REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEURE ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE IBN KHALDOUN - TIARET

MEMOIRE

Présenté à :

FACULTÉ DES MATHEMATIQUES ET DE l'INFORMATIQUE
DÉPARTEMENT D'INFORMATIQUE

Pour l'obtention du diplôme de :

MASTER

Spécialité : Génie Informatique

Par :

BRAHIM Abderrahmane
BENSAADIA Mohamed

Sur le thème

Modèle hybride pour la prévision de séries chronologiques

Soutenu publiquement le .. / .. / 2023 à Tiaret devant le jury composé de :

Mme.	BOUBEKEUR Aisha	MAA	Université Ibn-Khadoun	Présidente
M.	KOUADRIA Abderrahmane	MCB	Université Ibn-Khadoun	Encadrant
M.	MOKHTARI Ahmed	MAA	Université Ibn-Khadoun	Examinateur

2022-2023

Abstract

Improving time series forecasting accuracy is an active area of research, several models and techniques have been proposed to enhance models performance and improve forecasts accuracy, this work proposes two novel hybrid deep learning models applied for direct multi-step ahead time series forecasting, a combination of Convolutional Neural Network (CNN), Gated Recurrent and Deep Temporal Convolutional Network (TCN), and a hybridization of Convolutional Neural Network (CNN), Long-Short Term Memory network (LSTM) and Graph Neural Network (GNN) were proposed, both architectures were tested on three different types of datasets and were compared to a collection of eleven state-of-the-art models. Our novel models outperformed the baseline models according to four evaluation metrics. This comparative study demonstrates the superiority of hybrid models over individual approaches to time series forecasting.

Résumé

L'amélioration de la précision de la prévision des séries temporelles est un domaine de recherche actif. Plusieurs modèles et techniques ont été proposés pour améliorer les performances des modèles et accroître la précision des prévisions. Ce travail propose deux nouveaux modèles hybrides d'apprentissage profond appliqués à la prévision directe à plusieurs pas en avant des séries temporelles. Il s'agit d'une combinaison de Convolutional Neural Network, de Gated Recurrent Unit network (GRU) et de Deep Temporal Convolutional Network (TCN), ainsi que d'une hybridation de Convolutional Neural Network, de Long-Short Term Memory network (LSTM) et de Graph Neural Network (GNN). Les deux architectures ont été testées sur trois types de datasets différents et comparées à une collection de onze modèles de l'état de l'art. Nos nouveaux modèles ont surpassé les modèles de référence selon quatre métriques d'évaluation. Cette étude comparative démontre la supériorité des modèles hybrides par rapport aux approches individuelles de prévision des séries temporelles.

Acknowledgments

First and foremost, we extend our utmost gratitude and praise to Allah Almighty for granting us the perseverance, courage, determination, and inspiration to successfully accomplish this endeavor.

We would like to express our deep appreciation and thanks to our beloved parents, relatives, and friends for their unwavering support and constant encouragement throughout this journey.

Our sincere gratitude goes to our esteemed mentor,**Mr Kouadria Abderrahmane**, for his patient guidance, wholehearted support, and invaluable insights that have greatly influenced and shaped our research work.

Furthermore, we would like to extend our sincere acknowledgments to the president of the jury and all the jury members for their keen interest in our research. We are grateful for their willingness to examine our work and enrich it with their valuable suggestions and recommendations.

Contents

List of Figures

List of Tables

List of Acronyms

ACF Autocorrelation Function.

API Application Programming Interface.

AR Autoregressive.

ARCH Autoregressive Conditional Heteroscedastic.

ARIMA Autoregressive Integrated Moving Average.

ARMA Autoregressive Moving average.

CART Classification And Regression Tree.

CNN Convolutional Neural Network.

CPU Central Processing Unit.

ES Exponential Smoothing.

GARCH Generalized Autoregressive Conditional Heteroscedastic.

GAT Graph Attention Neural Network.

GB Gigabyte.

GCNN Graph Convolutional Neural Network.

GHz Gigahertz.

GNN Graph Neural Network.

GPU Graphics Processing Unit.

GraphSAGE Graph Sample And Aggregate.

GRNN Graph Recurrent Neural Network.

GRU Gated Recurrent Unit.

LSTM Long Short-Term Memory.

MA Moving Average.

MAE Mean Absolute Error.

MAPE Mean Absolute Percentage Error.

MSE Mean Squared Error.

NLP Natural Language Processing.

PCF Partial Autocorrelation Function.

RAM Random Access Memory.

RMSE Root Mean Squared Error.

RNN Recurrent Neural Network.

SVM Support Vector Machine.

SVR Support Vector Regression.

TCN Temporal Convolutional Network.

TPU Tensor Processing Unit.

VRAM Video Random Access Memory.

XGBoost Extreme Gradient Boosting.

General introduction

Problem statement

Time series forecasting plays a crucial role in various domains such as finance, economics, government, social science, environmental science, medicine, politics, and more. Accurate and reliable predictions of future values of time series data have significant implications on decision-making and planning. As time series data often exhibit complex patterns and dependencies, assessing the exact nature of a time series and generating appropriate forecasts is often a challenging task, researchers and practitioners have developed a plethora of modeling techniques and approaches in the pursuit of improving forecasting performance.
Classic statistical models such as Autoregressive Integrated Moving Average (ARIMA) and Exponential Smoothing are widely used and well founded mathematical models for time series forecasting, although these models cover a wide range to time series types and have a desirable level of interpretability, they suffer numerous limitations such as explicit assumptions on the data that are not always verified by real world situations which limits their range of applicability, the need of a certain level of expertise to adequately prepare and transform the data in addition to configuring the models, difficulties to model non-linearity and high sensitivity to data irregularities and outliers. Deep learning models on the other hand demonstrated strong abilities to model non-linearity and capture complex patterns and relationships as well as long-term dependencies in the data. However, deep learning models often require large amounts of data and computational resources and may have limited interpretability due to their architectural complexity.

This thesis focuses on investigating and comparing different time series forecasting models, including classic , deep learning , and hybrid models, with a special emphasis on evaluating the performance of a novel hybrid model proposed within this study against state-of-the-art models.

Research objectives

The primary objective of this thesis is to explore, understand and evaluate various modeling approaches in order to design and build a robust and reliable model able to enhance the performance and accuracy of time series forecasting. Particularly, the major objectives of this work are as follows:

a) To study classic time series forecasting models, such as ARIMA, Exponential Smoothing among others and understand their strengths and limitations in handling different types of time series data.

b) To explore state-of-the-art deep learning models, including LSTMs, GRUs, TCNs, Transformers and others, and assess their ability to capture complex temporal patterns in time series data.

c) To implement and experiment with hybrid models that combine the strengths of different models to potentially achieve superior forecasting accuracy and generalization performance.

d) To propose a novel hybrid model that incorporates an innovative approach and model architecture, aiming to outperform existing individual and hybrid state-of-the-art models in various forecasting scenarios.

Contributions of the thesis:

This thesis contributes to the time series forecasting literature in the following ways:

- **Comprehensive Evaluation:** We systematically evaluated a range of classic models, deep learning models, and hybrid models, providing a thorough comparison of their performance across different types of time series data.
- **Novel Hybrid Model:** Our proposed hybrid models CNN-GRU-TCN and GNN-CNN-LSTM introduces innovative model architecture, resulting in improved forecasting accuracy and generalization. This model demonstrated a substantial advancement in forecasting performance compared to existing methods.

Thesis outline

This paper is divided into four parts:

1. **Introduction:** introduces the thesis by defining the problem treated in this work and delimiting the research objectives and the thesis layout.

2. **Chapter 1:** this chapter serves as introduction to time series forecasting, it provides a theoretical background on the topic and introduces commonly used methods and techniques for time series forecasting.

3. **Chapter 2:** dives into the time series forecasting literature and reviews a wide array of forecasting methodologies focusing more on deep learning and hybrid models found in recent literature.

4. **Chapter 3:** consists of the empirical part of this study, it highlights in detail the proposed and implemented models, the research environment and methodology as well as the subsequent presentation and discussion of the experiment results and evaluations.

Chapter 1

Time series analysis and forecasting

Introduction

Time series data arises naturally in many research fields including economics, business, engineering, environment, medicine and more, such an important presence has grabbed the attention of analysts and statisticians for the last few decades and have been thoroughly analyzed and studied in order to discern the internal relationships and patterns of the data and to elaborate models that are able to describe such processes and allow for generating future predictions.

In this chapter we will dive into time series analysis and forecasting by first: establishing a definition of time series and outline their main characteristics and principles. Subsequently we will move forward and discuss some of the major work that have been conducted in the field and introduce prominent approaches both classical and modern to time series analysis and forecasting where we will walk through statistical, machine and deep learning commonly used base models .

1.1 Time series

1.1.1 Definition

A time series refers to a collection of observations on a particular variable of interest that are recorded at **regular intervals** over time, creating a **chronological** or **time-oriented sequence**.

1.1.2 Examples of time series data

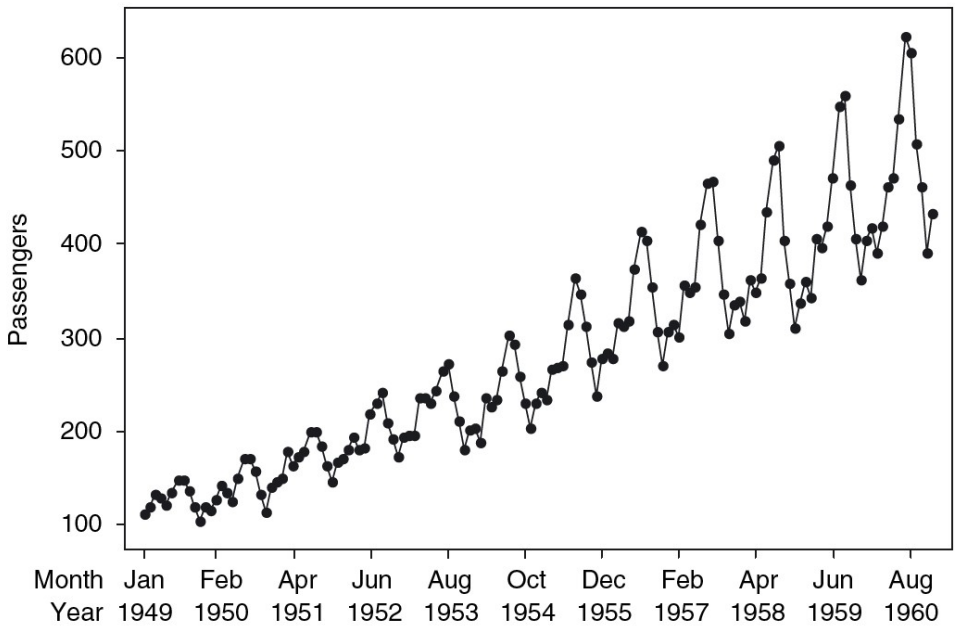


Figure 1.1: The number of US airline passengers from 1949 to 1960 [6].

Figures 1.1, 1.2 and 1.3 are time plots of : the number of US airline passengers, the annual sea levels for Copenhagen and the quarterly dollar sales of Marshall Field Company, this show how interest in time series data arises from different disciplinary fields such as business, finance, economy and climatology. The study and analysis of such data can help provide a deeper understanding and informative insights on the process of interest, which by turn contributes to future decision making and risk management.

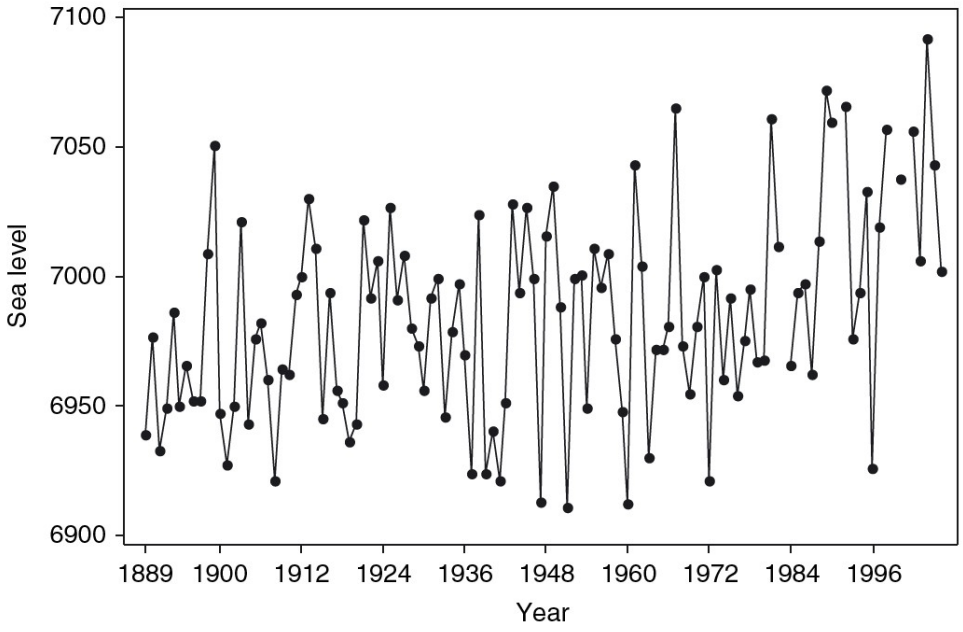


Figure 1.2: The annual sea levels in millimeters for Copenhagen, Denmark [6].

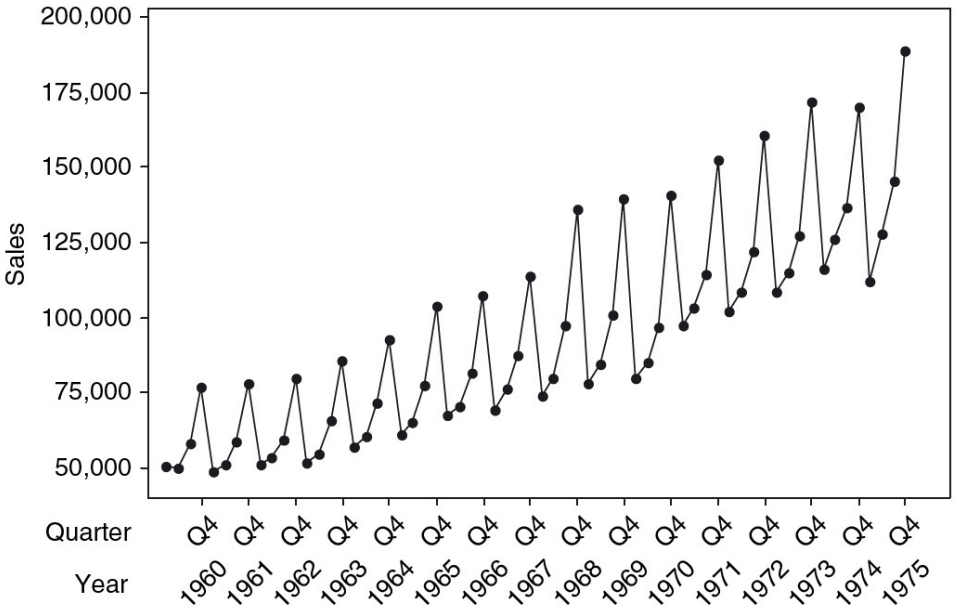


Figure 1.3: Quarterly dollar sales (in $1000) of Marshall Field & Company from 1960 to 1975 [6].

1.1.3 Time series analysis

Time series analysis is a branch of statistics that is interested in studying time series data in order to [44]:

- Understand the structure of the series and the time-dependency of the observations, identify and model the series components as well as extrapolating and recognizing patterns in historical data.
- Design control schemes and perform forecasts of future values of the variable of interest based on current and previous values.

1.1.4 Time series components

Decomposing a time series is a common and widely used technique in time series analysis, it isolates the series components and therefore allows a deeper analysis and a better understanding of the series behavior, its underlying patterns along side with irregularities and data anomalies.

The decomposition often follows one of two models [35, 36] :

- **Additive model:**
$$X_t = T_t + S_t + I_t \tag{1.1}$$
- **Multiplicative model:**
$$X_t = T_t \times S_t \times I_t \tag{1.2}$$

With T_t being **trend** (**trend-cycle**), S_t **seasonal** and I_t **irregular** (**random error, risidual**) components.

1.1.4.1 Trend

Trend is a general tendency over an extended observation time period, having either an increasing or decreasing slope. It may be **linear** or **nonlinear**, **deterministic** or **stochastic** [66].

Data in both Figure 1.1 and Figure 1.3 exhibit a clear upward trend.

1.1.4.2 Seasonality

Seasonality is a periodic recurrent pattern observed typically over a yearly or shorter (quarterly, monthly, weekly...) period of time.

Figure 1.1 and Figure 1.3 series exhibit seasonality.

1.1.4.3 Irregularity

The irregular component, also known as the residual, noise or random error, refers to the random and unpredictable fluctuations and deviations in the data that are not identified as part of the trend-cycle nor the seasonal components. It can represent any unexpected events, external influence, data collection and measurement errors or other factors.

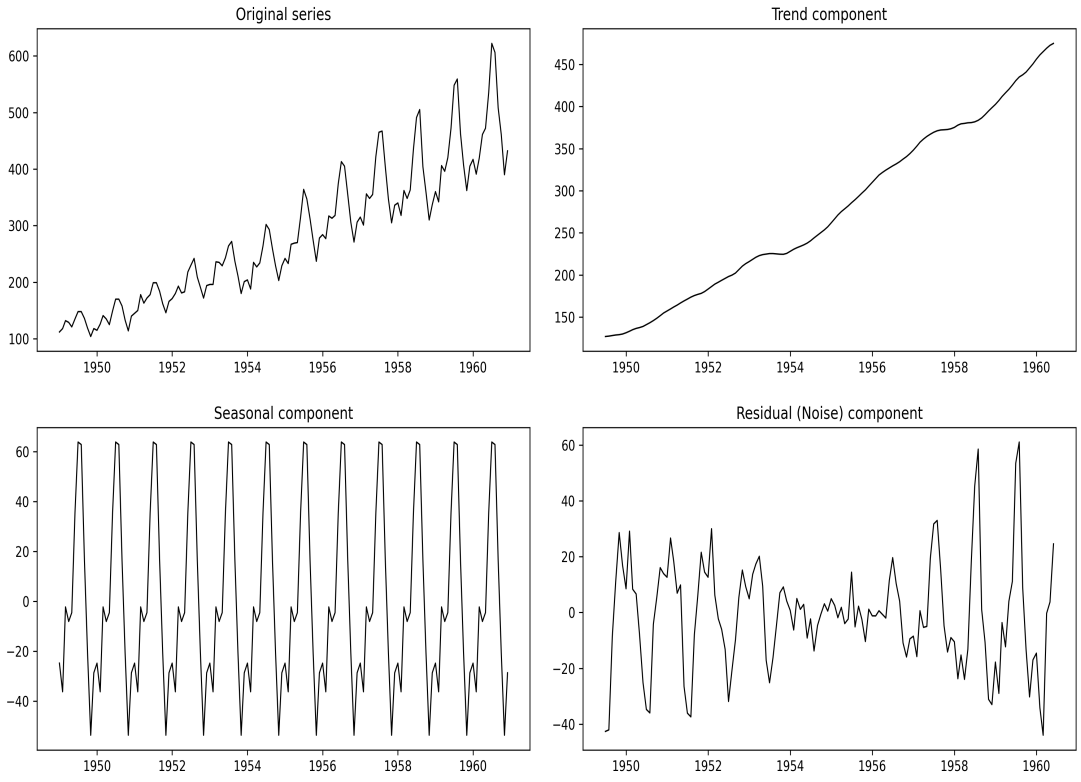


Figure 1.4: Additive decomposition of the number of US airline passengers time series (fig 1.1).

1.1.5 Other note-worthy features

1.1.5.1 Volatility

Volatility in a time series refers to the fluctuations of the series **variance** over time, it statically measures the dispersion of the series values around its mean. Volatility is often manifested in financial data.

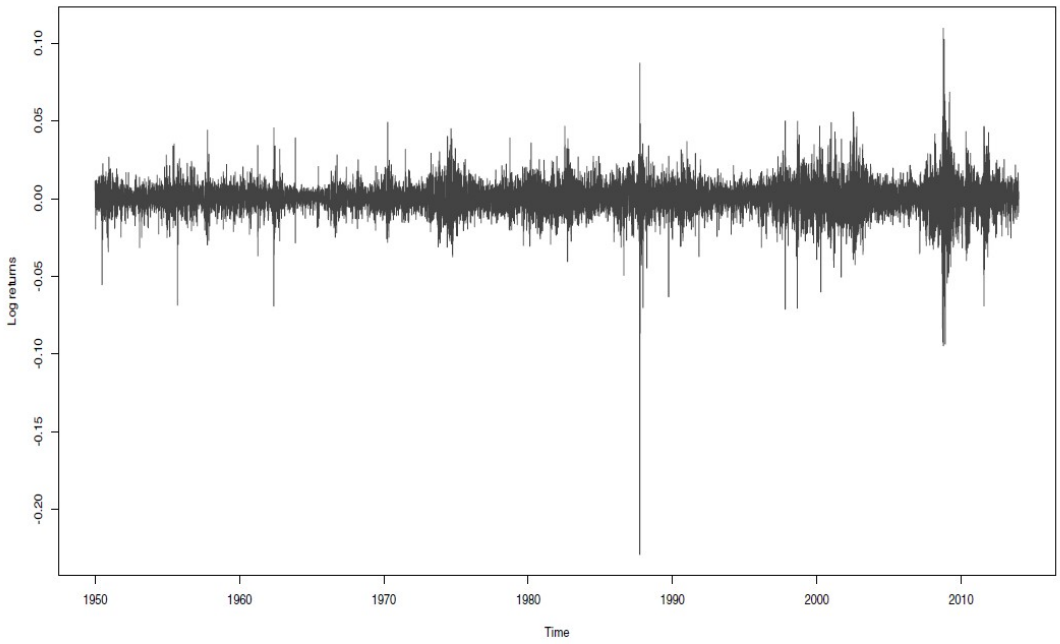


Figure 1.5: S&P500 daily log returns from Jan 1950 to Jan 2014 [40].

1.1.5.2 Stationarity

Stationarity is an essential concept in time series analysis, it implies a statistical **equilibrium** or **stability** in the series i.e the probability laws governing the series behavior remain invariant through time.
We would like to highlight two types of stationarity:

- **Strict stationarity:** a time series is said to be strictly stationary if [36] the *joint distribution* of Y_{t_1}, Y_{t_2}, Y_{t_3},..., Y_{t_n} is the same as the *joint distribution* of Y_{t_1+k}, Y_{t_2+k}, Y_{t_3+k},..., Y_{t_n+k} for all time points t_i and all lags k.
- **Weak stationarity**: a time series is said to be **weakly** or **second-order** stationary if [36] :
  (a) The expected value of the time series is time independent.
  (b) The *autocovariance function* $Cov(y, y_k)$, for any lag k is a function of k : $\gamma_y(k) = Cov(y, y_k)$.

However *strict stationarity* is hard to verify empirically [58], therefore in the literature stationarity often refers to *weak stationarity*.

1.1.5.3 Linearity

A time series is said to be *linear* if it can be written as [58]

$$y_t = \mu + \sum_{i=0}^{\infty} \psi_i a_{t-i} \tag{1.3}$$

Where μ is the mean of y_t, a_t is a sequence of independent and identically distributed random variables with mean zero and a finite variance, also known as *white noise*.

1.1.6 Types of time series

Several components and features discussed in the previous section, in addition to other unmentioned ones, serve as criteria for classifying time series. We would like to highlight the following classifications :

1.1.6.1 Stationary vs Nonstationary

A time series is considered *stationary* or *nonstationary* depending on whether it satisfies the stationarity condition (see 1.1.5.2) or not.

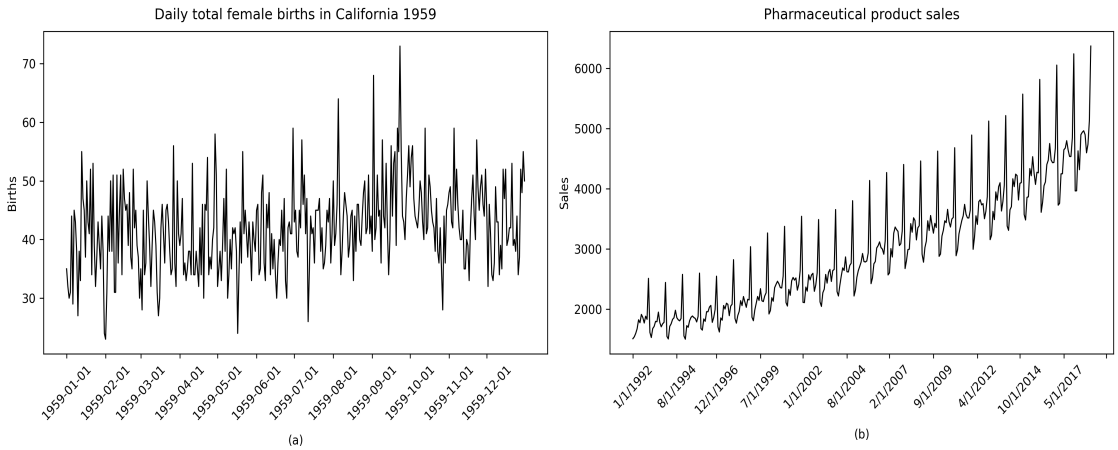


Figure 1.6: An example of stationary (a) and nonstationary (b) time series.

1.1.6.2 Linear vs Nonlinear

The linearity of a time series depends on the type of the relationship between its observation values which can be either *linear* or *nonlinear*. A formal definition of linearity is given in section 1.1.5.3.

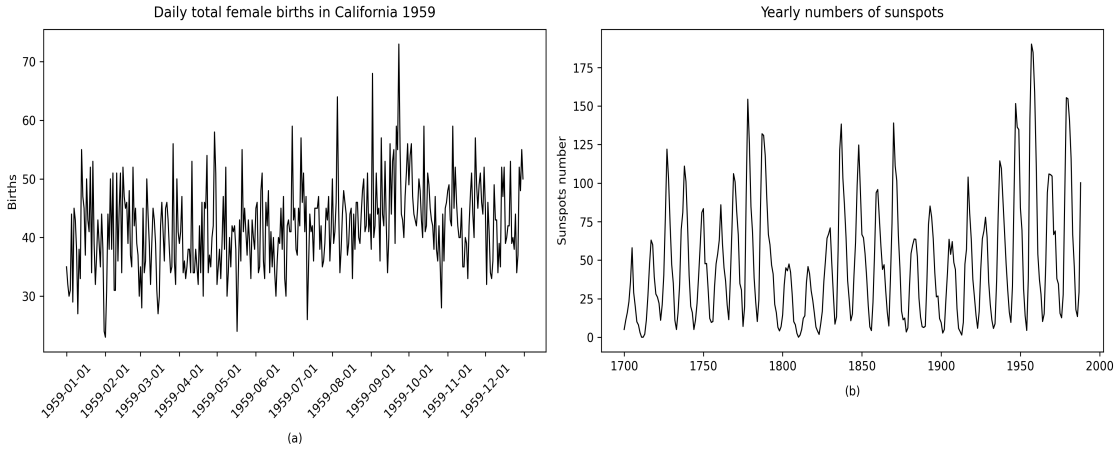


Figure 1.7: Linear (a) and nonlinear (b) time series.

1.1.6.3 Seasonal vs Nonseasonal

In time series analysis, many studied processes manifest seasonal patterns while others don't, this contrast creates an obvious distinction between the two classes *seasonal* and *nonseasonl* time series.

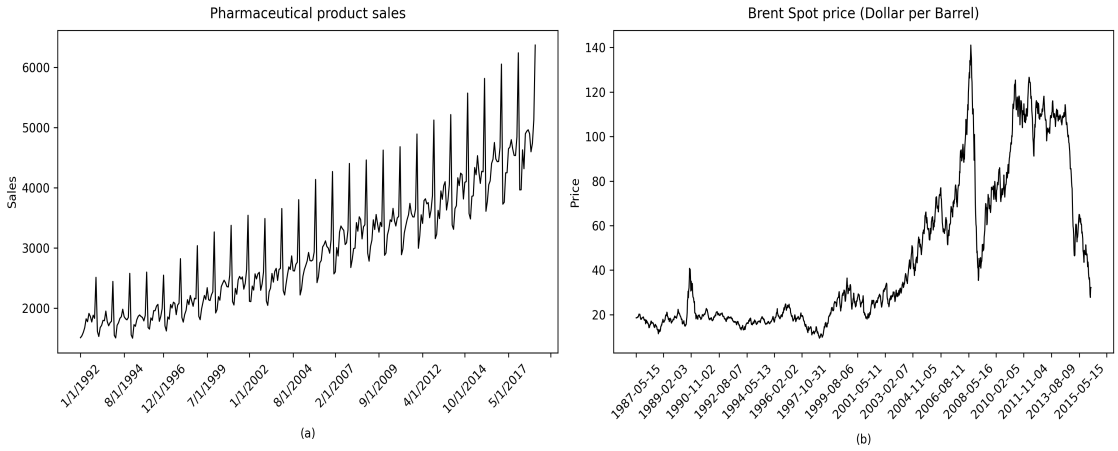


Figure 1.8: Seasonal (a) and nonseasonal (b) time series.

1.1.6.4 Univariate vs Multivariate

While *univariate* time series analysis consists of analyzing and modeling a **single** variable, in many fields such as business, economics and engineering, it is often practical and even necessary to consider **multiple** variables as they may be **contemporaneously related**. As a result *multivariate* time series analysis studies and models multiple series jointly in order to [7] :

- To understand the dynamic relationships among them.
- To improve forecasts accuracy for individual series.

A general definition is given as the following [56]:

$$z_t = (z_1 t, \ldots, z_k t)' \tag{1.4}$$

Where z_t is a vector of k univariate time series observed at equally spaced time points.

1.2 Time series forecasting

1.2.1 Definition

Forecasting is one of the main goals of time series analysis, it leverages statistical and machine learning methods to make predictions using historical and present data.

1.2.2 Applications

Forecasting spans a variety of fields and it is critical for *planning* and *decision making*, to name a few :

- **Meteorology:** Probably one of the most common applications is weather conditions prediction such as temperature, precipitation, wind speed..., the forecasts are crucial for planning for extreme weather conditions and natural disasters, more over they help make informed decisions in different sectors such as agriculture and transportation.
- **Economics :** Monetary and fiscal policy as well as budget and strategic planing require governments and financial institutions to forecast major economic variables such as production, consumption, job growth, unemployment, inflation, population growth...
- **Industrial process control:** The forecasting of quality characteristics in a production process can aid in making decisions on when to adjust or shut down the process. Control schemes utilize predictions of process output for monitoring and adjustment.
- **Finance and risk management:** Investors seek to predict the returns on their financial assets, ranging from stocks and bonds to commodities and currency exchange rates. The ability to forecast volatility is also critical to evaluating investment portfolios risk and pricing financial derivatives.

1.2.3 Types of time series forecasting

Let h be the *forecast horizon* or *lead time*, given a general model of forecasting as the following

$$\hat{y}_{t+h} = f(y_t, y_{t-1}, y_{t-2}, \dots) + \epsilon_{t+h} \tag{1.5}$$

Where $y_t, y_{t-1}, y_{t-2}, \dots$ are the elements of the time series up until the time point t and ϵ_{t+h} is the error, we distinguish two types of forecasting:

1.2.3.1 One step ahead forecasting

That is $h = 1$, this method is straight forward and utilizes the preceding values to anticipate the next value in the series.

1.2.3.2 Multi-step ahead forecasting

In this case $h > 1$, the forecasts can be performed in two ways :

a. Iterative

Also known as recursive, it consists of chaining multiple one step ahead forecasts, where each forecast is calculated using the previous ones.

$$\begin{aligned}
\hat{y}_{t+1} &= f(y_t, y_{t-1}, y_{t-2}, \dots) + \epsilon_{t+1} \\
\hat{y}_{t+2} &= f(\hat{y}_{t+1}, y_t, y_{t-1}, y_{t-2}, \dots) + \epsilon_{t+2} \\
&\vdots \\
\hat{y}_{t+h} &= f(\hat{y}_{t+h-1}, \hat{y}_{t+h-2}, \dots, y_t, y_{t-1}, \dots) + \epsilon_{t+h}
\end{aligned} \tag{1.6}$$

b. Direct

Unlike the iterative, the direct approach forecasts h-values using only the observed values by estimating a model per horizon [54].

1.3 Time series forecasting models

1.3.1 Statistical models

In this section, two assumptions are made, the time series in hand is assumed to be **linear** and **stationary** unless otherwise stated.

1.3.1.1 Autoregressive models

Denoted as **AR**, is a class of simple models that represent the time series as a *linear combination* of its past p values of the series plus *white noise*, **AR(p)** a general autoregressive process of **order p** is written as

$$y_t = \phi_0 + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t \tag{1.7}$$

Where the ϕ_i terms are the weights of y_i consecutive observations and ϵ_t is white noise [58].

1.3.1.2 Moving average models

Another class of simple models is the moving average class, denoted as **MA**, which is similar to the AR class, but instead of regressing on the series observations, MA models represent the time series as a *linear combination* of the present and previous noise terms, a general moving average model of **order q** denoted as **MA(q)** can be written as

$$y_t = \epsilon_t - \theta_1 \epsilon_{t-1} - \theta_2 \epsilon_{t-2} - \cdots - \theta_q \epsilon_{t-q} \tag{1.8}$$

Where θ_i, $i \in \{1, \ldots, q\}$ are the coefficients of the ϵ_i noise terms.

1.3.1.3 Autoregressive moving average models

In practice, modeling processes with AR or MA methods may produce a model of a high order, which tend to be hard to work with due to the large number of parameters to estimate [58], one solution to this is to combine both models in order to get a parsimonious representation [7], denoted as **ARMA**, a general model of **order (p, q)** can be written as

$$y_t = \phi_0 + \sum_{i=1}^{p} \phi_i \, y_{t-i} \; + \; \epsilon_t \; - \sum_{i=1}^{q} \theta_i \, \epsilon_{t-i} \tag{1.9}$$

1.3.1.4 Autoregressive integrated moving average models

The models we have discussed so far are only applied under the **stationarity condition**, however, in practice, most of the time series data tend to be **nonstationary** and require *transformation to stationarity*. One solution to this problem is **differencing**, the first difference of a time series y_t is defined as $\nabla y_t = y_t - y_{t-1}$, and generally the dth difference is $\nabla^d y_t = \nabla \left(\nabla^{d-1}(y_t) \right)$.

An autoregressive integrated moving average model of **order** p, q and **d** of a given time series y_t, is **a stationary ARMA(p, q) process of its *dth* difference**, it is called *integrated* due to the ability of reconstructing the series by summing or *integrating* the differences. The model is denoted as **ARIMA(p,d,q)** and can be written as

$$y_t = \sum_{i=1}^{p+d} \varphi_i y_{t-i} + \epsilon_t - \sum_{i=1}^{q} \theta_i \epsilon_{t-i} \tag{1.10}$$

Where $\boldsymbol{\varphi_i}$ result from $\boldsymbol{\phi_i}$ by applying differencing on the series.

1.3.1.5 Seasonal autoregressive integrated moving average models

While ARIMA can efficiently model a wide range of real world time series, it is still unable to handle seasonal data and usually require a *seasonal adjustment* on the data to remove seasonality, an extension to the ARIMA model has been proposed as a solution, that is **Seasonal ARIMA**. By introducing another type of differencing operation, **seasonal differencing** $\nabla_s y_t = y_t - y_{t-s}$ where s denotes *the seasonality period*, seasonal ARIMA, abbreviated as **SARIMA**, splits the data into *regular* and *seasonal* data.
Then, it applies a regular arima process on the regular data and, respectively, seasonal AR of **order P**, seasonal differencing of **order D** and seasonal MA of **order Q** on the seasonal data, at last both models are combined, typically via a **multiplicative model**, the result is denoted as $\boldsymbol{ARIMA(p,d,q) \times (P,D,Q)_s}$

1.3.1.6 Autoregressive conditional heteroscedastic models

In econometrics, particularly in asset pricing and risk management, data often exhibit **volatility** (Sec 1.1.5.1) also known as **heteroscedasticity** which violates one of the conditions the models mentioned so far have been relying on, that is the invariability of the error component $\boldsymbol{\epsilon_t}$ variance.

An autoregressive conditionally heteroscedastic model of **order p**, denoted as **ARCH(p)**, assumes that the time series is serially uncorrelated but dependent and that the dependence can be described by a simple quadratic function [36, 58], the model is formulated as

$$\begin{aligned} y_t &= \sigma_t e_t, \\ \sigma_t^2 &= \alpha_0 + \alpha_1 y_{t-1}^2 + \alpha_2 y_{t-2}^2 + \cdots + \alpha_p y_{t-p}^2 \end{aligned} \tag{1.11}$$

Where $\boldsymbol{e_t}$ is a sequence of independent and identically distributed random variables with *mean zero* and *variance 1* and $\boldsymbol{\sigma_t}$ denotes *the conditional variance* of $\boldsymbol{y_t}$ with $\boldsymbol{\alpha_0} > 0$ and $\boldsymbol{\alpha_i} \geq 0$.

1.3.1.7 Generalized autoregressive conditional heteroscedastic models

Empirically the **ARMA** model demonstrated a need to a high number of parameters to be able to model volatile processes adequately [7], **generalized ARCH** also denoted as **GARCH**, is an extension to the ARCH model that have been established to solve this issue, it assumes that the *conditional variance* of the time series σ_t does not depend only on the p previous observations of the series but also on the q **previous values of the conditional variance**, a **GARCH(p,q)** model of orders **p, q** is written as

$$\begin{aligned} y_t &= \sigma_t e_t, \\ \sigma_t^2 &= \alpha_0 + \alpha_1 y_{t-1}^2 + \alpha_2 y_{t-2}^2 + \cdots + \alpha_p y_{t-p}^2 \\ &\quad + \beta_1 \sigma_{t-1}^2 + \beta_2 \sigma_{t-2}^2 + \cdots + \beta_q \sigma_{t-q}^2 \\ \sigma_t^2 &= \alpha_0 + \sum_{i=1}^{p} \alpha_i y_{t-i} + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2 \end{aligned} \tag{1.12}$$

1.3.2 Machine learning models

Machine learning techniques and algorithms have gained popularity in time series analysis and forecasting due to their capability of capturing nonlinear relationships in time series data. In this section we will explore two of the most commonly used techniques: support vector regression and regression trees.

1.3.2.1 Support vector regression

Support vector regression or **SVR** is an extention to the **Support Vector Machine** classification algorithm [69], while SVM classification uses a *hyperplan* to separate the training data in a *multidimensional feature space*, SVR computes the *generalization error bounds* by introducing an ε-insensitive loss function that allows an error tolerance known as the **ε-insensitive tube** where errors below ε aren't penalized [1]. The aim of SVR is to minimize, via optimization, the generalization error bounds i.e making the ε-insensitive tube as narrow as possible [39, 69].
The regression function is given as [39]

$$J = w^T \phi(x) + b \tag{1.13}$$

Where ϕ is the feature of the input , w is the weight vector and b is the bias which are estimated by minimizing the regularized error function written as

$$J = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{M} L\left(y_i, f(x_i)\right) \tag{1.14}$$

Where

$$L = \begin{cases} 0 & if \quad |y_i - f(x_i)| \leq \varepsilon \\ |y_i - f(x_i)| - \varepsilon & otherwise \end{cases} \tag{1.15}$$

Where x_i are the inputs, y_i are the actual outputs, $f(x_i)$ is the predicted output for x_i, ε is the difference between the actual and predicted value, C is the regularization parameter and L is the linear ε-insensitive loss function [1, 39, 69].

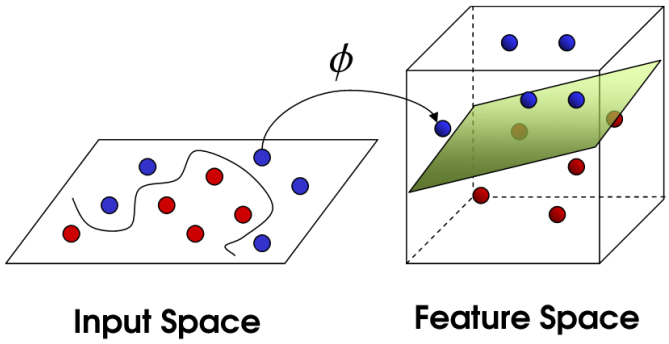


Figure 1.9: Transformation of the input space into a higher dimensional feature space using a mapping function ϕ.

As shown in Figure 1.9, to allow SVR to handle nonlinear data, a *non linear kernel* is used to map the input space where no linear separation of the data is possible to a *higher-dimensional feature space* where it can be linearly separated, this is also know as the "kernel trick" [69].

SVMs and SVR have known success in many application fields such as : pattern recognition, text classification, image recognition, **time series forecasting**, bioinformatics and more [47, 70].

1.3.2.2 Classification and Regression Tree

Classification and Regression Tree also known as **CART** is a supervised learning algorithm in data mining which aims to construct a binary tree by recursively partitioning the initial data into nonoverlapping subregions [14], in a regression case, we would like to predict the value of a **dependent variable** Y based on **predictor variables** X. The resulting tree consists of a *root* node plus *internal* and *terminal* nodes, where internal nodes represent a given condition of whether or not a split should be performed at this step, the condition consists of a predictor variable and its threshold value which are selected by minimizing, typically, the least square error in predicting the value of the dependent variable Y, the terminal nodes or *leaves* contain the regressed values [14].
The process of building a regression tree is carried out in three steps [11]

1. Constructing the maximum tree.
2. Pruning the tree.
3. Performing predictions with new data.

CART is widely used in many fields such as medicine, economy, ecology and many more due to its ability to process a variety of data types be it numerical or categorical or both as well as being able to handle linearity and nonlinearity relationships in the data [14].

Figure 1.10 shows a time plot of the quarterly growth rate of US real gross domestic product from 1947 to 2015 and Figure 1.11 shows an example of a fitted regression tree for the GDP rates data in order to predict the next growth rate value y_t based on its three past values y_{t-1}, y_{t-2} and y_{t-3} [57] which can be mathematically expressed as

$$y_t = g(y_{t-1}, y_{t-2}, y_{t-3}) + a_t \tag{1.16}$$

Where the tree represents a piecewise nonlinear approximation of g and a_t represents the error term [57].

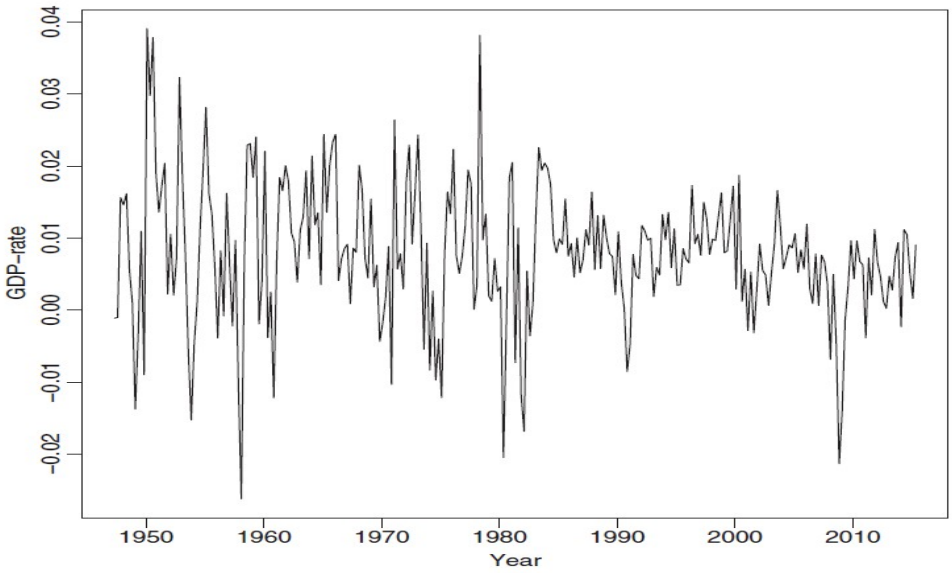


Figure 1.10: The quarterly growth rates of US real gross domestic product from 1947 to 2015 [57].

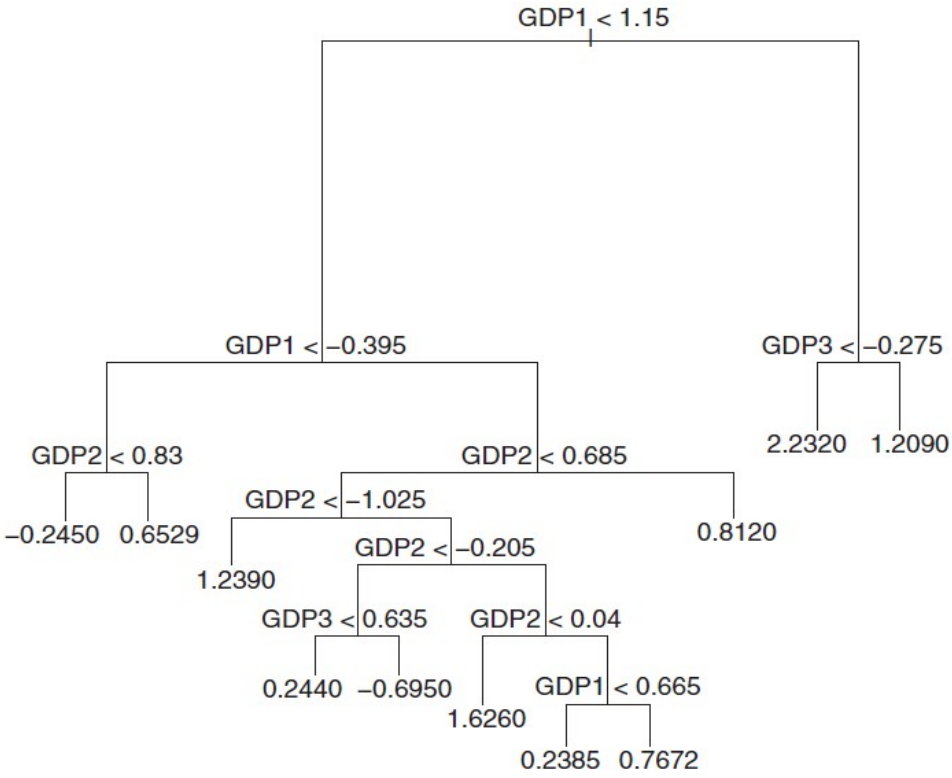


Figure 1.11: A regression tree for the quarterly growth rate of US real gross domestic product using three lagged values [57].

1.3.3 Deep learning models

Deep learning has been a subject of great interest in the last two decades and has been heavily studied and explored, in this section we will see how time series analysis takes advantage of deep learning's power and we will highlight some of the widely used techniques and algorithms in the literature.

1.3.3.1 Recurrent neural networks

Recurrent neural networks or **RNNs** are a *dynamic* type of neural networks [28] that are best suited for modeling sequential data [50] including time series data. The model's architecture allows it to capture the inherent order and dependence where the output value doesn't depend only on the current input but also on the previous values computed and stored in the *hidden state vector*, and therefore it is said that the model maintains *internal memory* [63]. As shown in the figure, the RNN model can include

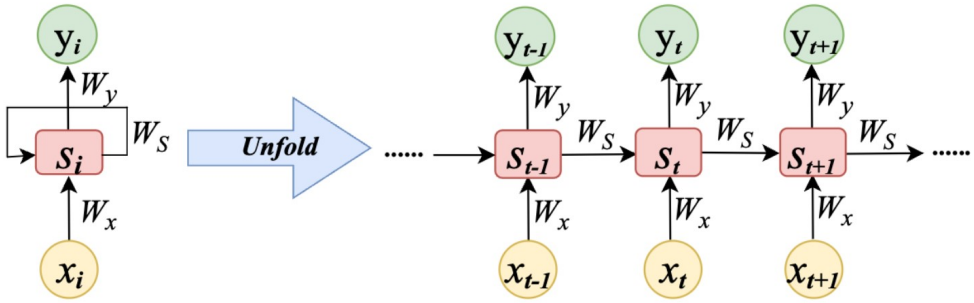


Figure 1.12: Folded and unfolded structure of an RNN [50].

multiple inputs and produce multiple outputs, one per time step, this input/output dynamic allows it to operate in three different ways [42] :

- One to many : as in image captioning.
- Many to one : as in sentiment analysis where the input is a given sentence.
- Many to many : as in video classification, a label for each frame.

Figure 1.10 shows the a single RNN unit/cell, the computation process is given mathematically as the following:

$$\begin{aligned} S_t &= \sigma_h(W_{xs} \cdot (x_t \oplus S_{t-1}) + b_s) \\ y_t &= \sigma_y(W_y \cdot S_t + b_y) \end{aligned} \tag{1.17}$$

Where x_t is the input vector, S_t is the hidden state, W's and b's are the learned parameters (weights and biases respectively) and the σ's are activation functions.

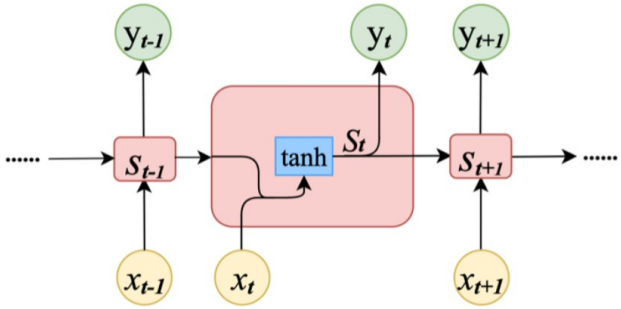


Figure 1.13: RNN cell structure and computation process [50].

RNNs can be trained using the **backprobagation through time** algorithm [49], the algorithm unrolls the RNN, sums the errors calculated for each time step, applies an optimization algorithm such as **SGD, ADAM ...etc** to adjust the weights accordingly and finally it folds the RNN back to its original form [42, 49].

The vanishing and exploding gradient problem

The main drawback with RNNs is that for long sequences processing and due to the repeated multiplication of weights through time steps, the gradient can become too small or too large and causes the RNN to lose the long-term information/dependencies, this is known as the *vanishing/exploding gradient problem* [50, 63], the LSTM model was specifically developed to address this problem.

1.3.3.2 Long short-term memory neural networks

The Long Short-Term Memory or **LSTM** model was developed specifically to address the vanishing/exploding gradient problem and allows the network to learn the long term dependencies in the data [67]. This is achieved by introducing a new, more sophisticated and complicated cell structure shown in Figure 1.11.
In a LSTM network, information flows in two main streams, long and short term memory streams, which are controlled and manipulated by the **gating mechanism**, LSTM cells consist typically of three different gates [67] :

- **The forget gate :** f_t is the gate responsible of determining, based on the current input S_{t-1}, how much of the previous *cell state* C_{t-1} should be *remembered*, an output of 1 *retains* all the previous information while a value of 0 means to totally *forget* the previous information.

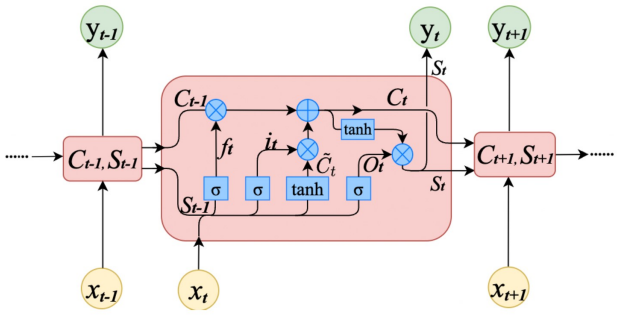


Figure 1.14: LSTM cell structure and computing process [50].

- **The input gate :** i_t creates new potential values $\tilde{C}_t$ to be added to the cell state based on the current input value and the previous hidden state.
- **The output gate :** O_t is the final gate, based on the current input and the previous hidden state, it operates on the information resulted form the previous two gates and decide how much of the the current cell state it is going to output as the **next hidden state** and eventually the *final output* of the network.

The cell state represents the long term memory and the hidden state represents the short term memory, this unit model can be mathematically represented as [50]:

$$\begin{aligned} f_t &= \sigma\left(W_f \cdot (S_{t-1} \oplus x_t) + b_f\right) \\ i_t &= \sigma\left(W_i \cdot (S_{t-1} \oplus x_t) + b_i\right) \\ \tilde{C}_t &= tanh\left(W_c \cdot (S_{t-1} \oplus x_t) + b_c\right) \\ C_t &= f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \\ O_t &= \sigma\left(W_o \cdot (S_{t-1} \oplus x_t) + b_o\right) \\ S_t &= O_t \cdot tanh(C_t) \\ y_t &= \sigma(W_y \cdot S_t + b_y) \end{aligned} \tag{1.18}$$

Where x_t is the input vector, W's and b's are the learned parameters (weights and biases respectively), C_t is the cell state , S_t is the hidden state, σ is the sigmoid activation function, tanh is the hyperbolic tangent activation function, f_t, i_t, O_t are the forget, input and output gates respectively.

1.3.3.3 Gated recurrent unit

Gated recurrent unit **GRU** is a variant of the LSTM model [67] that was conceived as an alternative in order to reduce the computational cost of the LSTM cell

without a significant loss in its performance and while still being able to handle the vanishing/exploding gradient problem. GRU is considered a *lighter* LSTM [67] and the complexity reduction is achieved by integrating the input and forget gates of the LSTM cell into one gate called the **update gate** which, in addition to the second gate the **reset gate**, constitutes the GRU and reduces the total number of the unit parameters [67].

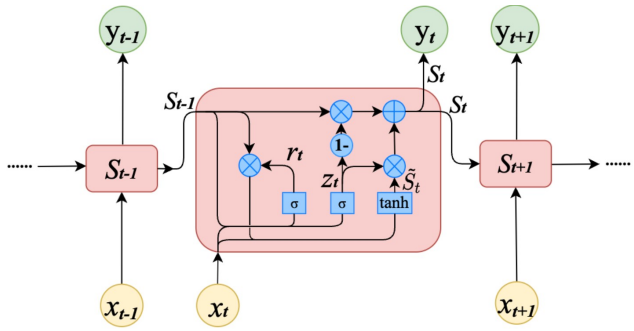


Figure 1.15: GRU structure and computation process [50].

The GRU is mathematically expressed as the following [50]

$$
\begin{aligned}
r_t &= \sigma\left(W_r \cdot (S_{t-1} \oplus x_t) + b_r\right) \\
z_t &= \sigma\left(W_z \cdot (S_{t-1} \oplus x_t) + b_z\right) \\
\tilde{S}_t &= tanh\left(W_s \cdot (S_{t-1} \cdot r_t \oplus x_t) + b_s\right) \\
S_t &= (1 - z_t) \cdot S_{t-1} + z_t \cdot \tilde{S}_t \\
y_t &= \sigma(W_y \cdot S_t + b_y)
\end{aligned}
\tag{1.19}
$$

The recurrent neural network class (vanilla RNN, LSTM, GRU) has a great ability to process sequential data and had a tremendous success in many applications such as [28] :

- Time series forecasting.
- Language modeling.
- Image and video captioning.
- Robotic control.

Conclusion

The diversity of time series types and features makes forecasting a complicated task with no known universal model. This lead to the development of numerous methods and techniques both statistical and artificially intelligent, each of which has its strengths and weaknesses.

In this chapter we have defined what time series are, their types, components and important features. Furthermore we introduced time series forecasting and detailed its types and applications. Finally we presented some of the commonly used techniques in time series analysis and forecasting, in the next chapter we will present state of the art models and see how it is possible to combine two or more models to improve forecasting results by taking advantage of each model's strength and compensating for their limitations.

Chapter 2

Literature review of time series forecasting models

Introduction

Given to the crucial role time series forecasting plays across various domains and the impact it has on decision making, enhancing forecast accuracy became an undeniable imperative. The subject was consequently intensively studied in the last few decades yielding a rich literature that encompasses several models originating from across multiple study fields including : statistics, machine learning and deep learning. To address these models' limitations and to further improve forecasting accuracy, researchers explored the potential of combining existing models, the advantages it offers and the challenges it comes with.

The aim of this chapter is to provide a comprehensive and up-to-date review of state-of-the art models in the time series forecasting literature. The review is organized into three sections, in the first section, we begin by examining classical techniques along with other well-established machine learning approaches. The second section delves into the advancements in deep learning models for time series forecasting and present a detailed overview of these models. In the final section we shed the light on hybrid models and we discuss the challenges and considerations in regard to this approach.

2.1 A survey of other forecasting methods

Expanding on the models introduced in the preceding chapter, this section delves into an exploration of three notable non-deep-learning methodologies employed in time series forecasting. These models, hailing from distinct perspectives including statistics, machine learning, and Bayesian analysis, are highlighted and discussed in detail to shed light on their respective approaches.

2.1.1 Exponential smoothing

Exponential smoothing is yet another family of traditional statistical methods for time series analysis and forecasting and is one of the earliest set of techniques to be developed in the literature [66]. Exponential smoothing splits the series into *signal* and *noise* components, where the signal represents patterns caused by intrinsic dynamics (trend, seasonality) while the noise represents any irregular variations in the data [36, 66]. For forecasting tasks, Exponential smoothing applies a weighted average on the current and previous observation to generate forecasts, this is accomplished by allowing observations to influence the new value depending on their position in the historical data timeline, the further an observation is from the present the less influence it has on the forecast, therefore the weight assigned to each value is increased *exponentially* as the observation chronologically approaches the current observation of the series [66].

2.1.2 XGBoost

XGBoost, known as eXtreme Gradient Boosting, is a powerful supervised machine learning algorithm that implements the gradient boosted trees technique. It is specifically engineered for scalability and efficient computation [10], making it suitable for handling large-scale datasets. Taking part of the *ensemble learning* methods family, XGBoost harnesses the collective strength of multiple weak learners, namely classification and decision trees (CARTs), to attain superior predictive performance beyond what each base learner can achieve individually [5].

The fundamental concept underlying XGBoost is the iterative enhancement of the model's predictive capabilities by sequentially adding and training decision trees. At each iteration, XGBoost intelligently assesses the errors made by the ensemble of previously trained trees and focuses on rectifying those errors with the introduction of new successor trees [19]. This iterative process allows the model to capitalize on the strengths of each weak learner and gradually refine its predictions, ultimately leading to enhanced performance and heightened precision in making accurate forecasts.

2.1.3 Facebook Prophet

Prophet is a time series forecasting model and an open-source library developed by Facebook's Core Data Science team and is available in R and Python languages. It serves as a tool for both experts and non-experts to produce high quality forecasts at *scale* by providing a set of intuitive parameters that can be adjusted with having minimal knowledge of the details of the underlying model [55].

Prophet is an *additive regression* model based on time series decomposition with three core components: trend, seasonality and holidays [55] described in the following equation:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \tag{2.1}$$

Where $g(t)$ is the trend component which models non-periodic changes in the values of the time series, $s(t)$ represents the seasonal component that models periodic changes in the values of the series, $h(t)$ models the holidays effect which may occur irregularly over variant time periods and are manually provided by the user as input to the model, lastly ϵ_t is the error term representing any changes that are not accommodated by the model. The trend component is modeled in two ways by Prophet :

- **Nonlinear, Saturating Growth :** which is modeled by logistic growth, which in its most basic form is given as

$$g(t) = \frac{C}{1 + exp(-k(t - m))} \tag{2.2}$$

  Where C is the carrying capacity, k is the growth rate and m is an offset parameter. When considering a time varying carrying capacity and a non-constant growth rate, the piece-wise logistic growth model is written as

$$g(t) = \frac{C(t)}{1 + exp(-(k + a(t)^\intercal\delta)(t - (m + a(t)^\intercal\gamma)))} \tag{2.3}$$

  Where a, δ and γ are vectors of rate adjustments and changepoints.

- **Linear trend with changepoints :** Covers cases where the series doesn't exhibit saturating growth which is often useful [55], the piece-wise linear model becomes parsimonious and is given as

$$g(t) = (k + a(t)^\intercal\delta)t + (m + a(t)^\intercal\gamma) \tag{2.4}$$

Prophet grew popular in the data science community due to its out of the box tools for forecasting and its great ability to handle outlines, missing values and dramatic changes in the time series.

2.2 An overview of deep learning techniques for time series forecasting

In recent years, several breakthroughs have been achieved in various fields and approaches to many complex tasks have been revolutionized thanks to deep learning advancements, time series forecasting was no exception to this revolution. Motivated by their ability to model non-linearity, to extract high level features and to handle complex data, deep learning models have been adopted to time series forecasting tasks. In this section we will highlight a collection of state of the art models :

2.2.1 Recurrent neural networks family

Due to their recurrent nature, RNNs are sequential by design which makes them well suited for sequence processing and modeling problems including time series forecasting [50], the key component that empowers RNNs effectiveness is the *recurrent layer* which allows the network to inherently capture the temporal order and dependence of the data [18] while the nonlinear activation functions grant modeling the nonlinear dependency and the complex dynamics of the series, finally, the recurrence mechanism combined with the hidden state ensure that the network *remembers* information from previously processed observations.
For the aforementioned reasons, RNNs, particularly LSTM and GRU architectures have grown popular and became the standard in tackling sequence modeling tasks [4], however these architectures suffer from their own shortcomings that motivated the exploration of other architectures :

- Training difficulty: RNNs are notorious for being hard to train [41], improved variants such as LSTMs and GRUs have more complex cell architectures which introduce a high computational complexity as a result of the increased number of parameters [67] which, by turn, requires longer training time and larger data samples [45].

- Long memory limitations: different time series data can exhibit different types of temporal dependency ranging from short to long term dependencies, while some types of RNNs such as LSTMs can efficiently model short, medium and even moderate long term dependency, their effectiveness seems to drop as the sequences grow longer and therefore these networks eventually fail to capture considerably long term dependencies [71].

2.2.2 Temporal convolutional networks

Convolutional neural networks or CNNs have gained an immense popularity in the

last decade given their notable success in different computer vision related tasks such as *image classification*, *object detection* and *image segmentation* as well as reaching state of the art performance in natural language processing tasks including *machine translation*, *character* and *word level language modeling* [4]. Driven by this success, investigations have been conducted to whether similar results can be achieved by applying CNN architectures in time series analysis.
Temporal convolutional networks or **TCNs** are a recent network architecture that have been originally developed for *audio synthesis* [59] and later adopted for *action segmentation* [27] and *time series forecasting problems* and showed promising results [26, 30], they are based on ***one dimensional dilated causal convolution*** and ***residual blocks*** which we will discuss in detail in this section.

2.2.2.1 Causal convolution

In order to respect the temporal order and avoid information leakage from future values, TCNs use a special type of convolution known as *causal convolution* where an output at time step t is obtained using only inputs from time steps prior or equal to t [4].

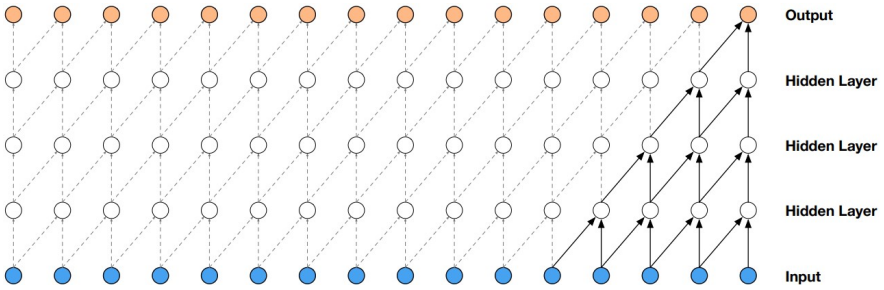


Figure 2.1: Visualization of a stack of causal convolution layers [59].

2.2.3 Dilated convolution

In time series forecasting and in sequence modeling in general, it is crucial to the model's performance and accuracy to be able to capture dependencies that may span lengthy ranges, to give TCNs the ability to effectively handle history of large size *dilated convolution* is used [4], that is, between every two kernel elements a fixed step of size d is inserted, where $d = 1$ gives the standard convolution as shown in Figure 2.2.
This adjustment in the kernel structure enables an *exponentially large receptive field* [4] and therefore the TCN is able to capture long term patterns in the sequence.

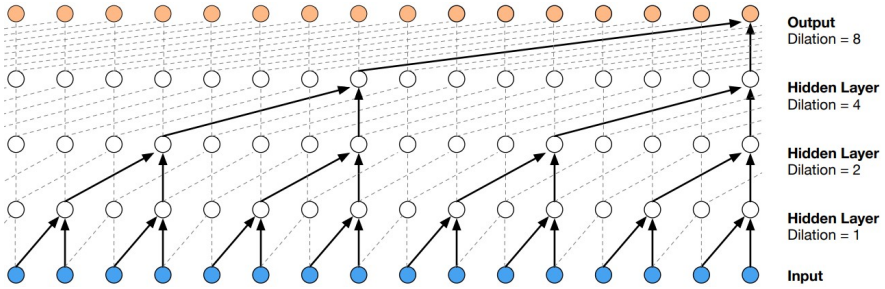


Figure 2.2: Visualization of a stack of dilated causal convolution layers [59].

2.2.3.1 Residual connection

In practice, to further expand the network's receptive field beyond its elementary parameters (kernel size, dilation factor) the depth of the network is increased, however, this increases the learning complexity as well and gives rise to the degradation problem, residual connections were opted for in order to counteract against these drawbacks as they were proven to benefit deep neural architectures [4].
A residual connection or block consists of adding the output of a TCN block to its input x and is generally given as [26] :

$$o = Activation(x + F(x)) \tag{2.5}$$

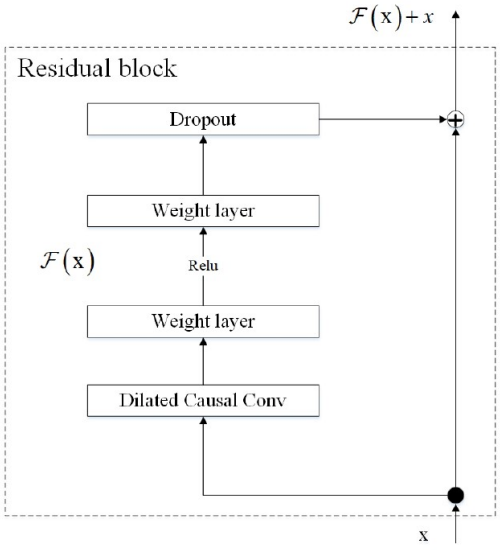


Figure 2.3: An example of a TCN residual block.

A deep temporal convolutional network consists of stacking multiple TCN blocks such as the output of each is the input to next, each block has a stack of dilated causal convolution layers (Figure 2.2) and a residual connection. The TCN blocks preserve the input size by zero padding the input to each convolutional layer. This architecture improves upon the existing RNN class of networks in mainly two ways [4, 26] :

- **Parallelism :** An input sequence can be processed as whole instead of sequentially as in RNNs given that the convolution can be performed concurrently, resulting in a faster training and evaluating time.
- **Stable gradients :** TCNs do not suffer from gradient problems as they are trained using the standard backpropagation algorithm unlike RNNs that rely on the backpropagation through time algorithm.

2.2.4 Transformers

The transformer architecture is an **attention** based model that made a huge success in the NLP domain by outperforming state of the art models in machine translation [60] and other NLP tasks, it empowers large language models such as **OpenAI's ChatGPT** and **Google's Bard**.
Transformers follow the *Encoder-Decoder* structure and are entirely based on the *attention mechanism* while disposing of any recurrence which yielded significant improvements in speed and performance [60]. It wasn't long until this novel architecture and its variants started being exploited in other deep learning areas including time series forecasting [68, 73, 63], in the following, we will present the main components and concepts that make up this powerful model.

2.2.4.1 Attention

The attention mechanism is at the core of the Transformer model and it's the basis of two important layers in the Transformer architecture: **self-attention layer** and **encoder-decoder attention**, it allows the model to attend to different parts in the input sequence and how each element relates to the other elements of the input, this gives the model a broader understanding of the dependencies and the contextual relationships in the sequence.
Attention can be described as a function that maps a set of ***Query, Key*** and ***Value*** inputs to an output [60] and it's given as the following :

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.6}$$

Where

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V \tag{2.7}$$

X is a matrix of the packed input sequence where each row corresponds to a single value of the sequence, W^Q, W^K and W^V are learned matrices and d_k is the dimension of K [60, 17].
In order to take full advantage of the attention mechanism and its ability to be parallelizable, transformer models often use what's called a **Multi-Head Attention**, that is: applying, in parallel, multiple *attention heads* (Equation 2.6) with separate learned matrices, this allows the model to diversely attend to information at different positions form different attention heads.
Figure 2.4 is a visualization of the attention mechanism in the case of time series analysis where each element in a layer is allowed to attend elements in the other layer up to its position only.

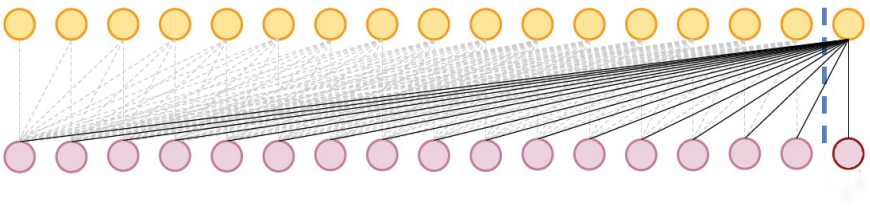


Figure 2.4: Visualization of a restricted attention mechanism.

We like to note that the term **self-attention** mechanism refers to the case where the attention is performed between the sequence and itself, both layers in Figure 2.4 would then correspond to the same sequence, hence the *"self"* reference.

2.2.4.2 Input embedding

Transformer models employ an *input layer* at both the encoder and decoder levels where input elements are mapped to vectors by a *learned embedding* [60].

2.2.4.3 Positional encoding

Unlike RNNs that inherently capture the sequential order and TCNs which use causal convolutions, Transformers do not have recurrence nor convolution and thus they need a way to model order and positioning, therefore **positional encoding** has been introduced as an effective way to insert order information into the input. The canonical Transformer model [60] use *sine* and *cosine* functions to generate positional encodings which are added to the input embedding from the previous layer, the gener-

ating functions are given as the following :

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{2.8}$$

Where i is the positional encoding dimension and d_{model} is the dimension of the input embedding resulting from the input layer, this choice of generating functions allows the model to extrapolate to sequence of variante length including ones longer than sequences encountered during training [60].

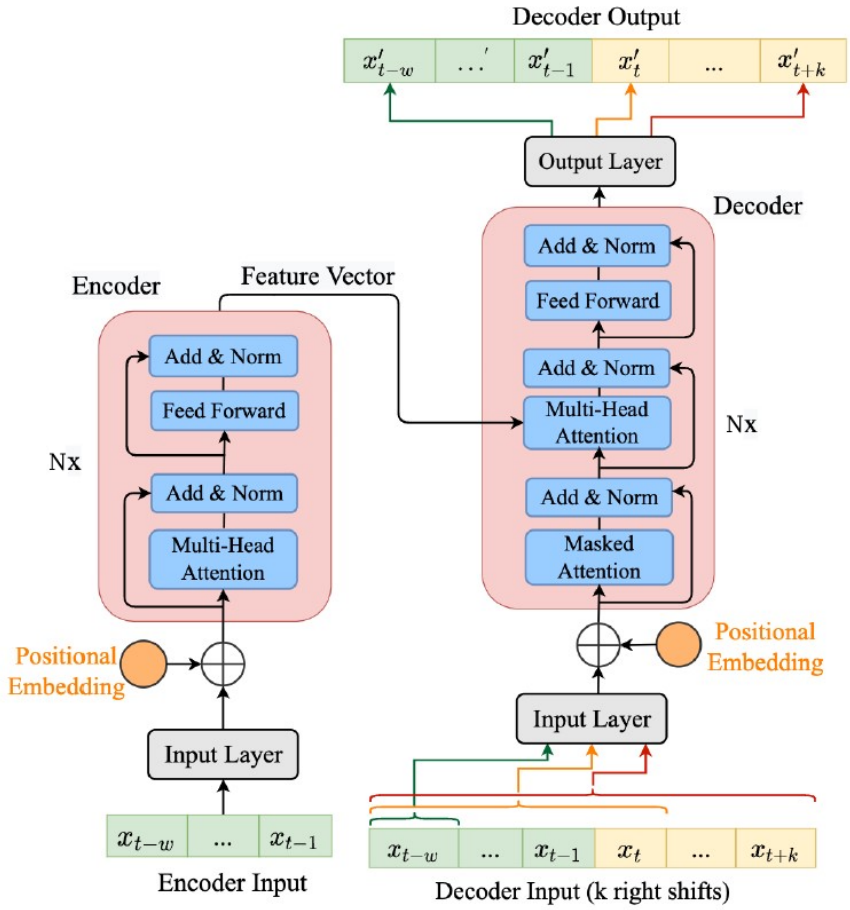


Figure 2.5: Transformer model architecture applied to time series forecasting [50].

2.2.4.4 Encoder

The Transformer encoder structure consists of N stacked identical layers, each of which has two sub-layers, the first being the multi-head self-attention mechanism and

the second is a fully connected feed-forward network, around each of these two layers, a residual connection is applied followed by layer normalization.

2.2.4.5 Decoder

The decoder has a similar architecture to the encoder as it's also composed of N stacked identical layers with one main difference, the decoder has a third sub-layer that applies multi-head attention on the encoders output, another subtle difference resides in the self-attention layer, unlike the encoder the decoder employs a masked multi-head attention to prevent attending to future values provided as its input.

2.2.5 Graph neural networks

Graph neural networks are a subset of the broader field of *geometric deep learning* that attempts to generalize neural networks for *non-Euclidean* data structures such as graphs and manifolds [8], they are designed to operate on graph-structured data and are based on *message passing* techniques to iteratively update each node's state using its previous state and its neighbors', GNNs leverage the expressive power of graphs to address problems where modeling entities, their relationships and their interactions is required, they have been successfully applied to various problems such as molecular chemistry, recommendation systems, intelligent transport systems, social, financial and biological networks. GNNs have also found applications in the field of time series analysis, more extensively in *multivariate time series forecasting*. Numerous studies in the literature have explored the use of GNNs for this task.
In this section, we aim to shed the light on a selection of some of the prominent variants of GNNs :

2.2.5.1 Graph recurrent neural networks

Graph Recurrent Neural Networks (GRNNs) aim to generalize traditional RNNS and extend their capabilities in order to be able handle data with graph structures, they are designed to exploit the power of recurrence to capture temporal dependencies within sequences of varying lengths while also considering the underlying graph structure [46]. To enhance the information propagation and retention across the graph, GRNNs incorporate a gating mechanism inspired by RNNs[31], this gating mechanism significantly improves the ability and effectiveness of the Graph Neural Network (GNN) to exchange and retain relevant information. In GRNNs, the state of each node is recurrently updated by exchanging information with its neighboring nodes through a diffusion mechanism [64]. This mechanism enables the nodes to share and aggregate information from their neighborhoods and thus facilitating the integration of local

and temporal dependencies and ensuring that each node's state reflects the collective influence of its neighboring nodes over time.

2.2.5.2 Graph convolutional neural networks

Analogous to GRNNs, Graph convolutional neural networks (GCNNs) generalize conventional CNNs to graph-structured data, the main idea is to take the convolution operation beyond the regular grid data and apply it on graph data where each node's representation is obtained by aggregating its features vector and its neighbors'. GCNNs often stack multiple layers with different weights for each layer allowing the network to extract high-level node representations [64]. GCNNs are mainly divided into two categories *spectral-based* and *spatial-based* networks.

- **Spectral-based GCNNs :** operate in the spectral domain, they assume the graph to be undirected and represent it by a *normalized graph Laplacian matrix* which is further decomposed to obtain a matrix of eigenvectors that form an orthonormal space. The input signal is then projected to the space from the previous step by receiving the *graph Fourier transform*. Now a convolution operation is performed by multiplying a learnable filter with the transformed signal [64]. Spectral-based GCNNs are computationally expensive due to the eigen-decomposition and several network variants were introduced to improve their computational complexity.

- **Spatial-based GCNNs :** operate in the spatial domain and are algorithmically similar to the well known 2D image convolution, in this context, the representation of each node in the graph is derived by convolving a filter with its previous representation and the neighboring nodes' representations [64]. This convolutional operation propagates information along the graph edges allowing the network to capture local spatial relationships and dependencies which can be further extended to learn complex and more abstract patterns across the graph by employing multiple convolution layers.

2.2.5.3 Graph attention neural networks

Graph attention neural networks (GATs) were introduced as an innovative adoption of the attention mechanism into GNNs [74] as it gained a wide recognition following its remarkable success in various sequence-related tasks. GAT leverages the self-attention mechanism to compute each node's features by attending to its neighbor nodes where each of which is assigned a distinct weight [74]. Similar to the Transformer model, in order to maximize the effectiveness of the attention mechanism and to stabilize the learning process, GAT integrates *multi-head attention* mechanism [61] yielding a diverse and more nuanced comprehension of the data dependencies and relationships.

2.3 Hybrid approach to time series forecasting

As shown in the previous two sections, a variety of models of different categories including statistics, machine learning and deep learning have been developed in the time series forecasting literature. Each of these models possesses its own strengths and weaknesses, some are more suited for short-term forecasting capturing immediate fluctuations with precision and others excel in long-term predictions providing insights into broader horizons, certain models make explicit assumptions about the inherent characteristics of the time series data, such as linearity or stationarity, further tailoring their applicability to specific contexts.
Although the development of new techniques and methods is an active area of research, the emergence of new models is infrequent, and while the new models may improve upon their predecessors they are likely to follow the same pattern of having their own benefits and shortcomings. Therefore, researchers and practitioners have opted for hybrid approaches, attempts to combine multiple models with the aim of leveraging the composing models strong points and overcoming their individual limitations, ultimately obtaining a potentially higher performance beyond what each model can achieve separately.

Some of the important advantages the hybrid approach might offer are:

- Accuracy enhancement: Combining models in a complementary manner can lead to improvements in the forecast performance [36] as the amalgamation of information from different models provides diverse insights into the underlying structure and behavior of the time series.

- Adaptability and Robustness: Different models operate on different types of data and may require specific conditions which narrows their ability to adapt and generalize for different forecasting tasks. Merging multiple models can extend their natural capability of handling data outside of what they usually operate on and performing tasks that they may not be designed for.

In this section we will showcase a selection of hybrid models that combine different types of algorithms and techniques. Later on in this section, we will briefly discuss challenges and consideration associated with designing and building hybrid models.

2.3.1 ARIMA and RNNs combination

ARIMA and RNNs (LSTM and GRU particularly) have long been the go-to for time series analysis and forecasting, both have their strengths and weaknesses, ARIMA lacks the ability to model non-linearity in time series, on the other hand, LSTMs and

GRUs are able to model both linearity and non-linearity, however their training requires a significant amount of time and data [12]. Therefore, it can be said that search for a midpoint is intuitive and the hybridization of ARIMA and LSTM/GRU follows as a natural subsequent step. This combination is common in the forecasting literature and several works have studied and successfully applied it [3, 53, 65, 12].
The main idea behind this hybrid model is to separate the time series into a combination of linear L_t and non-linear N_t components, the linear part is then given to the selected ARIMA model and the non-linear component is handled by the LSTM/GRU network, the predicted values obtained by both models are then added to produce the final forecast :

$$y(t) = L_t + N_t \tag{2.9}$$

This statistical-deep learning combination of two of the most well-known forecasting algorithms demonstrated a better performance than the individual models.

2.3.2 Exponential smoothing and RNNs

Another interesting hybridization of classic and deep learning techniques is the combination of Exponential smoothing (ES) methods and RNNs, one particularly notable framework involved merging the Holt and Winters variation of Exponential smoothing with dilated LSTMs. This framework emerged victorious in 2018 in the prestigious M4 forecasting competition. The combination leverages the ES ability to disassemble the time series and capture its main components (trend and seasonality) while serving as an instrument to preprocess and deseasonalize the series, the RNN contributes by capturing long-term dependencies and modeling non-linearity in the data [51].
Driven by this success, numerous works were based on this framework and variants from both methodological classes have been effectively combined and applied to various forecasting tasks, including short and midterm load forecasting, crime prediction, stock market analysis, and more [24, 9, 52, 51, 13].

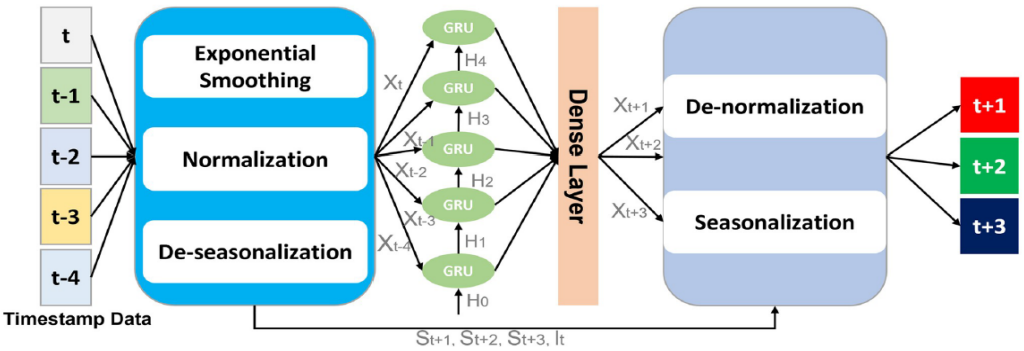


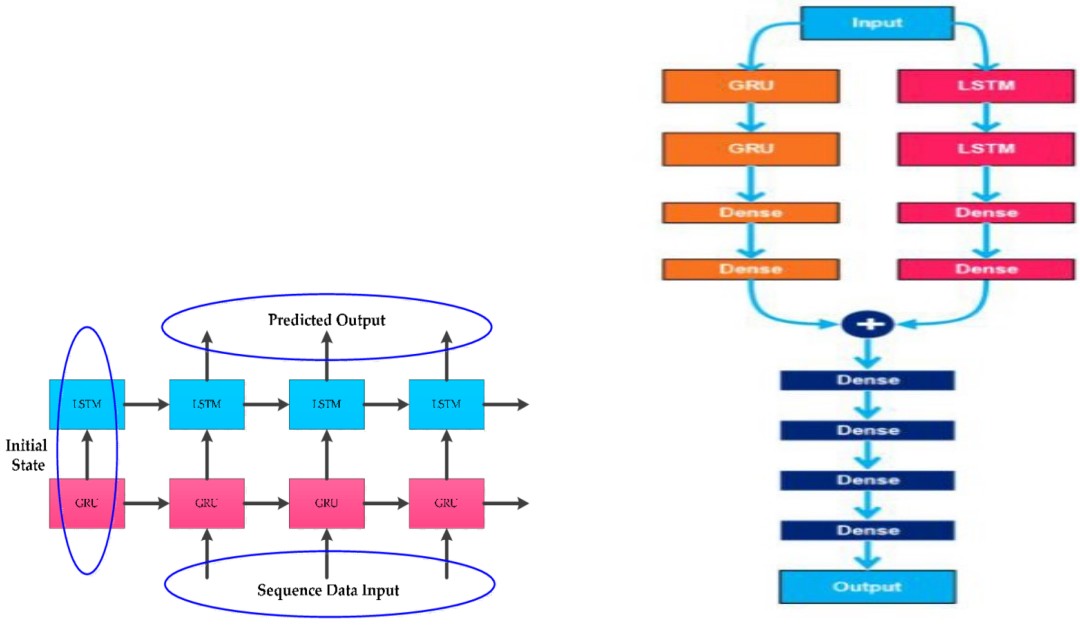
Figure 2.6: An example of hybrid ES-RNN framework [9].

2.3.3 LSTM and GRU hybridization

LSTM and GRU neural networks are two of the most commonly used deep learning models for time series forecasting due to their capability of memorizing long information sequences and capturing complex non-linear patterns in the data. Researchers have explored the coupling of the two architectures and investigated the performance and results [21, 37, 20, 29, 48].
The framework involving both architecture is theoretically conceivable in several way, the two frequent amongst are :

- Stacked combination: this design approach consists of stacking GRU network layers on top of the LSTM network layers or vice-versa, thus the output of one block serves as an input to the other creating an hierarchical structure as depicted in Figure 2.7a.

- Parallel combination: as shown in Figure 2.7b, this architecture separates the LSTM block from the GRU block, both receive the same input and perform their own computations in parallel, using an aggregation function of choice, the outputs from each block are then combined for further processing or to directly obtain the final forecasting results.



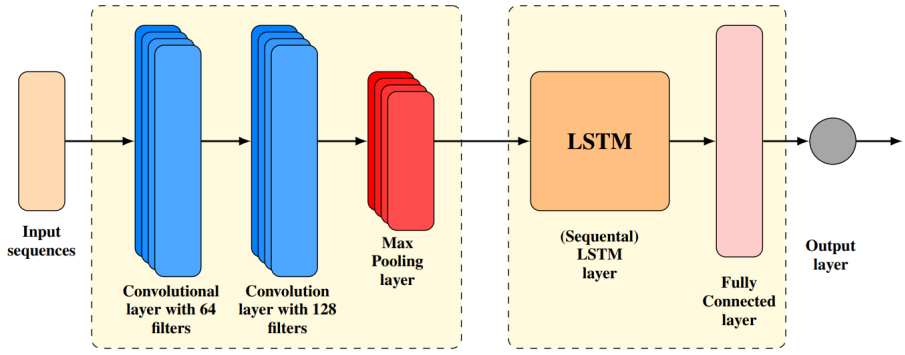
(a) (b)

Figure 2.7: LSTM-GRU hybrid model: (a) Stacked combination [16], (b) Parallel combination [48].

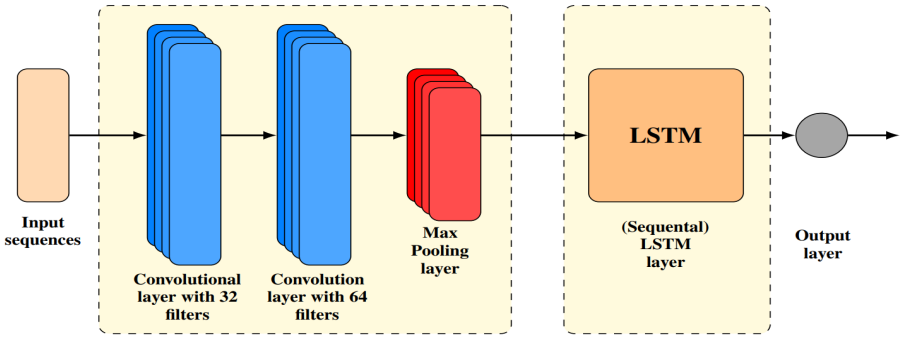
The hybrid model achieved superior performance over individual LSTM/GRU networks.

2.3.4 CNN-LSTM/GRU

This hybrid model exploits the CNN ability to extract high-level features from the data while the LSTM/GRU layers are employed to capture short and long-term dependencies as well as complex patterns and irregularities. This combination is able to successfully learn spatial and temporal relationships in the time series data and have been applied to several forecasting tasks [33, 23, 32, 22, 34]. The results showed a significant improvement over the independent networks in the prediction accuracy and the convergence rate.



(a) A CNN-LSTM network variation.



(b) A CNN-LSTM network variation (smaller filters, no FC layer).

Figure 2.8: An example of CNN-LSTM hybrid architecture [32].

The hybrid CNN-LSTM architecture is highly variable: the size and number of convolution filters, whether or not pooling, dropout, up-sample and fully connected layers are integrated, Figure 2.8 depicts two examples of CNN-LSTM network variations.

2.3.5 ARIMA-CNN-LSTM model

Yang et al. [72] proposed a hybrid ARIMA-CNN-LSTM framework for stock price forecasting, the model consists of three hierarchical layers and splits the time series into linear and non-linear components, the first layer is the ARIMA layer and it serves as a linear filter, it receives the time series sequence as input and produces predictions for the linear components, the real targets are then subtracted from these predictions to get the error terms, the residuals or the error series is regarded as the non-linear component and is processed by the CNN-LSTM network. The CNN is responsible for extracting feature patterns while a sequence to sequence LSTM networked generates non-linear predictions. A fully connected layer is employed to concatenate results from ARIMA and Seq2Seq layers, thus obtaining the final forecast values, the model can be summarized in the following equations :

$$\begin{aligned} \hat{x}_{t+h} &= ARIMA(x_t) \\ e_{t+h} &= \hat{x}_{t+h} - x_{t+h} \\ \tilde{e}_{t+h} &= CNN(e_{t+h}) \\ \hat{e}_{t+h} &= Seq2Seq(\tilde{e}_{t+h}) \\ y_{t+h} &= FC(\hat{e}_{t+h}, \hat{x}_{t+h}) \end{aligned} \tag{2.10}$$

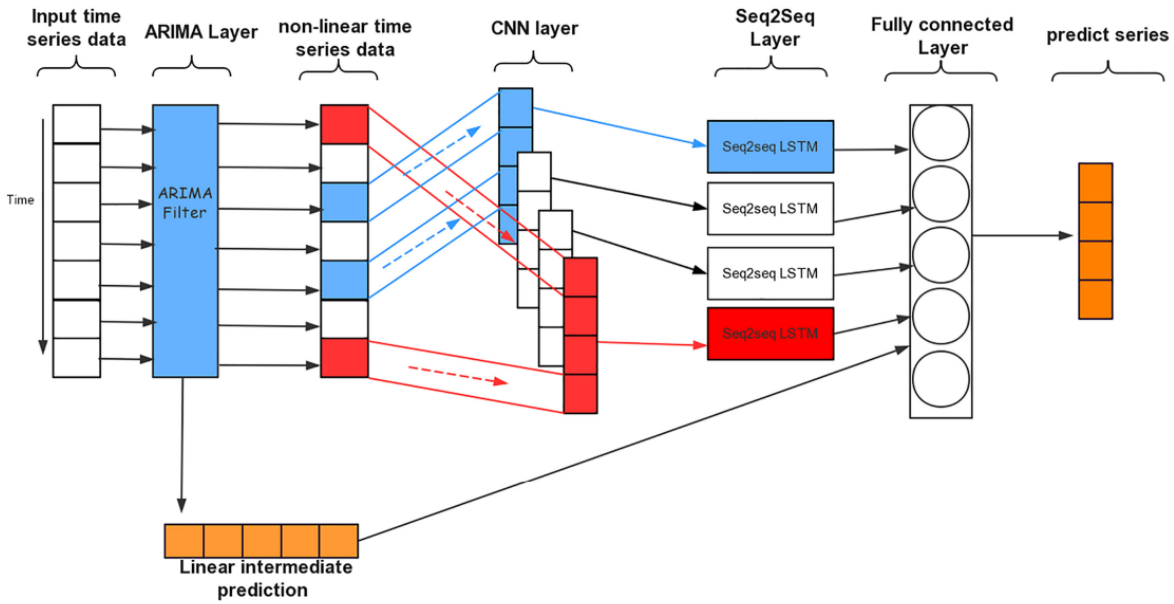


Figure 2.9: The proposed ARIMA-CNN-LSTM architecure [72].

The model demonstrated higher accuracy than the independent models by a sig-

nificant margin.

2.3.6 CNN-LSTM-Transformer model

Hajji et al. [2] proposed a comprehensive framework for solar energy production forecasting that amalgamates three influential deep learning mechanisms: convolution, recurrence and attention. The proposed model employs a sequential combination of three distinct network types: a CNN, an LSTM, and a Transformer network. The data undergoes preprocessing using a self-organizing map, which organizes it into four distinct seasons. Subsequently, the data mapped into four clusters is fed into a convolutional network to extract spatial features. The LSTM then captures temporal dependencies from the processed data, and the resulting outputs are further passed to a Transformer network. The inclusion of the Transformer enhances the model's memory capacity and compels the LSTM to focus on pertinent historical features.
The model achieved top performance compared to baseline models and outperforms CNN-LSTM hybridization and other combinations.

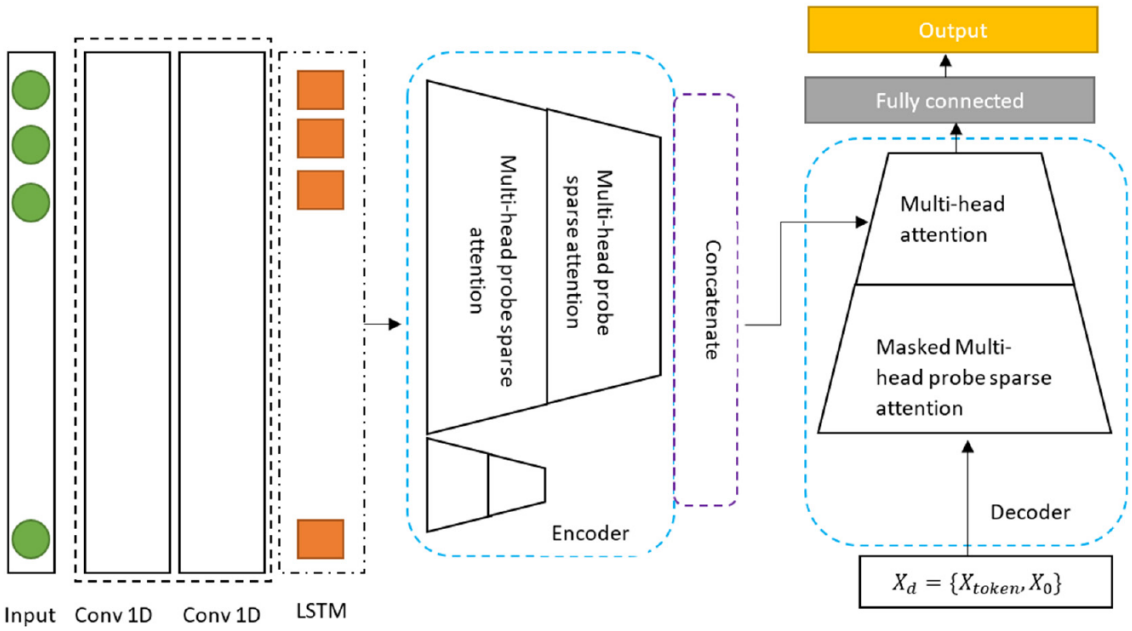


Figure 2.10: The proposed CNN-LSTM-Transformer hybrid model [2].

2.3.7 Challenges and considerations to the hybrid approach

While the hybrid approach to time series forecasting offers many advantages and performance improvements, it is still a complex task that presents several challenges

and difficulties that should be accounted for when considering this approach. Here are some points that the analyst or practitioner should be aware of when designing or opting for a hybridization:

- **Model selection, composition and parameter tuning :** The time series forecasting literature offers a rich selection of models and techniques to choose from, therefore selecting an adequate combination prove to be complicated and requires careful evaluation of the strengths and weaknesses of each model, deep understanding of the nature of the data at hand and the forecasting task. Integrating the candidate models is no less complicated than choosing them, the models compatibility must be verified and appropriate interfaces between the models are to be conceived. The final challenge in the composing step is parameter tuning, as each model has its own set of parameters, these sets need to be tuned with respect to their respective model as well as to the the framework as a whole which may be a complex optimization problem.

- **Complexity, computational resources and scalability :** Merging different models together often increases the englobing framework's complexity, which introduces higher computational requirements such as processing power and memory capacity and often leads to longer training times. As the model's complexity grows, the ability to scale it to larger datasets and to integrate new features and functionalities becomes more and more challenging.

- **Model interpretability and maintenance :** Individual models design, functionality and purpose make them inherently comprehensible and highly interpretable. However, hybridizing models result in high model complexity and may reduce the global interpretability of the framework. The less interpretable a hybrid model is the harder it is to maintain it, particularly when unexpected results or behavior occur.

Conclusion

In conclusion, when facing a forecasting problem, developing or selecting a suitable model hasn't always a clear path, the literature review presented in this chapter highlights the diversity of models available for time series forecasting and provides a comprehensive overview. Based on the insights gained from this review we can implement existing models and propose new ones. We have also emphasized the importance of considering the specific characteristics of the dataset, the type of prediction task at hand, the desired level of interpretability, and the computational resources available when selecting or designing an appropriate forecasting model.

Chapter 3

Proposition and evaluation

Introduction

The quest for improving forecast accuracy has always been at the center of time series analysis and forecasting field, several models and approaches have been proposed ranging from classic and machine learning to recent deep learning models as well as different combinations of these classes.

The research question guiding this chapter is: "How do classic forecasting models compare to recent deep learning techniques and does the hybrid approach guarantee performance improvements ?"

To address this question we have conducted a comprehensive experiment to investigate the effectiveness of different forecasting models, the experiment aims to assess and compare the performance of various state-of-the-art baseline models and our two novel hybrid model architectures.

This chapter is outlined as the following: we first introduce our proposed hybrid deep learning models, we then walk through the experimental setup presenting the tools, datasets and environment in which this study took place. Subsequently we showcase the baseline models details and architectures as well as the evaluation metrics used in assessing these models performance. Furthermore, we explain the forecast strategy on which we based our work. Finally the results of our experiment are presented, analysed and discussed in order to draw the final conclusion of this work.

3.1 Proposed models

3.1.1 CNN-GRU-TCN

In this section we present our novel hybrid model of CNN-GRU and deep TCN networks combination. The main idea behind this choice of architecture is to combine strength of two of the prominent sequence modeling architectures while using CNN as a complementary component and a performance enhancer.
This model aims to take advantage of the constituent networks strong points and merges them in a complementary manner:

- Short-term dependencies and local patterns: thanks to the convolution operation, TCNs are capable of capturing local complex patterns and dependencies across temporally close observations.

- Powerful long-term dependency modeling: GRUs were developed to address the vanishing/exploding gradient problem, thus giving them the ability to span longer sequences and therefore having larger sequential memory and retaining deep information history. TCNs on the other hand leverage stacked, increasingly dilated convolution layers to expand their receptive field to be eventually able to memorize, in a non-sequential manner, long information history and temporal dependencies. Combining both can benefit the model as it diversifies its long-memory information representations and allows it to learn from different parts of the sequence.

Following Section 3.3.3.3 demonstrations, we employ a 1D-convolution layer to enhance the performance of the GRU network, as discussed discussed in Section 3.3.3.3 the CNN plays a key role as a spatial feature extractor and a signal smoother, we consider it a necessary upgrade and we view the CNN-GRU/LSTM model hybridization as a single functioning block given the performance/complexity ratio of adding the 1D-convolutional layer. Few works have been accomplished in investigating the TCN-GRU hybridization potential but none have added CNN to the combination as a performance enhancer of the general framework.
The model consists of two parallel blocks, the first is a deep TCN block composed of N stacked residual blocks (Figure 3.11) and the second is the CNN-GRU model shown in Figure 3.18, the outputs from each block are concatenated and passed through two fully connected layers to obtain the final results, the full architecture is illustrated in Figure 3.1.

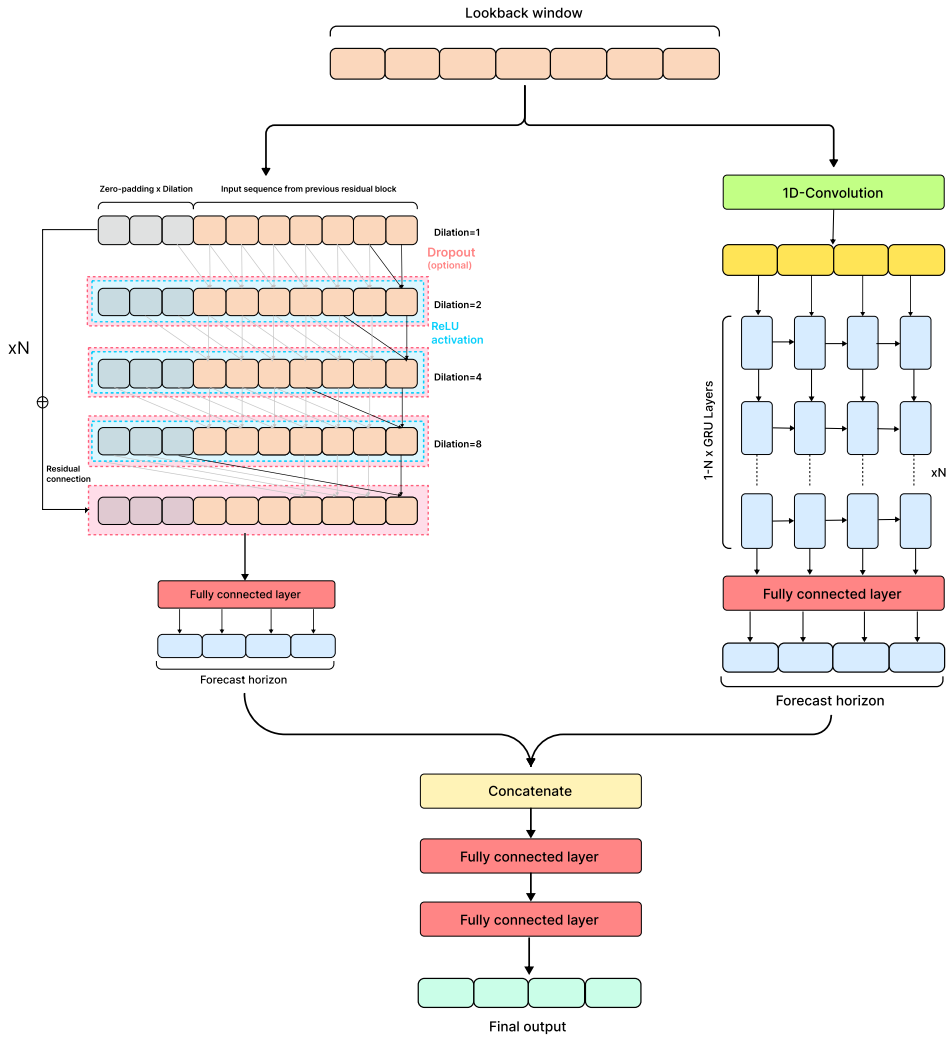


Figure 3.1: Proposed CNN-GRU-TCN model architecture.

3.1.2 GNN-CNN-LSTM

Our second proposed model combines the GraphSAGE (Sample and aggregate) graph neural network proposed by Hamilton et al. [15] with the CNN-LSTM hybrid neural network. The main motivation behind this hybrid approach is to gain deep insights into the time series by learning temporal and spatial patterns and dependencies from two distinct representations of the series: sequential grid-like and non-Euclidean graph data structures.

The visibility graph offers a non-conventional and unique representation of the time series as the generated graph structure depends entirely on the underlying structure and characteristics of the time series data. The visibility graph breaks the sequential

nature of the series and exposes an excellent spatial structure of it, where for example two observations at the extremeties of the sequence may have a direct visibility link while two closer observations may not. The GraphSAGE neural network constitutes a powerful graph modeling tool, unlike other graph neural networks that learn node embeddings, it trains a set of aggregator functions, these functions learn to aggregate feature information from a node's local neighborhood, each function learns from a different search depth in a way that one function might be aggregating features from direct neighbors of a given node while another might be aggregating information from distant and indirect neighbors as shown in Figure 3.2. This technique provides two major benefits, first the network is able to learn local and global context, patterns and contributions of the graph nodes, second, since the network is learning aggregator functions and not direct embeddings it has a great ability to generalize to unseen data [15]. The network and the model by extension gain in-depth insight into the spatial information and structure of the time series.

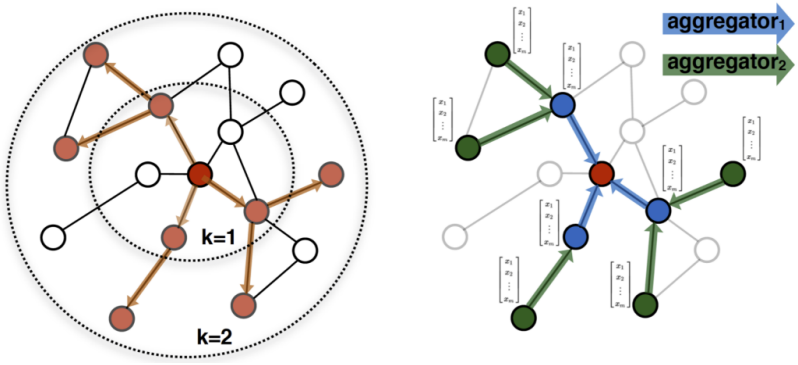


Figure 3.2: Aggregating feature information from different sample neighborhoods [15].

On the other hand the CNN-LSTM block handles the sequential grid-like structure of the time series data and learns temporal and long-term dependencies of the series. Combining both models makes the hybrid framework perfectly apt to learn complex spatial and temporal patterns and relationships and covers more than one perspective on the underlying structure of the time series, which therefore improves the model's robustness and accuracy.

Our model consists of two parallel blocks, the first is the GraphSAGE network block which is similar to GAT architecture presented in Section 3.3.2.4 with the difference being the incorporation of three GraphSAGE network layers instead of three GAT layers, for the second block we implement an LSTM instance of Section 3.3.3.3 architecture and we zero-pad the input accordingly to preserve its size after the convolution, we also employ a residual connection to both blocks outputs, the input sequence is subjected to 1D-convolution of kernel size=1 and is then added to the aforementioned outputs,

result sequences are subsequently concatenated and passed through two fully connected layer to finally obtain the prediction values.

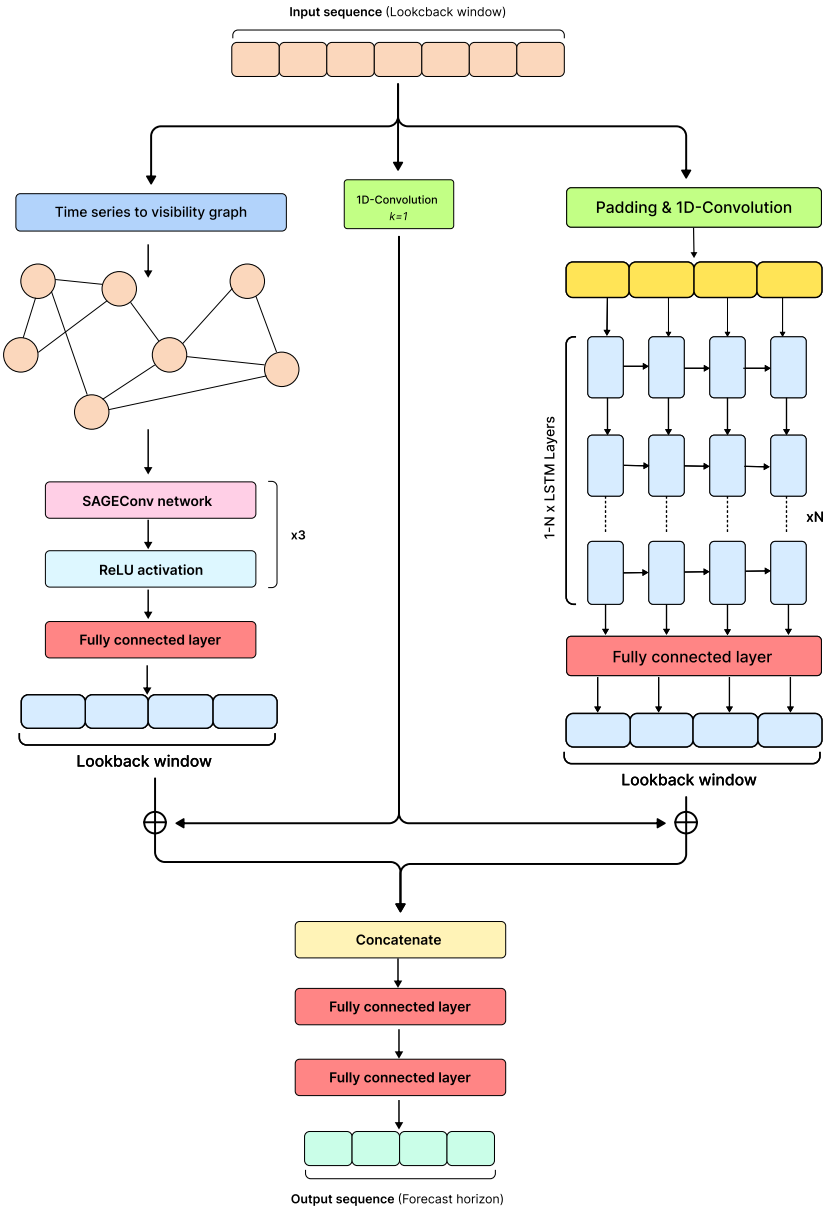


Figure 3.3: GraphSAGE-CNN-LSTM

3.2 Experimental setup

3.2.1 Implementation Tools and Environment

3.2.1.1 Google Colaboratory

All of our experimentations have been conducted on Google Colaboratory or "Colab" which is a cloud-based platform designed for data science and AI research purposes, it provides students, data scientists and AI researchers with a web-integrated Jupyter Notebook environment with a large set of pre-installed libraries and packages, it also offers up several runtimes with a decent hardware setup with the following specs (free plan) :

- Intel Xeon CPU @2.20 GHz.
- 13GB of RAM.
- Nvidia Tesla K80 GPU 12 GB GDDR5 VRAM.
- GPU runtime comes with NVIDIA Tesla T4 GPU 16GB VRAM.
- TPU runtime offers NVIDIA A100 GPU with 40GB of VRAM.

Additionally Colab offers other important features such as: version control integration, history tracking, Notebook sharing and remote collaboration...etc.

3.2.1.2 Python

Python is a versatile and widely adopted programming language, its rich ecosystem of libraries and frameworks makes it a preferred choice for data science, machine learning and deep learning applications.
In addition to the language's simple syntax and clarity that provide an easier written and highly maintainable code, the extensive collection of specifically designed libraries such as NumPy and Pandas destinated for numerical computation and data analysis, as well as machine and deep learning libraries including Tensorflow, Pytorch, Keras and others, offers pre-built functionalities and off-the-shelf tools that accelerate productivity and allow researchers and practitioners to focus on designing and composing models rather than dealing with implementation details and mathematical complexity thanks to the provided high-level and reliable tooling.
Leveraging Python in our implementation ensures that we can effectively harness the existing tools and resources and enables us to focus on the core objective of comparing and evaluating different time series forecasting models.

3.2.1.3 Statsmodels

Statsmodels is a Python library that specializes in statistical modeling, econometrics and exploratory data analysis. It is particularly suitable for time series analysis and offers a set of classes and functions for :

- Estimating different time series statistical models such as ARIMA and SARIMAX and their respective parameters.
- Component decomposition of time series.
- Generating ACF and PCF plots as well as confidence intervals.
- Running statistical inference and performing hypothesis and diagnoses tests.

The Statsmodels module is integrated with the Pandas library which facilitates results manipulation and data exchange between the two libraries APIs.

3.2.1.4 PmdARIMA

PmdARIMA is a library built on top of the Statsmodels library and provide useful extra-functionalities, notabely the AutoARIMA function that offers automatic ARIMA model selection given time series. The function performs several types of tests in order to determine the optimal parameter set for an ARIMA model, the function provides a model summary as well which highlights relevant information about the model such as the values of different information criteria involved in the selection process as well as the results of additional tests conducted during the model selection such as Ljung-Box and Jarque-Bera tests.

3.2.1.5 PyTorch

PyTorch is an open-source deep-learning framework developed by Facebook's AI research group, it has established itself as the *de facto* research framework [38] it shares the main advantageous features with other popular frameworks such as TensorFlow and Keras:

- High-level abstraction: the APIs offered by such frameworks provide high-level abstractions and simplify implementing complex neural network architectures thanks to the pre-built layers, modules and functions which reduce the manual effort going into developing and optimizing at a low-level.
- Automatic differentiation: automatic differentiation algorithms are at the core of deep learning frameworks, these algorithms efficiently compute gradients required for neural networks optimization, thus by automating this process the need for

manual derivation is eliminated and the implementation of complex models is significantly simplified.

- Computational efficiency: with the incredible advancement in computer hardware particularly GPUs and TPUs, it became a necessity to leverage the computational power of these components and such do modern deep learning frameworks. When performing large and intensive computations such as large matrix multiplications, complex and costly mathematical operations and tensors manipulation, these frameworks allow users to take full advantage of the hardware acceleration and therefore hasten and optimize execution, inference and learning times.

PyTorch also has its own unique features that eventually convinced us to opt for among other available options such as TensorFlow:

- Customizability: while PyTorch offers a wide range of pre-built layers, neural networks and models it is still exceedingly customizable at a very low-level which provides a high flexibility and freedom when building models.

- Dynamic computational graphs and flexibility: deep learning frameworks represent neural networks architectures as directed graphs where variables, weights, biases, gradients, loss functions are portrayed as graph nodes. PyTorch builds these graphs at runtime or *"on-the-fly"* which allows for a gain of flexibility regarding models with inputs and outputs of variable lengths that are unknown prior to runtime. This feature wasn't introduced to TensorFlow until its 2.0 version.

- Research focus: PyTorch currently dominates the deep learning research community and it is becoming the standard framework for building state-of-the-art models with more of 92% of HuggingFace models being PyTorch exclusive and about 80% of recently published research papers used PyTorch [38].

3.2.1.6 PyTorch Geometric

PyTorch Geometric is a deep learning library built on top of PyTorch and specifically designed for *geometric deep learning* including graph-structured data. It has a user friendly API closely similar to PyTorch's which ensures a seamless integration, it also comes with a wide array of built-in graph neural network from various published papers all while having PyTorch's flexibility, thus allowing for modifications to existing models or creating new architectures.

3.2.1.7 TS2VG

Time Series To Visibility Graph is a python package that provides an efficient algorithm for building *visibility graphs* from time series data. The employed algorithm

is highly performant and able to efficiently generate visibility graphs and some of their properties even from time series data with large numbers of observations. The package provides two types of visibility graphs natural and horizontal as well as multiple variations including weighted, directed, parametric and other visibility graphs.

3.2.1.8 Pandas

Pandas is a Python package that has established itself as an essential tool for data scientists and analysts, it offers a wide variety of tools and functionalities that simplify and facilitate working with data. The package encompasses powerful and robust features namely :

- The flexible and efficient DataFrame object: a two-dimensional tabular data structure with with labeled axis (rows and columns) allowing users to store and structure data in SQL-like manner.
- Time series and Date native functionalities: a comprehensive set of date and time functionalities as well as a one-dimensional data structure with time indexing possibility making it easier to handle time series data and to seamlessly perform time-based operations.
- Data manipulation: Pandas is a data oriented library therefore it provides a plethora of tools and functionalities including indexing, slicing, filtering, reshaping, merging, and grouping data as well as the ability to handle missing data. The library can be easily integrated with other libraries such as NumPy and Matplotlib providing a unique and complete environment for data processing.

3.2.1.9 NumPy

NumPy is the fundamental package and a lead contributor to Python's success in scientific computing. It provides powerful array object that supports multiple dimensions, along with a diverse range of derived objects like masked arrays and matrices. With a rich collection of functions and operations tailored for efficient array processing, NumPy empowers users with an extensive array of capabilities, including rapid execution of mathematical and logical operations, seamless manipulation of array shapes, efficient sorting and selection methods, input/output handling, discrete Fourier transforms, fundamental linear algebra operations, essential statistical computations, random simulation capabilities, and a host of other functionalities to cater to diverse scientific computing needs.

The core data structure of the NumPy package, is the nd-array object that encapsulates n-dimensional arrays of homogeneous data types, it comes with various

performance optimized operations due to its implementation in C language and the efficient memory management it provides.

3.2.1.10 Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. As visualization plays a key role in data science and machine learning by providing helpful and intuitive insights via different charts and graphs, Matplotlib became an essential tool when tackling any data related task as it helps :

- Create publication quality plots.
- Make interactive figures with zooming, panning and updating abilities.
- Make extensive visual style and layout customization.
- Export to many file formats.
- Embed in JupyterLab and Graphical User Interfaces.
- Use a rich array of third-party packages built on Matplotlib.

3.2.1.11 Scikit-learn

Also known as sklearn, is an open-source machine leraning library, it simplifies predictive data analysis by offering a variety of utilities including: data preprocessing and visualization, machine learning algorithms, evaluation metrics and more.

3.2.2 Data description, preprocessing and splitting

A reliable analysis and forecasting of time series data is heavily dependent on the quality and characteristics of the datasets involved. This section provides an overview of the datasets utilized in the implementation and the experimental evaluation, highlighting their key features, data sources, and relevance to the research objectives. We chose to diversify the source domains of the data, allowing for a comprehensive and well founded assessment of the implemented models in different real-world scenarios.

3.2.2.1 Air Passengers dataset

For our first dataset we chose a classic in the time series literature used in the work Box and Jenkins, the data shows monthly totals of US international airline passengers for the 12-year period from January 1949 to December 1960 consisting of a total of 141 observations. The series exhibits an uptrend due to the increasing number of passengers from year to year as well as a visible seasonality related to the holiday seasons.

Figure 3.4: Plot of US monthly international airline passengers time series.

We saw that this series would be a great case study for assessing the models' performance against seasonal data. The dataset is available for download on Kaggle website.

3.2.2.2 Covid-19 casualties dataset

For the second case study we opted for mortality analysis of the Covid-19 epidemic, the dataset was obtained from Kaggle website. Consisting of 188 observations representing daily new death cases from Jan 22 to Jul 27 of the year 2020, the dataset underwent four different non-linearity tests using the R programming language: Tsay, Keenan, Terasvirta and white neural network tests , all four tests yielded p-values significantly lower than the 0.05 threshold suggesting therefore that the dataset is unlikely to be linear.
Motivated by these results we bilieve that this time series is a proper assessment of models' ability to model and predict non-linearity in the data.

3.2.2.3 Brent spot price dataset

To finalize our selection of datasets we decided to include a financial time series, as these types of time series are known for having several characteristics that make them challenging to model such as high volatility, non-linearity and non-stationarity. Our selected financial dataset is the weekly Brent crude oil spot price time series, it consists of 1500 observations from 1987 to 2016 and is available on the DataHub website.

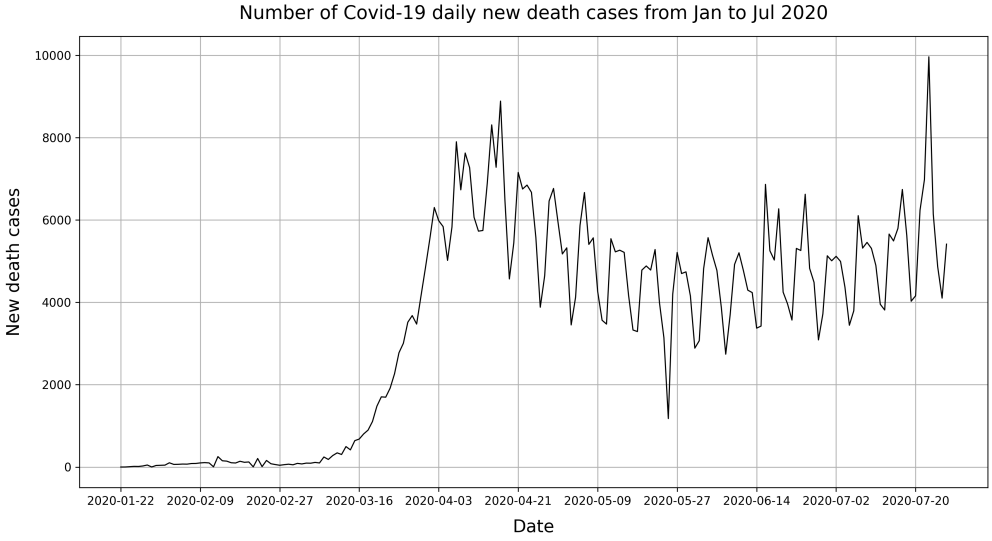


Figure 3.5: Plot of Covid-19 new daily casualties time series.

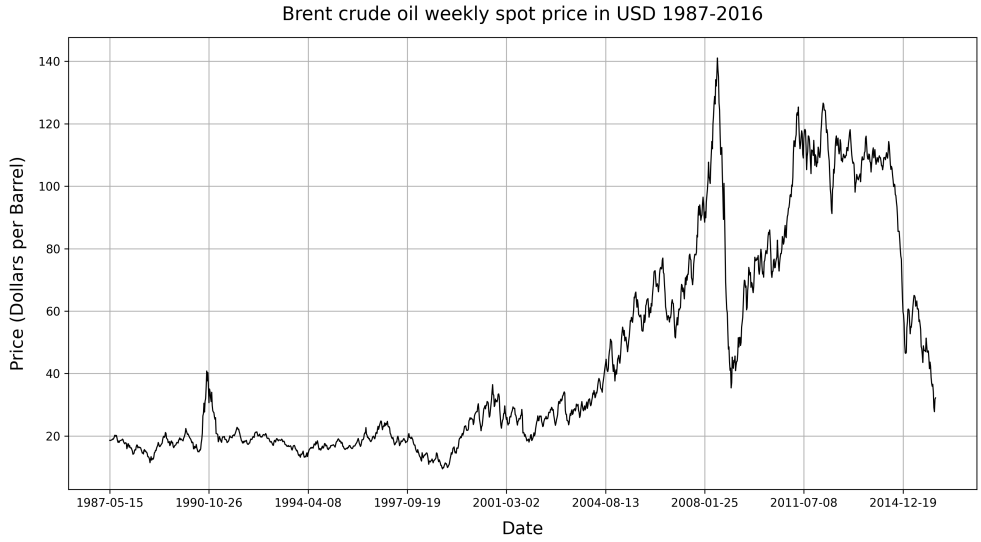


Figure 3.6: Plot of Brent oil spot price time series.

3.2.2.4 Data preprocessing and splitting

Feature scaling

Throughout our experimentation process, we employed feature scaling as a crucial preprocessing step of the time series datasets. It is a technique that transforms the numerical features of a dataset ensuring that they are all on an identical scale. This is

particularly helpful with time series data as it metigates against the presence of outliers and the variable magnitude across observation values.
Additionally, feature scaling plays an important role in improving deep learning models performance and helps reaching better and faster convergence [62] which is a phenomenon we experienced empirically during the training process of the models we implemented as some of them were persistently underfitting the non-scaled data.
All three datasets employed in this work were scaled between 0 and 1 using the **Min-Max scaler** from the Scikit-learn library [43]. The transformation formula is given by:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.1}$$

Where x_{min}, x_{max} are the minimum and the maximum values of the time series respectively.

Train-Test data split

For models fitting and evaluation, a train-test split of 70% and 30% was used to partition the datasets. It's a common split ration in machine and deep learning applications as it provides a good balance between providing an adequate amount of data for training while reserving a sufficient portion for independent evaluation. The training portion constituting of 70% of the data is used for training and optimizing the models parameters, by being exposed to a significant portion of the data, the models can learn the patterns and dependencies underlying the time series. On the other hand, the remaining unseen 30% of the data is large enough to test the models generalization and expose any underfitting while providing a good assessment to the forecast accuracy.

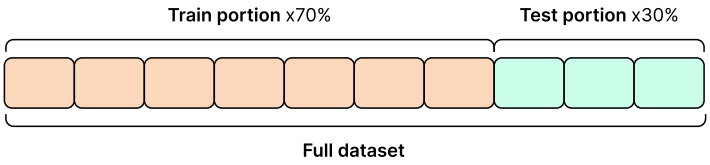


Figure 3.7: Train-Test split visualization.

3.3 Implemented baseline models

3.3.1 Classic models

Classic time series forecasting models are well established techniques conceived with thoughtful considerations and strong foundations in mathematics and statistics, in this section we will present the classic model we included in this work:

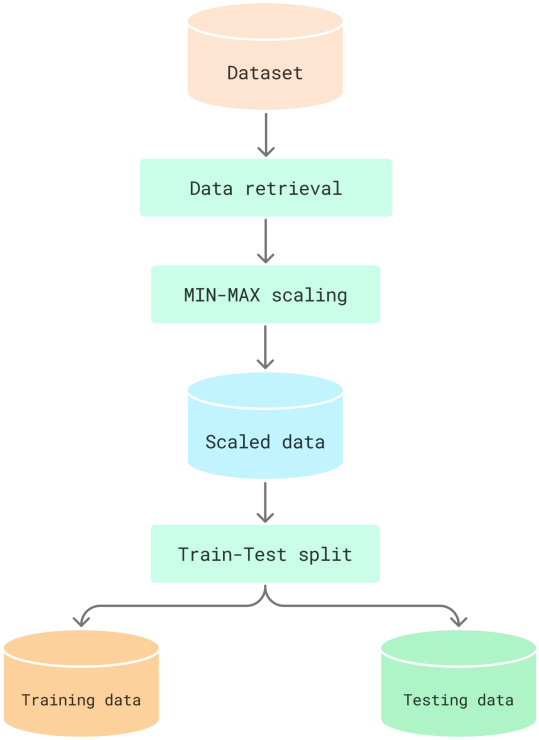


Figure 3.8: Data preprocessing steps.

3.3.1.1 ARIMA

ARIMA is the first and the only classic baseline model we have implemented in this comparative study. The Statsmodels library provides ARIMA class which we used to initiate the model, the p, d and q parameters were then obtained using the ***auto_arima*** function provided by PmdARIMA library, given a time series to fit the function estimates the parameters by searching and testing for the optimal values that minimize the information criterion provided.
Figure 3.9 depicts the general process of building and forecasting with ARIMA model.

3.3.2 Deep learning models

Deep learning models leverage the power of neural networks flexibility and their ability to capture non-linearity, memorize long data history as well as their sophisticated architectures and computing mechanisms, this section presents our selection of deep learning models to be implemented and evaluated:

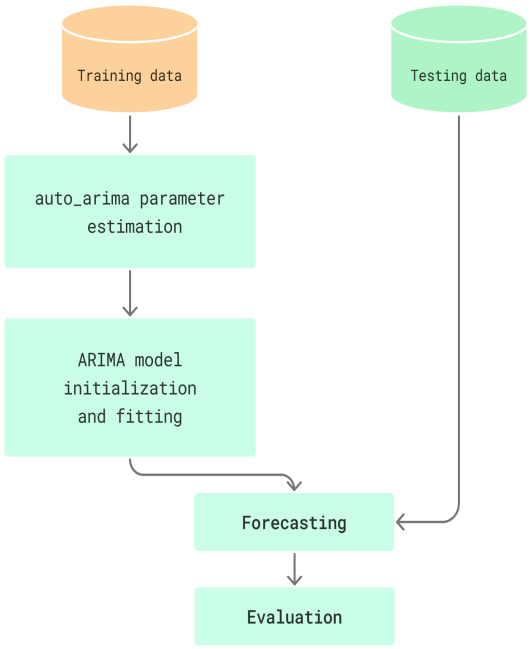


Figure 3.9: High-level view of the ARIMA model building and forecasting process.

3.3.2.1 LSTM/GRU

The second and third baseline models we implemented are the LSTM and GRU neural networks.The memory gating mechanism alongside with the inherent recursion qualify these two models to be powerful tools for sequence modeling and they are widely used in time series forecasting tasks.
The model consists of 1 to N stacked LSTM/GRU network layers followed by a fully-connected layer, which maps the network's output from a hidden size dimension to a single dimension. N is a hyperparameter to be tuned for each dataset.

3.3.2.2 TCN

Deep Temporal Convolution Networks are a raising architecture in the domain of sequence modeling and time series forecasting, therefore we decided to include it in this comparative study and test their efficiency against other state-of-the-art models.
Our implementation of the deep TCN network consists of stacking N ***residual blocks***, each of which comprises four stacked ***causal dilated 1D-convolutional*** layers. We use convolution kernels of fixed $size = 2$ and a dilation of factor 2^i at each layer where i is the 0-based order of the layer in question.
A non-linear activation function of type ReLU is applied to the outputs of the first

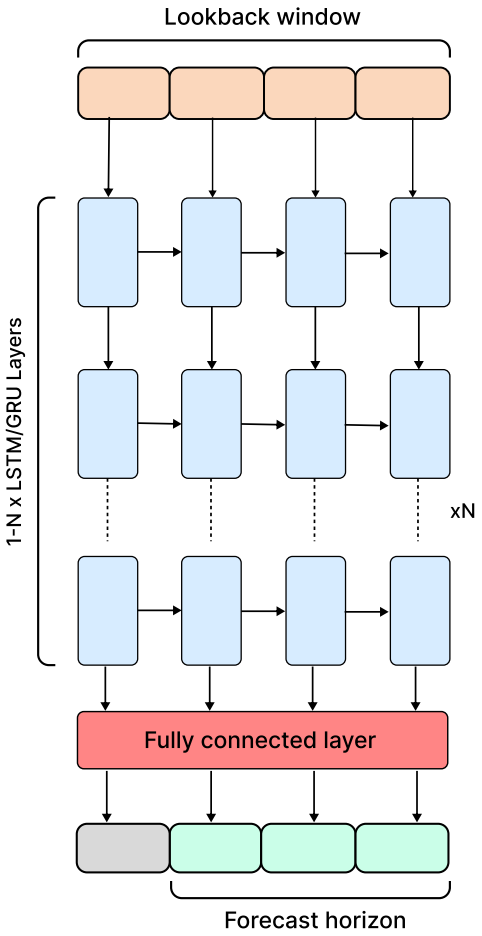


Figure 3.10: LSTM/GRU model architecture.

three layers followed by an optional dropout layer across all four layers.
In order to preserve the input size and the causality of the convolution, we left-zero-pad the input to each layer with an amount equal to its dilation factor, since we have kernels of $size = 2$ this approaches ensures the convolution causality and the preservation of the input size. Finally we employ a ***residual connection*** which adds the input to the first convolutional layer to the output of the last one, thus yielding the final output of a single residual block which serves as input to the next block.

3.3.2.3 Transformer

Motivated by their success in NLP and computer vision fields and other fruitful adoptions to time series forecasting (Section 2.2.4) we decided to include the Transformer neural network in our list of implemented models. However due its operational mechanism, the Transformer architectures inherently requires a recursive forecasting approach, not aligning therefore with our forecast strategy which we'll discuss in detail

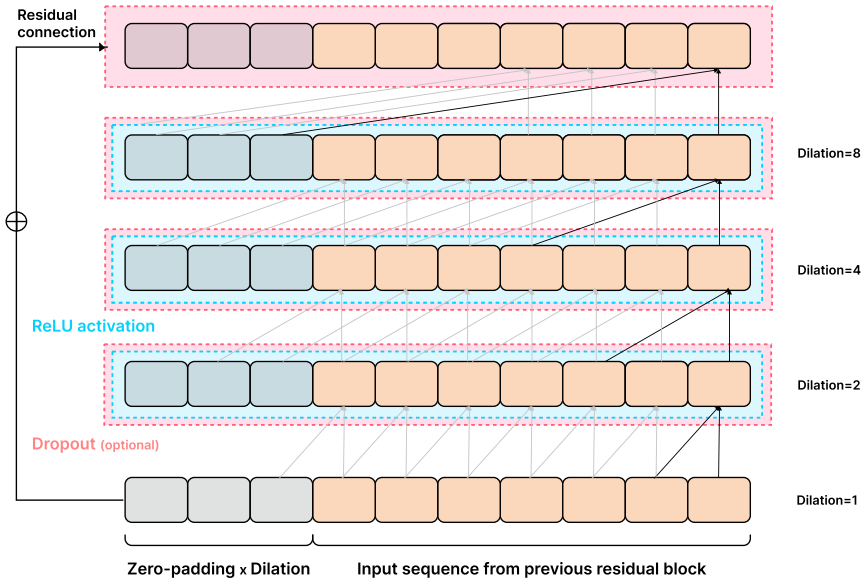


Figure 3.11: Architecture of the employed TCN Residual Block.

later in this chapter. Thus, we have chosen to rely solely on the encoder part, we used the canonical encoder from [60] paper.

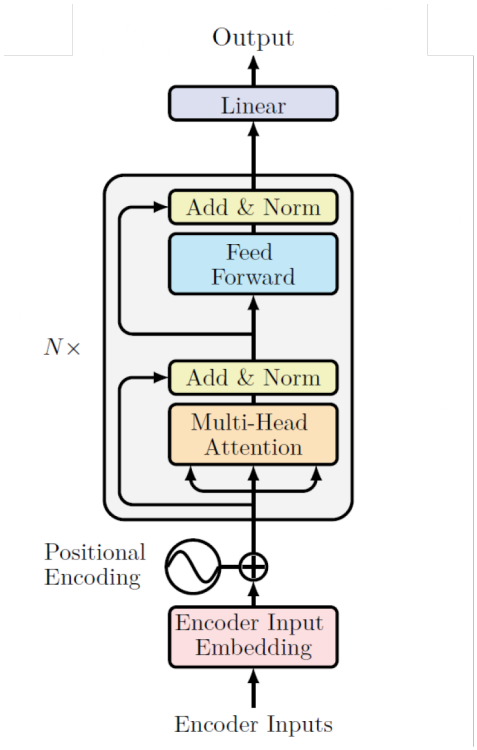


Figure 3.12: The employed transformer architecture [60].

The model consists of an input embedding layer followed by positional encoding followed by N stacked encoder layers and finally a linear layer mapping the output of the encoder layers from d_{model} to one dimension.

3.3.2.4 GAT

Even though graph neural network have been successfully applied to many deep learning problems, not much work have been done exploring their full potential in time series forecasting problems [49]. This motivated us to investigate their capabilities and efficiency in modeling time series and compare them to other state-of-the-art models.

The Graph neural network we chose to implement is the Graph Attention Network from [61] paper. Our approach to model the time series data we have is to transform it ***undirected, unweighted visibility graph*** using the transformation proposed by Lacasa et al. [25] which is provided by the TS2VG library. We then initialize our Graph Attention (GAT) model consisting of three stacked layers of GAT networks each of which is followed by a ReLU activation layer, finally we employ a fully-connected layer to map the output from input to output length.

Figure 3.13 shows the visibility scope of the 50 first observations of the Air Passengers dataset, the final result of the time series to visibility graph transformation is depicted in Figure 3.14.

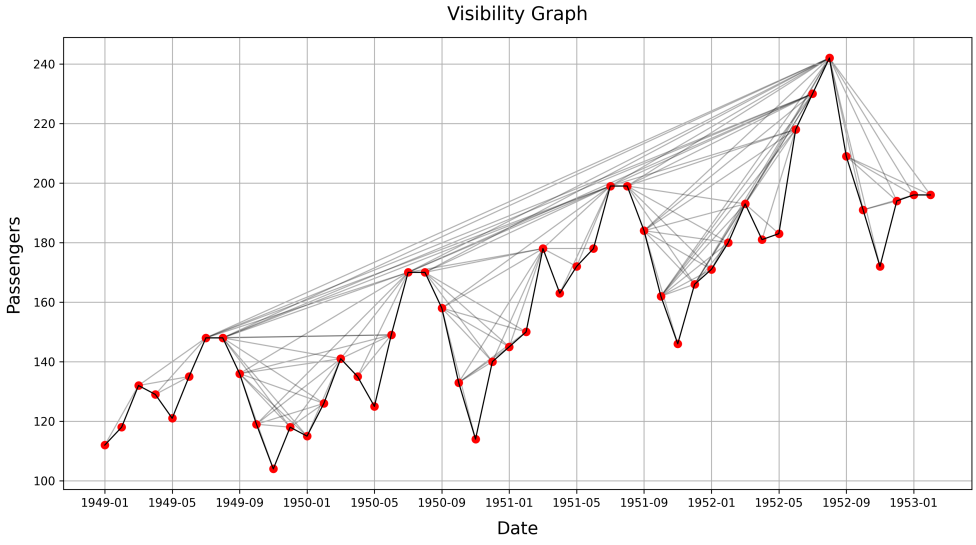


Figure 3.13: Time series to visibility graph of first 50 observations of the Air Passengers dataset.

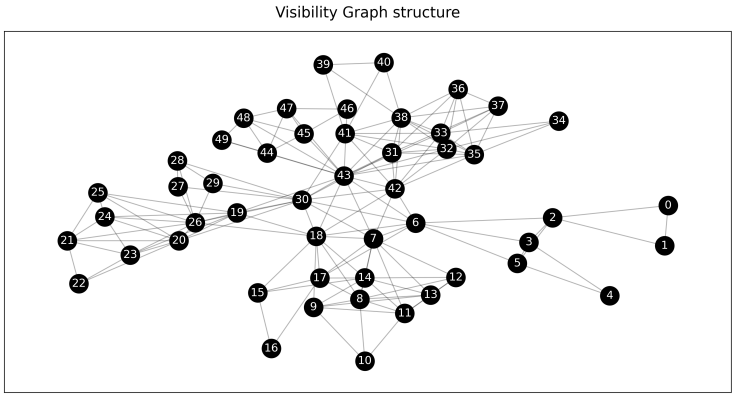


Figure 3.14: The visibility graph data structure of the Air Passengers sub-dataset.

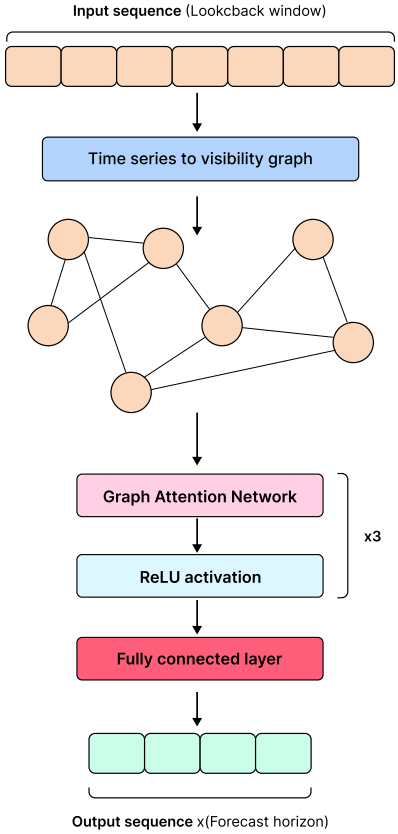


Figure 3.15: The Graph Attention model architecture.

3.3.3 Hybrid models

Models hybridization aims to combine their strengths and overcome their individual limitations, this section highlights the structures and architectures of the hybrid models covered in this comparative study:

3.3.3.1 ARIMA-LSTM/GRU

ARIMA-RNN hybridization is a well known combination in time series forecasting literature, it exploites ARIMA's great ability to capture linearity and the RNNs capabilities of retaining long history of information and capturing non-linearity in the data. we implement two separate instances of this hybridization, the first being ARIMA-LSTM and the second is ARIMA-GRU. The model's working methodology is illustrated in Figure 3.16.

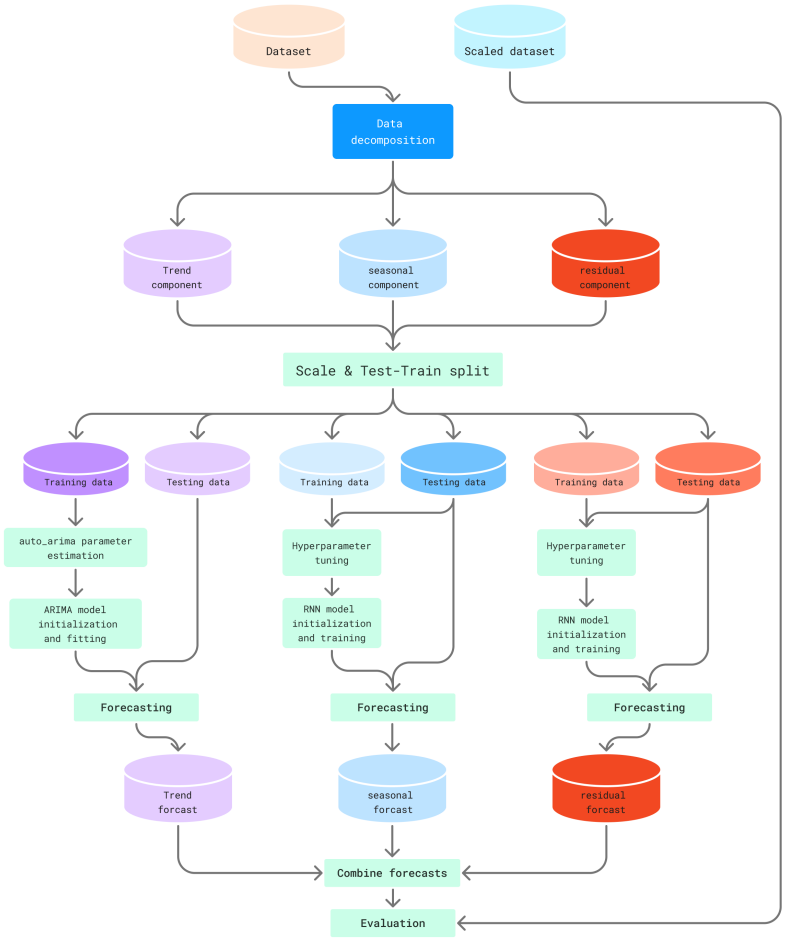


Figure 3.16: Visual demonstration of ARIMA-LSTM/GRU forecasting process.

The model is based on additive decomposition of the time series into Trend, Seasonal and Residual components, each of of these components is then scaled and split into training and testing portions. The trend component is fed into ARIMA for fitting and forecasting while the seasonal and residual components are handled by two separate LSTM/GRU models (Figure 3.10). After obtaining the forecast values of the three models, the forecasts are then added together forming the final predictions.

3.3.3.2 LSTM-GRU

While LSTM and GRU networks belong to the same neural networks family they have different cell structure as demonstrated in chapter 1, the aim of this combination is to leverage the strength of both gating mechanisms and strike a good balance between learning long-term dependencies and low cost, efficient computation.
Our implementation of this model consists of stacking two GRU layers over two LSTM layers followed by a fully connected layer mapping the output size from hidden size to one dimension.

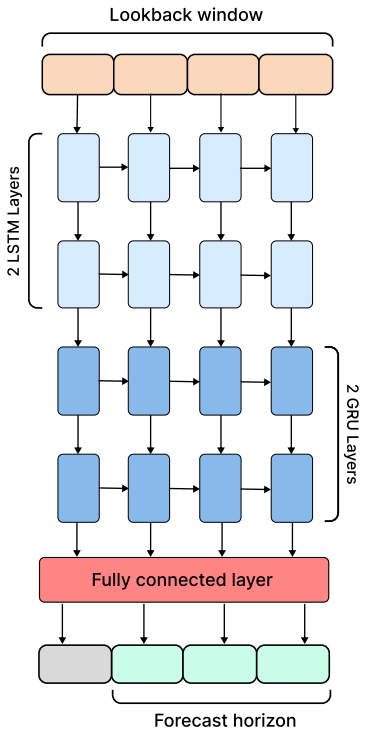


Figure 3.17: Hybrid LSTM-GRU model.

3.3.3.3 CNN-LSTM/GRU

As convolutional neural networks revolutionized the image processing field, it wasn't

long until their great abilities of extracting spatial features from grid-like data started being adopted and applied in domains other than computer vision. Time series data can be represented as one-dimensional grid-like data making them perfectly suitable for subjecting convolution operations, by combining CNN with LSTM/GRU networks we obtain a model capable of modeling and extracting spatio-temporal patterns and dependencies. In order to incorporate this hybridization into our study we've built a

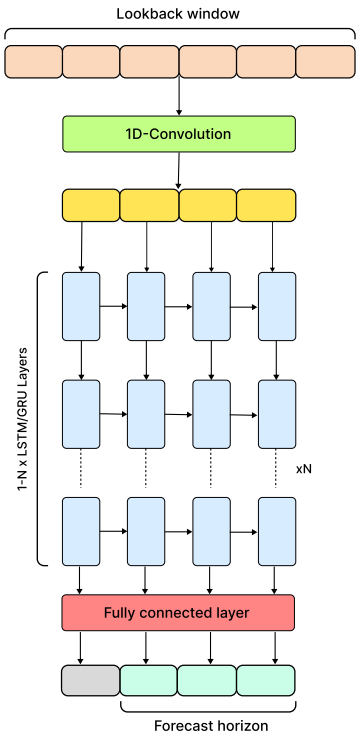


Figure 3.18: Hybrid CNN-LSTM/GRU model.

model consisting of 1D-convolutional layer followed by LSTM/GRU model from Section 3.3.2.1. The convolution layer ensures spatial feature extracting from the input sequence and acts as denoising layer and provides the LSTM/GRU with a smoother signal, thus disposing of rough fluctuations and allowing for better generalization. It uses default fixed dilation factor and stride size both equal to 1. The convolution kernel size and the number of LSTM/GRU layers need to be tuned separately for each dataset. Figure 3.18 shows the hybrid CNN-LSTM/GRU model architecture.

3.3.4 Hyperparameter selection and tuning

Accurate and robust time series forecasting models rely heavily on appropriate hyperparameter selection and tuning, this process involves exploration of different combinations in order to find the optimal set of value that allows the model to reach its best performance. The search and selection of optimal hyperparameter values requires carefull consideration as they play a crucial role in the assessment of the model's suitability to the forecast task at hand, a poor tuning of a model risks overlooking its true potential and capabilities.
In order to ensure the attainment of the models best performance possible, we employed a two steps hyperparameter tuning strategy:

1. **Grid search:** in order to narrow down the possibilities and minimize the manual effort we incorporated a grid search, that is, for each hyperparameter to be tuned, we set a fixed interval of values to be tested, then we apply an extensive search that explores every possible combination, finally the combination that maximizes the model's accuracy is retained.

2. **Trial and Error:** after obtaining the initial optimal combination we then attempt to further tune and optimize the selection by applying smaller adjustments to the parameters.

Model	**Hyperparameters**
ARIMA	p, d, q
TCN	N_{layers}
GAT	-
Transformer*	N_{layers}, N_{heads}, d_{val}, ff_{size}
LSTM/GRU	N_{layers}
LSTM-GRU	-
CNN-GRU-TCN	k, N_{layers}, N_{resid_block}
GNN-CNN-LSTM	N_{layers}, k
ARIMA-LSTM/GRU	p, d, q, N_{layers}
CNN-LSTM/GRU	k, N_{layers}

We would like to note that this process doesn't apply to ARIMA model as we use *auto_arima* function provided by PmdARIMA package which helps estimate the best set of parameters by minimizing the provided information criterion.

3.4 Model evaluation

In this section we will present the evaluation metrics and methodology we used in

measuring the models performance:

3.4.1 Evaluation metrics

3.4.1.1 Mean squared error

The mean squared error or MSE is a measure that calculates the average squared difference between true and predicted values, the quadratic operation places a higher penalty on larger errors and is more forgiving towards smaller ones, making it sensitive to outliers.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2 \tag{3.2}$$

3.4.1.2 Root mean squared error

The root mean squared error or RMSE as the name indicates is the square root of MSE, it still penalizes larger errors while providing information in the same scale as the variable in question.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2} \tag{3.3}$$

3.4.1.3 Mean absolute error

Also known as MAE measures the average absolute difference between true and predicted values, it provides a straight forward interpretation of the magnitude of the errors as it tells the average inaccuracy to be expected. MAE is also a scale-dependent measure, however unlike MSE and RMSE, it is less sensitive to outliers.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y_i}| \tag{3.4}$$

3.4.1.4 Mean absolute percentage error

Mean absolute percentage error or MAPE measures the average absolute difference between each actual and forecast values, relative to the actual value. MAPE is a scale-independent measure since it provides a percentage-based measure of the error making it suitable for comparing accuracy across multiple use cases and datasets.

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y_i}}{y_i}\right| \tag{3.5}$$

3.4.2 Evaluation methodology

After careful parameter selection and tuning as well as ensuring chosen hyperparameters optimality and in order to guarantee fair comparison and models stability we train and test each model in three distinct instances, the forecast results are saved alongside with evaluation metrics values for each instance. We then average and save error values across the three instances, the forecast results from the best iteration are selected to be plotted and represent the models best performance.

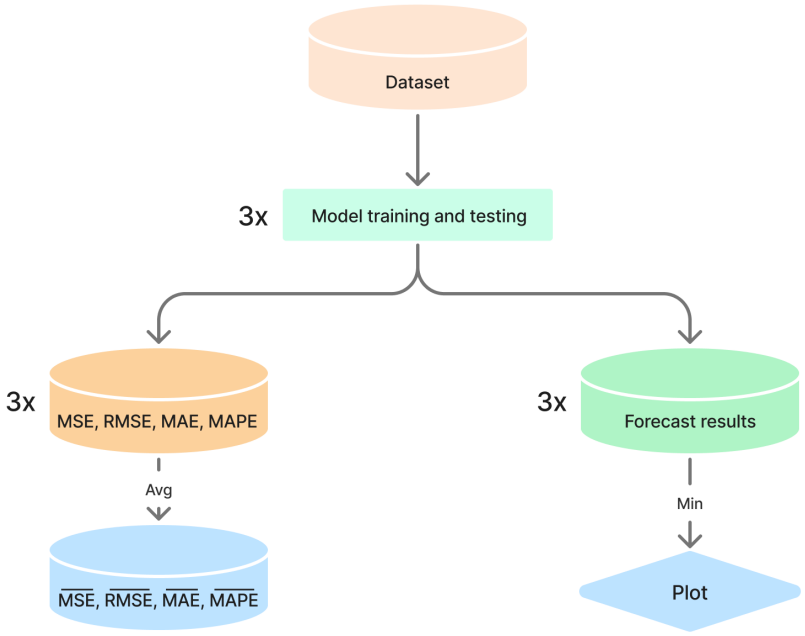


Figure 3.19: Model evaluation process.

3.5 Forecast strategy

The forecast strategy considered in this study is the ***Direct Multi-Step Ahead*** strategy, where models are trained to forecast multiple future time steps simultaneously instead of recursively forecasting one step at a time.
For each dataset, models are adequately provided with a fixed *lookback window* and *forecast horizon* sizes to be trained on.

For both training and testing, we employed a ***rolling window*** technique with a *stride size* = *forecast horizon* unlike the convention of sliding by one step at a time. This technique offers several advantages, first it allows the model to produce perfectly adjacent forecast sequences, disposing therefore from the need to process overlapping

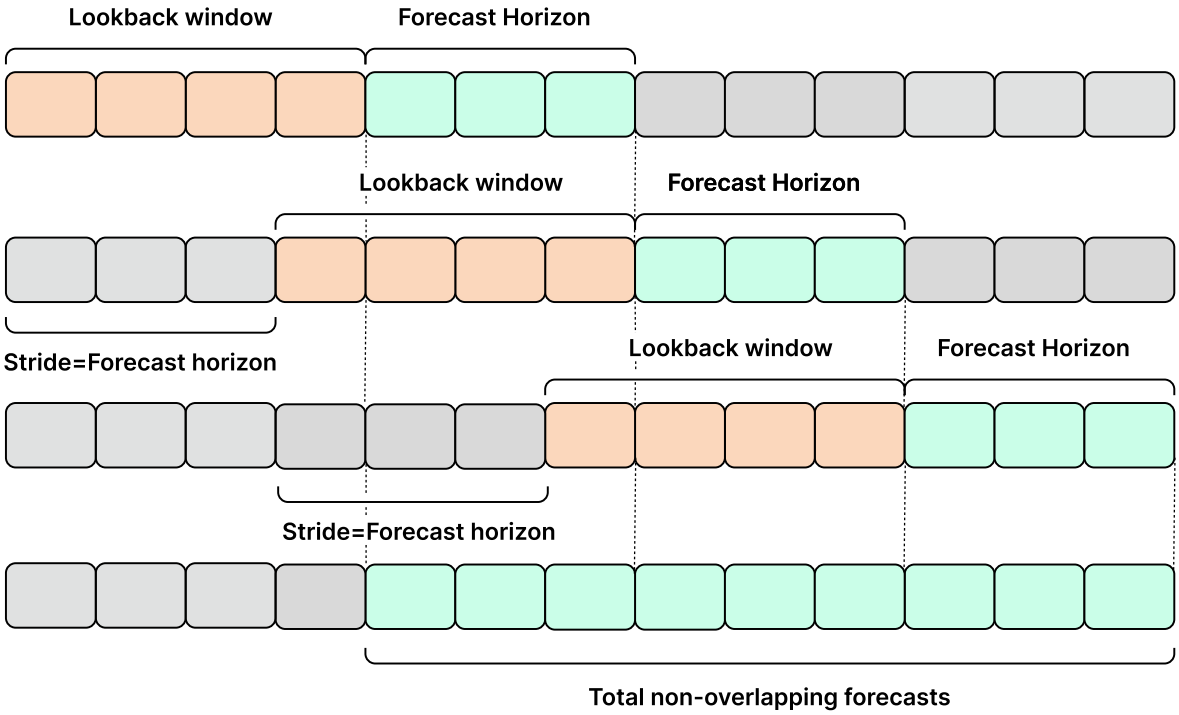


Figure 3.20: Horizon-strided rolling window forecast strategy.

Table 3.1: Lookback window and forecast horizon settings for each dataset.

Settings / Dataset	Loockback window	Forecast horizon
Covid-19 casaulities	15	5
Air passengers	15	5
Brent spot price	50	12

forecast time points, second by setting the sliding factor equal to the forecast horizon, we significantly reduce the time amount required for training the models.
To study the influence of strided-rolling window training on test outcomes and training time, we tested four of the implemented models on each dataset with two different sliding factors: single step slide and horizon size slide.
The results aligned with our expectations, the training time is significantly affected by the stride size, on the Brent Spot Price dataset, which is the largest among the three with 1500 observations, the training time differences were tremendous and went from less than 10 minutes to nearly and over an hour for certain models, this significant difference was the first motive for us to discard the conventional one-strided rolling window.
The unexpected finding regarding the second and more important criterion, forecast

accuracy, as shown in Figure **??** for all datasets and all four models , the horizon-strided training yielded more accurate forecasts upon testing.
We then concluded that our choice for the stride factor was beneficial and didn't require any Speed/Accuracy trade-offs.

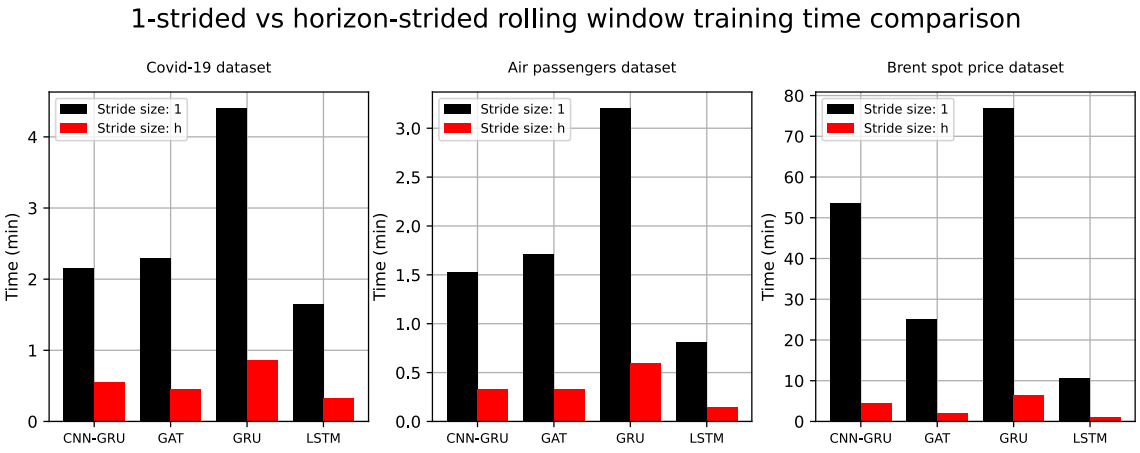


Figure 3.21: Impact of stride size on model training time.

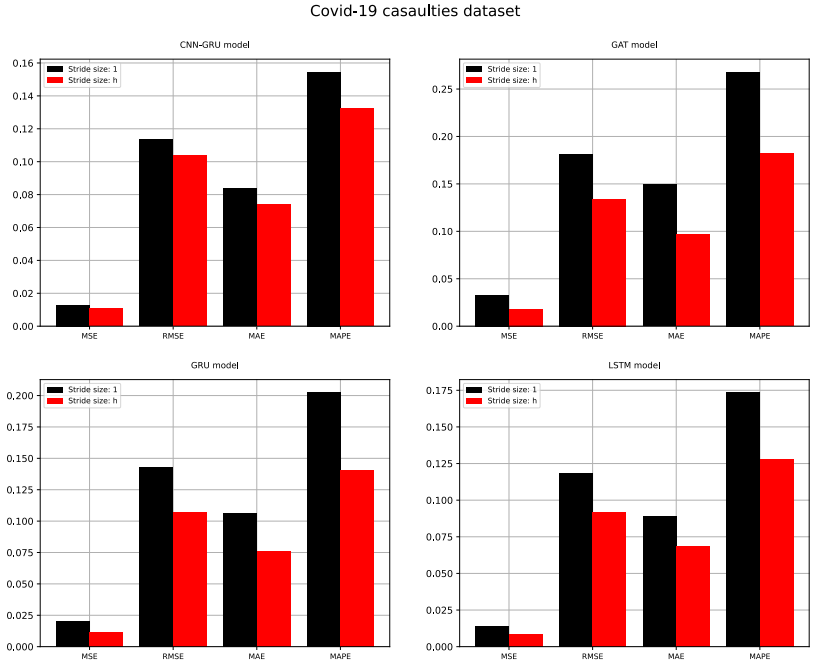


Figure 3.22: Impact of stride size on test forecasts accuracy: Covid-19 casualties dataset.

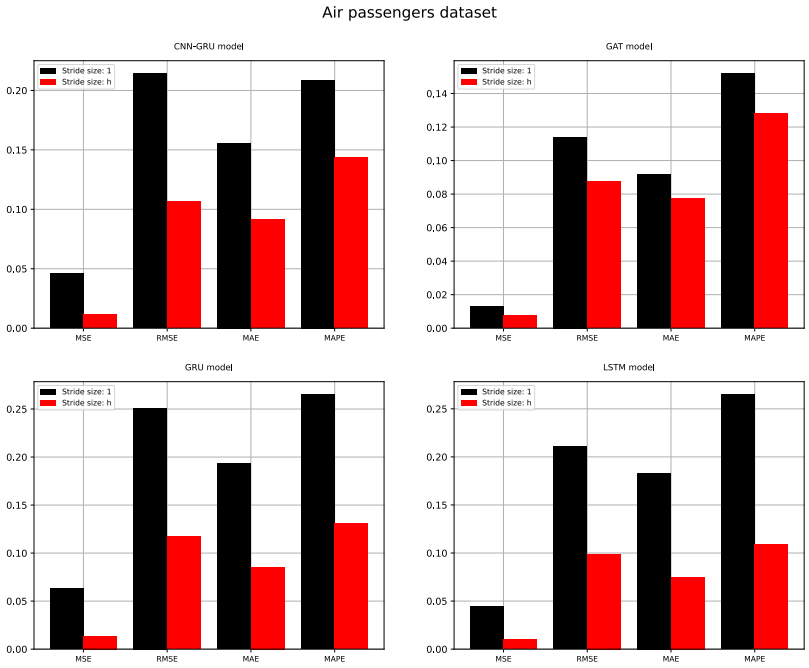


Figure 3.23: Impact of stride size on test forecasts accuracy: Air passengers dataset.

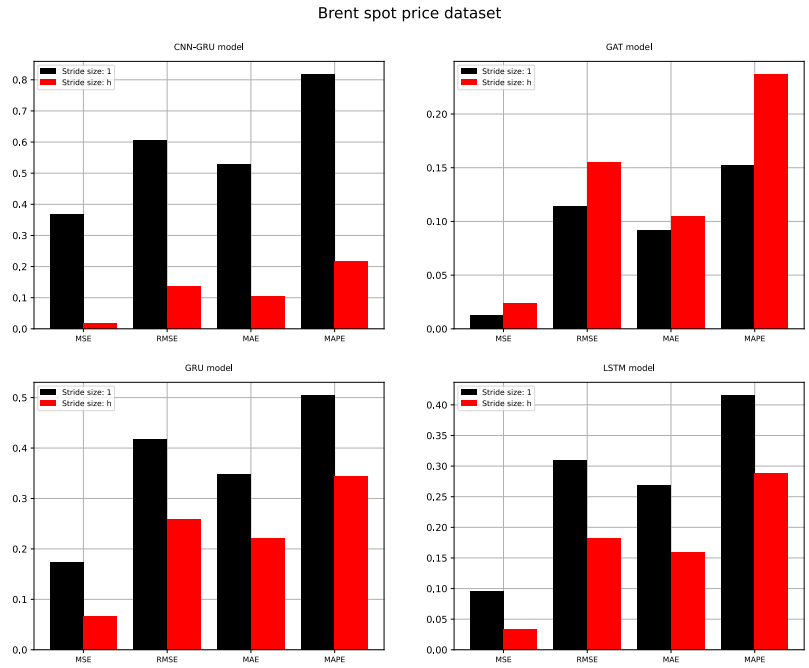


Figure 3.24: Impact of stride size on test forecasts accuracy: Brent spot price dataset.

3.6 Results

In this section we present, analyse and discuss the results of our comparative study, where we conducted experiments on three different real world datasets using thirteen forecasting models. This work compares the performance of classic, deep and hybrid models as well as our proposed architectures. For each model, visual and quantitative performance evaluations are provided alongside with their respective hyperparameter sets and configurations.
The implementation of these models was carried carefully, each model was trained on the respective datasets, hyperparameter optimization was incorporated to ensure a fair comparison.
All of the deep-learning models included in this study were trained for 250 epochs using the Adam optimizer and the MSE as the loss function.

3.6.1 Baseline models performance

3.6.1.1 ARIMA

Table 3.2: ARIMA model hyperparameters.

Parameter \ Dataset	Covid-19 casualties	Air passengers	Brent spot price
p	5	4	1
d	1	1	1
q	2	2	0

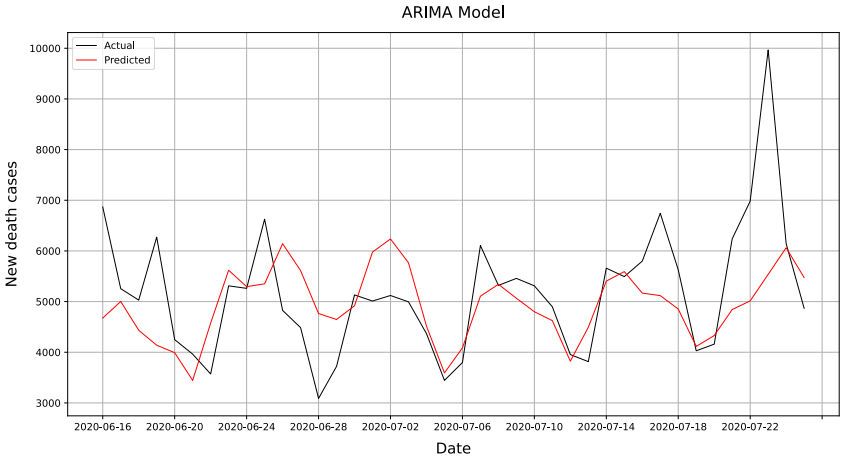


Figure 3.25: Forecast results with ARIMA model on Covid-19 casualties dataset.

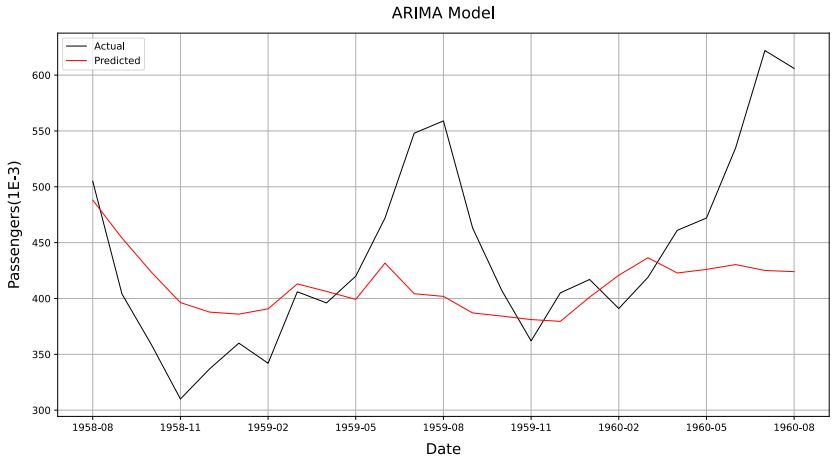


Figure 3.26: Forecast results with ARIMA model on Air Passengers dataset.

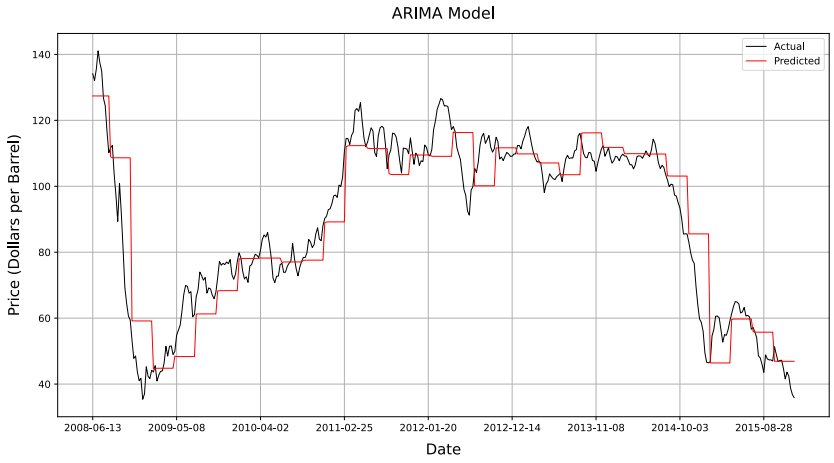


Figure 3.27: Forecast results with ARIMA model on Brent Spot Price dataset.

Table 3.3: ARIMA performance metrics.

Covid-19 casualties				**Air passengers**				**Brent oil spot price**			
MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE
0.0209	0.1089	0.1444	0.2111	0.0302	0.1328	0.174	0.1961	0.0732	0.2143	0.2706	0.4477

3.6.1.2 LSTM

Table 3.4: LSTM model hyperparameters.

Parameter \ Dataset	Covid-19 casualties	Air passengers	Brent spot price
Learning rate	1e-3	4e-3	5e-5
Hidden size	25	25	25
Number of layers	2	2	2

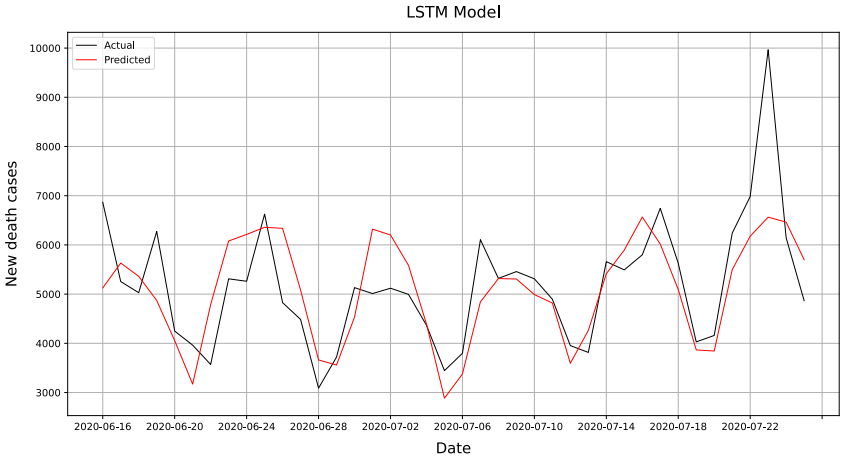


Figure 3.28: Forecast results with LSTM model on Covid-19 casualties dataset.

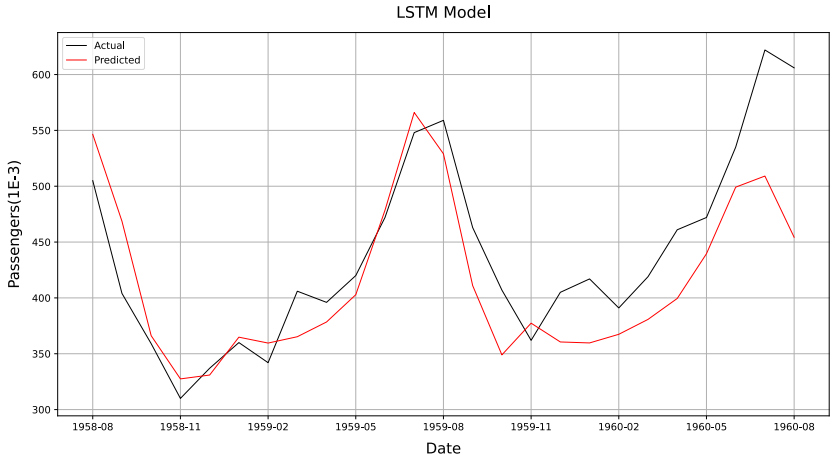


Figure 3.29: Forecast results with LSTM model on Air Passengers dataset.

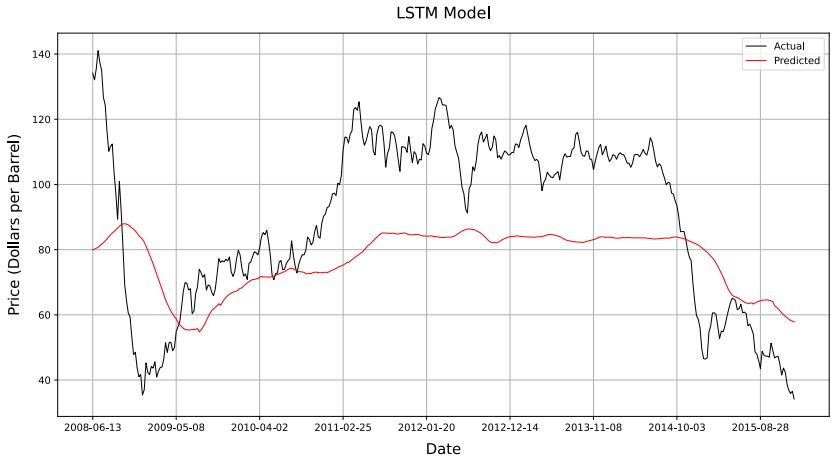


Figure 3.30: Forecast results with LSTM model on Brent Spot Price dataset.

Table 3.5: LSTM performance metrics.

Covid-19 casualties				**Air passengers**				**Brent oil spot price**			
MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE
0.0121	0.1053	0.1093	0.1447	0.0187	0.1078	0.1317	0.1602	0.0386	0.1816	0.1959	0.2989

3.6.1.3 GRU

Table 3.6: GRU model hyperparameters.

Dataset / **Parameter**	**Covid-19 casualties**	**Air passengers**	**Brent spot price**
Learning rate	1e-3	1e-3	2e-4
Hidden size	25	25	25
Number of layers	2	2	2

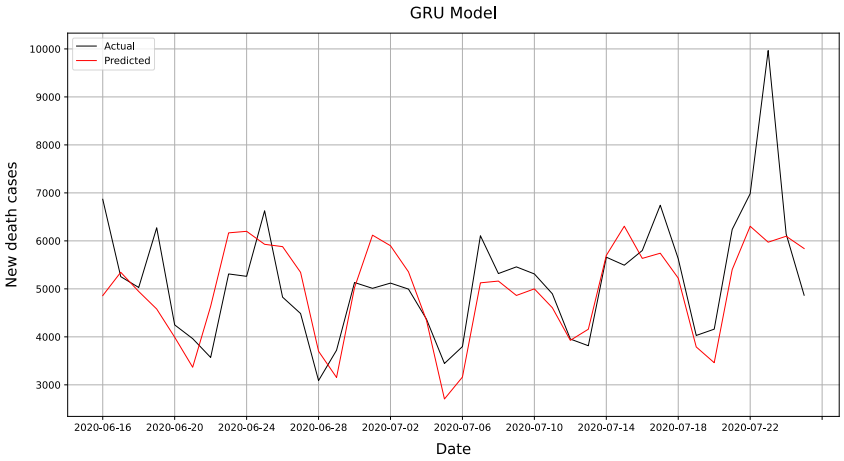


Figure 3.31: Forecast results with GRU model on Covid-19 casualties dataset.

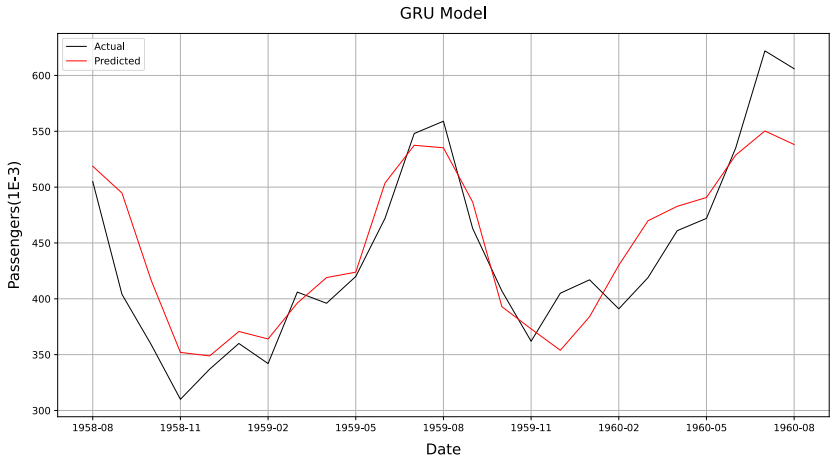


Figure 3.32: Forecast results with GRU model on Air Passengers dataset.

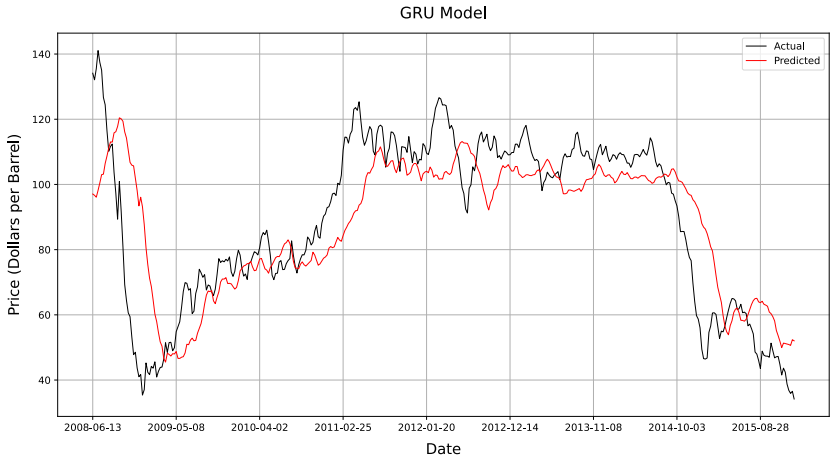


Figure 3.33: Forecast results with GRU model on Brent Spot Price dataset.

Table 3.7: GRU performance metrics.

Covid-19 casualties				**Air passengers**				**Brent oil spot price**			
MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE
0.0115	0.0763	0.1069	0.1404	0.0089	0.0736	0.0927	0.1115	0.0329	0.1339	0.1701	0.2455

3.6.1.4 GAT

Table 3.8: GAT model hyperparameters.

Dataset / **Parameter**	**Covid-19 casualties**	**Air passengers**	**Brent spot price**
Learning rate	5e-3	5e-3	7e-4
Number of layers	3	3	3
Activation function	ReLU	ReLU	ReLU

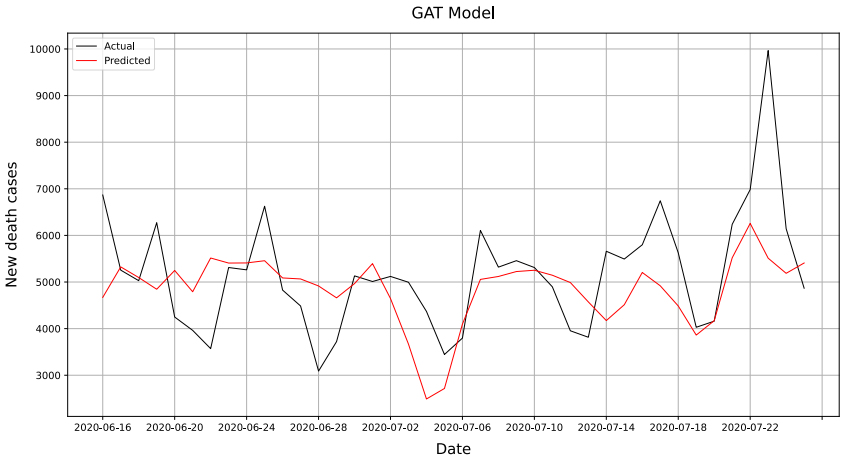


Figure 3.34: Forecast results with GAT model on Covid-19 casualties dataset.

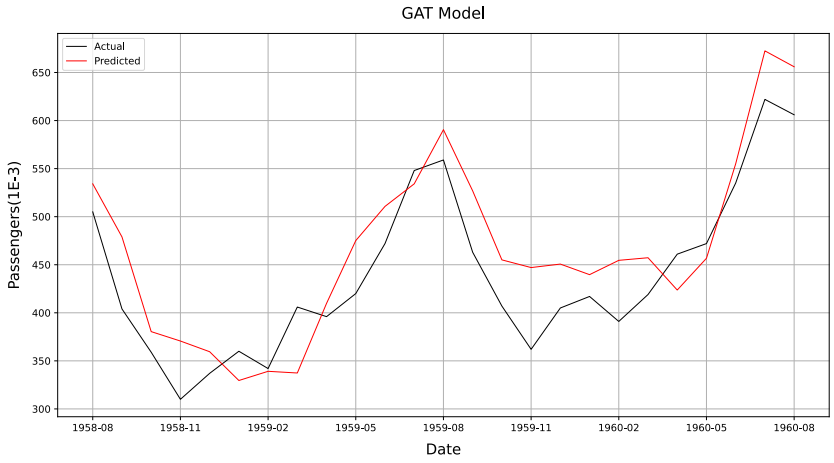


Figure 3.35: Forecast results with GAT model on Air Passengers dataset.

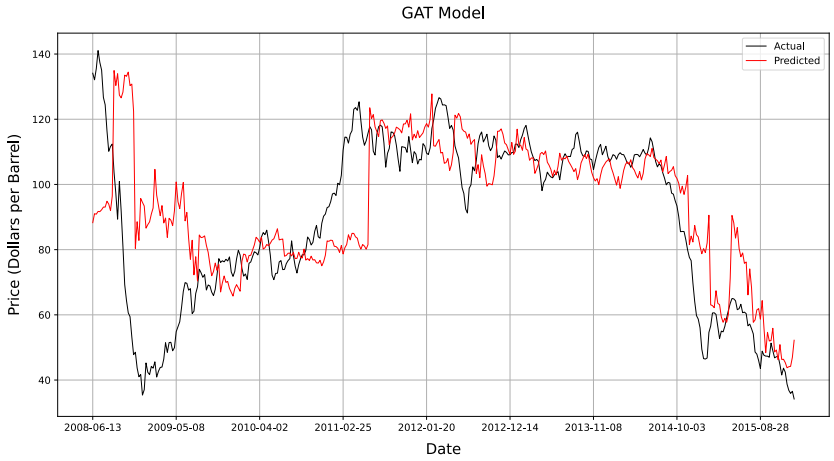


Figure 3.36: Forecast results with GAT model on Brent Spot Price dataset.

Table 3.9: GAT performance metrics

Covid-19 casualties				**Air passengers**				**Brent oil spot price**			
MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE
0.0161	0.0916	0.1268	0.1748	0.0101	0.0889	0.0998	0.1491	0.0335	0.1367	0.1804	0.2726

3.6.1.5 Transformer

Table 3.10: Transformer model hyperparameters.

Parameter \ **Dataset**	**Covid-19 casualties**	**Air passengers**	**Brent spot price**
Learning rate	5e-3	5e-3	7e-4
Number of layers	4	4	4
Attention heads	6	6	6
d_{val}	24	24	24
Feed forward dimension	64	64	64

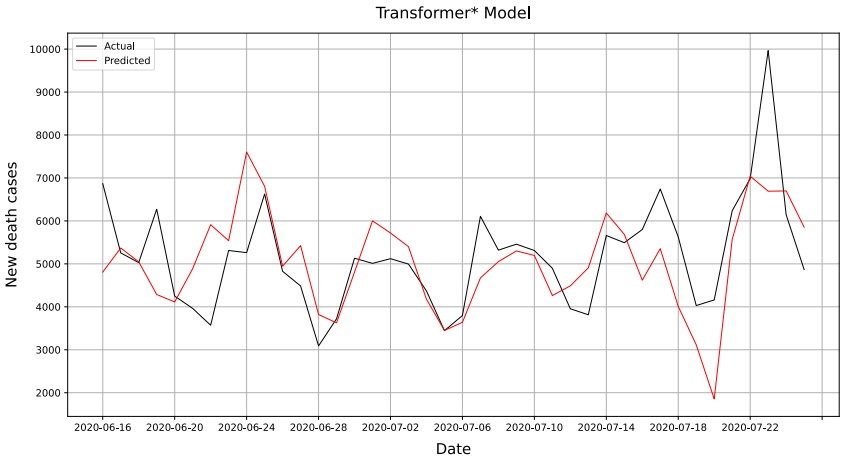


Figure 3.37: Forecast results with Transformer model on Covid-19 casualties dataset.

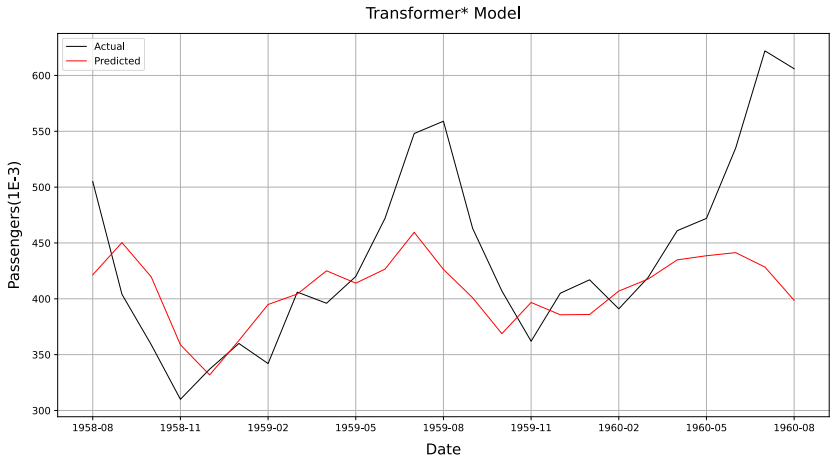


Figure 3.38: Forecast results with Transformer model on Air Passengers dataset.

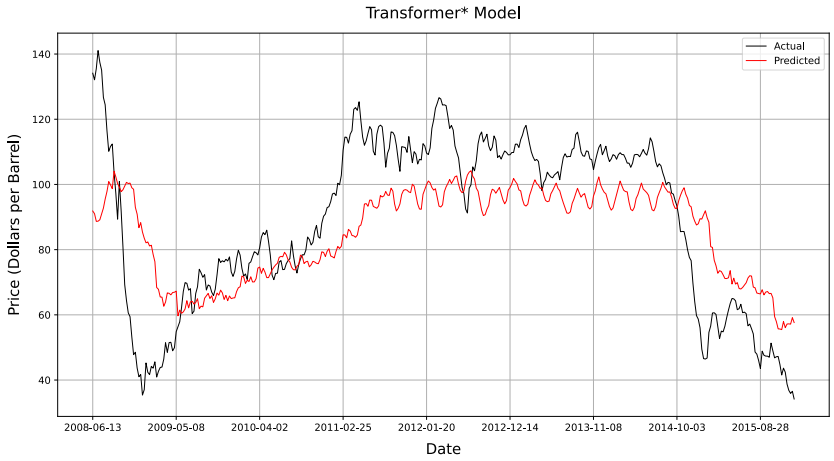


Figure 3.39: Forecast results with Transformer model on Brent Spot Price dataset.

Table 3.11: Transformer performance metrics.

Covid-19 casualties				**Air passengers**				**Brent oil spot price**			
MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE
0.0303	0.1353	0.1742	0.2702	0.0374	0.1467	0.1898	0.2205	0.0508	0.1904	0.2238	0.3776

3.6.1.6 TCN

Table 3.12: TCN model hyperparameters.

Dataset / **Parameter**	**Covid-19 casualties**	**Air passengers**	**Brent spot price**
Learning rate	4e-3	2e-3	1e-3
Activation function	ReLU	ReLU	ReLU
Number of residual blocks	12	12	12

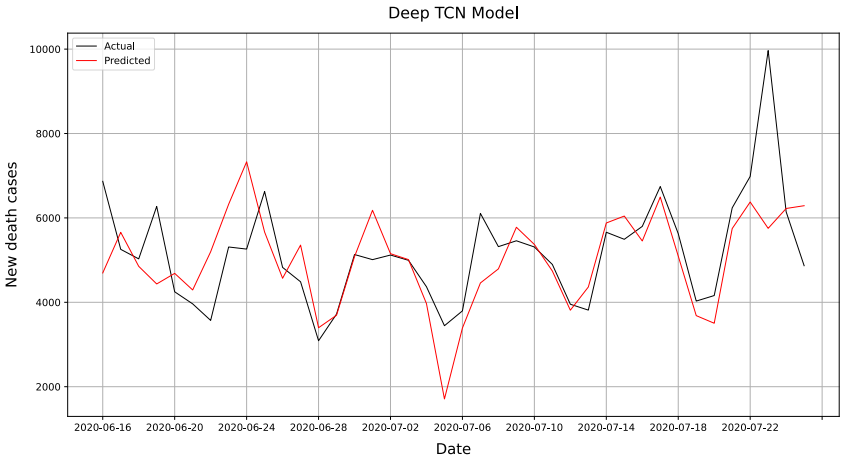


Figure 3.40: Forecast results with Deep TCN model on Covid-19 casualties dataset.

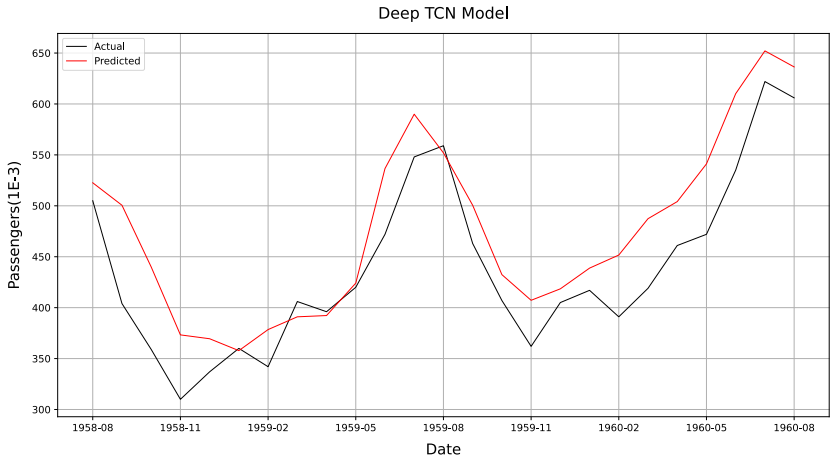


Figure 3.41: Forecast results with Deep TCN model on Air Passengers dataset.

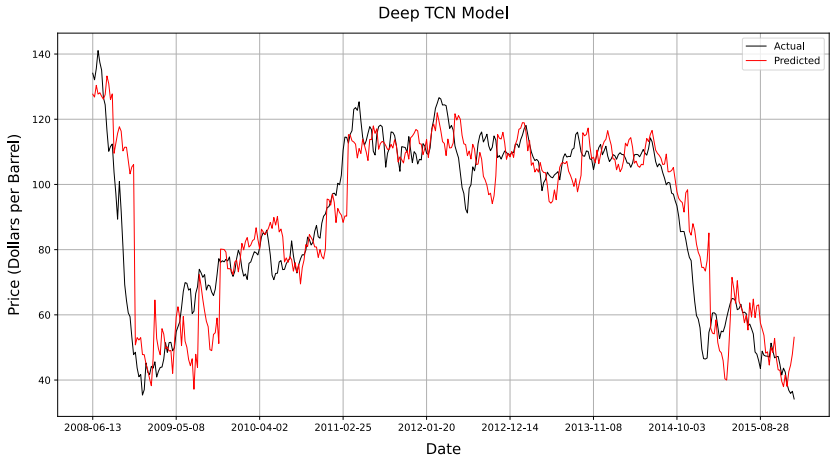


Figure 3.42: Forecast results with Deep TCN model on Brent Spot Price dataset.

Table 3.13: Deep TCN performance metrics.

Covid-19 casualties				**Air passengers**				**Brent oil spot price**			
MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE
0.0174	0.1806	0.1308	0.0963	0.0141	0.0939	0.1162	0.1471	0.0159	0.0866	0.1209	0.1708

3.6.1.7 ARIMA-LSTM

Table 3.14: ARIMA-LSTM model hyperparameters.

Dataset / **Parameter**	**Covid-19 casualties**	**Air passengers**	**Brent spot price**
p, d, q	2,1,0	2,1,2	2,1,1
Learning rate	1e-3	2e-4	1e-4
Number of layers	2	2	2
Hidden size	25	25	25

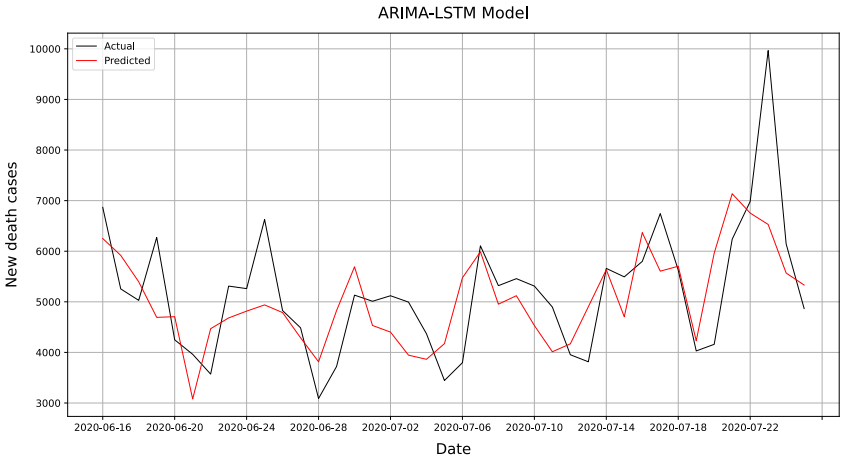


Figure 3.43: Forecast results with ARIMA-LSTM model on Covid-19 casualties dataset.

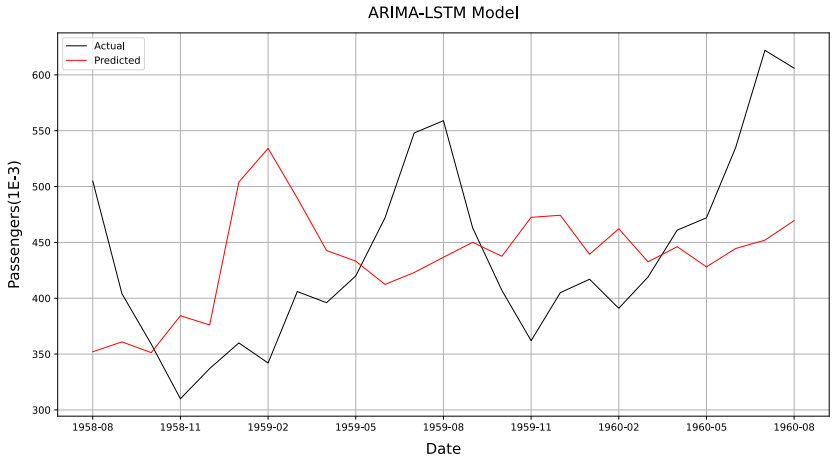


Figure 3.44: Forecast results with ARIMA-LSTM model on Air passenger dataset.

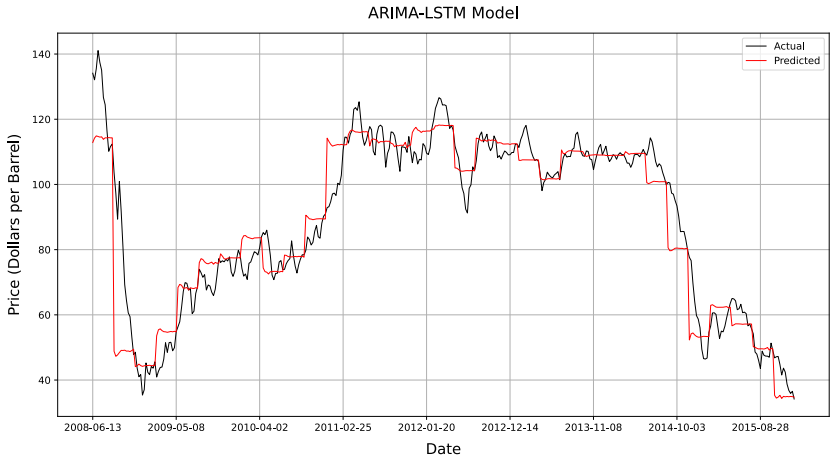


Figure 3.45: Forecast results with ARIMA-LSTM model on Brent Spot Price dataset.

Table 3.15: ARIMA-LSTM performance metrics.

Covid-19 casualties				**Air passengers**				**Brent oil spot price**			
MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE
0.0323	0.1537	0.1791	0.3193	0.1082	0.2792	0.3292	0.4417	0.0196	0.1165	0.1401	0.2201

3.6.1.8 ARIMA-GRU

Table 3.16: ARIMA-GRU model hyperparameters.

Dataset / **Parameter**	**Covid-19 casualties**	**Air passengers**	**Brent spot price**
p, d, q	2,1,0	2,1,2	2,1,1
Learning rate	1e-3	1e-3	5e-4
Number of layers	2	2	2
Hidden size	25	25	25

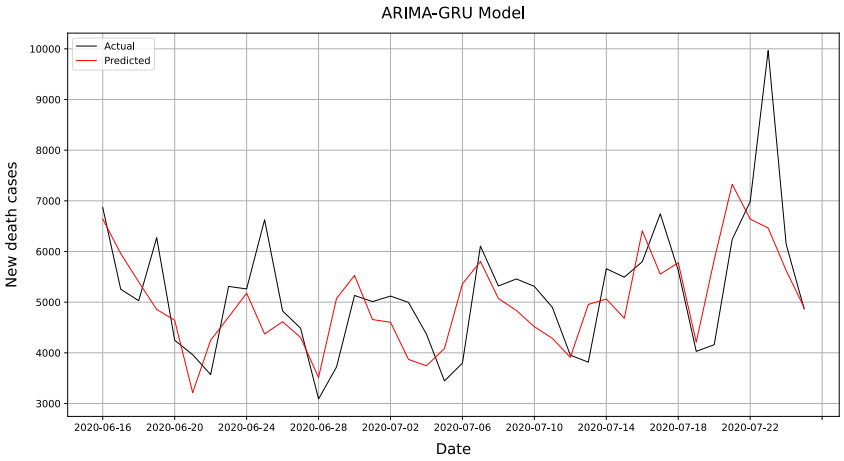


Figure 3.46: Forecast results with ARIMA-GRU model on Covid-19 casualties dataset.

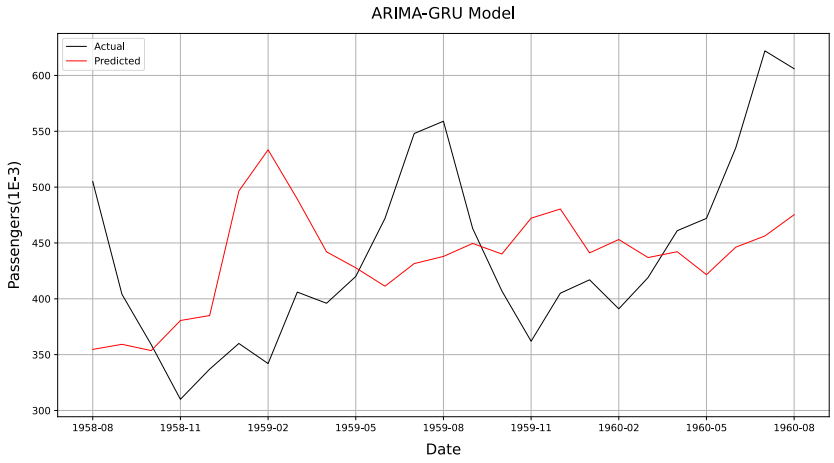


Figure 3.47: Forecast results with ARIMA-GRU model on Air passenger dataset.

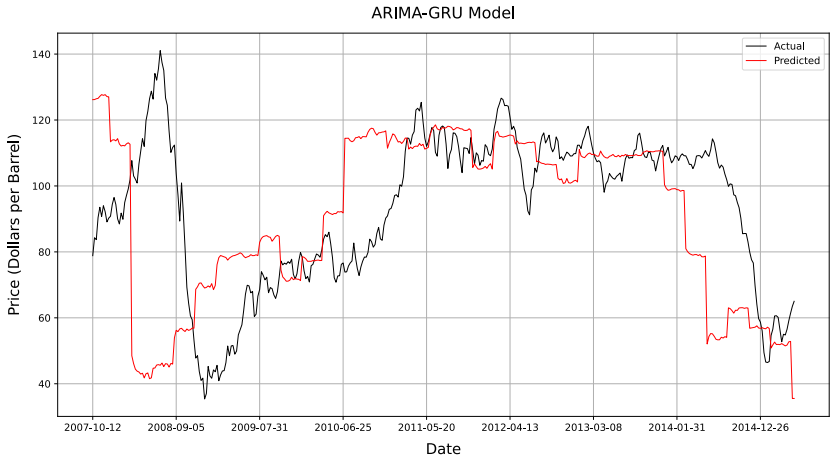


Figure 3.48: Forecast results with ARIMA-GRU model on Brent Spot Price dataset.

Table 3.17: ARIMA-GRU performance metrics.

Covid-19 casualties				**Air passengers**				**Brent oil spot price**			
MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE
0.0368	0.1623	0.1911	0.3341	0.1101	0.2818	0.3318	0.445	0.1161	0.1161	0.1137	0.216

3.6.1.9 LSTM-GRU

Table 3.18: LSTM-GRU model hyperparameters.

Parameter \ **Dataset**	**Covid-19 casualties**	**Air passengers**	**Brent spot price**
Learning rate	1e-3	1e-3	1e-4
Number of layers	2, 2	2, 2	2, 2
Hidden size	25, 25	25, 25	25, 25

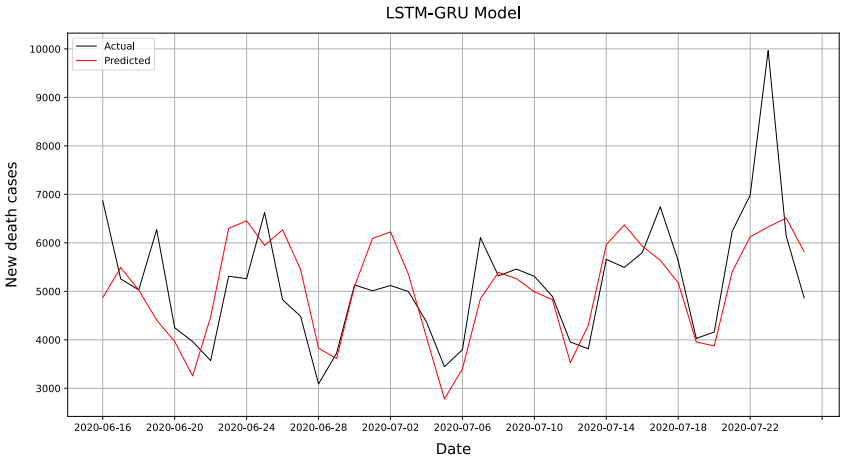


Figure 3.49: Forecast results with LSTM-GRU model on Covid-19 casualties dataset.

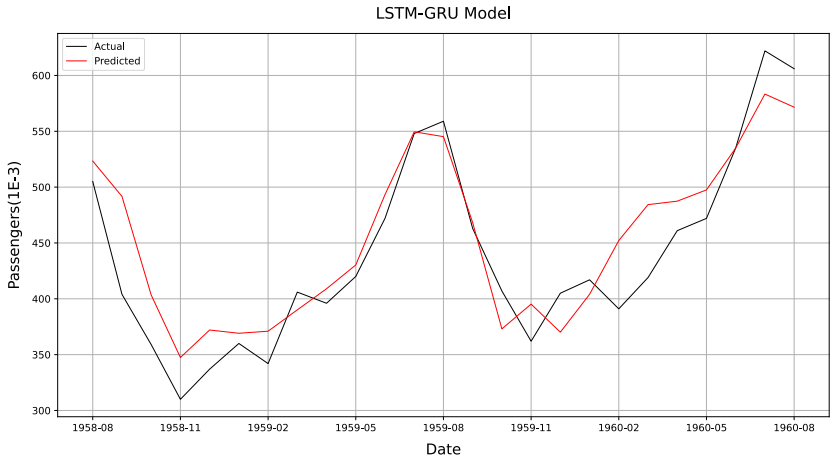


Figure 3.50: Forecast results with LSTM-GRU model on Air passenger dataset.

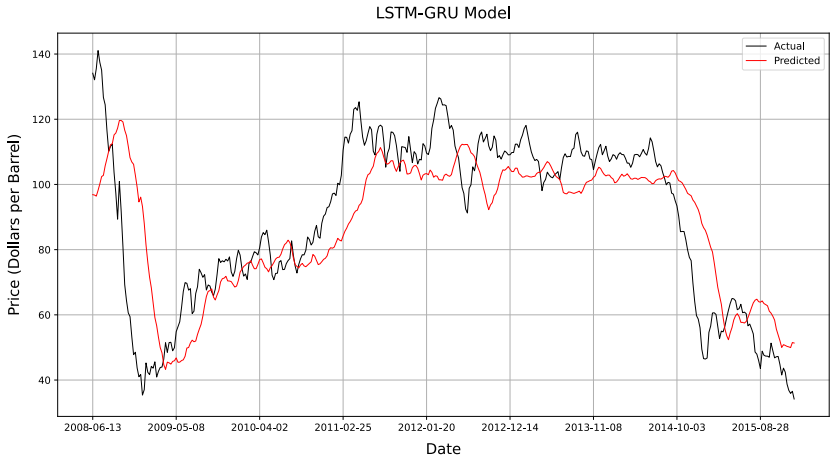


Figure 3.51: Forecast results with LSTM-GRU model on Brent Spot Price dataset.

Table 3.19: LSTM-GRU performance metrics.

Covid-19 casualties				**Air passengers**				**Brent oil spot price**			
MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE
0.0119	0.0788	0.1089	0.1446	0.0088	0.0731	0.0925	0.1105	0.0163	0.0951	0.1298	0.2017

3.6.1.10 CNN-LSTM

Table 3.20: CNN-LSTM model hyperparameters.

Parameter \ **Dataset**	**Covid-19 casualties**	**Air passengers**	**Brent spot price**
Learning rate	6e-3	2e-3	4e-5
Hidden size	1	1	1
Number of layers	2	2	2
Kernel size	5/2	5	5

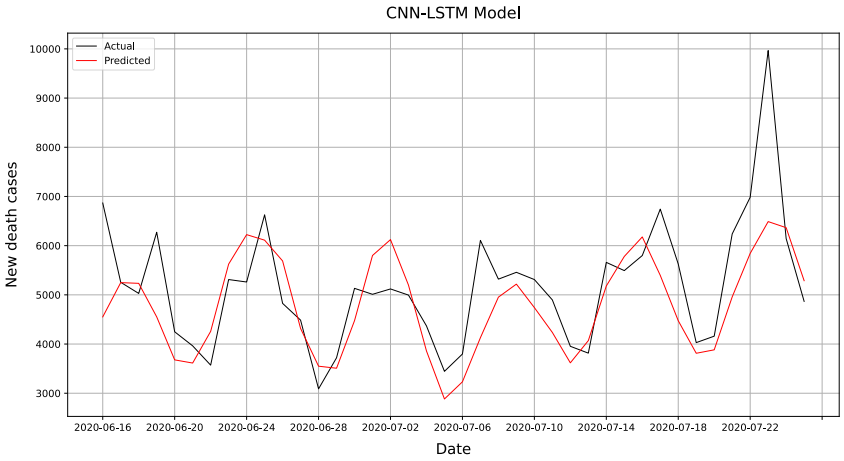


Figure 3.52: Forecast results with CNN-LSTM model on Covid-19 casualties dataset.

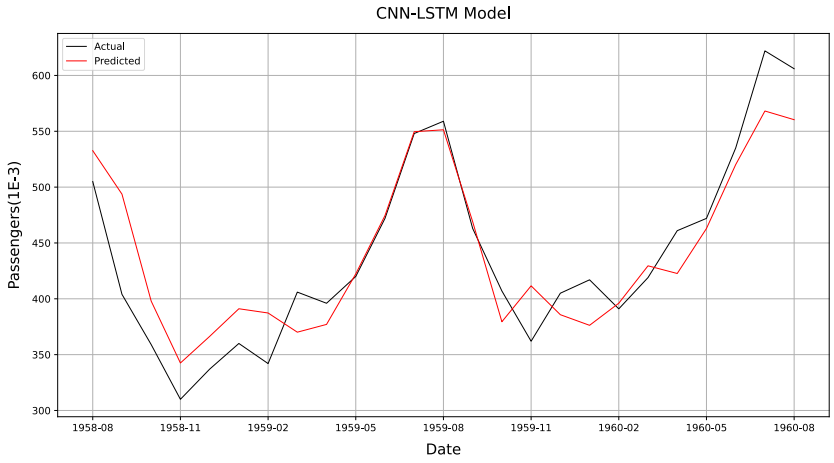


Figure 3.53: Forecast results with CNN-LSTM model on Air passenger dataset.

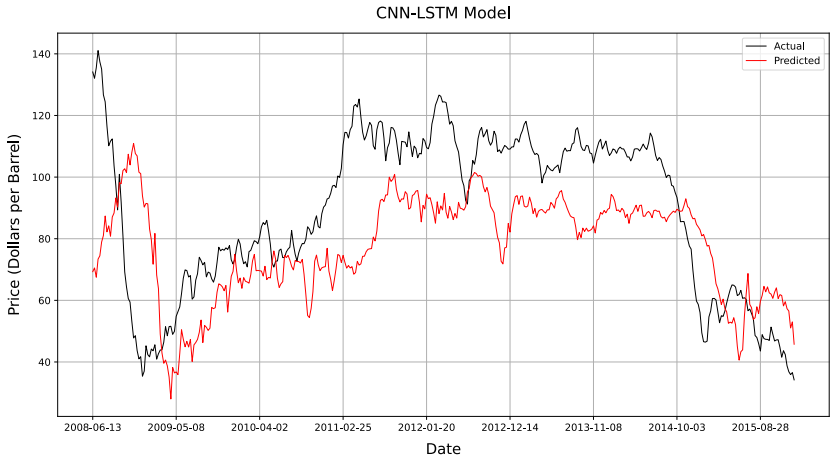


Figure 3.54: Forecast results with CNN-LSTM model on Brent Spot Price dataset.

Table 3.21: CNN-LSTM performance metrics.

Covid-19 casualties				**Air passengers**				**Brent oil spot price**			
MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE
0.0102	0.0727	0.1011	0.1292	0.0068	0.0649	0.0814	0.1078	0.0213	0.1251	0.1457	0.1982

3.6.1.11 CNN-GRU

Table 3.22: CNN-GRU model hyperparameters.

Parameter \ **Dataset**	**Covid-19 casualties**	**Air passengers**	**Brent spot price**
Learning rate	1e-3	3e-3	1e-4
Hidden size	1	1	1
Number of layers	2	2	2
Kernel size	5/2	5	5

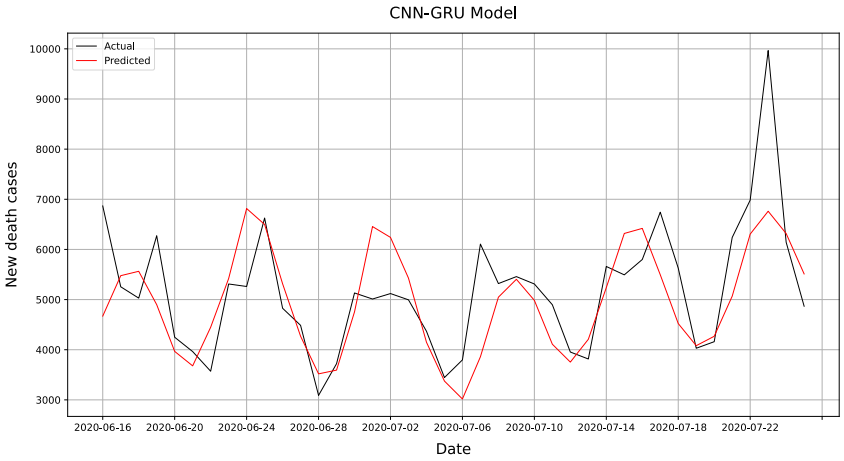


Figure 3.55: Forecast results with CNN-GRU model on Covid-19 casualties dataset.

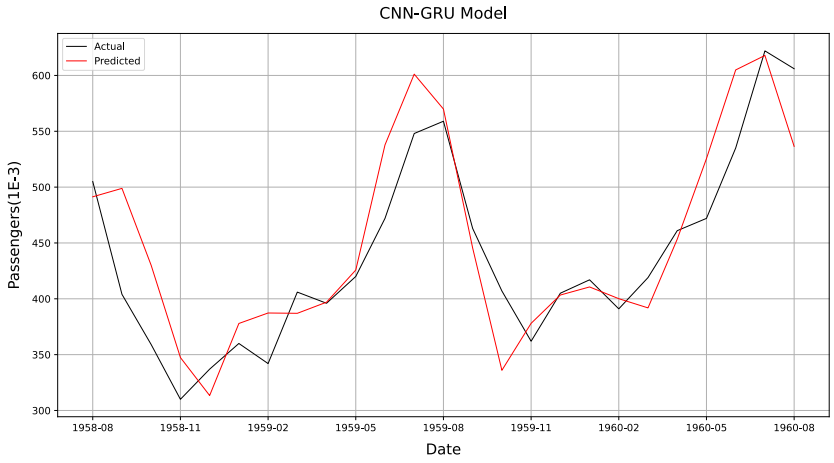


Figure 3.56: Forecast results with CNN-GRU model on Air passenger dataset.

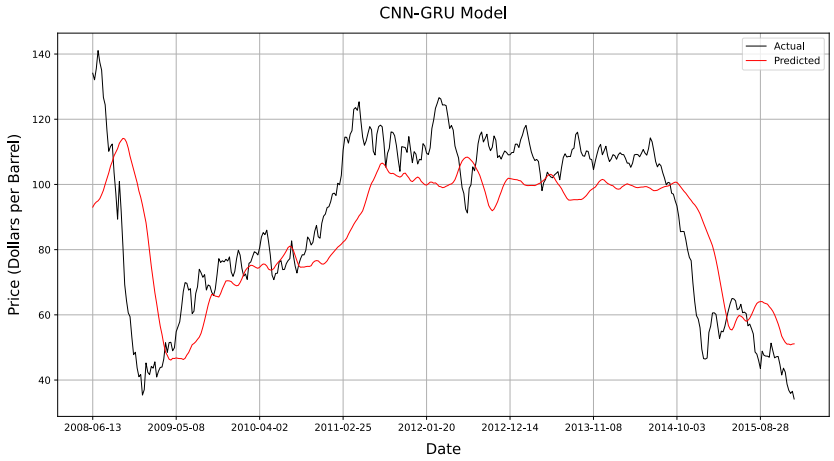


Figure 3.57: Forecast results with CNN-GRU model on Brent Spot Price dataset.

Table 3.23: CNN-GRU performance metrics.

Covid-19 casualties				**Air passengers**				**Brent oil spot price**			
MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE
0.0104	0.0739	0.1019	0.132	0.0123	0.0899	0.1085	0.1425	0.0187	0.1011	0.1367	0.2183

3.6.2 Proposed models performance

3.6.2.1 CNN-GRU-TCN

Table 3.24: CNN-GRU-TCN model hyperparameters.

Dataset / **Parameter**	**Covid-19 casualties**	**Air passengers**	**Brent spot price**
Learning rate	1e-3	2e-2	1e-3
Number of residual blocks	12	5	12
Hidden size	25	25	25
Number of layers	2	2	2
Kernel size	3	3	3

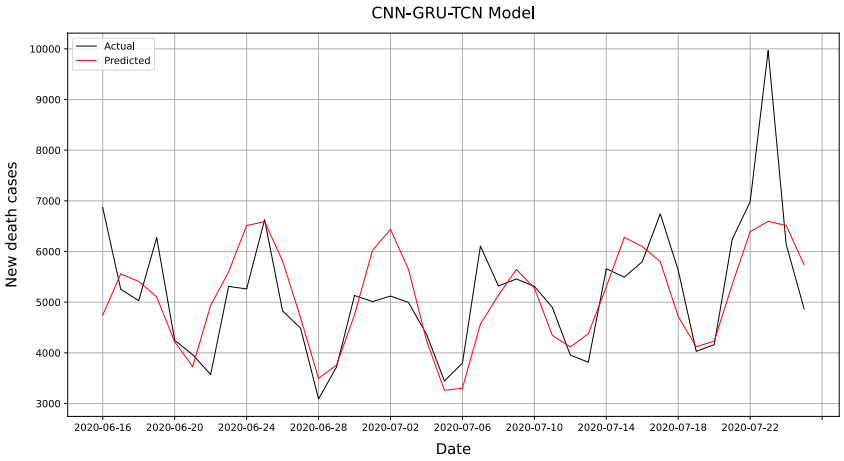


Figure 3.58: Forecast results with CNN-GRU-TCN model on Covid-19 casualties dataset.

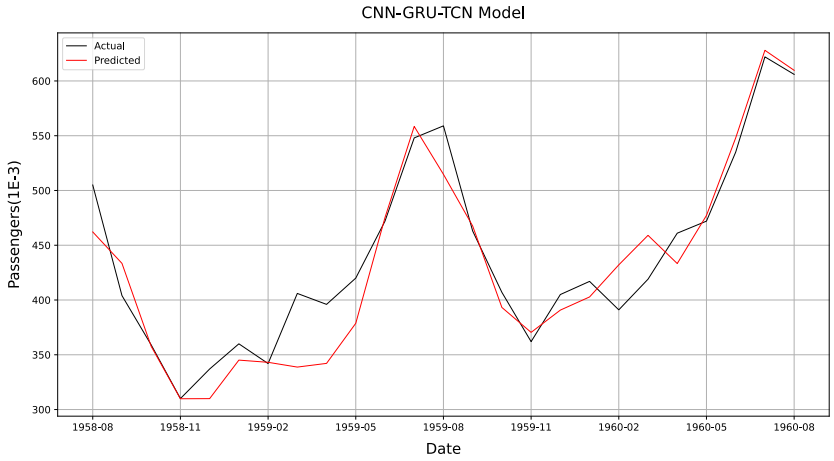


Figure 3.59: Forecast results with CNN-GRU-TCN model on Air passenger dataset.

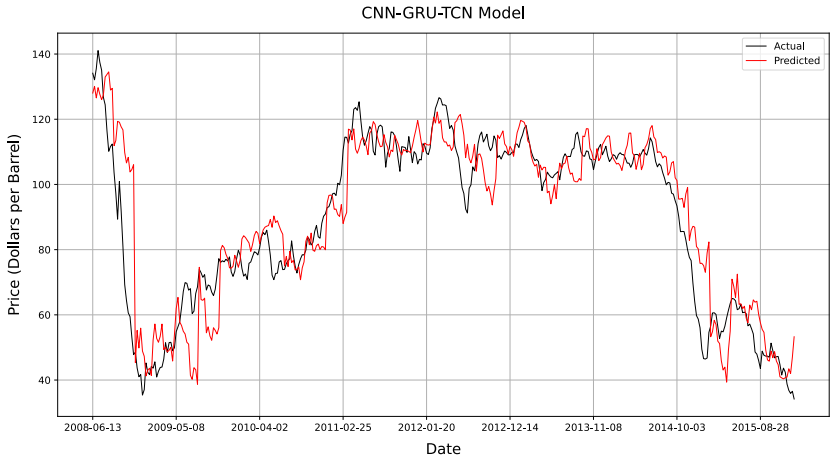


Figure 3.60: Forecast results with CNN-GRU-TCN model on Brent Spot Price dataset.

Table 3.25: CNN-GRU-TCN performance metrics.

Covid-19 casualties				**Air passengers**				**Brent oil spot price**			
MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE
0.0094	0.0671	0.0973	0.1195	0.0052	0.0566	0.0709	0.0937	0.0071	0.0614	0.0889	0.1294

3.6.2.2 GNN-CNN-LSTM

Table 3.26: GNN-CNN-LSTM model hyperparameters.

Dataset / **Parameter**	**Covid-19 casualties**	**Air passengers**	**Brent spot price**
Learning rate	1e-3	1e-2	1e-3
Number of GNN layers	3	3	3
Hidden size	25	25	25
Number of LSTM layers	2	1	2
Kernel size (CNN)	3	2	5

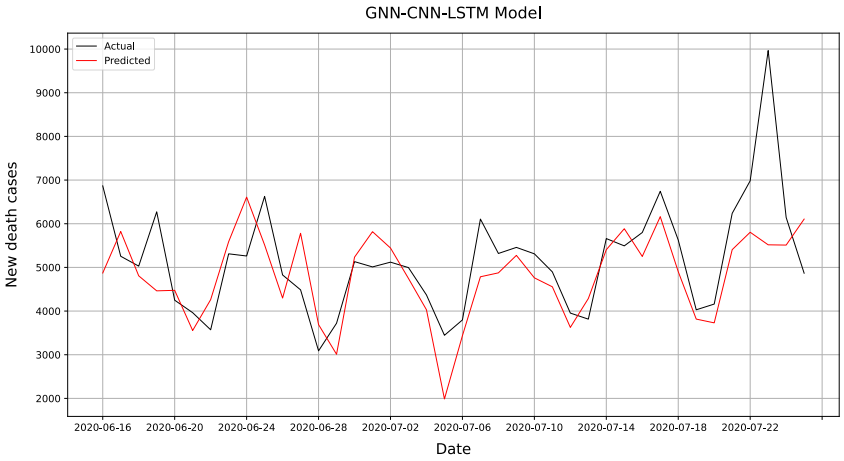


Figure 3.61: Forecast results with GNN-CNN-LSTM model on Covid-19 casualties dataset.

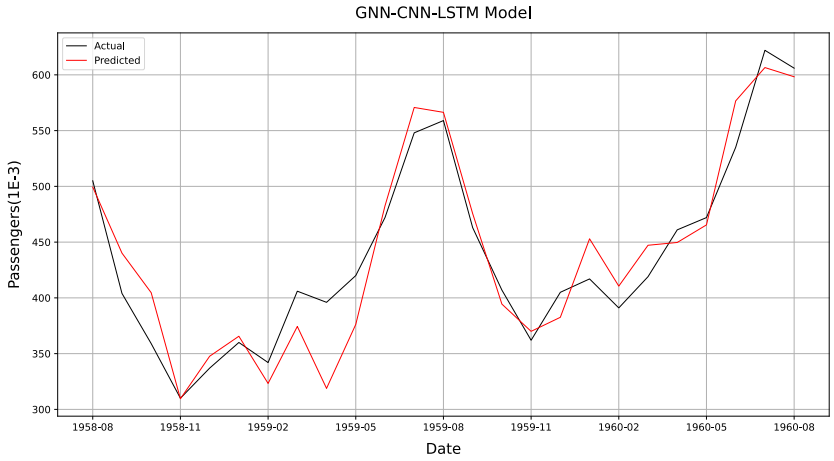


Figure 3.62: Forecast results with GNN-CNN-LSTM model on Air passenger dataset.

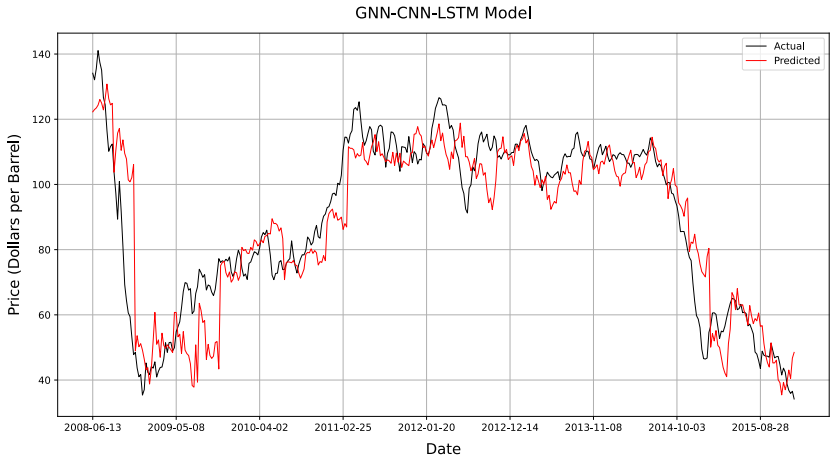


Figure 3.63: Forecast results with GNN-CNN-LSTM model on Brent Spot Price dataset.

Table 3.27: GNN-CNN-LSTM performance metrics.

Covid-19 casualties				**Air passengers**				**Brent oil spot price**			
MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE
0.0114	0.0766	0.1071	0.1408	0.0034	0.0451	0.0583	0.0746	0.0115	0.0771	0.1063	0.1548

Table 3.28: Model performance summary table.

	Covid-19 casualties				**Air passengers**				**Brent oil spot price**			
	MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE	MSE	MAE	RMSE	MAPE
GAT	0.0161	0.0916	0.1268	0.1748	0.0101	0.0889	0.0998	0.1491	0.0335	0.1367	0.1804	0.2726
LSTM	0.0121	0.1053	0.1093	0.1447	0.0187	0.1078	0.1317	0.1602	0.0386	0.1816	0.1959	0.2989
GRU	0.0115	0.0763	0.1069	0.1404	0.0089	0.0736	0.0927	0.1115	0.0329	0.1339	0.1701	0.2455
LSTM-GRU	0.0119	0.0788	0.1089	0.1446	0.0088	0.0731	0.0925	0.1105	0.0163	0.0951	0.1298	0.2017
CNN-GRU	0.0104	0.0739	0.1019	0.132*	0.0123	0.0899	0.1085	0.1425	0.0187	0.1011	0.1367	0.2183
CNN-LSTM	0.0102	0.0727	0.1011	0.1292	0.0068*	0.0649*	0.0814*	0.1078*	0.0213	0.1251	0.1457	0.1982
CNN-GRU-TCN	**0.0094**	**0.0671**	**0.0973**	**0.1195**	0.0052	0.0566	0.0709	0.0937	**0.0071**	**0.0614**	**0.0889**	**0.1294**
GNN-CNN-LSTM	0.0114*	0.0766*	0.1071*	0.1408*	**0.0034**	**0.0451**	**0.0583**	**0.0746**	0.0115	0.0771	0.1063	0.1548
Transformer*	0.0303	0.1353	0.1742	0.2702	0.0374	0.1467	0.1898	0.2205	0.0508	0.1904	0.2238	0.3776
TCN	0.0174	0.1806	0.1308	0.0963	0.0141	0.0939	0.1162	0.1471	0.0159*	0.0866*	0.1209*	0.1708*
ARIMA	0.0209	0.1089	0.1444	0.2111	0.0302	0.1328	0.174	0.1961	0.0732	0.2143	0.2706	0.4477
ARIMA-GRU	0.0368	0.1623	0.1911	0.3341	0.1101	0.2818	0.3318	0.445	0.1161	0.1161	0.1137	0.216
ARIMA-LSTM	0.0323	0.1537	0.1791	0.3193	0.1082	0.2792	0.3292	0.4417	0.0196	0.1165	0.1401	0.2201

Table 3.28 provides a summary of the models performance, best results are in **bold**, second best are underlined and third best are noted by an asterisk*.

3.7 Discussion

, we would like to carry the comparative analysis in the following points:

3.7.1 ARIMA vs Deep learning techniques

Based on the findings presented in Table 3.28, it is evident that the ARIMA model shows inferior performance in direct multi-step ahead forecasting when compared to deep learning techniques. In fact, with the exception for the Air passengers dataset where it outperformed the Transformer model, ARIMA has the worst score among all models. This highlights that the autoregressive nature of ARIMA makes it ill-suited for direct multi-step ahead forecasting. Instead, ARIMA aligns more effectively with the recursive strategy.
The performance disparity between ARIMA and deep learning in direct multi-step ahead forecasting reinforces the limitations of its autoregressive method and emphasizes the importance of adopting the appropriate forecasting strategy depending on the task's specific requirements.

3.7.2 Hybrid vs Individual models

When compared to their individual counterparts, deep learning models demonstrated a general supremacy in forecasting accuracy as they improved upon their individual component networks in almost all cases as shown in Table 3.28: LSTM-GRU combination yielded higher forecasting accuracy in all datasets compared to LSTM and GRU networks with improvement reaching up to 58% and 51% decrease in MSE on Brent spot price dataset. Similar results were achieved with CNN-GRU and CNN-LSTM combinations. As for hybridizations including ARIMA, forecast accuracy didn't improve, on the contrary, the forecast accuracy deteriorated and this is probably due to ARIMA's influence on the hybridization as it was used to predict future trend values. This result may also suggests that the component decomposition strategy with the employment of ARIMA is not be optimal for direct multi-step ahead forecasting.

3.7.3 Comparison against proposed models

To assess the effectiveness and robustness of our proposed models, we tested and compared them against eleven baseline models, the results were indeed positive, our novel architectures showcased improved overall accuracy with significant margins on

some datasets.
In terms of the Covid-19 casualties dataset, the performance results were quite competitive among the models. The CNN-GRU-TCN model emerged as the top performer, closely followed by the CNN-LSTM model. The CNN-GRU and GNN-CNN-LSTM models secured the third and fourth positions, respectively, with MSE scores of 0.0094, 0.0102, 0.0104, and 0.0114.

However, for the Air passengers and Brent spot price datasets, our proposed models showcased remarkable dominance, outperforming other models by significant margins. The GNN-CNN-LSTM and CNN-GRU-TCN networks demonstrated exceptional performance on the Air passengers dataset, achieving a 50% and 23% reduction in MSE, respectively, compared to the next best model CNN-LSTM. Similarly, on the Brent spot price dataset, the GNN-CNN-LSTM and CNN-GRU-TCN models achieved a significant 28% and 55% decrease in MSE, respectively, compared to the third-best performing model.

These results highlight the superiority of our proposed models in capturing the intricate patterns and dependencies and in accurately predicting future values and outperforming competing state-of-the-art approaches.

Conclusion

In this chapter we conducted an experimental evaluation and a comparative analysis to investigate the effectiveness of the single and hybrid approaches to univariate multi-step ahead time series forecasting. We tested and compared a collection state-of-the-art models on three real world time series datasets. The covered models range from classic, deep learning and hybrid models including our two novel hybrid deep learning frameworks proposed in this thesis.
The experiment results yielded several important findings:

- Classic ARIMA model demonstrated unsatisfactory performance and exhibited limitations in directly predicting multiple future time points.
- Deep learning models showcased superior performance and succeeded to a satisfactory degree in modeling complex patterns and relationships and providing an acceptable forecast accuracy.
- Hybrid deep learning models showed varying degrees of improvement over individual models.

Finally our proposed models showcased remarkable improvements and outperformed both individual and hybrid state-of-the-art models.

The experiment results provide compelling evidence that deep learning models outperform classic models in terms of forecasting accuracy, adaptability to complex patterns in direct multi-step ahead univariate time series forecasting. It proves as well that the hybridization of these models is an effective way to improve models performance and forecast accuracy.

General conclusion and future work

The primary aim of this thesis was to investigate the effectiveness of the hybrid approach to time series forecasting and develop a novel hybrid model that leverages strengths of different models, enhances forecast accuracy, and eventually outperforming existing methods.

Throughout the course of this thesis we provided a sufficient theoretical background on time series analysis and forecasting, we then presented a thorough literature review of the subject where a large selection of forecasting models were showcased, ranging from classic and machine learning to deep learning and hybrid models, a discussion on the challenges and considerations to the hybrid approach concluded the review.

The final chapter highlighted our proposition of two novel deep learning based hybrid architectures (CNN-GRU-TCN and GNN-CNN-LSTM) and the comprehensive experiment of testing them for direct multi-step ahead univariate time series forecasting and comparing them against eleven state-state-of-the-art baseline models using three time series datasets. The evaluation was conducted based on four different regression performance metrics (MSE, RMSE, MAE, MAPE) and as shown in Table 3.28 our proposed models outperformed the baseline models in most experimental scenarios, fulfilling therefore the purpose of this work.

Finally we hope that the findings of this research will have practical impact as the enhanced accuracy of the proposed model can help improve real world forecasting tasks and lead to better planning decision-making.

Direction of future research

Based on the findings of this work, the scope of the paper can be extended in several directions, namely:

- **Better hyperparameter optimization:** due to the limited hardware, time and financial resources a limited hyperparameter tuning was employed, we believe that a more thorough process can be driven to further tune the proposed models.

- **Testing the models on other datasets :**the limited time resource had a direct impact on the number of datasets employed for the comparative analysis, the models are to be tested on more datasets to solidfy the findings of this study.
- **Applications on multivariate time series forecasting:** we performed our experiments and based our results on univariate time series forecasting, further experiments on different types of time series can be conducted.
- **Experiments with variant lookback windows and forecast horizons:** investigating the proposed models performance on varying lookback windows and forecast horizon is still an open research problem.
- **Exploration of other forecasting strategies:** This work's experiments and results were based on the direct multi-step ahead forecasting strategy, the proposed models can be tested and evaluated with other forecasting strategies such as the recursive multi-step ahead and hybrid direct-recursive approaches.

Bibliography

[1] Nesreen K Ahmed, Amir F Atiya, Neamat El Gayar, and Hisham El-Shishiny. An empirical comparison of machine learning models for time series forecasting. *Econometric reviews*, 29(5-6):594–621, 2010.

[2] Elham M. Al-Ali, Yassine Hajji, Yahia Said, Manel Hleili, Amal M. Alanzi, Ali H. Laatar, and Mohamed Atri. Solar energy production forecasting based on a hybrid cnn-lstm-transformer model. *Mathematics*, 11(3), 2023.

[3] Noor Ali and Firas Mohammed. The use of arima, lstm and gru models in time series hybridization with practical application. *International Journal of Nonlinear Analysis and Applications*, 14(1):1371–1383, 2023.

[4] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, 2018.

[5] Candice Bentéjac, Anna Csörgő, and Gonzalo Martínez-Muñoz. A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, 54:1937–1967, 2021.

[6] Søren Bisgaard and Murat Kulahci. *Time series analysis and forecasting by example*. John Wiley & Sons, 2011.

[7] G.E.P. Box, G.M. Jenkins, G.C. Reinsel, and G.M. Ljung. *Time Series Analysis: Forecasting and Control*. Wiley Series in Probability and Statistics. Wiley, 2015.

[8] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, jul 2017.

[9] Umair Muneer Butt, Sukumar Letchmunan, Fadratul Hafinaz Hassan, and Tieng Wei Koh. Hybrid of deep learning and exponential smoothing for enhancing crime forecasting accuracy. *PLOS ONE*, 17(9):1–22, 09 2022.

[10] Tianqi Chen and Carlos Guestrin. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2016.

[11] Bahram Choubin, Gholamreza Zehtabian, Ali Azareh, Elham Rafiei-Sardooi, Farzaneh Sajedi-Hosseini, and Özgür Kişi. Precipitation forecasting using classification and regression trees (cart) model: a comparative study of different approaches. *Environmental earth sciences*, 2018.

[12] Emmanuel Dave, Albert Leonardo, Marethia Jeanice, and Novita Hanafiah. Forecasting indonesia exports using a hybrid model arima-lstm. *Procedia Computer Science*, 179:480–487, 2021.

[13] Grzegorz Dudek, Paweł Pełka, and Slawek Smyl. A hybrid residual dilated lstm and exponential smoothing model for midterm electric load forecasting. *IEEE Transactions on Neural Networks and Learning Systems*, 33(7):2879–2891, 2022.

[14] Snezhana Georgieva Gocheva-Ilieva, Desislava Stoyanova Voynikova, Maya Plamenova Stoimenova, Atanas Valev Ivanov, and Iliycho Petkov Iliev. Regression trees modeling of time series for air pollution analysis and forecasting. *Neural Computing and Applications*, 2019.

[15] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[16] Shipeng Han, Zhen Meng, Xingcheng Zhang, and Yuepeng Yan. Hybrid deep recurrent neural networks for noise reduction of mems-imu with static and dynamic conditions. *Micromachines*, 12:214, 02 2021.

[17] Yaru Hao, Li Dong, Furu Wei, and Ke Xu. Self-attention attribution: Interpreting information interactions inside transformer, 2021.

[18] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1):388–427, 2021.

[19] Ahmedbahaaaldin Ibrahem Ahmed Osman, Ali Najah Ahmed, Ming Fai Chow, Yuk Feng Huang, and Ahmed El-Shafie. Extreme gradient boosting (xgboost) model to predict the groundwater levels in selangor malaysia. *Ain Shams Engineering Journal*, 12(2):1545–1556, 2021.

[20] M.S. Islam and E. Hossain. Foreign exchange currency rate prediction using a gru-lstm hybrid network. *Soft Computing Letters*, 3:100009, 2021.

[21] Aissam Jadli, El Habib Ben Lahmer, et al. A novel lstm-gru-based hybrid approach for electrical products demand forecasting. *International Journal of Intelligent Engineering & Systems*, 15(3), 2022.

[22] Lipeng Ji, Chenqi Fu, Zheng Ju, Yicheng Shi, Shun Wu, and Li Tao. Short-term canyon wind speed prediction based on cnn—gru transfer learning. *Atmosphere*, 13(5):813, 2022.

[23] Tae-Young Kim and Sung-Bae Cho. Predicting residential energy consumption using cnn-lstm neural networks. *Energy*, 182:72–81, 2019.

[24] C. Koushik, M. V. Pranav, R. K. Arjun, and S. Shridevi. Hybrid exponential smoothing-lstm-based univariate stock market prediction for financial sectors in nifty50. In Rabindra Nath Shaw, Sanjoy Das, Vincenzo Piuri, and Monica Bianchini, editors, *Advanced Computing and Intelligent Technologies*, pages 357–368, Singapore, 2022. Springer Nature Singapore.

[25] Lucas Lacasa, Bartolo Luque, Fernando Ballesteros, Jordi Luque, and Juan Carlos Nuno. From time series to complex networks: The visibility graph. *Proceedings of the National Academy of Sciences*, 105(13):4972–4975, 2008.

[26] Pedro Lara-Benítez, Manuel Carranza-García, José M. Luna-Romera, and José C. Riquelme. Temporal convolutional networks applied to energy-related time series forecasting. *Applied Sciences*, 10(7), 2020.

[27] Colin Lea, Michael D. Flynn, Rene Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks for action segmentation and detection, 2016.

[28] ND Lewis. *Deep time series forecasting with Python: an intuitive introduction to deep learning for applied time series modeling*. ND Lewis, 2016.

[29] Yiwei Liu, Zhiping Wang, and Baoyou Zheng. Application of regularized gru-lstm model in stock price prediction. In *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, pages 1886–1890, 2019.

[30] Yujie Liu, Hongbin Dong, Xingmei Wang, and Shuang Han. Time series prediction based on temporal convolutional network. In *2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS)*, pages 300–305, 2019.

[31] Zhiyuan Liu and Jie Zhou. *Introduction to graph neural networks*, volume 14. Morgan & Claypool Publishers, 2020.

[32] Ioannis E Livieris, Emmanuel Pintelas, and Panagiotis Pintelas. A cnn–lstm model for gold price time-series forecasting. *Neural computing and applications*, 32:17351–17360, 2020.

[33] Wenjie Lu, Jiazheng Li, Yifan Li, Aijun Sun, and Jingyang Wang. A cnn-lstm-based model to forecast stock prices. *Complexity*, 2020:1–10, 2020.

[34] Thabang Mathonsi and Terence L. van Zyl. A statistics and deep learning hybrid method for multivariate time series forecasting and mortality modeling. *Forecasting*, 4(1):1–25, 2022.

[35] Terence C Mills. *Applied time series analysis: A practical guide to modeling and forecasting*. Academic press, 2019.

[36] D.C. Montgomery, C.L. Jennings, and M. Kulahci. *Introduction to Time Series Analysis and Forecasting*. Wiley Series in Probability and Statistics. Wiley, 2015.

[37] Abdullahi Uwaisu Muhammad, Adamu Sani Yahaya, Suhail Muhammad Kamal, Jibril Muhammad Adam, Wada Idris Muhammad, and Abubakar Elsafi. A hybrid deep stacked lstm and gru for water price prediction. In *2020 2nd International Conference on Computer and Information Sciences (ICCIS)*, pages 1–6, 2020.

[38] Ryan O'Connor. Pytorch vs tensorflow in 2023. https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/. Accessed on Jun 25, 2023.

[39] Ping-Feng Pai, Kuo-Ping Lin, Chi-Shen Lin, and Ping-Teng Chang. Time series forecasting by a seasonal support vector regression model. *Expert Systems with Applications*, 2010.

[40] Wilfredo Palma. *Time series analysis*. John Wiley & Sons, 2016.

[41] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2013.

[42] Josh Patterson and Adam Gibson. *Deep learning: A practitioner's approach*. " O'Reilly Media, Inc.", 2017.

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

[44] Daniel Pena, George C Tiao, and Ruey S Tsay. *A course in time series analysis*. John Wiley & Sons, 2011.

[45] Fei Qian and Xianfu Chen. Stock prediction based on lstm under different stability. 2019.

[46] Luana Ruiz, Fernando Gama, and Alejandro Ribeiro. Gated graph recurrent neural networks. *IEEE Transactions on Signal Processing*, 68:6303–6318, 2020.

[47] Nicholas I Sapankevych and Ravi Sankar. Time series prediction using support vector machines: a survey. *IEEE computational intelligence magazine*, 4(2):24–38, 2009.

[48] Yuslena Sari, Yudi Firmanul Arifin, Novitasari Novitasari, and Mohammad Reza Faisal. Deep learning approach using the gru-lstm hybrid model for air temperature prediction on daily basis. *International Journal of Intelligent Systems and Applications in Engineering*, 10(3):430–436, Oct. 2022.

[49] Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied soft computing*, 2020.

[50] Jimeng Shi, Mahek Jain, and Giri Narasimhan. Time series forecasting (tsf) using various deep learning models. *arXiv preprint arXiv:2204.11115*, 2022.

[51] Slawek Smyl. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, 36(1):75–85, 2020. M4 Competition.

[52] Slawek Smyl, Grzegorz Dudek, and Paweł Pełka. Es-drnn: A hybrid exponential smoothing and dilated recurrent neural network model for short-term load forecasting, 2021.

[53] Ying Sun, Zijun Zhao, Xiaobin Ma, and Zhihui Du. Short-timescale gravitational microlensing events prediction with arima-lstm and arima-gru hybrid model. In *Big Scientific Data Management: First International Conference, BigSDM 2018, Beijing, China, November 30–December 1, 2018, Revised Selected Papers 1*, pages 224–238. Springer, 2019.

[54] Souhaib Ben Taieb and Rob J Hyndman. Recursive and direct multi-step forecasting: the best of both worlds. 11 2022.

[55] Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[56] R.S. Tsay. *Multivariate Time Series Analysis: With R and Financial Applications*. Wiley Series in Probability and Statistics. Wiley, 2013.

[57] R.S. Tsay and R. Chen. *Nonlinear Time Series Analysis*. Wiley Series in Probability and Statistics. Wiley, 2018.

[58] Ruey S. Tsay. *Analysis of financial time series*. Wiley series in probability and statistics. Wiley-Interscience, 2010.

[59] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016.

[60] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[61] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[62] Xing Wan. Influence of feature scaling on convergence of gradient iterative algorithm. *Journal of Physics: Conference Series*, 1213(3):032021, jun 2019.

[63] Neo Wu, Bradley Green, Xue Ben, and Shawn O'Banion. Deep transformer models for time series forecasting: The influenza prevalence case. *arXiv preprint arXiv:2001.08317*, 2020.

[64] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, jan 2021.

[65] Dehe Xu, Qi Zhang, Yan Ding, and De Zhang. Application of a hybrid arima-lstm model based on the spei for drought forecasting. *Environmental Science and Pollution Research*, 29(3):4128–4144, 2022.

[66] Robert A Yaffee and Monnie McGee. *An introduction to time series analysis and forecasting: with applications of SAS® and SPSS®*. Elsevier, 2000.

[67] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 2019.

[68] George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. A transformer-based framework for multivariate time series representation learning, 2020.

[69] Fan Zhang and Lauren J O'Donnell. Support vector regression. In *Machine learning*, pages 123–140. Elsevier, 2020.

[70] Tong Zhang. An introduction to support vector machines and other kernel-based learning methods. *Ai Magazine*, 2001.

[71] Jingyu Zhao, Feiqing Huang, Jia Lv, Yanjie Duan, Zhen Qin, Guodong Li, and Guangjian Tian. Do rnn and lstm have long memory?, 2020.

[72] Yang Zhao and Zhonglu Chen. Forecasting stock price movement: New evidence from a novel hybrid deep learning model. *Journal of Asian Business and Economic Studies*, 29(2):91–104, 2022.

[73] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting, 2021.

[74] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications, 2021.