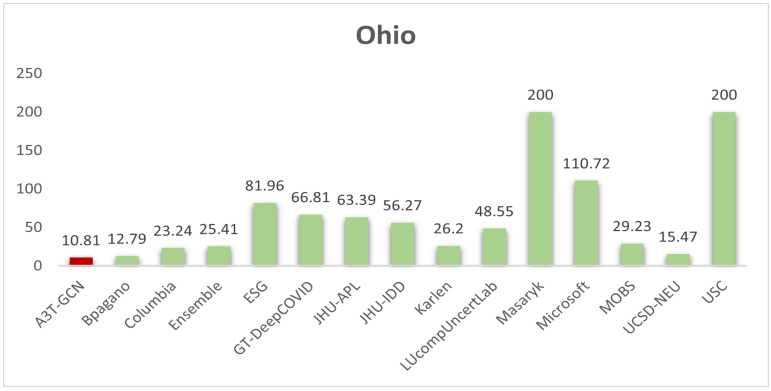


Figure A.6: A3T-GCN comparison with CDC models for Ohio using sMAPE

Appendix B

Statistical Transformations and Normalization in Transformers Supplement

B.1 Preprocessing Methods

The definitions of the transformation methods utilized in Chapter 4 of the dissertation are outlined below.

B.1.1 Standard Scaler

StandardScaler is one of the most widely used scaling techniques in machine learning. It transforms each feature by subtracting its mean and dividing by its standard deviation, effectively centering the data around zero with a unit variance. In mathematical terms, for each value y, the transformation is given by:

$$y_{scaled} = \frac{y - \mu}{\sigma} \tag{B.1}$$

where y is the original value, μ is the mean, and σ is the standard deviation of the feature. The result, y_{scaled}, is the standardized value. The mean and standard deviation are obtained from only the training set instead of the entire dataset to avoid data leakage during training Kuhn and Johnson, 2019.

However, this method assumes that the data follows a roughly normal (Gaussian) distribution. If the data deviates from normality or contains significant outliers, standardization might not produce reliable results Pedregosa et al., 2011. Inverse scaling is obtained by adding the mean back and multiplying by the standard deviation.

B.1.2 Logip

The $log1p$ function adds one to each value and then takes the natural logarithm of the result. In other words, for a given number y,

$$log1p(y) = ln(1 + y) \tag{B.2}$$

This approach is beneficial for values close to zero because adding one helps avoid numerical problems that can arise when directly applying the natural logarithm to very small or negative numbers. If you pass in real (i.e., non-complex) numbers to the function, it always tries to return a real result. However, if $(1 + y)$ is not a real number or infinity, $log1p$ outputs NaN (not a number) Harris et al., 2020.

For the inverse transformation of $log1p$, we use $expm1$, which calculates $e^y - 1$. This function reverses the effect of $log1p$, meaning that if you apply $log1p$ to a number and then use $expm1$ on the result, you get the original number.

B.1.3 Square Root

Another way to handle skewed data is the power transformation, Square Root, where you simply take the square root of the variable y Miller, 2022. Like the logarithmic transformation, this also helps shrink large values and makes the distribution more symmetric. To reverse the transformation, you take the square of the transformed values to get the original scale. Unlike the log transformation, the square root transformation can handle zeros but not negative values.

B.1.4 Box-Cox

The Box-Cox transformation, a combination of both logarithm and power transformation, was proposed by George Box and David Cox Box and Cox, 1964. It is used to stabilize variance, reduce skewness in the data, and achieve normality when desired. The Box-Cox transformation is defined as:

$$y_{\text{trans}} = \begin{cases} \dfrac{y^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0, \\ \log y & \text{if } \lambda = 0. \end{cases} \tag{B.3}$$

where y is the input variable and λ is the transformation parameter. An optimal parameter λ is learned from the input data, which determines the specific power function applied to y; for example, λ = 1 gives the original value, λ = -1 gives the reciprocal transformation, and λ = 2 corresponds to a square transformation. The log transformation is applied when λ is equal to zero. The optimal parameter λ is chosen via the maximum likelihood estimation.

The inverse of the Box-Cox transformation is defined as follows Pedregosa et al., 2011:

$$y = \begin{cases} \exp\left(y_{\text{trans}}\right) & \text{if } \lambda = 0, \\ \left(y_{\text{trans}}\lambda + 1\right)^{\frac{1}{\lambda}} & \text{if } \lambda \neq 0. \end{cases} \tag{B.4}$$

where y_{trans} is the transformed variable. To obtain the original data, the inverse transformation is applied using the fitted lambdas. It is important to note that the Box-Cox transformation can only be applied when all data values are positive. If your data includes negative values, you can either add a constant to shift the distribution into the positive range or opt for the Yeo-Johnson transformation, which accommodates negative values.

B.1.5 Yeo-Johnson

The Yeo-Johnson transformation, introduced by Yeo and Johnson Yeo and Johnson, 2000, is an extension of the Box-Cox transformation. It is defined as:

$$y_{\text{trans}} = \begin{cases} \dfrac{(y+1)^{\lambda}-1}{\lambda} & \text{if } y \geq 0 \text{ and } \lambda \neq 0, \\ \log(y+1) & \text{if } y \geq 0 \text{ and } \lambda = 0, \\ -\dfrac{(-y+1)^{2-\lambda}-1}{2-\lambda} & \text{if } y < 0 \text{ and } \lambda \neq 2, \\ -\log(-y+1) & \text{if } y < 0, \lambda = 2. \end{cases} \tag{B.5}$$

Here, y is the input variable and λ is the transformation parameter. This transformation can be applied without any restrictions on y, which means that it is designed for variables that include zeros, negatives, and positive values. For positive values, the Yeo-Johnson transformation functions similarly to the Box-Cox transformation but is applied to $y+1$. This shift ensures that the transformation is well-defined (since logarithms require positive arguments) and helps in handling zeros. When the values are strictly negative, it is also equivalent to Box-Cox, but this time it is applied to $-y+1$ with a power of $2-\lambda$. If the data has positive and negative values, it combines the two approaches by using different powers for the positive and negative parts of the data Weisberg, 2001.

B.1.6 Differencing

Transformations like logarithms can help make the variance of a time series more stable. Differencing, on the other hand, helps stabilize the mean by removing shifts in the data over time. This can reduce or even eliminate trends and seasonal patterns, making the non-stationary time series stationary Hyndman, 2018. In this paper, we explore two different types of differencing techniques: First, and Seasonal.

First Differencing. For First differencing, compute the difference between consecutive time steps by subtracting each previous time step from the current one Hyndman, 2018. For reverse differencing, combine the initial seed value from the original input with the predicted differences and perform a cumulative sum to rebuild the original series, dropping the seed afterward.

Seasonal Differencing. For Seasonal differencing, subtract each data point from the data point that is a fixed number (`seasonal` in the code) of time steps behind it Hyndman, 2018. For reverse differencing, combine the `seasonal` seed values with the predictions and add back the value from the corresponding previous `season` at each step, finally removing the `seasonal` seeds to reconstruct the original series.

B.2 Statistical Tests

We apply several statistical tests to evaluate the properties of the COVID-19, ILI, and RSV target variables.

B.2.1 Stationarity Test

To assess stationarity, we use both the Augmented Dickey–Fuller (ADF) and the Kwiatkowski–Phillips–Schmidt–Shin (KPSS) tests. The tests results are provided in the table B.1. The KPSS test is computed for both `regression=c` and `ct`, where `c` specifies a level-stationary null (stationarity around a constant mean) and `ct` specifies a trend-stationary null (stationarity around a deterministic linear trend).

For the COVID-19 deaths series, the ADF test fails to reject the null of a unit root, which is consistent with non-stationary, unit-root behaviour, whereas both KPSS specifications (`regression=c` and `ct`) fail to reject stationarity. These mixed results place the series near the boundary between stationary and non-stationary behaviour. Given the pandemic dynamics and pronounced trends in COVID-19 data, we treat this series as effectively non-stationary in practice. For the ILITOTAL series, the ADF test rejects the null of a unit root, while the KPSS test with `regression=c` rejects level stationarity and the KPSS test with `regression=ct` does not reject trend stationarity. Taken together, these results suggest that ILITOTAL is better characterised as trend-stationary rather than level-stationary. For the RSV detections series, all three tests (ADF and both KPSS specifications) are consistent with stationarity, indicating no evidence of a unit root or non-stationary behaviour.

B.2.2 Normality and Heteroskedasticity Tests

We use the Shapiro–Wilk test for normality and Breusch–Pagan test for heteroskedasticity. The test results are provided in the table B.2. All three time series fail the normality test, indicating that none of the datasets follow a normal distribution. The COVID-19 series shows no evidence of heteroscedasticity under the Breusch–Pagan test, suggesting homoscedastic residual behaviour. In contrast, both the RSV and ILI series reject the null hypothesis of homoscedasticity and are therefore classified as heteroskedastic.

B.3 Full Implementation Details

B.3.1 Transformers

For the PatchTST model, we implemented it using the original code provided in the paper to replicate the reported results. The code is available on their GitHub[1], and we used the exact hyperparameters specified in the Illness script[2]. The results reported in PatchTST Nie, 2022 were generated by dropping the last test samples that did not meet the batch size (i.e., with `drop_last` set to `True`). In contrast,

[1] https://github.com/yuqinie98/PatchTST/tree/main/PatchTST_supervised
[2] https://github.com/yuqinie98/PatchTST/blob/main/PatchTST_supervised/scripts/PatchTST/illness.sh

Table B.1: Stationarity test results for the COVID-19, ILI and RSV series.

Series	Measure	ADF	KPSS (reg = c)	KPSS (reg = ct)
COVID-19	**Statistic**	-2.6872	0.0781	0.0742
	p-value	0.0763	0.1	0.1
	Critical values	1%: -3.4918 5%: -2.8884 10%: -2.5811	10%: 0.347 5%: 0.463 2.5%: 0.574 1%: 0.739	10%: 0.119 5%: 0.146 2.5%: 0.176 1%: 0.216
	Decision	Non-stationary	Stationary	Stationary
ILI	**Statistic**	-6.1612	1.7704	0.06557
	p-value	7.16×10^{-8}	0.01	0.1
	Critical values	1%: -3.4372 5%: -2.8645 10%: -2.5683	10%: 0.347 5%: 0.463 2.5%: 0.574 1%: 0.739	10%: 0.119 5%: 0.146 2.5%: 0.176 1%: 0.216
	Decision	Stationary	Non-stationary	Stationary
RSV	**Statistic**	-8.1931	0.105	0.03193
	p-value	7.57×10^{-13}	0.1	0.1
	Critical values	1%: -3.4431 5%: -2.8671 10%: -2.5697	10%: 0.347 5%: 0.463 2.5%: 0.574 1%: 0.739	10%: 0.119 5%: 0.146 2.5%: 0.176 1%: 0.216
	Decision	Stationary	Stationary	Stationary

while reproducing these results under the same experimental conditions, we set `drop_last` to `False` and present our results accordingly. The original paper's results and our reproduced results are shown in Table B.3. The table shows the Mean Absolute Errors (MAEs) on the normalized multivariate time series for the Illness dataset.

For the TimeXer and iTransformer models, we used the code from TimeXer's GitHub repository[3]. For all other transformer-based models that do not include experiments on the ILI, RSV, or COVID-19 datasets, we tuned the hyperparameters using Ray Tune Liaw et al., 2018. All experiments were tuned with `StandardScaler` as the preprocessing technique to ensure a fair comparison. The full set of hyperparameters is provided in the Tables B.5 and B.6. Other hyperparameters, such as epochs and patience,

[3] https://github.com/thuml/TimeXer

Table B.2: Normality and Heteroskedasticity test results for the COVID-19, ILI and RSV series.

Series	Measure	Shapiro–Wilk	Breusch–Pagan
COVID-19	**Statistic**	0.9398	0.0603
	p-value	5.60×10^{-5}	0.806
	Decision	NOT normally distributed	Homoskedastic
ILI	**Statistic**	0.6990	97.9633
	p-value	2.40×10^{-38}	4.26×10^{-23}
	Decision	NOT normally distributed	Heteroskedastic
RSV	**Statistic**	0.7915	9.9966
	p-value	3.01×10^{-25}	0.001568
	Decision	NOT normally distributed	Heteroskedastic

Table B.3: PatchTST reproduced results for the Illness dataset. Values shown are MAEs computed on the normalized multivariate time series.

Weeks	Original Paper	Reproduced
24	0.814	0.913
36	0.834	0.890
48	0.854	0.916
60	0.862	0.875
Avg	0.841	0.895

that are not listed in the hyperparameter table remain the same as in the original papers Y. Liu et al., 2023; Nie, 2022; Y. Wang et al., 2024.

B.3.2 Preprocessing Techniques

The COVID-19 dataset contains zero values; we added an epsilon of $1e^{-6}$ to all entries before applying Box–Cox, as the transformation only supports strictly positive values. Both classic Box–Cox and multivariate Box–Cox+ experiments were performed on this adjusted dataset.

For preprocessing, we used the NumPy library to implement Log1p, Square Root, and Differencing transformations. The Box–Cox and Yeo–Johnson transformations were implemented using Scikit–Learn's `PowerTransformer`, which estimates the transformation parameters via Maximum Likelihood Estimation (MLE) and optimizes them using Brent's method. For the implementation of multivariate Box-Cox, we used the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm as an optimization method. The estimated λ values for the seven ILI features were:

- Box–Cox: [-0.28602651, -0.503773, 0.29751579, 0.14257209, 0.19154931, 1.18640383, 0.8973477],
- Yeo–Johnson: [-1.05905535, -1.35406619, 0.29698175, 0.14196202, 0.1913529, 1.18688668, 0.89734852],
- Multivariate Box–Cox: [9.40573957e-01, 6.49478498e-06, -1.88604996e-01, -4.64516897e-03, 2.14544321e-06, 7.31035204e-01, -5.70633053e-06] ,

where $ILITOTAL$ corresponds to index 4.

For the COVID-19 dataset, the estimated λ values were:

- Box–Cox: [0.4729845, 0.13878323, 0.13108939, 0.57212189, -0.00723948, -0.00893658, 0.38290023],
- Yeo–Johnson: [0.47470482, 0.26700578, 0.23867696, 0.69053757, -0.01413478, -0.02115433, 0.3829135],
- Multivariate Box–Cox: [0.59915354, 0.02555853, 0.02416443, 0.86095119, -0.21504889, -0.26126894, 0.38628694]

where new_deaths corresponds to index 0.

For the RSV dataset, the estimated λ values were (Only applied to RSV variables):

- Box–Cox: [-0.00771988, 0.03359194, 0.7787307, 0.77655533],
- Yeo–Johnson: [-0.01010536, 0.03347787, 0.77877008, 0.77656596],
- Multivariate Box–Cox: [0.05281147, 0.08573934, 1.04814387, 1.61049883]

where $RSV\,Detections$ corresponds to index 0.

The seasonal differencing procedure follows Hyndman, 2018; Nau, 2016. For the ILI dataset, which exhibits clear annual seasonality, we experimented with seasonal lags of 13, 26, and 52 weeks and found that a lag of 26 produced the lowest forecasting errors for the transformer-based models. RSV also displays a well-defined yearly cycle, so we used a 52-week seasonal lag. In contrast, the COVID-19 dataset does not exhibit a stable annual seasonal pattern; instead, it consists of a few epidemic waves of varying duration. For this dataset, we therefore treated the seasonal differencing lag as a tuning parameter rather than a fixed seasonal period. A lag of 10 weeks yielded the smallest sMAPEs across models, so we adopted it as the differencing lag for COVID-19. Plots of all three datasets are shown in the Figure B.1. For each prediction horizon, we experimented with both first differencing and seasonal differencing and report the configuration that achieved the lowest sMAPE. As observed in the experiments, short-term horizons generally benefited from first differencing, while longer horizons benefited from seasonal differencing. Table B.4 summarizes the differencing choices across models and horizons.

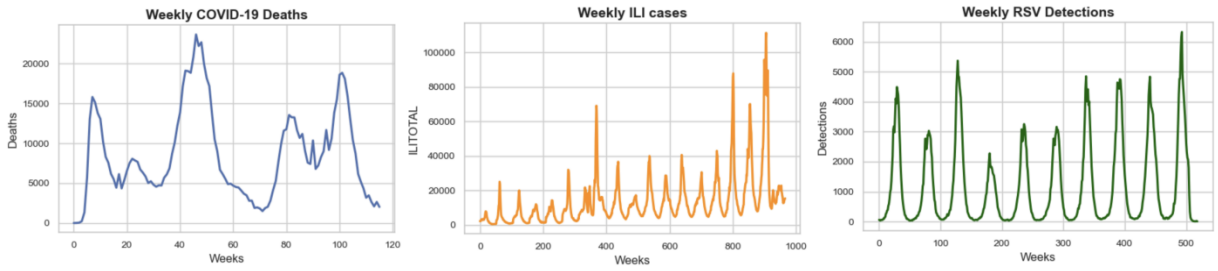


Figure B.1: The figure displays weekly COVID-19 deaths, ILI cases, and RSV detections. Unlike ILI and RSV, which exhibit clear and regular seasonal patterns, the COVID-19 series shows irregular epidemic waves rather than true seasonality.

B.3.3 Evaluation

To evaluate the model, we use two main metrics: the symmetric mean absolute percentage error (sMAPE) and the mean absolute error (MAE). sMAPE measures the absolute difference between the actual and forecasted values, normalized by the absolute values of both Miller, 2022. The sMAPE is a bounded metric where 0% indicates that the forecast is perfect, with no discrepancies between predicted and actual values, whereas a value of 200% implies that the predictions are completely off, with the forecasted values having opposite signs to the true values Chicco et al., 2021. The sMAPE is calculated using equation B.6

$$\text{sMAPE} = \frac{200}{n} \sum_{t=1}^{n} \frac{|y_t - \hat{y}_t|}{|y_t| + |\hat{y}_t|} \tag{B.6}$$

where y_t is the true value, and $\hat{y}_t$ is the forecasted value.

MAE measures the mean of the absolute differences between the predicted values and the actual values. The MAE is an unbounded metric, and it is calculated using equation B.7

$$\text{MAE} = \frac{1}{n} \sum_{t=1}^{n} |y_t - \hat{y}_t| \tag{B.7}$$

where y_t is the true value, and $\hat{y}_t$ is the forecasted value.

B.3.4 RSV Experiments

We merge the RSV dataset with weather covariates, as environmental factors such as temperature and humidity often exhibit nonlinear relationships with viral transmission. This relationship is also reflected in the correlation matrix between lagged variables and RSV Detections (the target variable), shown in

Table B.4: Type of differencing (first or seasonal) applied across forecasting horizons for the ILI, COVID-19, and RSV datasets. Seasonal differencing uses a lag of 26 for ILI, 10 for COVID-19, and 52 for RSV.

Week(s)	PatchTST	iTransformer	TimeXer
ILI			
6	First	First	First
12	First	First	First
24	Seasonal	Seasonal	First
36	Seasonal	Seasonal	Seasonal
48	First	Seasonal	Seasonal
60	Seasonal	Seasonal	Seasonal
COVID-19			
1	First	First	First
2	First	First	First
3	First	First	First
4	First	First	First
5	First	Seasonal	First
6	Seasonal	Seasonal	First
RSV			
6	First	First	First
12	Seasonal	Seasonal	Seasonal
24	Seasonal	Seasonal	Seasonal
36	Seasonal	Seasonal	Seasonal
48	Seasonal	Seasonal	Seasonal

Figure B.2. The strongest correlations are observed up to lag 2; therefore, lag-1 and lag-2 versions of each weather variable are included in our experiments. We evaluate the models both with and without weather variables and find that the best performance is achieved when incorporating the lagged weather features. The corresponding results are presented in Table B.7.

Next, we examine the distribution of RSV Detections in Figure B.3, which reveals a right-skewed pattern where the median is lower than the mean. This indicates the presence of several weeks with exceptionally high detection counts, likely corresponding to outbreak peaks.

B.3.5 Correlation Heatmaps

Figure B.4 presents the correlation heatmaps for the COVID-19, ILI, and RSV datasets. Higher positive values indicate stronger linear relationships between variables, whereas negative values indicate inverse associations. As expected, each variable shows a value of 1 along the diagonal, representing perfect self-correlation. The RSV dataset exhibits a mix of low to high correlations among its features, while the COVID-19 and ILI datasets display comparatively stronger correlations across variables.

These correlation patterns help explain the performance differences among models: iTransformer, which uses variate-wise attention to capture inter-feature dependencies, achieves the lowest overall sMAPEs —25.90 on ILI and 29.83 on COVID-19—and remains competitive in MAE. Its strong performance on COVID-19 and ILI aligns with the high degree of cross-feature correlation present in these datasets.

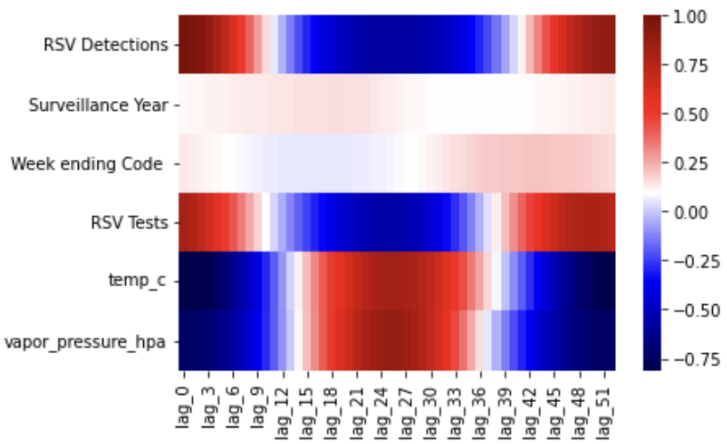


Figure B.2: Correlation between RSV Detections and lagged RSV and weather variables.

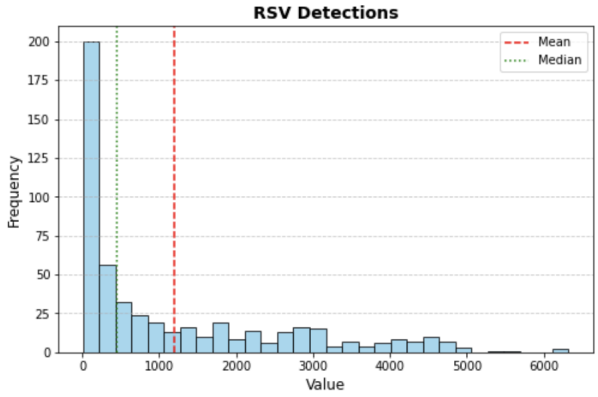


Figure B.3: Distribution of RSV Detections showing right skew, with mean (red) and median (green).

B.3.6 Tuned Hyperparameters for Log1p

In the final comparison between StandardScaler and other transformations, all hyperparameters were initially tuned using the StandardScaler, and the same settings were then applied across all transformations for unbiased comparisons. However, as shown in Table B.8, performance improves further—particularly for the Log1p transformation in the TimeXer model—when hyperparameters are individually optimized for each transformation. The corresponding tuned hyperparameters are listed in Table B.9.

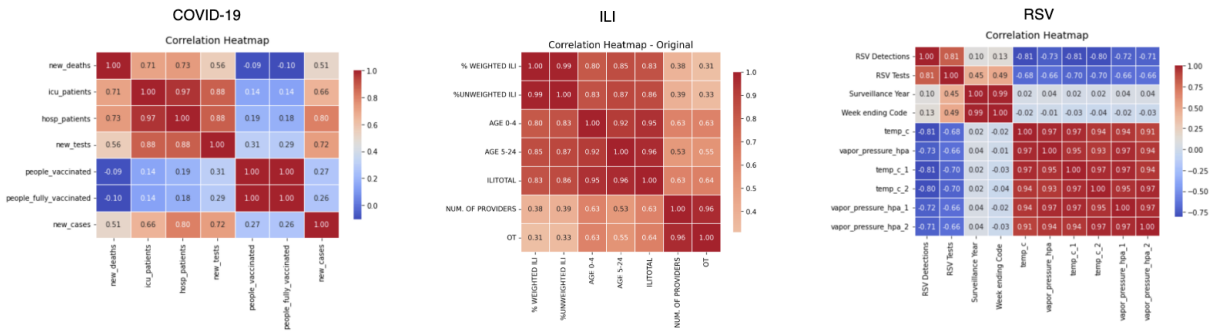


Figure B.4: Correlation heatmaps for COVID-19, ILI, and RSV datasets.

B.4 More Detailed Explanation of Results

B.4.1 Preprocessing techniques

Detailed results for each horizon are reported in Tables B.10, B.11, and B.12. Among the power and log transformations, the best results for $ILITOTAL$ are achieved using the log1p. Logarithmic transformations are more interpretable because changes in the logarithmic scale directly reflect relative (or percentage) changes in the original data Hyndman, 2018. The primary objective of these transformations is to make the variations in the time series more homogeneous. Our dataset exhibits fluctuating magnitudes, small at certain points and larger at others. This means that the data shows structural breaks, which lead to differing mean levels across segments and considerable variance. These structural breaks can be identified by comparing the means of different segments Salles et al., 2019. By applying these transformations, we aim to stabilize the data across its entire range. Figure B.5 illustrates that the high fluctuations are notably diminished in the logarithmically transformed series compared to the square root method. The log1p approach effectively lowers the variance, while the mean becomes nearly constant at the later stage, though not entirely stabilized. Although the square root transformation moderates the fluctuations somewhat, significant variability remains. In contrast, the logarithmic transformation yields a series of more consistent fluctuations, simplifying the modeling process.

Figure B.6 compares the plots of the first-differenced series with and without the log1p transformation. Although both series oscillate around zero, their variance behaviors differ significantly. In particular, the log1p-differenced series performs better than the series that has been only differenced, a finding that is also supported by Hossain et al., 2019; Lütkepohl and Xu, 2012. The log1p-differenced $ILITOTAL$ series exhibits considerable variance in the early years, which then stabilizes during the middle and later periods, as shown in the plots and confirmed using $np.var()$. In contrast, the series that underwent only differencing shows a trend of increasing variance over time. We conducted Levene's test Virtanen et al., 2020 on both series, and according to the p-values, the null hypothesis of equal variances across segments

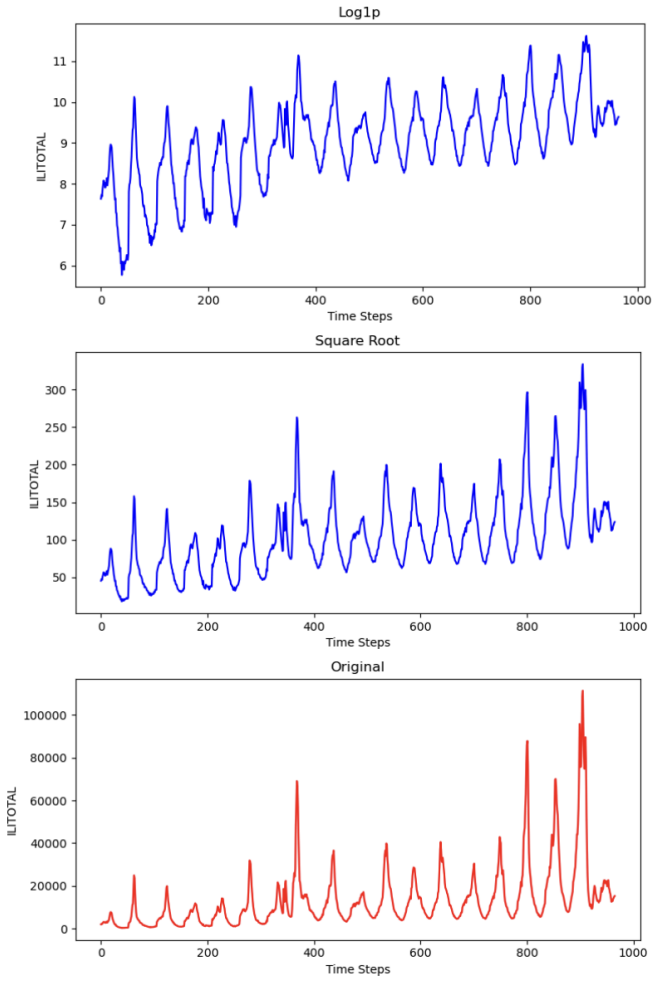


Figure B.5: The above plots display the transformed ILITOTAL series alongside the original data, allowing you to compare the variations before and after the Log1p and Square Root transformations.

is rejected for both transformations. However, the log1p-differenced series displays reduced heteroscedasticity in the later segments ($p \approx 0.000014$), while the only differenced series exhibits even greater variance inconsistency ($p \approx 1.40 \times 10^{-15}$). Although neither method achieves perfect homoscedasticity, log1p-differencing offers superior variance stabilization, making it a more effective transformation for achieving uniform variance over time. That said, additional preprocessing may still be required to fully normalize the variance, particularly in the early data.

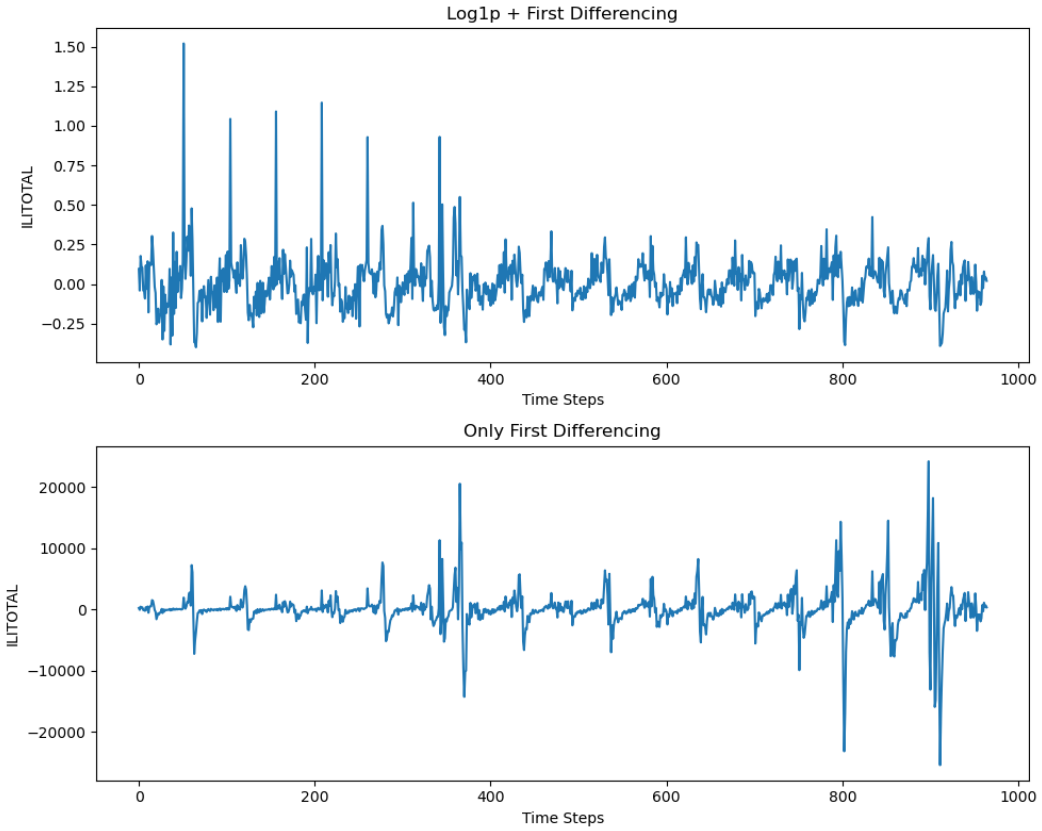


Figure B.6: In the figure above, the top plot shows the ILITOTAL series after applying both a log1p transformation and first differencing, while the bottom plot shows the series after applying only first differencing.

B.5 Instance Normalization Techniques

We experimented with another instance normalization technique in addition to CoIN. We implemented a moving average–based normalization approach in which, for each look-back window, the average of the last h steps is used for normalization. For example, if $i = 1, 2, \ldots, h$, we compute the average of the last i steps at each step, subtract it from the input, and divide by the standard deviation of the look-back window for normalization. If i=5, then we subtract the average of the last 5 steps. For denormalization, we use the final computed average, the mean of the last h steps, which would be the mean of the full look-back window. We tested this approach using both h and $\sqrt{h}$ steps. Table B.13 presents the results for PatchTST, TimeXer, and iTransformer on the ILI and COVID-19 datasets. CoIN consistently outperforms all other techniques, including both moving average-based methods and the default RevIN variants that use mean or last-value normalization.

Table B.5: Hyperparameters across models and preprocessing/normalization experiments. For both datasets, the hyperparameters listed below are for all prediction lengths. Any hyperparameters not listed here are the default ones from the models' scripts.

		ILI			COVID-19		
Experiment	**Hyperparameter**	**PatchTST**	**iTransformer**	**TimeXer**	**PatchTST**	**iTransformer**	**TimeXer**
StandardScaler	Sequence length	104	60	104	8	7	6
	Patch length/Stride	24/2	-	8	6/2	-	6
	Batch size	16	16	32	16	8	8
	d_{model}	16	256	256	64	256	256
	d_{ff}	128	2048	128	2048	128	512
	Learning rate	0.0025	0.00040	0.00059	0.0001	0.00276	0.00101
	Dropout	0.3	0.109	0.149	0.05	0.4848	0.394
	e_layers	3	3	3	2	1	2
	n_heads	4	8	8	8	8	8
Log1p	Sequence length	104	60	104	8	7	12
	Patch length/Stride	24/2	-	8	6/2	-	6
	Batch size	16	16	32	8	8	8
	d_{model}	16	256	256	512	256	256
	d_{ff}	128	2048	128	2048	192	512
	Learning rate	0.0025	0.00040	0.00059	0.0001	0.00276	0.000846
	Dropout	0.3	0.109	0.149	0.05	0.4848	0.0259
	e_layers	3	3	3	3	2	3
	n_heads	4	8	8	8	8	8
Square Root	Sequence length	104	60	104	12	15	7
	Patch length/Stride	24/2	-	8	6/2	-	6
	Batch size	16	16	32	8	16	8
	d_{model}	16	256	256	256	192	192
	d_{ff}	128	2048	128	2048	256	16
	Learning rate	0.0025	0.00040	0.00059	0.0001	0.00276	0.0022
	Dropout	0.3	0.109	0.149	0.05	0.4848	0.2389
	e_layers	3	3	3	3	1	3
	n_heads	4	8	8	8	8	8
Box–Cox	Sequence length	104	60	104	12	9	12
	Patch length/Stride	24/2	-	8	6/2	-	6
	Batch size	16	16	32	8	8	8
	d_{model}	16	256	256	256	256	192
	d_{ff}	128	2048	128	2048	16	16
	Learning rate	0.0025	0.00040	0.00059	0.0001	0.00078	0.00134
	Dropout	0.3	0.109	0.149	0.05	0.4769	0.08717
	e_layers	3	3	3	3	3	2
	n_heads	4	8	8	8	8	8
Yeo–Johnson	Sequence length	104	60	104	12	10	12
	Patch length/Stride	24/2	-	8	6/2	-	6
	Batch size	16	16	32	8	16	8
	d_{model}	16	256	256	256	192	256
	d_{ff}	128	2048	128	2048	256	2048
	Learning rate	0.0025	0.00040	0.00059	0.0001	0.00276	0.00201
	Dropout	0.3	0.109	0.149	0.05	0.4848	0.4062
	e_layers	3	3	3	3	3	3
	n_heads	4	8	8	8	8	8
Multi Box–Cox	Sequence length	104	60	104	12	7	9
	Patch length/Stride	24/2	-	8	6/2	-	6
	Batch size	16	16	32	16	8	16
	d_{model}	16	256	256	256	256	256
	d_{ff}	128	2048	128	2048	128	512
	Learning rate	0.0025	0.00040	0.00059	0.0001	0.00276	0.0031
	Dropout	0.3	0.109	0.149	0.05	0.4848	0.06705
	e_layers	3	3	3	3	1	2
	n_heads	4	8	8	8	8	8
CoIN	Sequence length	104	60	104	7	7	6
	Patch length/Stride	24/2	-	8	6/2	-	6
	Batch size	16	16	32	16	8	8
	d_{model}	16	256	256	64	256	256
	d_{ff}	128	2048	128	2048	128	512
	Learning rate	0.0025	0.00040	0.00059	0.0001	0.00276	0.00101
	Dropout	0.3	0.109	0.149	0.05	0.4848	0.394
	e_layers	3	3	3	2	1	2
	n_heads	4	8	8	8	8	8

Table B.6: Hyperparameters across models and preprocessing experiments for RSV dataset. The hyperparameters listed below are for all prediction lengths. Any hyperparameters not listed here are the default ones from the models' scripts.

		RSV		
Experiment	**Hyperparameter**	**PatchTST**	**iTransformer**	**TimeXer**
All Transformations	Sequence length	96	96	48
	Patch length/Stride	16/2	–	24
	Batch size	16	8	8
	d_{model}	256	512	192
	d_{ff}	512	512	512
	Learning rate	0.00043	0.0004934	0.000272
	Dropout	0.3697	0.1	0.028033
	e_layers	2	4	2
	n_heads	4	8	8

Table B.7: Performance comparison of RSV forecasting under different settings and transformations.

Weeks	**RSV + Lagged Weather**		**RSV + Weather (no lags)**		**RSV only**	
	StandardScaler	**Log1p**	**StandardScaler**	**Log1p**	**StandardScaler**	**Log1p**
			sMAPE			
6	40.92	29.07	50.51	25.98	37.98	25.10
12	32.36	28.02	56.06	27.39	70.73	26.45
24	42.61	24.39	55.12	24.14	62.37	23.65
36	44.07	25.79	63.40	22.17	73.42	24.48
48	54.25	23.16	42.83	22.37	74.16	22.21
Average	**42.84**	**26.09**	**53.58**	**24.41**	**63.73**	**24.38**
			MAE			
6	267.75	280.52	299.92	261.43	272.82	229.04
12	311.27	360.62	410.77	336.20	491.49	323.82
24	363.88	390.36	414.46	388.57	474.16	399.17
36	328.94	458.06	455.76	364.96	529.40	342.24
48	344.29	355.03	300.26	347.70	423.58	269.55
Average	**323.22**	**368.92**	**376.23**	**339.77**	**438.29**	**312.76**

Table B.8: Performance comparison between Log1p and Tuned Log1p transformations for ILI dataset using the TimeXer model.

Weeks	Log1p	Tuned Log1p
	MAE	
6	22.80	20.30
12	31.32	26.90
24	30.50	30.24
36	29.36	28.28
48	33.10	27.87
60	31.03	27.20
Average	**29.69**	**26.80**
	MSE	
6	7452.06	6601.13
12	9735.79	8765.07
24	10816.45	10220.49
36	10436.13	9914.20
48	11079.11	10240.89
60	10610.94	10197.76
Average	**10021.75**	**9323.26**

Table B.9: Model Hyperparameters Across Forecast Horizons for tuned Log1p for TimeXer model.

Parameter	6	12	24	36	48	60
Sequence length	48	48	96	96	96	96
Batch size	16	16	16	16	32	32
Encoder layers (e_layers)	2	2	2	5	3	2
Model dimension (d_model)	256	512	512	192	512	512

Table B.10: Performance of preprocessing methods on PatchTST, iTransformer, and TimeXer for the Illness dataset across all horizons.

Weeks	StandardScaler	Log1p	SQRT	Box–Cox	Yeo–Johnson	Log1p+Diff
PatchTST – ILLNESS (sMAPE)						
6	35.40	22.20	24.87	23.56	23.77	18.81
12	49.40	30.20	33.64	35.06	35.28	25.74
24	46.71	28.50	32.30	36.08	35.22	31.80
36	47.97	27.46	35.75	32.51	31.57	33.11
48	58.11	26.82	35.05	29.40	30.43	28.87
60	38.75	32.23	32.47	34.32	33.52	29.61
Avg	46.06	27.90	32.35	31.82	31.63	27.99
PatchTST – ILLNESS (MAE)						
6	8273.13	6373.05	7749.31	6554.55	6655.77	5542.20
12	11437.84	9263.33	10197.19	10205.94	10190.11	7383.88
24	12210.70	9525.51	10196.25	9403.71	9555.62	11989.62
36	11391.07	9161.96	10427.53	10520.07	10362.76	12069.84
48	11507.16	9020.55	10039.06	9267.83	9531.32	9317.11
60	10812.97	9161.72	10418.69	11180.91	10900.16	9836.89
Avg	10938.81	8751.02	9838.01	9522.17	9532.62	9356.59
iTransformer – ILLNESS (sMAPE)						
6	25.01	19.59	20.45	21.22	21.03	18.71
12	33.94	25.61	27.11	27.79	27.56	28.32
24	38.72	29.23	32.77	32.73	32.53	30.52
36	39.48	27.39	33.52	32.26	32.02	30.23
48	40.18	25.97	29.28	28.08	28.03	30.28
60	45.61	27.61	31.48	28.66	28.4	31.61
Avg	37.15	25.90	29.10	28.46	28.26	28.28
iTransformer – ILLNESS (MAE)						
6	6771.75	6156.49	6089.06	6538.90	6509.05	5827.22
12	9200.27	8372.22	8430.59	8937.26	8916.59	8749.94
24	11252.74	10896.27	11158.67	11491.66	11484.22	10862.99
36	11713.13	9932.19	11263.40	11015.40	10952.86	10265.56
48	11420.25	9609.47	10018.254	9864.82	9826.63	10584.52
60	12637.90	10050.91	10641.27	10119.55	10037.58	10967.57
Avg	10499.34	9169.59	9600.21	9661.26	9621.16	9542.97
TimeXer – ILLNESS (sMAPE)						
6	27.33	22.80	22.60	22.95	22.95	18.49
12	35.20	31.32	29.85	29.65	29.61	28.49
24	35.60	30.50	30.74	30.56	30.51	37.56
36	34.56	29.36	28.95	28.86	28.79	36.81
48	33.85	33.10	36.46	33.17	32.97	34.77
60	35.30	31.03	32.59	30.98	30.79	36.11
Avg	33.64	29.69	30.20	29.36	29.27	32.04
TimeXer – ILLNESS (MAE)						
6	7710.90	7452.06	7079.36	7506.29	7504.02	5782.98
12	10221.03	9735.79	9478.35	9589.29	9596.13	8532.50
24	11215.61	10816.45	10660.61	10773.22	10762.17	11374.30
36	10757.35	10436.13	10000.81	10051.17	10034.75	12430.45
48	10574.79	11079.11	11283.16	10912.04	10843.48	12036.34
60	11103.93	10610.94	10959.96	10342.57	10264.97	12101.90
Avg	10263.94	10021.75	9910.38	9862.43	9834.25	10376.41

Table B.11: Performance of preprocessing methods on PatchTST, iTransformer, and TimeXer for the COVID-19 dataset across all horizons.

Weeks	StandardScaler	Log1p	SQRT	Box–Cox	Yeo–Johnson	Log1p+Diff
PatchTST – COVID-19 (sMAPE)						
1	12.49	15.08	11.12	11.34	11.27	12.95
2	20.11	20.29	17.30	16.26	15.97	22.48
3	29.01	31.34	25.06	21.25	21.28	31.24
4	40.05	38.92	34.94	35.83	35.26	40.47
5	50.92	54.81	44.17	44.35	43.48	50.00
6	61.39	66.76	51.81	56.67	55.59	52.72
Avg	35.66	37.87	30.73	30.95	30.48	34.98
PatchTST – COVID-19 (MAE)						
1	1340.56	1602.64	1172.77	1184.82	1184.74	1373.24
2	2299.54	2369.02	1886.85	1819.25	1797.44	2540.97
3	3289.08	3242.41	2637.44	2381.44	2367.28	3465.01
4	4148.29	3928.87	4036.35	3944.62	3898.38	4312.17
5	5250.25	5228.75	4638.83	4528.56	4446.36	5257.46
6	5962.91	6326.23	5145.46	5152.98	5092.66	4540.18
Avg	3715.11	3782.99	3252.95	3168.61	3131.14	3581.51
iTransformer – COVID-19 (sMAPE)						
1	26.99	11.42	15.79	10.61	9.72	12.33
2	28.82	19.34	14.88	16.91	16.49	20.14
3	25.51	27.72	18.09	24.82	23.26	23.61
4	24.77	40.72	31.74	35.51	32.11	37.20
5	30.99	51.51	41.96	50.34	44.77	32.86
6	43.95	71.30	56.55	57.38	60.84	53.51
Avg	30.17	37.00	29.83	32.59	31.20	29.93
iTransformer – COVID-19 (MAE)						
1	1867.33	1232.09	1612.27	996.69	1126.93	1328.49
2	1862.07	2141.20	1836.60	1816.38	1708.02	2127.21
3	2037.14	2947.22	2064.12	2353.61	2606.74	2621.44
4	2523.21	3991.75	3658.94	3207.09	3195.12	3828.78
5	3208.39	5166.39	4416.06	4156.71	4095.33	2718.10
6	3919.48	6638.36	5328.85	4944.54	5507.34	4518.37
Avg	2569.60	3686.17	3152.80	2912.50	3039.91	2857.06
TimeXer – COVID-19 (sMAPE)						
1	18.32	11.98	15.33	13.25	14.34	16.52
2	23.89	23.31	20.10	20.45	19.64	31.54
3	36.31	27.61	24.21	25.80	26.41	36.16
4	46.99	37.62	34.98	35.42	43.37	47.54
5	47.99	51.75	44.68	48.08	51.99	57.59
6	55.24	62.16	54.50	54.43	64.97	64.11
Avg	38.12	35.74	32.30	32.91	36.79	42.24
TimeXer – COVID-19 (MAE)						
1	1833.92	1421.23	1769.84	1476.11	1638.66	1731.44
2	2467.32	2573.73	2098.87	2310.00	2060.79	3434.08
3	3453.34	2866.34	2466.51	2806.27	2848.90	3806.61
4	4219.89	3538.08	3104.75	3578.18	3993.36	4767.98
5	4427.96	4675.19	3930.41	4555.40	4563.34	5548.12
6	5060.09	5488.16	4463.01	5030.01	5540.94	6011.54
Avg	3577.07	3427.12	2972.23	3292.66	3440.56	4216.63

Table B.12: Performance of preprocessing methods on PatchTST, iTransformer, and TimeXer for the RSV dataset across all horizons.

Weeks	StandardScaler	Log1p	SQRT	Box–Cox	Yeo–Johnson	Log1p+Diff
PatchTST – RSV (sMAPE)						
6	40.92	29.07	26.89	29.12	29.13	27.37
12	32.36	28.02	27.60	28.07	28.08	33.63
24	42.61	24.39	29.50	24.44	24.44	29.48
36	44.07	25.79	24.67	25.85	25.91	25.20
48	54.25	23.16	28.65	23.30	23.29	24.13
Avg	42.84	26.09	27.46	26.16	26.17	27.96
PatchTST – RSV (MAE)						
6	267.75	280.52	240.74	281.78	282.20	326.17
12	311.27	360.62	290.07	362.34	363.55	346.32
24	363.88	390.36	372.80	393.05	392.94	363.11
36	328.94	458.06	310.22	462.50	464.64	399.97
48	344.29	355.03	291.50	358.80	359.16	380.88
Avg	323.22	368.92	301.07	371.69	372.50	363.29
iTransformer – RSV (sMAPE)						
6	35.58	32.62	31.37	32.77	32.79	25.56
12	56.52	29.74	31.31	29.94	29.95	28.66
24	42.77	26.14	28.00	26.45	26.48	27.57
36	43.78	22.96	27.48	23.13	23.16	26.05
48	62.68	23.89	40.82	24.12	24.14	27.21
Avg	48.27	27.07	31.80	27.28	27.30	27.01
iTransformer – RSV (MAE)						
6	326.45	388.59	348.57	397.19	397.66	391.74
12	408.71	416.53	387.29	426.48	427.06	385.93
24	456.62	459.83	406.60	472.35	473.24	466.08
36	384.95	363.89	353.79	369.75	370.39	411.22
48	502.34	375.18	478.36	381.06	381.69	425.75
Avg	415.81	400.81	394.92	409.37	410.01	416.15
TimeXer – RSV (sMAPE)						
6	38.25	30.51	28.54	30.51	30.53	27.63
12	39.82	29.17	28.90	29.16	29.19	28.77
24	38.52	25.89	26.22	25.90	25.92	25.36
36	38.62	22.23	22.85	22.23	22.26	25.77
48	47.85	21.71	22.72	21.75	21.77	26.82
Avg	40.61	25.90	25.85	25.91	25.93	26.87
TimeXer – RSV (MAE)						
6	306.15	368.76	327.88	368.86	369.35	345.33
12	341.74	395.20	363.84	395.64	396.20	346.29
24	371.11	418.69	371.48	419.28	419.75	412.64
36	336.03	356.61	325.31	357.39	357.80	429.48
48	327.01	325.27	286.38	326.64	327.07	410.04
Avg	336.41	372.91	334.98	373.56	374.04	388.76

Table B.13: Performance comparison of different normalization strategies across models and datasets, evaluated using sMAPE.

Horizon	Subtract_Last	Subtract_Mean	Subtract_Mov_Avg ($\sqrt{H}$)	Subtract_Mov_Avg (H)	CoIN
PatchTST (ILI Dataset)					
6	22.74	35.40	28.71	32.05	22.56
12	38.66	49.40	50.72	38.20	30.50
24	54.84	46.71	56.18	45.44	41.57
36	43.25	47.97	51.46	45.84	39.72
48	37.98	58.11	44.82	42.09	37.98
60	35.97	38.75	42.57	37.89	35.97
Average	**38.91**	**46.06**	**45.74**	**40.25**	**34.72**
iTransformer (ILI Dataset)					
6	26.62	25.01	26.39	27.87	20.37
12	35.33	33.94	35.65	37.83	28.13
24	41.66	38.72	40.34	45.64	38.07
36	43.45	39.48	41.98	41.08	39.09
48	40.05	40.18	41.32	38.81	37.59
60	42.07	45.61	42.48	38.08	36.81
Average	**38.20**	**37.15**	**38.03**	**38.22**	**33.34**
TimeXer (ILI Dataset)					
6	23.31	27.33	31.30	28.28	21.26
12	33.46	35.20	38.91	40.75	31.32
24	40.13	35.60	45.47	42.35	32.58
36	48.28	34.56	52.93	50.73	32.13
48	36.37	33.85	39.30	38.02	33.09
60	37.24	35.30	38.52	43.43	34.11
Average	**36.47**	**33.64**	**41.07**	**40.59**	**30.75**
Horizon	**Subtract_Last**	**Subtract_Mean**	**Subtract_Mov_Avg ($\sqrt{H}$)**	**Subtract_Mov_Avg (H)**	**CoIN**
PatchTST (COVID Dataset)					
1	13.38	12.49	13.52	13.98	14.60
2	20.65	20.11	20.70	22.02	18.81
3	25.83	29.01	27.42	28.28	25.33
4	37.26	40.05	37.07	38.59	37.98
5	44.72	50.92	47.88	46.66	45.46
6	59.96	61.39	56.63	62.26	55.69
Average	**33.63**	**35.66**	**33.87**	**35.30**	**32.98**
iTransformer (COVID Dataset)					
1	11.26	26.99	29.72	16.16	11.95
2	19.20	28.82	47.75	21.60	21.09
3	24.01	25.51	57.18	30.75	22.18
4	34.55	24.77	64.92	40.14	27.18
5	39.58	30.99	73.41	42.31	35.68
6	52.13	43.95	80.67	58.37	49.33
Average	**30.12**	**30.17**	**58.94**	**34.89**	**27.90**
TimeXer (COVID Dataset)					
1	14.44	18.32	17.85	13.58	12.77
2	22.35	23.89	26.23	23.49	20.54
3	30.18	36.31	36.87	30.72	26.56
4	38.82	46.99	48.20	43.55	40.66
5	48.61	47.99	54.86	50.17	44.41
6	53.88	55.24	66.86	57.90	53.52
Average	**34.71**	**38.12**	**41.81**	**36.57**	**33.08**

Bibliography

Ahsan, M. M., Mahmud, M. A. P., Saha, P. K., Gupta, K. D., & Siddique, Z. (2021). Effect of data scaling methods on machine learning algorithms and model performance. *Technologies*, *9*(3). https://www.mdpi.com/2227-7080/9/3/52

Andersen, T. G., Bollerslev, T., Christoffersen, P., & Diebold, F. X. (2005). Volatility forecasting.

Andrews, D., Gnanadesikan, R., & Warner, J. (1971). Transformations of multivariate data. *Biometrics*, 825–840.

Bai, J., et al. (2021). A3t-gcn: Attention temporal graph convolutional network for traffic forecasting. *ISPRS International Journal of Geo-Information*, *10*(7), 485.

Bandara, K., Hewamalage, H., Liu, Y.-H., Kang, Y., & Bergmeir, C. (2020). Improving the accuracy of global forecasting models using time series data augmentation. https://arxiv.org/abs/2008.02663

Bartlett, M. S. (1936). The square root transformation in analysis of variance. *Supplement to the Journal of the Royal Statistical Society*, *3*(1), 68–78.

Box, G. E. P., & Jenkins, G. M. (1976). *Time series analysis: Forecasting and control* (Revised Edition).

Box, G. E., & Cox, D. R. (1964). An analysis of transformations. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *26*(2), 211–243.

Bpagano by bob pagano [Accessed: 2025-10-14]. (n.d.).

Cao, D., et al. (2020). Spectral temporal graph neural network for multivariate time-series forecasting. *Advances in Neural Information Processing Systems*, *33*, 17766–17778.

CDC. (2022). Covid-19 forecasts: Deaths, cases, and hospitalizations [Accessed: 2025-10-20].

Cdc covid tracker [Accessed: 2025-10-14]. (n.d.).

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Chicco, D., Warrens, M. J., & Jurman, G. (2021). The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation. *PeerJ Computer Science*, *7*, e623.

Chimmula, V. K. R., & Zhang, L. (2020). Time series forecasting of covid-19 transmission in canada using lstm networks. *Chaos, Solitons & Fractals*, *135*, 109864.

Chinazzi, M., et al. (2020). The effect of travel restrictions on the spread of the 2019 novel coronavirus (covid-19) outbreak. *Science*, *368*(6489), 395–400.

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Empirical Methods in Natural Language Processing (EMNLP)*.

Columbia by columbia university [Accessed: 2025-10-14]. (n.d.).

Computational uncertainty lab by prof. thomas mcandrew [Accessed: 2025-10-14]. (n.d.).

Cramer, E. Y., et al. (2022). Evaluation of individual and ensemble probabilistic forecasts of covid-19 mortality in the united states. *Proceedings of the National Academy of Sciences*, *119*(15), e2113561119.

Cramer, E. Y., Huang, Y., Wang, Y., Ray, E. L., Cornell, M., Bracher, J., Brennen, A., Rivadeneira, A. J. C., Gerding, A., House, K., et al. (2022). The united states covid-19 forecast hub dataset. *Scientific data*, *9*(1), 462.

Cramer, E. Y., Ray, E. L., Lopez, V. K., Bracher, J., Brennen, A., Castro Rivadeneira, A. J., Gerding, A., Gneiting, T., House, K. H., Huang, Y., et al. (2022). Evaluation of individual and ensemble probabilistic forecasts of covid-19 mortality in the united states. *Proceedings of the National Academy of Sciences*, *119*.

Curran-Everett, D. (2018). Explorations in statistics: The log transformation. *Adv Physiol Educ*, *42*(2), 343–347.

Dastjerdi, F. R., Robinson, D. A., & Cai, L. (2022). α-hmm and optimal decoding higher-order structures on sequential data. *Journal of Computational Mathematics and Data Science*, *5*, 100065.

de Amorim, L. B., Cavalcanti, G. D., & Cruz, R. M. (2023). The choice of scaling technique matters for classification performance. *Applied Soft Computing*, *133*, 109924. https://doi.org/https://doi.org/10.1016/j.asoc.2022.109924

Dong, E., Du, H., & Gardner, L. (2020). An interactive web-based dashboard to track covid-19 in real time. *The Lancet Infectious Diseases*, *20*(5), 533–534.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Esg by robert walraven [Accessed: 2025-10-14]. (n.d.).

Feng, C., Wang, H., Lu, N., Chen, T., He, H., Lu, Y., & Tu, X. M. (2014). Log-transformation and its implications for data analysis. *Shanghai archives of psychiatry*, *26*(2), 105–109.

Feng, C., Wang, H., Lu, N., & Tu, X. M. (2013). Log transformation: Application and interpretation in biomedical research. *Statistics in medicine*, *32*(2), 230–239.

Ferguson, N. M., et al. (2006). Strategies for mitigating an influenza pandemic. *Nature*, *442*(7101), 448–452.

Fritz, C., Dorigatti, E., & Rügamer, D. (2021). Combining graph neural networks and spatio-temporal disease models to predict covid-19 cases in germany. *arXiv preprint arXiv:2101.00661*.

Galli, S. (2024). Python feature engineering cookbook: A complete guide to crafting powerful features for your machine learning models.

Gelfand, S. J. (2015). Understanding the impact of heteroscedasticity on the predictive ability of modern regression methods.

Gt-deepcovid by georgia institute of technology, college of computing [Accessed: 2025-10-14]. (n.d.).

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*.

Hossain, Z., Rahman, A., Hossain, M., & Karami, J. H. (2019). Over-differencing and forecasting with non-stationary time series data. *Dhaka University Journal of Science*, *67*(1), 21–26.

Hyndman, R. (2018). *Forecasting: Principles and practice*. OTexts.

Jhu-apl by johns hopkins university, applied physics lab [Accessed: 2025-10-14]. (n.d.).

Jhu-idd by johns hopkins university, infectious disease dynamic lab [Accessed: 2025-10-14]. (n.d.).

Kapoor, A., et al. (2020). Examining covid-19 forecasting using spatio-temporal graph neural networks. *arXiv preprint arXiv:2007.03113*.

Karlen by karlen working group [Accessed: 2025-10-14]. (n.d.).

Kim, T., Kim, J., Tae, Y., Park, C., Choi, J.-H., & Choo, J. (2021). Reversible instance normalization for accurate time-series forecasting against distribution shift. *International conference on learning representations*.

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kondo, K., Ishikawa, A., & Kimura, M. (2019). Sequence to sequence with attention for influenza prevalence prediction using google trends. *Proceedings of the 2019 3rd International Conference on Computational Biology and Bioinformatics*, 1–7.

Kuhn, M., & Johnson, K. (2019). *Feature engineering and selection: A practical approach for predictive models*. Chapman; Hall/CRC.

Kumar, P., et al. (2020). Forecasting the dynamics of covid-19 pandemic in top 15 countries in april 2020: Arima model with machine learning approach. *medRxiv*, 2020–03.

Kumar, S., et al. (2021). Forecasting the spread of covid-19 using lstm network. *BMC Bioinformatics*, *22*(6), 1–9.

Learned-Miller, E. G. (2013). Entropy and mutual information.

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., & Stoica, I. (2018). Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*.

Lippi, M., Bertini, M., & Frasconi, P. (2013). Short-term traffic flow forecasting: An experimental comparison of time-series analysis and supervised learning. *IEEE Transactions on Intelligent Transportation Systems*, *14*(2), 871–882.

Lirio, R. B., Sosa, P. V., Marqui, R. P., Jiménez-Sobrino, J., Amador, A. A., & Garcia, L. G. (1989). Multivariate box-cox transformations with applications to neurometric data. *Computers in biology and medicine*, *19*(4), 263–267.

Liu, L., Han, M., Zhou, Y., & Wang, Y. (2018). Lstm recurrent neural networks for influenza trends prediction. *Bioinformatics Research and Applications: 14th International Symposium, ISBRA 2018, Beijing, China, June 8–11, 2018, Proceedings 14*, 259–264.

Liu, Q., Fung, D. L. X., Lac, L., & Hu, P. (2021). A novel matrix profile-guided attention lstm model for forecasting covid-19 cases in usa. *Frontiers in Public Health*, *9*.

Liu, Y., Hu, T., Zhang, H., Wu, H., Wang, S., Ma, L., & Long, M. (2023). Itransformer: Inverted transformers are effective for time series forecasting. *arXiv preprint arXiv:2310.06625*.

Liu, Z., Cheng, M., Li, Z., Huang, Z., Liu, Q., Xie, Y., & Chen, E. (2023). Adaptive normalization for non-stationary time series forecasting: A temporal slice perspective. *Advances in Neural Information Processing Systems*, *36*, 14273–14292.

Lütkepohl, H., & Xu, F. (2012). The role of the log transformation in forecasting economic variables. *Empirical Economics*, *42*(3), 619–638. https://doi.org/10.1007/s00181-010-0440-1

Lv, Y., Duan, Y., Kang, W., Li, Z., & Wang, F.-Y. (2014). Traffic flow prediction with big data: A deep learning approach. *Ieee transactions on intelligent transportation systems*, *16*(2), 865–873.

Mahmud, S., et al. (2021). A human mobility data driven hybrid gnn+rnn based model for epidemic prediction. *2021 IEEE International Conference on Big Data (Big Data)*.

Mathieu, E., Ritchie, H., Rodés-Guirao, L., Appel, C., Gavrilov, D., Giattino, C., Hasell, J., Macdonald, B., Dattani, S., Beltekian, D., Ortiz-Ospina, E., & Roser, M. (2020). Coronavirus (covid-19) deaths [https://ourworldindata.org/covid-deaths]. *Our World in Data*.

Microsoft by microsoft ai [Accessed: 2025-10-14]. (n.d.).

Miller, J. A. (2022). *Introduction to computational data science using scalation*.

Mobs by northeastern university, laboratory for the modeling of biological and socio-technical systems [Accessed: 2025-10-14]. (n.d.).

Mudelsee, M. (2019). Trend analysis of climate time series: A review of methods. *Earth-science reviews*, *190*, 310–322.

Nau, R. (2016). Statistical forecasting: Notes on regression and time series analysis. *Stepwise and All Possible Regressions. Available online: https://people. duke. edu/˜ rnau/regstep. htm (accessed on 2 May 2019)*.

Nie, Y. (2022). A time series is worth 64words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*.

Ogasawara, E., Martinez, L. C., De Oliveira, D., Zimbrão, G., Pappa, G. L., & Mattoso, M. (2010). Adaptive normalization: A novel data normalization approach for non-stationary time series. *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

Panagopoulos, G., Nikolentzos, G., & Vazirgiannis, M. (2021). Transfer graph neural networks for pandemic forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*(6).

Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., & Iosifidis, A. (2019). Deep adaptive input normalization for time series forecasting. *IEEE transactions on neural networks and learning systems*, *31*(9), 3760–3765.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, *32*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Perone, G. (2022). Using the sarima model to forecast the fourth global wave of cumulative deaths from covid-19: Evidence from 12 hard-hit big countries. *Econometrics*, *10*(2), 18.

Proietti, T., & Luetkepohl, H. (2011). Does the Box-Cox transformation help in forecasting macroeconomic time series?

Rahman, M., & Pearson, L. M. (2009). Estimation of the multivariate box-cox transformation parameters. In *Advances in multivariate statistical methods* (pp. 173–183). World Scientific.

Raju, V. N. G., Lakshmi, K. P., Jain, V. M., Kalidindi, A., & Padma, V. (2020). Study the influence of normalization/transformation process on the accuracy of supervised classification. *2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)*, 729–735.

Rana, S., Barna, N. H., & Miller, J. A. (2023). Exploring the predictive power of correlation and mutual information in attention temporal graph convolutional network for covid-19 forecasting. *International Conference on Big Data*, 18–33.

Rana, S., Miller, J. A., Nesbit, J., Barna, N. H., Aldosari, M., & Arpinar, I. B. (2024). How effective are time series models for pandemic forecasting? *International Conference on Big Data*, 3–17.

Ray, E. L., et al. (2020). Ensemble forecasts of coronavirus disease 2019 (covid-19) in the us. *medRxiv*, 2020–08.

Ray, E. L., Brooks, L., Bien, J., Bracher, J., Gerding, A., Rumack, A., Biggerstaff, M., Johansson, M., Tibshirani, R., & Reich, N. (2021). Challenges in training ensembles to forecast covid-19 cases and deaths in the united states. *International Institute of Forecasters*.

Ritchie, H., Mathieu, E., Rodés-Guirao, L., Appel, C., Giattino, C., Ortiz-Ospina, E., Hasell, J., Macdonald, B., Beltekian, D., & Roser, M. (2020). Our world in data. *Scientific Data*.

Rozemberczki, B., et al. (2021). Pytorch geometric temporal: Spatiotemporal signal processing with neural machine learning models. *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*.

Salles, R., Belloze, K., Porto, F., Gonzalez, P. H., & Ogasawara, E. (2019). Nonstationary time series transformation methods: An experimental review. *Knowledge-Based Systems*, *164*, 274–291. https://doi.org/https://doi.org/10.1016/j.knosys.2018.10.041

Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with python. *9th Python in Science Conference*.

Smyl, S., Ranganathan, J., & Pasqua, A. (2018). M4 forecasting competition: Introducing a new hybrid es-rnn model.

Srivastava, A. (2022). The variations of sikjalpha model for covid-19 forecasting and scenario projections. *arXiv preprint arXiv:2207.02919*.

Sundar, S., Schwab, P., Tan, J. Z. H., Romero-Brufau, S., Celi, L. A., Wangmo, D., & Penna, N. (2021). Forecasting the covid-19 pandemic: Lessons learned and future directions. *medRxiv*.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, *27*.

Ucsd-neu university of california, san diego and northeastern university [Accessed: 2025-10-14]. (n.d.).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*.

Velilla, S. (1992). A note on the multivariate box-cox transformation to normality. *Nº.: Working Papers 1992-08*.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Wang, L., et al. (2020). Using mobility data to understand and forecast covid-19 dynamics. *medRxiv*.

Wang, Y., Wu, H., Dong, J., Qin, G., Zhang, H., Liu, Y., Qiu, Y., Wang, J., & Long, M. (2024). Timexer: Empowering transformers for time series forecasting with exogenous variables. *Advances in Neural Information Processing Systems*, *37*, 469–498.

Weisberg, S. (2001). Yeo-johnson power transformations. *Department of Applied Statistics, University of Minnesota. Retrieved June*, *1*, 2003.

Weron, R. (2014). Electricity price forecasting: A review of the state-of-the-art with a look into the future. *International journal of forecasting*, *30*(4), 1030–1081.

Wu, H., Xu, J., Wang, J., & Long, M. (2021). Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in neural information processing systems*, *34*, 22419–22430.

Wu, H., Zhou, H., Long, M., & Wang, J. (2023). Interpretable weather forecasting for worldwide stations with a unified deep model. *Nature Machine Intelligence*, *5*(6), 602–611.

Wu, N., Green, B., Ben, X., & O'Banion, S. (2020). Deep transformer models for time series forecasting: The influenza prevalence case. *arXiv preprint arXiv:2001.08317*.

Xue, J., et al. (2022). Multiwave covid-19 prediction from social awareness using web search and mobility data. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Yeo, I.-K., & Johnson, R. A. (2000). A new family of power transformations to improve normality or symmetry. *Biometrika*, *87*(4), 954–959.

Zeng, A., Chen, M., Zhang, L., & Xu, Q. (2023). Are transformers effective for time series forecasting? *Proceedings of the AAAI Conference on Artificial Intelligence*, 11121–11128.

Zeroual, A., Harrou, F., Dairi, A., & Sun, Y. (2020). Deep learning methods for forecasting covid-19 time-series data: A comparative study. *Chaos, solitons & fractals*, *140*, 110121.

Zhang, Y., Long, M., Chen, K., Xing, L., Jin, R., Jordan, M. I., & Wang, J. (2023). Skilful nowcasting of extreme precipitation with nowcastnet. *Nature*, *619*(7970), 526–532.

Zhang, Y., & Yan, J. (2023). Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. *The eleventh international conference on learning representations*.

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2021). Informer: Beyond efficient transformer for long sequence time series forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 11106–11115.

Zhou, T., Ma, Z., Wen, Q., Wang, X., Sun, L., & Jin, R. (2022). Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. *International Conference on Machine Learning*, 27268–27286.