ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

PREDICTIVE ERROR COMPENSATED WAVELET NEURAL NETWORKS FRAMEWORK FOR TIME SERIES PREDICTION

M.Sc. THESIS

Serkan MACİT

Department of Computer Engineering

Computer Engineering Programme

JULY 2024

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

PREDICTIVE ERROR COMPENSATED WAVELET NEURAL NETWORKS FRAMEWORK FOR TIME SERIES PREDICTION

M.Sc. THESIS

Serkan MACİT
(504221532)

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Prof. Dr. Burak Berk ÜSTÜNDAĞ

JULY 2024

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

ZAMAN SERİSİ TAHMİNİ İÇİN HATA TAZMİNLİ DALGACIK DÖNÜŞÜMLÜ SİNİR AĞLARI ÇERÇEVE YAZILIMI

YÜKSEK LİSANS TEZİ

Serkan MACİT
(504221532)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Burak Berk ÜSTÜNDAĞ

TEMMUZ 2024

Serkan MACİT, a M.Sc. student of ITU Graduate School student ID 504221532 successfully defended the thesis entitled "PREDICTIVE ERROR COMPENSATED WAVELET NEURAL NETWORKS FRAMEWORK FOR TIME SERIES PREDICTION", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Burak Berk ÜSTÜNDAĞ** Istanbul Technical University	.....
Jury Members :	**Dr. Yusuf Hüseyin Şahin** Istanbul Technical University	.....
	Associate Prof. Dr. Taner Arsan Kadir Has University	.....

Date of Submission : **1 May 2024**
Date of Defense : **22 July 2024**

To my family,

FOREWORD

I would like to thank my family, whose encouragement and support have never stopped. I would also like to express my gratitude to my advisor for all of his guidance and assistance with my studies. The creation of this work has been substantially impacted by his thoughts. I thank Mustafa Abdullah Hakkoz, Ajla Kulaglic, Halil Özgür for their contributions to the implementation and experimental aspects of the developed algorithm, and Alexandra Denisenko for her assistance with the presentation of drawings and data, which enriched the thesis. Lastly, I would like to thank all my fellow students working in our laboratory for assisting me in completing tasks faster.

I am appreciative of each person's contributions, as they have all been instrumental in bringing my thesis to a successful finish.

July 2024

Serkan MACİT
Computer Scientist

TABLE OF CONTENTS

ABBREVIATIONS

ML	**:** Machine Learning
PECNET	**:** Predictive Error Compensated Wavelet Neural Networks
GPU	**:** Graphics Processing Unit
CPU	**:** Central Processing Unit
MLP	**:** Multi-Layer Perceptron
NN	**:** Neural Network
CNN	**:** Convolutional Neural Network
RNN	**:** Recurrent Neural Network
LSTM	**:** Long Short-Term Memory
ARIMA	**:** Auto-Regressive Integrated Moving Average
DWT	**:** Discrete Wavelet Transform
DN	**:** Denormalization
EF(D)	**:** Electric Field (Data)
ER	**:** (Seismic) Energy Release
MSE	**:** Mean Squared Error
RMSE	**:** Root Mean Squared Error
GELU	**:** Gaussian Error Linear Unit

SYMBOLS

$\mathbf{T}$	: Period
$\mathbf{t}$	: Timestamp
$\mathbf{X_{i=1,2,...,n}}$	: Multivariate Input Data
$\mathbf{y,\ \hat{y}}$	: Target, Predicted Data
$\mathbf{e,\ \hat{e}}$	: Error, Predicted Error
$\mathbf{N}$	: Normalization
$\mathbf{W}$	: Wavelet Transform
$\mathbf{J}$	: Joule
$\mathbf{M}$	: Magnitude
$\mathbf{E}$	: Energy
$\mathbf{R^2}$	: Coefficient of Determination
$\mathbf{r_{Xy}}$	: Pearson Correlation Coefficient

LIST OF TABLES

Page

LIST OF FIGURES

PREDICTIVE ERROR COMPENSATED WAVELET NEURAL NETWORKS FRAMEWORK FOR TIME SERIES PREDICTION

SUMMARY

Machine learning algorithms have gotten considerable attention and recognition in the context of addressing time series prediction problems; however, the task of constructing an accurate machine learning model with optimal architecture and hyperparameters becomes highly challenging if the data is non-linear, encompasses multi variable characteristics with chaotic or stochastic properties, and has sensitivity to environmental factors. In this context, common issues encountered in time series prediction models and frameworks include overfitting, a machine learning problem arising in situations with limited labeled data but a large variety of input data; generalization issues due to insufficient or inadequate input data; the need for extensive feature engineering to properly set internal weights in artificial neural networks; dependency on network parameters in developed solutions and limited adaptability to different problems; and being computationally expensive and time-consuming.

Predictive Error Compensated Wavelet Neural Networks (PECNET) is an innovative artificial neural network architecture for time series predictions. It avoids overfitting by training the data separately in cascaded networks based on different frequency bands and using the remaining errors of each subsequent network as target data. In the PECNET architecture, data is fed into the networks with a low-frequency band in a wide time window, and the subsequent network is trained with narrower time windows and high-frequency data while using the error from the previous low-frequency network as the target data. This method improves the orthogonality of data features across time windows and lets you choose orthogonal features in data fusion applications. It also makes predictions more accurate as more networks are added, which lowers the risk of overfitting. Additionally, by applying wavelet transform as a feature extraction method to the various frequency components of each network , it is possible to distinguish and extract the variety of patterns present in the data. PECNET also overcomes the traditional normalization problems for non-stationary time series data by using adaptive normalization techniques. In conclusion, PECNET is a very good alternative for solving time series prediction problems due to its high prediction accuracy without overfitting, unique structure that allows adaptation to different problems independently of network parameters, and being computationally cheap and low time-consuming just with two-layer MLP structure.

The PECNET model, due to its composition of cascade networks, modular feature extractions, and fusion networks, presents challenges in its implementation at the coding level. PECNET contains many sequential cascaded neural networks that are

trained by each other's errors. When training the sequential networks with input data sequences, it is necessary to shift the input values to the previous time window. Otherwise, it would result in using data from the future, which is not feasible in practice. The continuous use of error data sequences as both label and input data in training and the use of the final error in the fusion network for predicting the remaining error value of the time series as both input and target data increase computational complexity for solutions involving numerous data sources and networks. This can lead to execution errors in time synchronization management. To overcome these challenges, PECNET framework software has been developed within the scope of this thesis, ensuring optimal use of memory and processor resources along with the modular design.

In the development of the PECNET framework software, the C compiler and Python interpreter were utilized. NumPy, Pandas, PyWavelets, and Matplotlib libraries were used for data processing tasks. The PyTorch library was chosen for constructing the artificial neural network model due to its extensive modification features and options for interacting with the graphics processing units (GPUs). The design adhered strictly to object-oriented programming principles, and a syntax similar to Keras was used. Additionally, the machine learning application cycle followed the sklearn flow (fit-predict-eval).

The PECNET framework is made up of various modules that work together. In the "models" module, the BasicNN class forms the core of the neural network architecture. This class manages key tasks like initializing the network, fitting data, computing loss, and adjusting the model during training. In the "network" module, specialized classes like ErrorNetwork, FinalNetwork, and VariableNetwork handle specific stages of the prediction process. ErrorNetwork focuses on correcting prediction errors; FinalNetwork integrates predictions from previous networks for a final output; and VariableNetwork manages training input data across different frequencies and integrates data fusion mechanism for multivariate data. In the same module, PECNET's functionality is encapsulated in the Pecnet class, which coordinates the workflow among different networks. It manages the data flow between cascaded networks, error compensation, and final prediction generation. The PecnetBuilder class provides a fluent interface to construct the Pecnet object. It sequentially adds various network components, ensuring a streamlined building process for the PECNET model. In the "preprocessing" module, the DataPreprocessor class plays a crucial role in data preparation. It lets users change how data is processed into different frequency bands, sampling periods, and sequence sizes that are appropriate for each of cascaded networks in the model. It also does scaling, adaptive normalization, denormalization, and wavelet transform with the help of other classes in the same module. This makes sure that the input data is in the best possible shape for the prediction pipeline. In the "utils" module, the Utility class facilitates hyperparameter optimization, offers tools for loading datasets and plotting results. Overall, PECNET's code-level functionality revolves around these classes, each contributing to the framework's ability to process and predict time series data efficiently.

PECNET has already demonstrated successful outcomes on various datasets as a discrete code, showing promising results against existing machine learning models such as ARIMA, MLP, CNN, LSTM. In this study, as a framework form, it is initially tested on a historical time series dataset of daily adjusted close prices of Apple stocks and then compared with LSTM, which is known for its strong memory and sequence comprehension capabilities. As a result, in terms of the RMSE metric, the LSTM model has had an error of $2.55, while PECNET has had an error of $1.24. In terms of the R2 metric, the LSTM model has achieved a value of 0.94, whereas PECNET has reached 0.98. Following that, it is comparatively tested with LSTM for seismic energy estimation on real-time chaotic Electric Field Data (EFD) which is collected within the scope of earthquake prediction research project conducted at Istanbul Technical University (ITU). In this experiment, in terms of the RMSE metric, the LSTM model has showed errors ranging between 300J-400J, while PECNET has showed errors ranging between 130J-150J. In terms of the R2 metric, the LSTM results has fluctuated between 0.2-0.3, while PECNET has achieved values between 0.5-0.6. In both scenarios, PECNET outperforms LSTM, and the developed framework is being integrated into the portal software for real-time earthquake prediction.

In conclusion, the developed modular and customizable framework facilitates the use of PECNET, which is highly performant and robust against overfitting, for various types of time series predictions by other developers in real-time machine learning systems without requiring specific coding knowledge of PECNET.

ZAMAN SERİSİ TAHMİNİ İÇİN HATA TAZMİNLİ DALGACIK DÖNÜŞÜMLÜ SİNİR AĞLARI ÇERÇEVE YAZILIMI

ÖZET

Makine öğrenmesi algoritmaları ile, zaman serisi tahmin problemlerinin çözümünde önemli bir yol katedilmiştir; ancak, veriler doğrusal olmayan yapıda, kaotik veya stokastik özellikler barındıran çok değişkenli karakterde ve çevresel faktörlere hassas olduğunda, optimal mimari ve hiperparametreler ile doğru bir makine öğrenimi modeli oluşturmak oldukça zorlaşmaktadır. Bu bağlamda, zaman serisi tahmin modelleri ve çerçeve yazılımlarında karşılaşılan yaygın sorunlar arasında; yeterli özellikte girdi verisi fakat sınırlı etiketli veriye sahip durumlarda ortaya çıkan aşırı uyum (overfitting), yetersiz veya uygun olmayan girdi verileri nedeniyle oluşan genelleştirme (underfitting) problemi, yapay sinir ağlarında iç ağırlıkların doğru bir şekilde ayarlanabilmesi için gereken kapsamlı veri özellikleri çıkarımı ihtiyacı; geliştirilen çözümlerde ağ parametrelerine bağımlılık ve farklı problemlere sınırlı adaptasyon; ve hesaplama açısından pahalı ve aşırı zaman maliyeti üretme sayılabilir.

Hata Tazminli Dalgacık Dönüşümlü Sinir Ağları (PECNET), zaman serisi tahminleri için yenilikçi bir yapay sinir ağı mimarisidir. PECNET, verileri farklı frekans bantlarına göre ayrı ayrı kaskad ağlarda eğiterek ve üst ağın kalan hatalarını alt ağda hedef veri olarak kullanarak aşırı uyuma engel olur. PECNET mimarisinde, veriler düşük frekans bandında geniş bir zaman penceresindeki ağla eğitime başlar ve sonraki ağ daha dar zaman pencereleri ve yüksek frekanslı verilerle, önceki ağın hatasını hedef veri olarak alıp eğitilir. Bu yöntem, zaman pencereleri boyunca veri özelliklerinin dikliğini artırır ve veri füzyonu uygulamalarında dik özellikleri seçme imkanı sunar. Ayrıca, daha fazla kaskad ağ eklendikçe tahminlerin başarımını artırarak aşırı uyum riskini azaltır. PECNET'te her ağın çeşitli frekans düzeylerindeki girdi verisine özellik çıkarma yöntemi olarak dalgacık dönüşümü uygulanır, bu da verilerdeki çeşitli desenleri ayırt etmeyi ve çıkarmayı mümkün kılar. PECNET ayrıca, adaptif normalizasyon tekniklerini kullanarak durağan olmayan zaman serisi verilerinde ortaya çıkan geleneksel normalizasyon problemlerini çözebilir. Sonuç olarak, PECNET overfit olmadan sağladığı yüksek tahmin doğruluğu, ağ parametrelerinden bağımsız olarak farklı problemlere uyum sağlama yeteneği ve hesaplama açısından ucuz ve az zaman maliyetli olması nedeniyle (sadece iki katmanlı bir yapay sinir ağı kullanır) zaman serisi tahmin problemlerini çözmek için güçlü bir alternatiftir.

PECNET modeli, kademeli ağlardan oluşması, modüler özellik çıkarımları ve füzyon mekanizması içermesi nedeniyle, kodlama seviyesinde uygulama zorlukları barındırmaktadır. PECNET birbirlerinin hatalarıyla eğitilen birçok ardışık kaskad sinir ağı içermektedir. Ardışık ağlar kendi girdi veri dizileriyle eğitilirken, hedef veri olarak kullanılacak hata verisi henüz oluşmadığından girdi verilerini önceki

zaman penceresine kaydırmak gereklidir. Aksi takdirde, pratikte mümkün olmayan gelecekteki verilerin kullanılması söz konusu olur. Hata veri dizilerinin hem etiket hem de girdi verisi olarak eğitimde sürekli kullanılması ve son hata verisinin zaman serisinin kalan hata değerini tahmin etmek için hata füzyon ağında hem girdi hem hedef veri olarak kullanılması çok sayıda veri kaynağı ve ağ içeren çözümlerde hesaplama karmaşıklığını artırır. Bu durum zaman senkronizasyonu yönetiminde yürütme hatalarına yol açabilir. Bu zorlukların üstesinden gelmek için, bu tez kapsamında PECNET için bir çerçeve yazılımı geliştirilmiştir ve modüler tasarımla birlikte bellek ve işlemci kaynaklarının optimal kullanımı sağlanmıştır.

PECNET çerçeve yazılımının geliştirilmesinde C derleyicisi ve Python yorumlayıcısından faydalanılmıştır. Verinin işlenmesi için NumPy, Pandas, PyWavelets ve Matplotlib kütüphaneleri kullanılmıştır. PyTorch kütüphanesi, geniş ayarlanabilir özellikleri ve grafik işleme birimleriyle (GPUs) etkileşim seçenekleri nedeniyle yapay sinir ağı modelinin oluşturulması için seçilmiştir. Tasarımda nesne yönelimli programlama ilkelerine sadık kalınmıştır ve Keras'a benzer bir sözdizimi kullanılmıştır. Ayrıca, makine öğrenimi uygulama döngüsü için sklearn akışı (fit-predict-eval) takip edilmiştir.

PECNET çerçevesi yazılımı, birlikte çalışan çeşitli modüllerden oluşmaktadır. "models" modülünde yer alan BasicNN sınıfı sinir ağı mimarisinin temelini oluşturur. Bu sınıf, ağın ilk değerlerini yükleme, verileri eğitime uyarlama, kayıp fonksiyonunu optimize etme ve eğitim sırasında modeli ayarlama gibi temel görevleri yönetir. "network" modülünde bulunan özelleştirilmiş ErrorNetwork, FinalNetwork ve VariableNetwork sınıfları tahmin sürecinin belirli aşamalarını ele alırlar. ErrorNetwork tahmin hatalarını düzeltmeye odaklanır; FinalNetwork önceki ağlardan gelen tahminleri nihai tahmin için birleştirir; VariableNetwork farklı frekanslardaki eğitim giriş verileri için kaskad ağları yönetir ve çok değişkenli veriler için veri füzyonu mekanizmasını entegre eder. Aynı modüldeki Pecnet sınıfı, bu sınıflar arasındaki iş akışını koordine eder. Bütün kaskad ağlar arasındaki genel veri aktarımını, hata tazminini ve nihai tahmin üretimini yönetir. PecnetBuilder sınıfı ise Pecnet nesnesini oluşturmak için kullanıcıya kolay bir arayüz sunar, çeşitli kaskad ağ bileşenlerini ardışıl ekler ve farklı farklı PECNET modellerinin oluşturulması için sorunsuz bir yapılandırma süreci sağlar. "preprocessing" modülündeki DataPreprocessor sınıfı verinin işlenmasi ve hazırlanmasında kritik bir rol oynar. Kullanıcıların, modeldeki tüm kaskad ağlar için uygun olan verileri farklı frekans bantları, örnekleme periyotları ve sekans boyutlarına göre oluşturmasına olanak tanır. Aynı modüldeki diğer sınıfların yardımıyla ölçeklendirme, adaptif normalizasyon, denormalizasyon ve farklı tiplerde dalgacık dönüşümü işlevleri sunar. Bu işlemler giriş verilerinin tahmin süreci için mümkün olan en iyi durumda bulunmasını sağlar. "utils" modülündeki Utility sınıfı hiperparametre optimizasyonunu kolaylaştırır, veri setlerini yükleme ve çizim için araçlar sunar. Genel olarak, PECNET'in kod seviyesindeki işlevselliği, bu sınıflar etrafında döner, her biri çerçeve yazılımının zaman serisi verilerini işleme ve tahmin etme yeteneğine katkıda bulunur.

PECNET daha önce ayrık kod şeklinde çeşitli veri setleri üzerinde ARIMA, MLP, CNN, LSTM gibi mevcut makine öğrenmesi modellerine karşı başarılı sonuçlar göstererek etkili olduğunu kanıtlamıştır. Bu çalışmada ise çerçeve yazılımı formunda

ilk olarak Apple hisse senetlerinin günlük düzeltilmiş kapanış fiyatlarından oluşan tarihsel bir zaman serisi veri setinde denenmiş ve ardından güçlü hafıza ve sekans anlama yetenekleriyle bilinen LSTM ile karşılaştırılmıştır. Sonuç olarak, RMSE metriği cinsinden LSTM 2.55$, PECNET 1.24$ yanılmıştır. R2 metriği cinsinden ise LSTM 0.94, PECNET 0.98 değerlerine ulaşmıştır. Ardından, İstanbul Teknik Üniversitesi (İTÜ) bünyesinde yürütülen deprem tahmini araştırma projesi kapsamında toplanan gerçek zamanlı kaotik Elektrik Alan Verisi (EFD) üzerinde sismik enerji tahminlemesi için yine LSTM ile karşılaştırmalı test edilmiştir. Bu deneyde, RMSE metriği cinsinden, LSTM 300J-400J arası, PECNET 130J-150J arasında yanılmalar göstermiştir. R2 cinsinden ise LSTM sonuçları 0.2-0.3 değerleri arasında seyrederken, PECNET 0.5-0.6 değerlerine ulaşmıştır. Her iki senaryoda da PECNET, LSTM'den daha başarılı performans göstermiştir ve geliştirilen çerçeve yazılımı, gerçek zamanlı deprem tahmini için oluşturulan portal yazılımına entegre edilmektedir.

Sonuç olarak, geliştirilen modüler ve özelleştirilebilir çerçeve yazılımı, yüksek performanslı ve aşırı uyuma karşı dayanıklı olan PECNET'in, PECNET'e özgü bir kodlama bilgisi gerektirmeden diğer geliştiriciler tarafından gerçek zamanlı makine öğrenimi sistemlerinde çeşitli zaman serisi tahminleri için kullanımını mümkün kılmıştır.

1. INTRODUCTION

Time series prediction has become increasingly important in diverse fields, ranging from financial forecasting to natural disaster management, in recent years [1]. The inherent complexity of the time series data, characterised by non-linear, chaotic and stochastic behaviours, poses substantial challenges to traditional machine learning techniques. The advancement of deep learning and neural network architectures has offered new opportunities for addressing these challenges, yet the development of a robust, adaptable, and efficient framework remains a pressing need in the field.

The Predictive Error Compensated Wavelet Neural Networks (PECNET) is a high-performance deep learning model that addresses issues commonly faced by traditional machine learning models, such as overfitting, low generalization ability, excessive dependence on network parameters, the need for extensive feature engineering, and high computational costs. PECNET consists of neural networks in a cascade structure that learn from each other's errors and minimize errors downward. The data is adaptively normalized at each network and subjected to discrete wavelet transformation. Additionally, vertical features that enhance the prediction performance of multivariate data are incorporated into the training process through data fusion [2].

The framework developed for the PECNET model presents all components of the model functionally under specific software packages, based on the fundamental machine learning process of preprocess-fit-predict-evaluate. The operation of each component of the PECNET model is explained both theoretically and practically, with visuals illustrating the learning process, and the use of relevant functions in the framework is demonstrated through example scenarios.

1.1 Problem & Hypothesis

Predictive Error Compensated Wavelet Neural Networks (PECNET), an innovative neural network architecture, can significantly improve the accuracy and generalizabil-

ity of time series predictions but it has operational and structural challenges inherent in its deployment for practical applications [2].

It's hypothesized that all challenges of PECNET can be effectively mitigated through a dedicated framework, encapsulating the complexities of the model and transforming it into an accessible tool for practitioners who have just fundamental programming knowledge. This transformation aims to address the current limitations in time series prediction models and frameworks in a way reducing computational power and time costs while maintaining high performance.

1.2 Purpose of Thesis

The purpose of this thesis is to develop a framework for PECNET and detail the design, testing, and real-time application of developed framework software, making this sophisticated model accessible for a wider range of time series prediction applications. The framework is designed to provide a comprehensive exploration of the PECNET model within a practical, user-friendly programming environment, demonstrating its superiority in addressing the complex challenges of time series prediction and contributing significantly to the advancement of machine learning applications in natural phenomenon prediction and beyond.

2. TIME SERIES PREDICTION

Time series data represents a sequence of values or observations obtained over successive periods of time. This type of data is characterised by being recorded at regular intervals and is arranged chronologically. Time series data is ubiquitous, found in many fields including economics, meteorology, social sciences, and natural phenomena. It is critical in forecasting stock market trends, weather patterns, consumer behaviour, and many other domain-specific applications.

The analysis and prediction of time series data involves understanding underlying patterns, forecasting future values, and discerning trends and cyclic behaviours.

2.1 Characteristics and Complexity of Time Series Data

Time series data, particularly from natural systems, is inherently complex due to several characteristics:

- **Non-linearity:** It often does not follow a straight-line pattern, making linear models insufficient for accurate predictions.
- **Chaotic nature:** It has the potential to be unpredictable, with its future dynamics strongly reliant on initial conditions. Small changes in initial values might result in radically different outcomes, making prediction difficult and emphasizing the significance of precisely capturing these initial states.
- **Stochastic components:** Randomness or noise is a common trait, complicating the task of extracting underlying patterns.
- **Sensitivity to environmental factors:** Time series data is frequently affected by external environmental factors, which can cause significant variations in the data, making it challenging to predict future trends accurately.

- **Multivariate complexity:** Time series data often contains multiple variables or dimensions, which significantly increases the complexity of the analysis. This multivariate aspect requires models to integrate and interpret interrelated data points from various sources or features, thereby complicating the prediction process.

2.2 Common Difficulties in Time Series Prediction

Considering the aforementioned characteristics of time series data, the challenges encountered in prediction models and frameworks can be outlined as follows:

- **Overfitting in the presence of non-linearity and chaos:** Overfitting refers to a situation where a model learns the specific details and noise present in the training data to such an extent that it impairs its ability to generalize effectively to new data. As the quantity of features to be learned increases relative to the amount of input data, or when the available labeled data is limited compared to the inputs, overfitting becomes inevitable, especially if the data characteristics are non-linear and chaotic [3].

- **Lack of local context and generalization ability:** This problem draws attention to the model's limitations in capturing significant information in specific parts or local features of the data set, as well as its inability to adjust to other data sets or environmental factors. It has to do with the model's incapacity of providing a general solution across several data sets along with its failure to understand the context of individual data points. When the model is introduced to fresh, unexplored data, this limitation often results in poor performance, indicating a lack of adaptability and understanding of various data properties [4].

- **Need for extensive feature engineering:** Identifying and engineering relevant features from raw data is both labor-intensive and requires substantial domain knowledge.

- **Dependency on network parameters and hard tuning:** Models are usually calibrated for specific problems and they often perform worse when applied to new problems with the same settings. This means that parameters must be constantly

modified for each new problem, which is a difficult and never-ending effort. Finding the optimal parameters for the best model performance in a variety of applications is still a difficult and time-consuming task.

- **Sensitivity to estimation errors:** The use of predicted values in training phase to estimate parameters of the model can lead to the accumulation of errors and small errors can cause significant inaccuracies in predictions.

- **High computational power and time cost:** In order to achieve high prediction accuracy, models often require substantial computational resources, both in terms of processing power and memory, and significant time costs during training and evaluation phases. This requirement increases with the complexity of the data set and the intricacy of the model, which raises difficulties for practical, real-time applications where quick and effective processing is essential.

2.3 Comparative Analysis of Prominent Models & Frameworks

In the context of the characteristics and complexities of time series data, as discussed in the first section, and considering the general challenges faced by time series prediction models outlined in the second section, a comparative analysis of prominent models and frameworks used in time series prediction can be summarized in Table 2.1 [5]–[7]. This table juxtaposes their strengths and weaknesses, illustrating how each model or framework aligns with or deviates from addressing the specific needs of time series prediction.

Table 2.1 : Comparative Analysis of Prominent Solutions.

Solution	Strengths(+)/Weaknesses(-)
ARIMA	**+** Effective for linear relationships, computationally cheap **-** Struggles with non-linearity, sensitive to estimation errors
LSTM	**+** Strong memory capability, good at handling sequences **-** Complex and parameter-dependent, prone to over-fitting
CNN	**+** Efficient in spatial data, automated feature engineering **-** Lacks temporal sequence modeling, extensive tuning
Prophet	**+** Good at seasonality modeling, robust to outliers **-** Limited customization, over-simplification of data
TimeGPT	**+** Effective with big data sets, adaptable to diverse domains **-** High data requirement, high computational power

The need for a robust, adaptable, and efficient framework in time series analysis becomes evident when considering these challenges and the limitations of current solutions.

In the following section, the Predictive Error Compensated Wavelet Neural Networks, abbreviated as PECNET, which offer solutions to the previously discussed time series prediction problems, will be introduced. Its effectiveness has been proved in experimental studies, where it outperformed ARIMA, MLP, CNN, and LSTM in a variety of time series data sets. Then, the reasons for transforming PECNET into a framework will be explained, followed by a detailed description of the structure and usage of the developed framework. Subsequently, the PECNET framework will be comparatively tested against LSTM, which has shown high performance in previous tests, on two different time series datasets.

3. PREDICTIVE ERROR COMPENSATED WAVELET NEURAL NETWORKS (PECNET)

PECNET is an innovative, highly efficient artificial neural network architecture for time series prediction, which achieves high generalisation without overfitting and does not require high computational resources and data [2,8].

PECNET divides data into different frequency bands and uses wavelet coefficients of each band to establish a cascaded neural network architecture that progresses from the widest to the narrowest temporal window. Successive networks are trained using the remaining errors of the previous ones, aiming to refine predictions and learn data characteristics ranging from general trends to finer details. A final neural network combines predictions from each network and error predictions generated by a separate error network to produce an improved prediction [9]–[11].

The subsections that follow will go into more depth about PECNET's internal structure, its components, and its general features.

3.1 Core Concepts & Architecture

PECNET is conceptually founded on five fundamental approaches. These five approaches, when diversified internally, enable the creation of various types of PECNET architectures.

The first approach of these is the application of time series data at different time windows with different sub-sampling frequencies simultaneously to cascaded neural network segments. PECNET networks receive data in low-frequency bands with wide time windows, and the next network is trained with data at higher frequencies and narrower time windows using the error from the previous low-frequency network. This approach enhances the orthogonality of data characteristics across time windows, and as additional networks are added, the likelihood of overfitting reduces while prediction performance improves [9]–[11].

The second approach is that after the first network, training is conducted not with the target data but with the error coming from the upper network, and in the last sub-network, the compensated errors are corrected and prepared for a final fusion network. In this final network, training is carried out using the target data predictions from the first network along with the error predictions from all sub-networks. Thus, each sub-network performs learning at different levels, gradually improving the overall prediction performance and prevents the model from merely memorising the samples from the data. Additionally, as the orthogonality in the input data characteristics for the final fusion network increases, prediction performance is enhanced [9]–[11].

The third approach involves performing data fusion in multivariate time series by connecting a second variable, which is most highly correlated with the error of the first variable's last network, to provide a comprehensive perspective on the target data. This method and the use of data fusion in subsequent networks at the right time and place can enhance the analysis of complex systems, thereby improving prediction performance [9,11].

The fourth approach is the use of adaptive normalisation to dynamically adapt to changes in the data set and preserve the relationships between features. This improves the efficiency of the neural network architecture while maintaining the properties of the time series data [9].

Lastly, it involves applying wavelet transformation to the input and using wavelet coefficients instead of the data itself during training. This representation enhances the orthogonality among the features, thereby improving the network performance [2].

An ideal PECNET architecture that is established for a multivariate time series based on these concepts could be shown as in the Figure 3.1. Here, each grey rectangle represents a network termed as a *cascaded neural network*, each of which is in charge of training the data based on a distinct frequency range. Cascaded neural networks belonging to a single data type collectively form what is known as a *variable network*. The last cascaded neural network is specifically named the *error network*, which is trained with the error of errors. All cascaded neural networks are connected to a *final network*, forming the *prediction pipeline*.

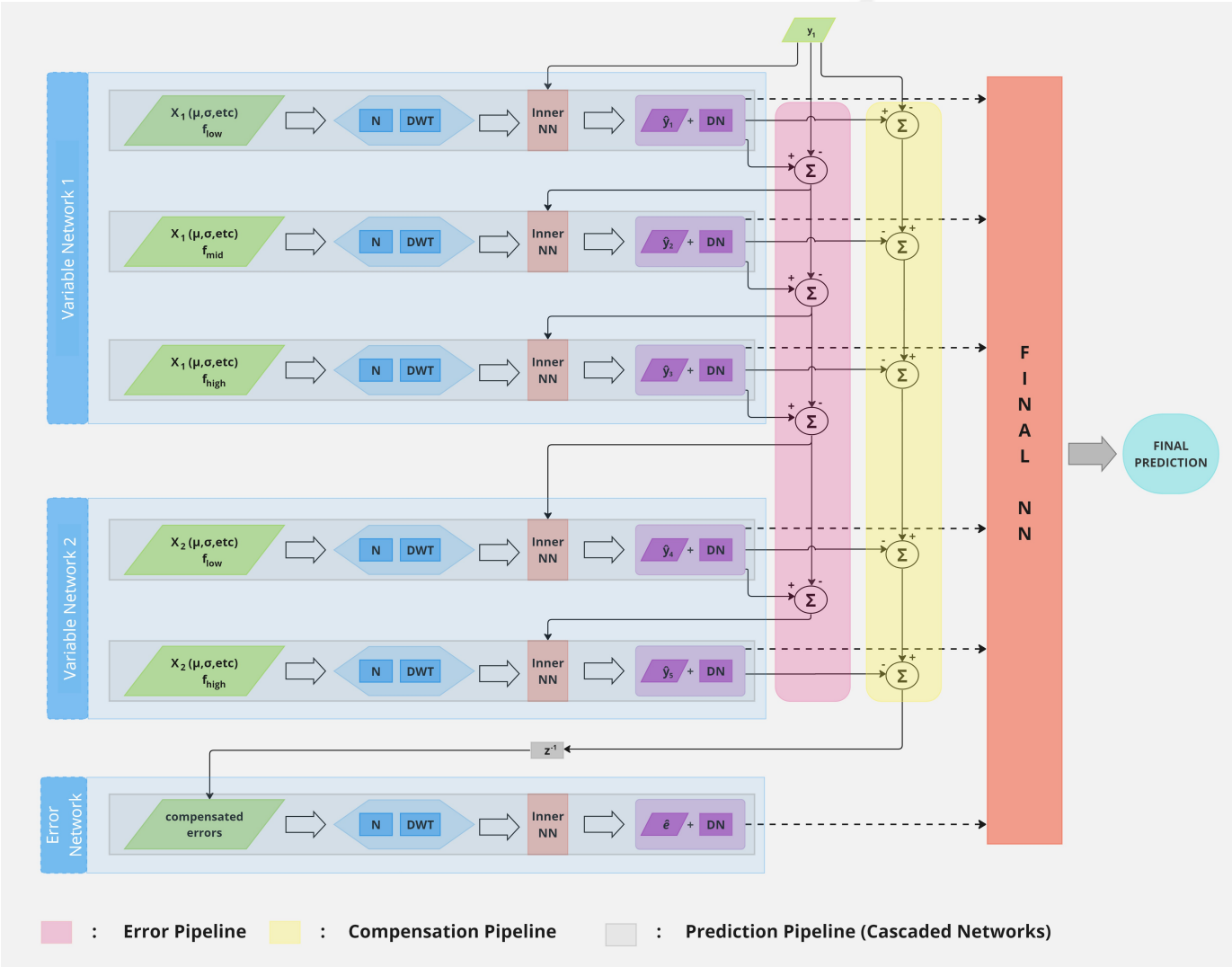

Figure 3.1 : Core Concepts and Architecture of PECNET.

The module responsible for calculating and transferring the errors that feed the error network is referred to as the *compensation pipeline*. Error values in this pipeline are shifted one timestamp back using the *unit delay operator* (z^{-1}) before being transferred to the error network to prevent the use of future data. The adjacent magenta vertical module is termed as *the error pipeline*, as it generates error target data for cascaded networks.

In the following subsections, the main components and functionality of this architecture will be examined.

3.1.1 Data preprocessing and transformation

This core section involves selecting a statistical method for sampling and windowing the data to create sequences, followed by normalization, generation of wavelet coefficients, and splitting the data for training and testing purposes for all PECNET networks. This process ensures that the data is appropriately structured and optimized for the subsequent stages of the predictive modeling pipeline. If we go through these steps one by one:

- **Sequence generation:** The data, specifically for each cascaded network, is first windowed according to the size determined by the product of the sampling period (T) and the chosen sequence length (l_s), as in (3.1) and (3.2).

$$s = T \times l_s \tag{3.1}$$

$$F_i = \begin{cases} None & i < s, \\ x_{i-s+1}, x_{i-s+2}, ..., x_i, & i \geq s \end{cases} \tag{3.2}$$

  where s is the window size , F_i is the i-th value of windowed data and x_i is the i-th value of the input. Afterward, for each cascaded neural network, the windowed data sequences created with different periods are sampled using a specified sampling technique (mean, etc.) by grouping window elements into sizes of F_i/T to equalize the sequence sizes. This process ensures that the data sequence to be processed in each cascaded neural network is segmented from wide to narrow time windows, all of which contain an equal number of elements.

- **Normalization:** Normalization is the process of scaling input data to fall within a specified range, typically between 0 and 1. This is crucial because it prevents features with larger scales from dominating the learning process, leading to more accurate and efficient models. By normalizing data, algorithms can converge faster and achieve better performance. On the other hand, classical normalization techniques, like min-max scaling and z-score normalization, can be thrown off by outliers, which can distort the data. Also, these methods often assume a certain data distribution, which isn't always true, leading to less effective scaling. This can affect how well machine learning models perform, especially with varied or unusual data distributions [12].

  PECNET uses adaptive normalization which is a technique that dynamically adjusts to the changing statistical properties of data, especially useful for non-stationary datasets. Unlike classical methods, it continuously updates normalization parameters like mean and variance, overcoming issues with fixed assumptions. This approach ensures more accurate scaling over time, enhancing the performance of machine learning models on varying data [13,14].

  In this step, the widely-used adaptive normalization technique, average subtraction, is applied to the input, as shown in formula (3.3). Prior to this procedure, in order to enhance model performance, the input may be scaled using a user-specified scale factor.

$$\tilde{x}_{ij} = x_{ij} - \mu_i, \quad 1 \leq j \leq l_F, \quad 1 \leq i \leq \frac{n}{l_F} \tag{3.3}$$

  where x_{ij} is the j-th element in the i-th window of sequential input data, μ_i is the average of the i-th window, l_F is the window size, n is the input length and $\tilde{x}_{ij}$ is the normalized value of the x_{ij}. The similar scaling and normalization procedure is also applied to the target data, taking care to avoid peeking.

- **Wavelet transform:** The wavelet transform, which is widely used in signal processing because it provides information in both the time and frequency domains, thus being more capable than traditional Fourier analysis, decomposes time series data into scaled and shifted versions of a base function that is known as the mother

wavelet [15]. Mathematically, this is represented as in (3.4).

$$W(a,b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} f(t) \cdot \psi^* \left(\frac{t-b}{a} \right) dt \tag{3.4}$$

where a is the scale parameter, b is the shift parameter, $f(t)$ is the time series data and ψ^* is the mother wavelet. The scale parameter a stretches or compresses the wavelet, allowing the analysis of the signal at different frequency bands, while the shift parameter b moves the wavelet along the time axis, enabling the examination of the signal at different time intervals.

By applying wavelet transform, we can separate the high-frequency components (details) from the low-frequency components (smooth trends) of a signal. This separation enhances the orthogonality of data features, meaning it makes different components of the data more distinct and independent from each other.

Wavelet transform is often used in its discrete form, as in (3.5) instead of (3.4) because it provides efficient computation and simpler analysis while retaining most of the relevant information.

$$W(j,k) = \frac{1}{\sqrt{M}} \sum_{t=0}^{M-1} f(t) \cdot \psi_{jk}(t) \tag{3.5}$$

where j is the scale level, k is the time level, $f(t)$ is the time series and $\psi_{jk}(t)$ is the discrete wavelet function. The Haar wavelet, as shown in (3.6), is widely used as a discrete wavelet function due to its fast computability and ease of implementation [16].

$$\psi(t) = \begin{cases} 1 & \text{if } 0 \leq t < \frac{1}{2}, \\ -1 & \text{if } \frac{1}{2} \leq t < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

In this step, the Haar wavelet transform (or other wavelets like Symlet, Daubechies, etc., depending on the data) is applied to each normalized sequence. The elements of the sequence are grouped in pairs, and then the sequence is divided into two equal parts, with one part consisting of the averages of each group (the *"approximation coefficients"*) and the other part consisting of half of the differences between the elements within each group (the *"detail coefficients"*) at the first level of decomposition. The approximation coefficients from the previous level are similarly divided into two equal halves at each succeeding level of decomposition,

and this procedure is repeated until the desired level of resolution is reached. At the end, the first wavelet coefficient (approximation coefficient) of each sequence is discarded since it's zeroed due to the average subtraction normalization before the wavelet transform. During the training in the relevant networks, these coefficients are used instead of the original data.

After these stages, the input data for the networks, up to the error network and final network, is processed and ready to be split into training and testing sets.

3.1.2 Cascaded neural network structure

A specific configuration of neural networks, which is called the *cascaded neural networks* inside PECNET and breaks down the prediction process into more manageable sub-problems, is intended to increase the model's predictive accuracy for time series data. In this structure, multiple neural networks are connected sequentially, with each network focusing on different aspects of the data. This sequential arrangement lowers the risk of overfitting and enables PECNET to capture and process various features and patterns at different levels, allowing it to make more accurate predictions by considering both high-level and low-level information in the time series data.

Cascaded neural networks operate collaboratively, with the output of one network being used to calculate an error, as in (3.7), which is then used as the target data for the subsequent network.

$$e_i = \hat{y}_i - y_i \tag{3.7}$$

where e_i is the error of i-th network and will be the target of $(i+1)$-th network, $\hat{y}_i$ is the i-th prediction set , and y_i is the i-th target data. The transmission of error in this manner as the target data begins with the training of the first network using the target data, and the error from this initial network is transferred to the next network. This procedure continues through the subsequent networks, where the error of error is transferred to the next one, culminating before reaching the *error network*. This approach allows for error reduction at each stage, ensuring that the lower-level networks learn from the

errors of their predecessors, and it helps refine the data patterns and features iteratively, ultimately leading to more accurate predictions [9].

3.1.3 Error analysis and compensation

The architecture of cascaded neural networks concludes with an *error network*, where all error predictions are compensated and transferred in a cascading manner through a *compensation pipeline*, synchronously with the cascading networks. This compensation is carried out downwards in a step-by-step manner according to the formulas in (3.8) and (3.9).

$$p_{compensated}(t+1) = \hat{y}_1(t+1) - \sum_{i=2}^{n} \hat{y}_i(t+1) \tag{3.8}$$

$$e_{compensated}(t+1) = p_{compensated}(t+1) - y(t+1) \tag{3.9}$$

where $(t+1)$ is the first timestamp at which predictions begin, following the forecasting process performed with data up to timestamp t. $\hat{y}_1$ is the target prediction and $\hat{y}_i$ is the error prediction of i-th network. $p_{compensated}$ is the *compensated prediction* which is calculated by subtracting the sum of all error predictions from target prediction and $e_{compensated}$ is the *compensated error* which is calculated by subtracting target data from *compensated prediction*.

The previously calculated *compensated error* data is shifted from timestamp $t+1$ to timestamp t to align with the input data in the other cascaded networks before undergoing analysis in the error network. Then, similar preprocessing steps such as sequencing, normalization, and wavelet transform are applied to the *compensated error* data in the same manner as in the other cascaded networks to prepare it as an input and target for the *error network*. The input error values are used to predict the error at the next time step. The resulting predictions are denormalized and prepared as input for the *final network*.

In conclusion, the error network takes into account the compensated errors of past predictions and learns to correct the errors of errors that will occur in the future. It plays a complementary role in obtaining the most refined prediction outcome by considering the error outputs of the upper cascaded networks [2,9,10].

3.1.4 Data fusion in multivariate data sources

Data fusion refers to the process of integrating multiple data sources to construct a more comprehensive and accurate representation of the underlying phenomena of the target data. The primary objective is to enhance the predictive performance of time series models by leveraging the richness and diversity of multivariate data. This approach is particularly beneficial in scenarios where single-source data is insufficient to capture the complexity of the dynamics involved. By fusing data from various sources, it is aimed to achieve a more robust and reliable forecasting model, which is critical in domains like finance, healthcare, and environmental studies. The benefits of this method include improved accuracy, better generalization, and a deeper understanding of intricate temporal relationships among multiple variables [17].

Data fusion brings together data from many sources. While this can be useful, it also has downsides. Mixing data can sometimes make things confusing. Different sources may not always match well, leading to unclear, repetitive, conflicting, and noisy information. This can make it harder to understand and use the data correctly. Also, preprocessing and making data ready for the model can take a lot of time and effort. It's like trying to make a puzzle fit when each piece comes from a different set.

In the PECNET architecture, each type of data used for predicting the target data participates in training within the *cascaded neural networks*, specifically in a component called the variable network. Inside the variable network, data can be represented through networks connected at different frequency bands. To perform data fusion, the highest frequency network of the upper variable network is connected to the lowest frequency network of the next variable network below.

In the cascaded structure, the top input data is chosen by finding the feature of the multivariate data that has the highest correlation with the target data. This is done using a specific formula, as in (3.10).

$$r_{X_m y} = \frac{\sum_{i=1}^{n}(x_{mi} - \bar{X}_m)(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_{mi} - \bar{X}_m)^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{3.10}$$

where X_m represents the m-th feature of input X, y is the target data, n is the total number of observations, x_{mi} and y_i are individual observations of the m-th feature of

input X and target data, respectively, $\bar{X_m}$ and $\bar{y}$ are the means of the m-th feature of input X and target data, and $r_{X_m y}$ is the Pearson correlation coefficient, which measures the strength and direction of the linear relationship between the target data y and each feature of multivariate input X.

For the next variable network, the input is selected based on which feature of the multivariate X_m data has the highest correlation, not with the target data itself but with the data series created by the prediction error of the previous network. For example, considering three features with correlations of 0.8, 0.5, and 0.3 with the target data, the first network applies the feature with a 0.8 correlation. However, the selection for the second network does not simply go to the feature with a 0.5 correlation over the one with 0.3. This is because if the 0.5 correlated feature is highly dependent on the first feature, say with a correlation of 0.9, and the 0.3 feature has a lower dependency, like 0.4, on the first feature, then despite its lower correlation, the third feature might contribute more to the overall prediction accuracy in the final network. Also, including highly dependent variables in the same neural network system can increase over-fitting issues during training. During data fusion, to avoid this situation and the negative impacts mentioned above, the feature of X_m that has the highest correlation with the prediction error coming from the upper network is determined as input for the subsequent cascaded network. By doing so, the maximum benefit is obtained from data fusion, thereby enhancing predictive performance. With this logic, data fusion with other features continues as long as the condition of achieving high correlation is met and the prediction performance is enhanced.

The network created with the first feature is called the *primary network*, and the other cascaded networks used for data fusion are referred to as *supplementary network*s in the PECNET architecture.

3.1.5 Finalizing the prediction pipeline

PECNET includes a *final network* that receives all predictions (target prediction, error predictions, and compensated error predictions) from entire cascaded neural networks (i.e., variable networks and error networks) after their training is completed, to produce a final target prediction. This network is a type of fusion network that attempts to

capture a mathematical connection associating target data with target data predictions and errors, and it performs nowcasting, not forecasting. The working cycle of the PECNET architecture concludes with this network, and after the obtained predictions are denormalized, the final output is determined [9,11].

3.2 Superiorities & Inferiorities

Considering the architecture and general functioning of PECNET, its superiorities can be summarized briefly in the following points:

- High prediction accuracy and generalization without over-fitting by effectively using cascaded neural network structure and multiple time scales
- Residual error-based learning for gradual and collaborative improvement of predictive accuracy
- Improved prediction performance through data fusion by incorporating effectively the most relevant and orthogonal features of multivariate data into the learning process
- Orthogonal feature extraction mechanism with discrete wavelet transform for accuracy improvement
- Overcoming traditional normalization problems for nonstationary data with adaptive normalization
- No sensitivity to estimation errors by not accumulating errors using predicted values
- Less dependent to network parameters and can be applied to two different time series problems.
- The ability to incorporate other prediction models within its network structure.
- Computationally cheap and low time-consuming.

When it comes to inferiorities:

- The need for seamless and ample dataset to unleash its true potential

- The difficulty of determining the optimal parameters (such as wavelet type, window size, etc.) for its advanced data processing steps
- Susceptibility to erroneous behaviors such as data leakage and peeking.

3.3 Why is a Separate Framework Required for PECNET?

A framework software, or simply a framework, is a standardized set of concepts, practices, and criteria for dealing with a common type of problem that can be used as a reference to help developers build more structured, efficient, and maintainable software [18].

In the field of machine learning, the use of framework software is very important due to its ability to provide a well-organized, efficient environment for handling complex algorithms and data processes. These frameworks offer pre-built components and tools, facilitating rapid development and testing of machine learning models. They also ensure consistency and best practices across projects, reducing the likelihood of errors and increasing the reproducibility of results. This is especially important in a domain where precision and reliability are crucial [19].

The need for developing framework software for PECNET, in addition to the above, arises fundamentally to implement its superiorities and address its inferiorities. Upon deeper consideration, we can attribute this need to the following reasons in more detail:

- The arrangement of multiple sequential networks trained with each other's errors, the management of data flows amongst them, and the aggregation of produced outputs require a high degree of organizational and coding complexity. In the form of a framework, PECNET addresses this difficulty by providing a practical environment for implementation and testing.
- It is difficult to maintain data integrity and control time synchronization during the learning process because of the segmentation and transformation of data into various frequency bands, the modification of data lengths in each network, and the shifting mechanism used for temporal alignment. The framework aims to overcome these challenges by rendering them into a modular and parametric form.

- Performing data fusion experiments by feeding different features of multivariate data into cascaded networks in different orders is a challenging task to manage. So, the number of features and their order in the cascaded networks will be functional and automatically manageable through the framework.
- Potential issues with discrete-coded PECNET implementations include data leaks, high memory utilization, unnecessary workload on processing units, and unmanageable workflow problems caused by code repetitions. The framework is expected to provide optimal performance and functionality to address these issues.

4. PECNET FRAMEWORK:ARCHITECTURE AND IMPLEMENTATION

PECNET framework is an advanced tool designed for working with time series data. Its architecture incorporates a series of specially designed neural networks, each focused on specific tasks of prediction and error correction. In terms of implementation, it is built on a solid Python base, integrating a variety of libraries and tools that work together for model training, evaluation, and prediction. This combination of architecture and implementation makes it a versatile and powerful tool in the field of time series analysis, capable of handling complex data scenarios with precision and adaptability.

4.1 Development Environment, Setup and Dependencies

The PECNET framework has been developed using the C compiler and Python interpreter, and package versions have been released for both Unix-based and Windows operating systems. The design strictly adheres to the principles of object-oriented programming and has been entirely coded in Python. Python has been chosen for its versatility and extensive support in data science and machine learning applications. Additionally, Python's ecosystem offers an array of libraries and tools, each contributing to different facets of the PECNET framework.

The successful deployment and operation of the PECNET framework involve a well-organized setup of various dependencies. In the context of software development, a *dependency* is essentially a software component or module that a particular program requires to function correctly. For PECNET framework, these dependencies include a range of Python libraries, each serving specific roles and often interdependent, meaning the functionality of one might rely on the prior installation of another. The primary dependencies and their roles in the framework are as follows:

- **NumPy:** This library is foundational for numerical computations in Python. It provides support for large, multi-dimensional arrays and matrices, which is a prerequisite for data manipulation tasks in PECNET.
- **Pandas:** Leveraging NumPy, Pandas is crucial for efficient data manipulation and reading operations. It simplifies handling tabular data and is instrumental in the initial stages of data processing in PECNET.
- **Matplotlib and Seaborn:** These libraries are used for data visualization. Matplotlib provides a wide range of plotting functions, while Seaborn, which is built on top of Matplotlib, offers a higher-level interface for statistical graphics. Their role in visualizing data patterns and model outputs is important for analysis in PECNET.
- **Scikit-learn (sklearn):** A fundamental machine learning library in Python, often used alongside NumPy and Pandas. It offers a wide range of tools for data preprocessing, model selection, evaluation, and various machine learning algorithms for PECNET.
- **PyWavelets:** This library is essential for performing wavelet transforms in PECNET. It offers the flexibility to use different types of wavelets easily and allows transformations at various levels and options.
- **PyTorch:** PyTorch is a cornerstone for building and training neural networks in PECNET. PyTorch's dynamic computation graphing capabilities, effective management of GPU-CPU interaction, and the flexibility to adjust almost every parameter of a neural network make it ideal for complex neural network architectures.

To set up the PECNET framework successfully, certain prerequisites must be met beyond just the dependencies. It requires a specific Python version that is compatible with the dependencies and adequate hardware and operating system specifications to function efficiently. The necessary criteria for a smooth, error-free setup and operation of the PECNET framework can be summarized as in Table 4.1:

Table 4.1 : Setup Requirements of PECNET.

Requirement Type	Details
Python Version	3.11.5
Operating System	Min. Windows 8 or above for Windows-based systems. No restriction for Unix-based systems. IOS, Android and others are not supported.
Hardware Requirements	Min. 2 Ghz 32-bit, 64-bit or ARM64 processor, 4 GB RAM, 10 GB storage, Min. GTX 1000 series GPU for CUDA support.
Dependency Versions	NumPy 1.26, Pandas 2.1, Matplotlib 3.8, Seaborn 0.13, Scikit-learn 1.2, PyWavelets 1.41, PyTorch 2.1, CUDA 11.8.

To complete the PECNET setup, two methods can be followed:

- **Method 1: Standard Setup**

  1. Install Python v. 3.11.5.
  2. Use Python's package installer, *pip*, to install PECNET by running the command `pip install pecnet`. This will automatically install all dependencies and integrate PECNET into your current project.
  3. It is recommended to create a new development environment using tools like *conda*, *venv* before installation. This prevents potential conflicts with existing dependencies.

- **Method 2: Custom Setup for Code-Level Modifications**

  1. Access the PECNET repository at `https://github.com/pecnet/pecnetframework` and download the source code into your project directory.
  2. Install Python 3.11.5.
  3. Install the dependencies within the PECNET directory using *pip* or another package installer of your choice. Note: For PyTorch, use the command "`pip`

`install ...`" generated based on your hardware and system requirements from the official PyTorch website.

4. With these steps, PECNET is ready for use with the flexibility for code-level customizations.

4.2 High-Level Overview and Component Interaction

PECNET framework is built upon four primary, interacting packages: `models`, `network`, `preprocessing`, and `utils`. Each package serves a distinct function within the framework.

- **models package:** This package is designed to host various machine learning models. Right now, it only has the `BasicNN` class, which is a simple MLP (Multi-Layer Perceptron) structure. The main job of this package is to build and train neural network models.
- **network package:** "network" package, with its classes `ErrorNetwork`, `FinalNetwork`, `VariableNetwork`, `Pecnet`, `PecnetBuilder`, is responsible for establishing the PECNET model, transferring data between networks, generating final predictions, and evaluating the outcomes. This package plays a key role in ensuring that each stage of the predictive process is efficiently managed and seamlessly integrated within the PECNET framework.
- **preprocessing package:** This package is responsible for transforming raw data into a format that is suitable for analysis and prediction within the PECNET networks. It includes classes like `DataPreprocessor`, `Normalizers`, etc., each playing a key role in this transformation process. Additionally, after the final predictions are made, this package also takes charge of converting the predictions back into a form that is understandable and meaningful for the user.
- **utils package:** The utility package offers essential support and utility functions, encapsulated in the `Utility` class. It provides functionalities such as data loading, visualization tools, and hyperparameter settings.

These packages work together to create an integrated system that transfers data from preprocessing to model prediction. The classes contained within these packages and their interactions are illustrated as a package diagram in Figure 4.1.

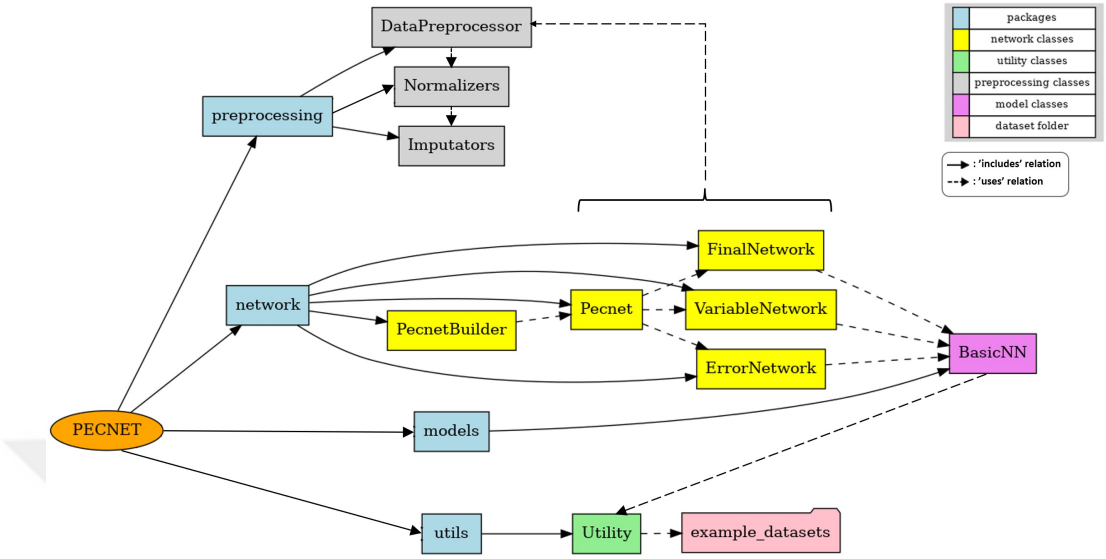

Figure 4.1 : Package Diagram of PECNET Framework.

The PECNET framework has been developed based on the machine learning software cycle steps of the *scikit-learn* library. These steps and the packages that fulfill them are summarized in Figure 4.2.

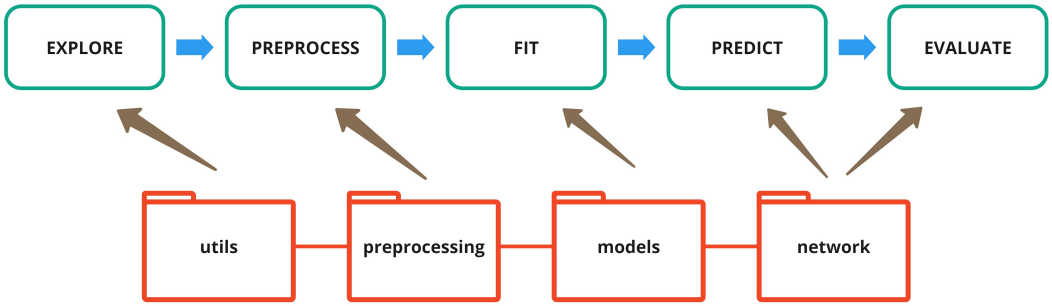

Figure 4.2 : Packages of PECNET Framework and Fullfilled ML Tasks.

4.3 Detailed Analysis of Internal Mechanics

This section will concentrate on the tasks carried out by the classes within the packages, and the subsections will use function-level examples of them to illustrate various stages of the prediction process.

The framework's static `Utility` class in the "utils" package is responsible for important tasks like using preloaded test data, plotting data for user-specific analysis, and setting various hyperparameters for the neural network architecture. *load_apple_test_dataset()*, *plot()*, *set_hyperparameters()* functions can be used for these purposes, respectively.

`DataPreprocessor` and `Normalizers` classes within "preprocessing" package are the key components for data preparation in the framework. The `DataPreprocessor` class is designed as a singleton to ensure a single global instance that manages preprocessing functions consistently and efficiently across the application. Data initially sampled through the `DataPreprocessor` class, based on defined statistical parameters and period information. Then, for the networks being built, data sequences are generated for each period using a sliding window approach. Subsequently, the `Normalizers` class uses average subtraction to adaptively normalize each sequence element. This normalization procedure follows a similar structure to sklearn's flow, with methods such as *fit()*, *transform()*, and *fit_transform()*. Following this stage, the `DataPreprocessor` class applies a wavelet transform to each normalized sequence based on the wavelet type selected. Finally, the data is divided into training and test sets in the desired ratio, and it is ready to feed into the network. If necessary, domain experts can scale the train and test data sequences using a specific coefficient via the inner `Scaler` class. All these processes are encapsulated within the *preprocess()* and *preprocess_errors()* wrapper methods for the main data and the compensated error data, respectively.

`BasicNN` class in "models" package is a basic neural network class that extends PyTorch's neural network module for creating and training neural network models. This class is intended to handle simple to moderately complex neural network tasks. It allows for the initialization of a neural network with a customizable number of hidden layers and units, as well as hyperparameters, and includes methods like *forward()*, *loss()*, *fit()*, and *predict()* for training the network and making predictions. This class also manages device interactions like the training of the model on the GPU and the transfer of outputs back to the CPU. It is designed to be used with the PECNET `Variable Network`, `Error Network`, and `Final Network` classes.

`PecnetBuilder` class in "network" package is coded using the builder design pattern and provides a fluent interface to sequentially add different components to build a PECNET model. It encapsulates the creation logic of the PECNET and its constituent networks, allowing for a more readable and manageable setup. It assembles the PECNET model using *add_variable_network()*, *add_error_network()* and *add_final_network()* methods. It finalizes the construction of the PECNET model and returns the instance with *build()* method.

The `Pecnet` class in "network" package, which serves as a container for the `VariableNetwork`, `ErrorNetwork`, and `FinalNetwork` classes, is coded using the composite design pattern. It operates as the framework's orchestrator, bringing these network classes together. It coordinates the flow of data and predictions across them, ensuring seamless interaction and processing. This class also employs the *switch_mode()* method to transition all components between train and test modes. The *predict()* method generates predictions based on unseen data and then *evaluate()* function compares the predicted values with the actual values and calculates several metrics to determine the model's accuracy and performance.

`VariableNetwork` class in "network" package, which is used in a predictive modeling pipeline to handle different frequency bands of one feature from input data, stores predictions, errors and compensated predictions generated from its cascaded networks. It supports both training and testing modes, and it can be instantiated iteratively to handle multi-dimensional input data.

`ErrorNetwork` class in "network" package represents the PECNET's error network, which is a part of the cascaded neural network structure. Its main function is to predict compensated errors in a separate prediction layer and then provide error predictions to the final network.

`FinalNetwork` class in "network" package is designed to integrate and finalize prediction pipeline coming from a cascaded sequence of networks. It works as a fusion network and generates a final, unified prediction using all network's outputs as input. These final predictions are presented via *get_final_predictions()* method after being unscaled and denormalized with the help of `DataPreprocessor` class.

The class diagram in Figure 4.3 summarizes the above-mentioned classes, their features and functions, as well as their interrelations .

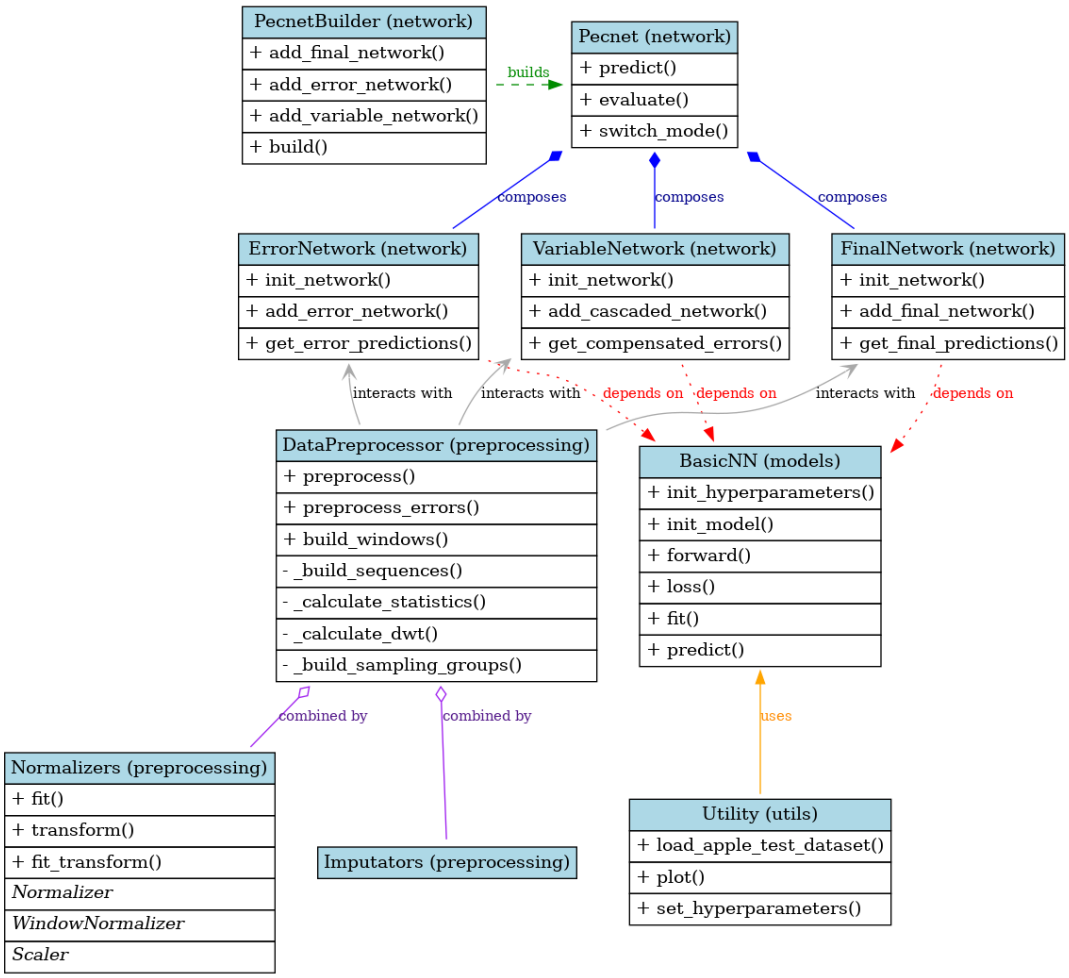

Figure 4.3 : Class Diagram of PECNET Framework.

On the diagram, standard arrows represent *association* relationships, dashed arrows indicate *dependency* relationships, arrows with an empty diamond show *aggregation* relationships, arrows with a filled diamond denote *composition* relationships, and the labels on the arrows specify the type of relationship. Also, '+' symbol next to the method name indicates its *public* accessibility, while '-' symbol signifies that the method is *private*.

4.3.1 From raw data to model-ready formats

This section will provide practical examples and relevant components of the framework to demonstrate the data preprocessing procedures previously explained in theory.

The examination and preparation of the data for PECNET are conducted through the *plot()* and *preprocess()* functions in the singleton Utility and DataPreprocessor classes, respectively. The *plot()* function displays time series of varying lengths on a single grid, aligned and without distortions caused by automatic completion of matplotlib, facilitating easy visualization. The *preprocess()* function, on the other hand, masks lower-level functions such as *build_windows(), _build_sampling_groups(), _calculate_statistics(), _calculate_dwt(), _build_sequences(), normalize_with_prewindow()*, etc., to quickly prepare the input data for PECNET variable networks. For the error network, the equivalent of the *preprocess()*, the *preprocess_errors()* function, is automatically used while network is constructed. The basic syntax of these two functions is as follows:

```
Utility.plot(
    aapl_timestamps,
    aapl_prices,
    title='Apple Stock Prices',
    xlabel='Date',
    ylabel='Price ($)',
    tick_size=5,
    save_location=None)
```

```
DataPreprocessor().preprocess(
                                data=aapl_prices,
                                sampling_periods=[1, 4, 8],
                                sampling_statistics=["mean"],
                                sequence_size=4,
                                error_sequence_size=4,
                                wavelet_type="haar",
                                test_ratio=0.01 )
```

How these two functions operate on an example of time series data (*aapl_prices*) will be explained in the following subsections through visuals. For lower-level functions and their purposes, check Appendix A.

4.3.1.1 Data loading techniques and visualization utilities

The PECNET framework hosts several different time series datasets for testing purposes in the `example_datasets` folder in .csv format. For instance, APPLE stock prices data can be imported into the software environment as follows:

```
aapl_timestamps,aapl_prices=Utility.load_apple_test_dataset()
```

A similar syntax can be used for other datasets as well.

The data and the related predictions can be visualized and examined by using the *plot()* function under the Utility class with different parameters. It aligns the data and predictions to be visualized according to the timestamp array it receives as a parameter and labels them with a default naming (if no label is provided as a parameter). If the *save* parameter is used, the visual can be exported at 600 dpi.

4.3.1.2 Sampling, windowing and sequence preparation

The input data will be transformed as illustrated in Figure 4.4 when the *preprocess()* function processes it for the PECNET model in an example scenario where the sampling period (window) is *[1, 4, 8]*, the sampling statistic is the *"mean"*, and the sequence size is *4*. Here, *t* represents the current timestamp, and μ denotes the mean of the values at any given interval.

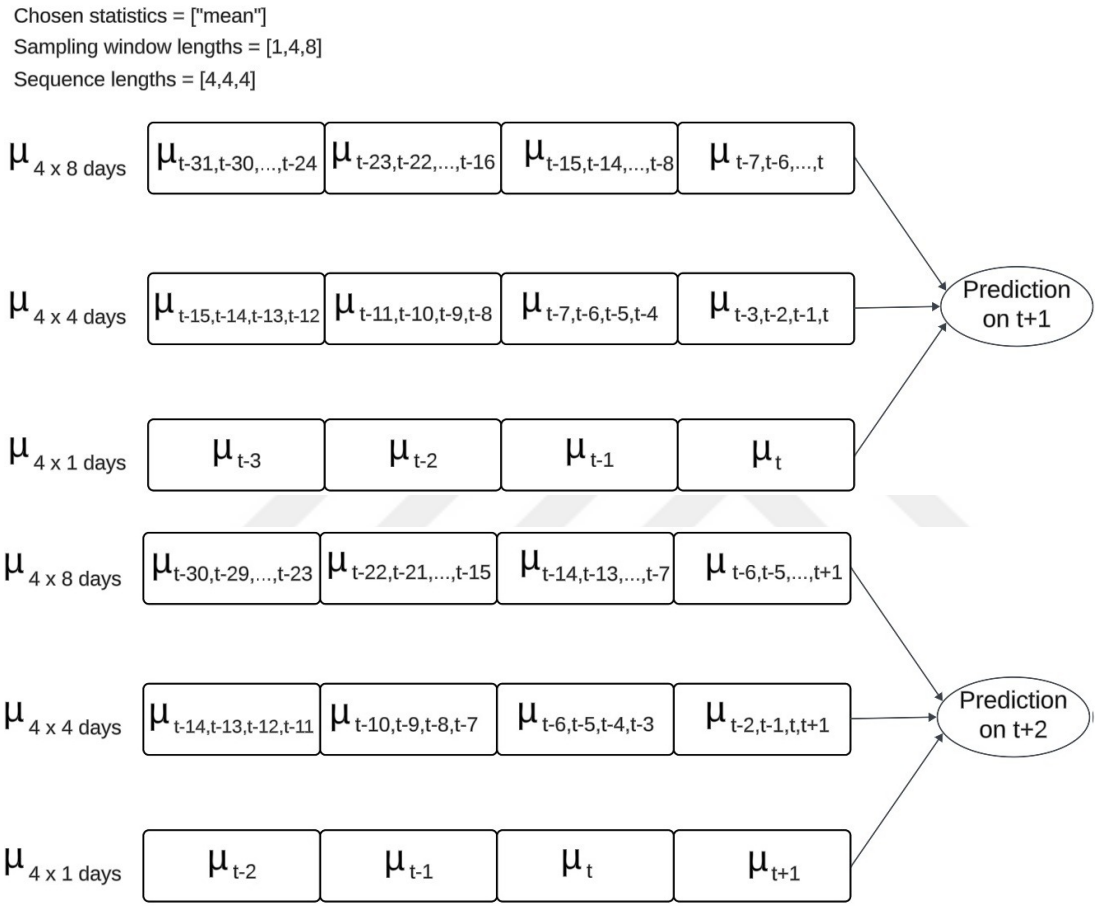

Figure 4.4 : Sampling, Windowing and Sequence Preparation on Input Data.

For the sake of demonstration, the resulting representation of input data collected at 1-day intervals is shown for only two separate time frames, *[t-31, t]* and *[t-30, t+1]*. This data, when processed for all moments as sliding windows of step size one with the specified three different window sizes, one statistic, and a sequence size of four, will yield three separate time series for use in the PECNET model. The prediction for

the next moment will be made with these three time series data, of course, after they have also undergone normalization and wavelet transform steps.

4.3.1.3 Normalization and wavelet transform strategies

After the input data is sampled at various frequencies, as shown in Figure 4.4, each sampled data is adaptively normalized in sequence using average subtraction. Scaling of the data can also be performed beforehand. Figure 4.5 illustrates how the adaptive normalization method is implemented to sampled input data in the range *[t-31, t]* using the given example values.

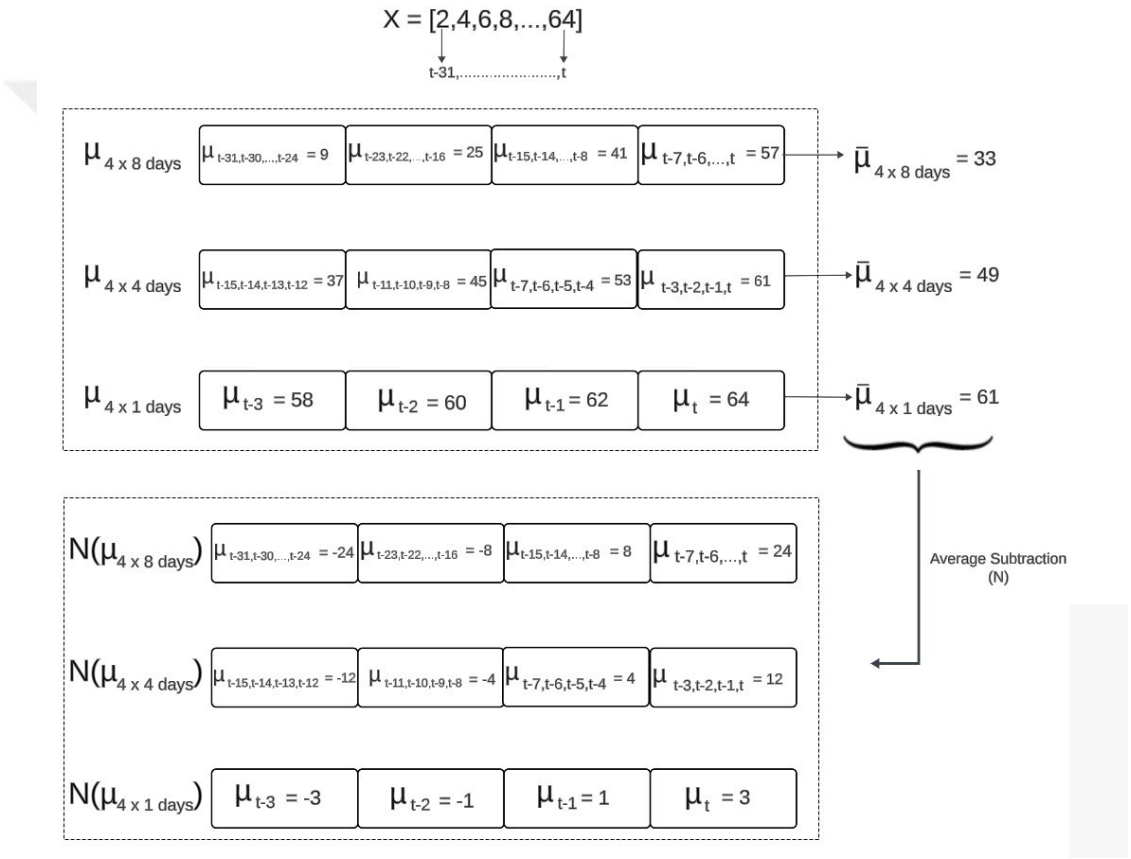

Figure 4.5 : Adaptive Normalization with Average Subtraction.

The processing of the input data concludes with the wavelet transformation phase following the normalization step. Depending on the internal structure of the data, transformations can be performed with different wavelet types (such as Haar, Symlet, Daubechies, etc.). The PECNET framework uses the Haar wavelet transformation by default due to its success in capturing sudden changes, simplicity, and speed. A Haar

wavelet is applied to an example sequence shown in Figure 4.5, as demonstrated with Mallat's algorithm in Figure 4.6 [20]. Once the transformation is completed for all sequences, the first coefficients of each sequence will be removed due to being zero as a result of average subtraction.

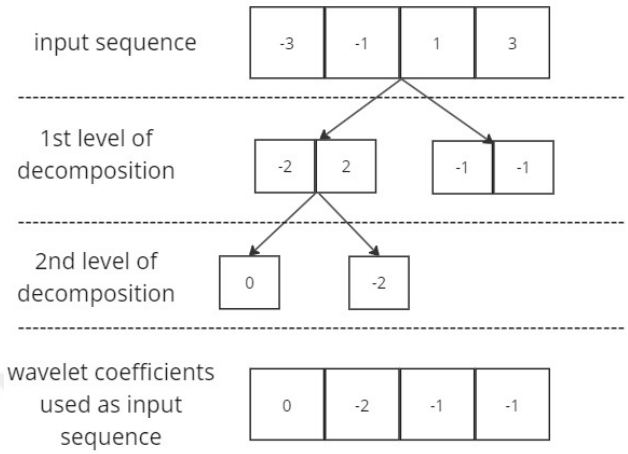

Figure 4.6 : Last Phase of Data Processing : Wavelet Transformation.

4.3.2 Neural network fundamentals

The default structure of the artificial neural network utilized in the PECNET framework networks, including the variable network, error network, and final network, is defined within the BasicNN class, as depicted in Figure 4.7.

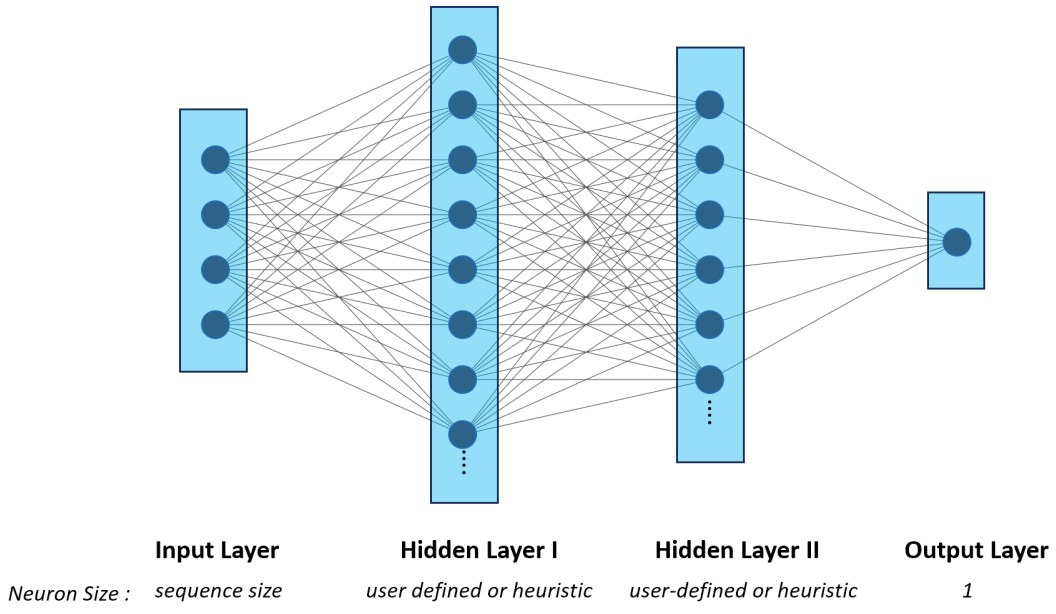

Figure 4.7 : Artificial Neural Network Structure in PECNET Framework.

The default configuration of this network consists of two hidden layers, where the number of neurons is automatically modified based on the size of the input data. The activation function it employs is the Gaussian Error Linear Unit, or *GELU*, as given in (4.1) and (4.2). GELU improves the generalization capabilities of deep learning models by generating adaptive threshold values based on inputs and enabling smooth transitions [21]. GELU also learns from negative information.

$$\mathrm{GELU}(x) = x \cdot \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \tag{4.1}$$

where

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{4.2}$$

Here, x is the value entering the neuron, and *erf(x)* is the mathematical expression of the Gaussian error function, illustrating how it is calculated as a function of the x value.

The *He initialization* method is employed for the initialization of neuron weights. The aim of He initialization is to stabilize the propagation of data in deep artificial neural networks by adjusting the variance of the weights according to the amount of input to the neurons. When employed in combination with ReLU-based activation functions such as GELU, this approach brings the weights to an ideal initial state, hence expediting the learning process and enabling better results [22].

As the loss function, *MSE (Mean Squared Error)*, which has a "punitive" structure against large errors, is used as given in (4.3).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{4.3}$$

The *Adam Optimizer* has been chosen as the optimization algorithm with a learning rate of 0.01 due to the fact that it offers adaptable learning rates for each parameter and that it converges quickly [23]. Optimization is performed iteratively in batches according to a batch size dynamically determined by the input size and a default number of 300 epochs.

The entire forecasting procedure in that structure, from setting up of training configurations until the presentation of results to the user, has been organized to operate on the GPU.

4.3.2.1 Hyperparameter tuning and heuristic approaches

The hyperparameters in the implemented artificial neural network architecture can be configured by using the *set_hyperparameters()* function within the `Utility` class, as demonstrated in the code snippet below.

```
Utility.set_hyperparameters(learning_rate=0.001,
                            epoch_size=500,
                            batch_size=96,
                            hidden_units_sizes=[64,32])
```

Additionally, when the same function is used as *set_hyperparameters(heuristic=True)*, the hyperparameters are set in the following manner:

- The *epoch size* is set to 300, and the *batch size* is set to the square root of the input data size.
- The *learning rate* for the Adam optimizer is set to 0.01.
- The *number of hidden layers* is set to 2 in total. This is because, currently, two hidden layers are sufficient for most tasks except some extreme cases. [24].
- The *number of neurons* in first hidden layer is $2/3$ the size of the neurons in input layer, plus the size of the neurons in output layer. The number of neurons in second hidden layer is calculated using the formula in (4.4).

  $$\frac{N}{8 \times (s_i + s_o)} - S \tag{4.4}$$

  where N is the number of observations in the training dataset, s_i and s_o are the numbers of input and output sequences, respectively, 8 is the scale factor, which can be changed between 1-10, and S is the number of neurons in the first hidden layer [25,26].

In both usages of the *set_hyperparameters()* function, the *optimizer* is configured as Adam, the *activation function* is set to GELU, and the *weight initialization method* is chosen as He Initialization.

4.3.2.2 Establishing the network infrastructure and learning procedures

The neural network infrastructure is established by creating an instance of the *BasicNN* class. At this stage, hyperparameters are set to their defaults, GPU-CPU device settings are finalized, and the model is preloaded.

The *forward()* and *loss()* functions included in BasicNN are utilized in the iterative learning process. The *forward()* function enables the flow of data across the layers in order to generate a prediction. The *loss()* function computes the error at each iteration and is employed to optimize the weights of the neurons. Another function, *fit()* uses these two functions with current hyperparameters and initiates a batch-by-batch learning procedure on the training data cast into PyTorch tensors, as illustrated in the following code snippet.

```
def fit(self, input_values, target_values):

        input_tensor = torch.tensor(input_values)
        output_tensor = torch.tensor(target_values)
        dataset = TensorDataset(input_tensor, output_tensor)

        train_loader=DataLoader(dataset=dataset,
                                batch_size=self.batch_size,
                                shuffle=False)

        for epoch in range(self.epoch_size):

            total_loss = 0.0

            for inputs, targets in train_loader:

                inputs, targets = inputs.to(self.device), targets.to(self.device)
                predict = self.forward(inputs)
                loss = self.loss(predict, targets)
                loss.backward()

                self.optimizer.step()
                self.optimizer.zero_grad()

                total_loss += loss.item()
                ...
```

Upon completion of the learning procedure, the model is switched to evaluation mode and predictions can be made with the test data using the *prediction()* method, as shown in the following code.

```
def predict(self, X):

    self.eval()

    with torch.no_grad():
        inputs = torch.tensor(X,device=self.device)
        predictions = self(inputs)
        return predictions.detach().cpu().numpy()
```

4.3.3 Methodology for building the PECNET architecture

The PECNET architecture is formed by joining variable networks, an error network, and a final network as previously shown in Figure 3.1. Each variable network is built with a different feature of the input data. Each used feature is processed based on the sampling periods and sampling statistics specified in the *preprocess()* function, and divided into multiple time series of different scales. These time series are first heuristically arranged sequentially according to statistics, and then, within the same statistic, they are arranged from highest to lowest period. After that, a neural network structure is established for each time series, where the first time series is trained with the target data, and each subsequent network is trained with the training error of the previous network, resulting in a hierarchically connected variable network. Variable networks created for other features are connected to the upper variable network chain. In this way, while forming a chained network structure, the order in which features are included is determined by looking at the maximum correlation with the training error produced by the last network. Then, errors coming through the compensation pipeline are connected to the error network as input data with a time shift. The error network predicts errors for the next step. Finally, the outputs of all networks (target prediction and error predictions) are given to the final network in an attempt to make the most accurate prediction for the target data. While all networks focus on predicting the next step, the final network focuses on producing results by establishing a relationship between the current step and its errors.

In the PECNET model, all sub-networks within the hierarchical structure manage their own unique `BasicNN` object, and this object is trained using the relevant data with the *fit()* method in train mode, and predictions are obtained with the *predict()* method in test mode.

4.3.3.1 Variable network: Managing band-specific networks

Variable networks are network units that create the statistical and frequency band-based network structure of time series data belonging to a feature. The `VariableNetwork` class manages the setup of this network structure as well as the storage and transfer of

the results of each inner network component. Its design is shown in the following pseudocode, Algorithm 1.

Algorithm 1 Training and Prediction in *VariableNetwork*

Require: X_bands, multidimensional input data across various frequency bands and statistics; y_band, target output data.

Ensure: Predictions ($\hat{y}$), errors (e), compensated predictions ($comp\_\hat{y}$), compensated errors ($comp\_e$)

Initialization: Create empty lists for nn models, predictions, errors, and compensated predictions.
for each frequency f **in** X_bands **do**
    for each statistic s **in** X_bands **do**
        Extract subset $x = X\_bands[frequency = f, statistic = s]$
        Set target y to y_band or previous errors, depending on the iteration.
        if mode $==$ **"train" then**
            $model$ = Train a new *BasicNN* with (x, y).
            Append $model$ to the models list.
        else if mode $==$ **"test" then**
            $model$ = Retrieve related model from models list.
        end if
        $\hat{y}$ = $model$ with (x, y).
        $e = \hat{y} - y$
        if first iteration **then**
            $comp\_\hat{y} = \hat{y}$
        else
            $comp\_\hat{y} = comp\_\hat{y} - \hat{y}$
        end if
        Update predictions, errors, compensated predictions lists with new values.
    end for
end for
Calculate compensated errors with $comp\_e = comp\_\hat{y} - y\_band$.
Return predictions for all networks, compensated errors, last compensated predictions and last errors.

The `VariableNetwork` object retains the following variables after each training session:

- All *predictions* generated within its internal networks, intended for transmission to the final network,
- The latest *compensated predictions* and the latest *errors*, to facilitate the continuation of the compensation pipeline and the error pipeline for newly added variable networks,

- The calculated *compensated errors*, intended for transfer to the error network.

4.3.3.2 Error network: Handling predictive inaccuracies

The error network receives compensated errors, shifted by one time unit, from variable networks through the use of a compensation pipeline. Then, it constructs a neural network model with this error sequence to predict what the next error will be. The setup and management of this model are handled by the `ErrorNetwork` class. This class completes the training and testing process with a BasicNN object and sends the error predictions to the final network so that the original predictions can be finely adjusted. It also stores variables that will continue the compensation and error pipeline, enabling the addition of other networks subsequently.

The error input entering the error network is processed through the same procedures (sampling, normalization, wavelet transform, etc.) as the input data used in variable network. For this purpose, the *preprocess_errors()* method within the DataPreprocessor class is utilized.

The fundamental design of the ErrorNetwork class can be summarized as described in Algorithm 2.

Algorithm 2 Error Correction in *ErrorNetwork*

Require: *error_band*, error data transmitted from the compensation pipeline; *last_compensated_preds*, compensated predictions after last variable network.
Ensure: Error predictions, errors of errors, and compensated error predictions for the Final Network.

1: Initialize model for storing error predictive model.
2: Set operational mode to 'train'.
3: **Data Preprocessing:** Adjust and format *error_band* suitable for the error network with *DataPreprocessor.preprocess_errors()*
4: **if mode** == **"train" then**
5: Train a new *BasicNN* model with preprocessed data.
6: Store the trained model.
7: **else if mode** == **"test" then**
8: Load the model for prediction.
9: **end if**
10: Calculate error predictions($\hat{e}$) with model predictions plus denormalization term.
11: Calculate compensated error predictions with $comp\_\hat{y} - \hat{e}$.
12: Calculate errors of errors as the difference between actual and predicted errors.
13: **return** Error predictions, errors of errors, and compensated error predictions.

4.3.3.3 Final network: Synthesizing predictions

The final network refines the target data prediction through nowcasting by using the prediction results from the variable networks and the error network, with only the first serving as the target data prediction and the others as cascaded error and compensated error predictions. The establishment, management of its neural network structure, and the processing of the produced predictions—including denormalizing and unscaling to prepare them for comparison with the original data—are conducted by the `FinalNetwork` class.

The basic algorithmic operation of the FinalNetwork is as follows in Algorithm 3.

Algorithm 3 Synthesizing Predictions in *FinalNetwork*

Require: $all\_cascaded\_predictions$, a 2D array of predictions from cascaded models.
Ensure: $final\_predictions$, denormalized and unscaled unified predictions.

On Initialization:
$model = None$
$mode =$ 'train'
$init\_network(all\_cascaded\_predictions)$

Method $init\_network(all\_cascaded\_predictions)$:
$y\_processed = DataPreprocessor.get\_final\_y\_processed()$
$preds = add\_final\_network(all\_cascaded\_predictions, y\_processed)$
$final\_predictions = generate\_final\_preds(preds)$

Method $add\_final\_network(x, y)$:
if $mode =$ 'train' **then**
    $model = BasicNN().fit(x, y)$
else if $mode =$ 'test' **then**
    Use pre-trained $model$ for prediction.
end if
return $model.predict(x)$

Method $generate\_final\_preds(preds)$:
$denorm\_preds = preds + DataPreprocessor.get\_final\_denormalization\_term()$
$unscaled\_preds = DataPreprocessor.scaler.unscale1D(denorm\_preds)$
return $unscaled\_preds$

Due to the differences in window sizes and time levels between the outputs of the variable networks and the error network, it is necessary to perform trimming and

time adjustment on original target data. In Algorithm 3, the *get_final_y_processed()* function returns time-adjusted and size-trimmed target y values for training or test purposes, depending on the network mode. Then, after predictions obtained, the *generate_final_preds()* function adds the time-adjusted and size-trimmed denormalization data to the predictions using the *get_final_denormalization_term()* function and then brings them to the same scale as the ground truths with the *unscale1D()* function.

The behaviour of final network and internal processes at any given time t is shown in Figure 4.8. Here, the predictions from a variable network with three inner layers and the compensated error predictions coming from the error network are used to predict the original target data $y(t) = 100$. For the sake of accurately demonstrating the relationship, it is assumed that the error network makes error-free predictions. Similarly, it is assumed that the final network can determine the relationship between $[104, 5, 2, -3]$ and $[100]$ (as $104 - 5 - 2 - (-3) = 100$) without errors.

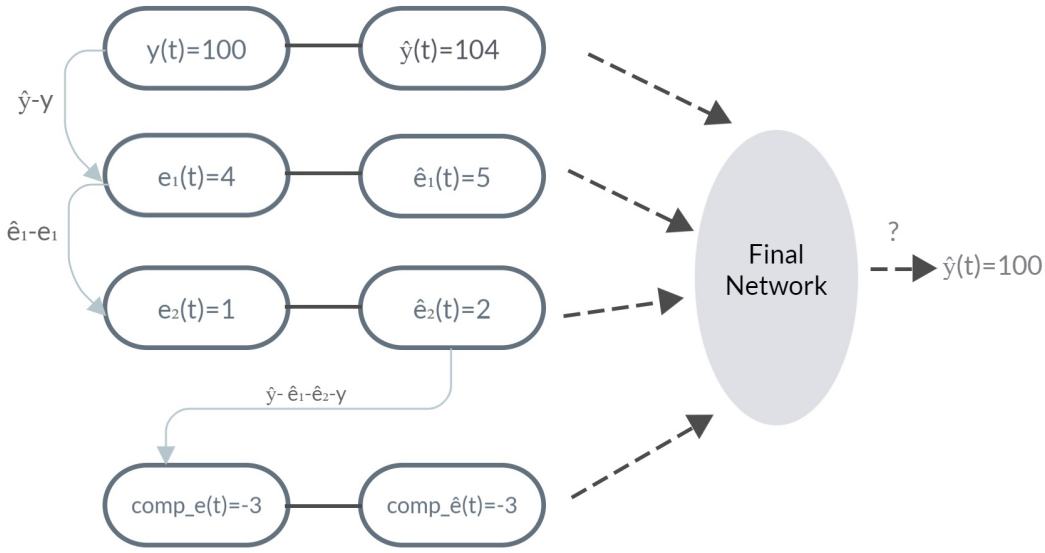

Figure 4.8 : Behaviour of Final Network in PECNET.

4.3.3.4 Encapsulating PECNET's functionality and learning cycle

The integrated setup of the PECNET model is done through `PecnetBuilder` class, and the operation of the learning process, data transfers between components, and the evaluation of predictions are conducted via the `Pecnet` class.

The general structure of the PecnetBuilder class can be illustrated as shown in Algorithm 4.

Algorithm 4 Structure of PecnetBuilder

```
Class PecnetBuilder
    Attributes:
        pecnet: An instance of the Pecnet class.

    Method __init__()
        Initialize Pecnet instance.

    Method add_final_network()
        Adds a final network to Pecnet.
        Returns self for fluent interface.

    Method add_error_network()
        Adds an error network to Pecnet.
        Returns self for fluent interface.

    Method add_variable_network(X_train, y_train)
        Adds a variable network to Pecnet.
        Returns self for fluent interface.

    Method build()
        Finalizes Pecnet construction.
        Returns the Pecnet instance.
```

A basic PECNET model or a model that includes data fusion can be created with the PecnetBuilder class without considering the complexities of data transfers, network interactions, etc. at the code level, as illustrated in Figure 4.9.

```
(PecnetBuilder().add_variable_network(X₁, y₁)
                .add_error_network()
                .add_final_network()
                .build())
```

```
(PecnetBuilder().add_variable_network(X₁, y₁)
                .add_variable_network(X₂,y₂)
                .add_error_network()
                .add_final_network()
                .build())
```

Figure 4.9 : Generation of Pecnet Model with PecnetBuilder.

The model constructed with PecnetBuilder is maintained on the Pecnet class. This class manages the coordinated operation of the variable network, error network, and final network components, transferring the data required by each. The entire network's prediction process is established within this class, and the prediction results are

measured according to specific metrics within this class as well. The basic algorithmic structure of the Pecnet class, based on methods, is documented in Algorithm 5.

Algorithm 5 Structure of Pecnet Class

Class Pecnet
    Description: Manages and integrates variable, error, and final networks within the Pecnet model.

    Attributes:
        final_network: Network for final predictions.
        error_network: Network for error predictions.
        variable_networks: Networks for target predictions and data fusion.

    Method predict(X_test, test_truth)
        Prepares networks for prediction.
        Uses variable_network for initial predictions.
        Adjusts for errors using error_network.
        Compiles final predictions with final_network.
        Returns final predictions.

    Method get_shifted_compensated_errors()
        Retrieves time-adjusted compensated errors for error network.

    Method get_last_compensated_predictions()
        Retrieves last compensated predictions from last variable_network.

    Method get_all_preds()
        Aggregates predictions from all networks for final_network.

    Method evaluate(pred, y)
        Compares predictions to actual values using R2, RMSE, MAPE.
        Returns evaluation results.

    Method switch_mode(mode)
        Toggles mode across all networks and other components in framework.

4.4 Step-by-Step PECNET Construction:Code-Level Demonstration

The solution steps for handling a time series prediction task from beginning to end, and their corresponding code equivalents in the framework, are provided below in Figure 4.8. Assuming the framework installation is successfully completed and the data is transferred to the software environment clean and without errors, this basic code flow shows how the PECNET model to be applied for many time series prediction tasks.

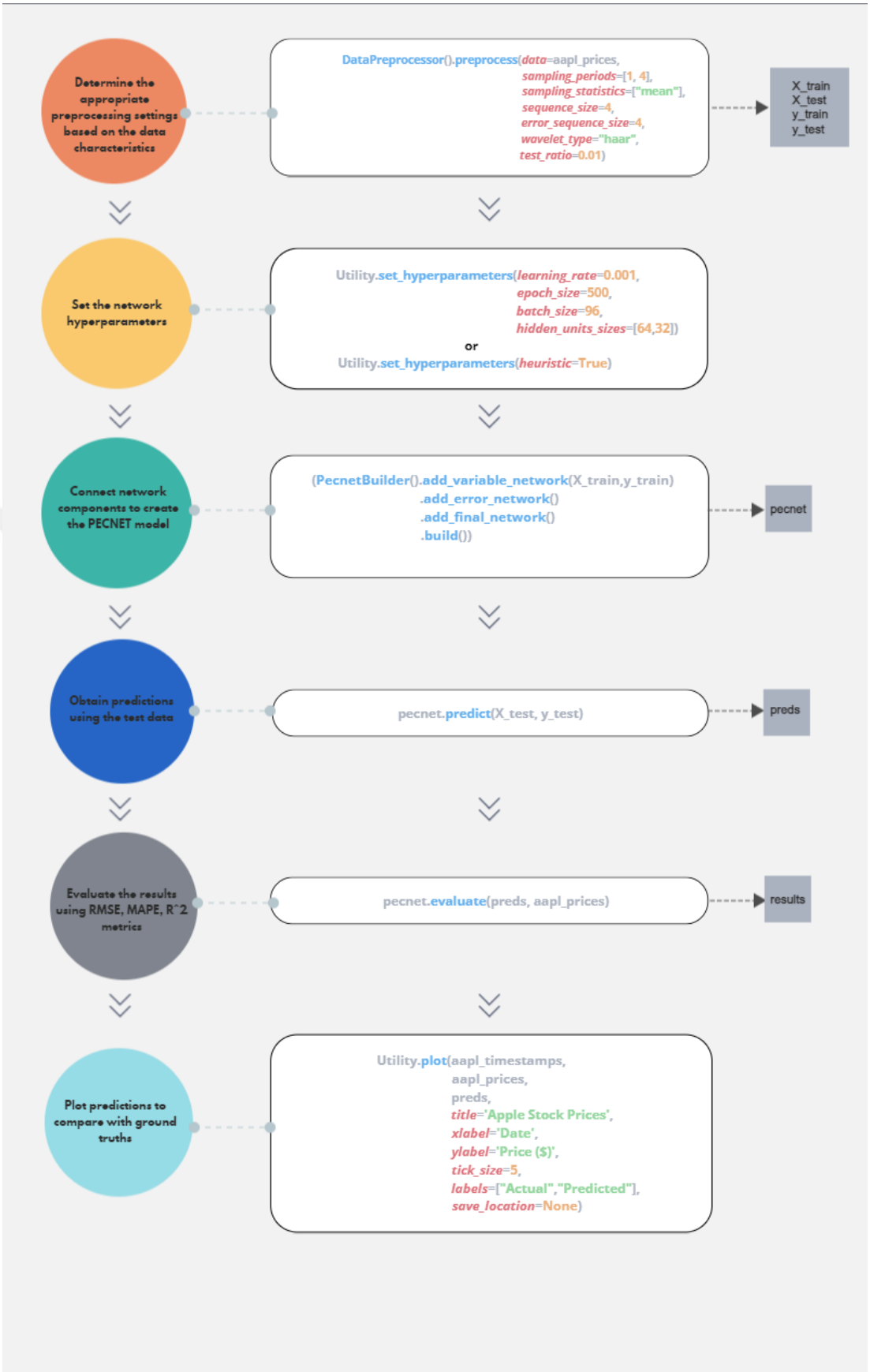

Figure 4.10 : Time Series Prediction Workflow with PECNET Framework.

In the flow above, univariate Apple stock data (*aapl_prices*) is used. An executable code version of this flow can be found in the *example.py* file in the framework's GitHub repository.

The flow described above requires modifications in two distinct scenarios. First, in cases where the target data is different from the input data, the target data also must be processed through the *preprocess()* function and then supplied to the variable network in step 3. Second, for a multivariate time series, training should first be completed with a single variable network using the input data's one feature that has the highest correlation with the target data. Then, the errors occurring during training should be accessed with the *get_last_variable_errors()* function under the Pecnet class, and a second variable network should be added with the second feature of the data that is most correlated with these errors. The target data for this network should be the errors we mentioned. This procedure should be repeated until no correlation is found or until the prediction performance deteriorates.

4.5 Technical Benefits

The troubles at the coding and usage level, which arise from the integration of components such as cascaded neural networks, modular feature extractions, and fusion networks in PECNET, have been eliminated. PECNET is now able to be utilized by users without the need for specialized coding knowledge.

The framework offers users customizable extensive feature engineering, data preprocessing, and neural network architecture design capabilities through its parametric and modular structure. Additionally, data fusion applications can be integrated into the existing model with just a single line of code. Consequently, an effective environment is provided for the faster development, testing, configuration, enhancement of predictive performance, and robustness of the designed model. Rapid deployment in real-time prediction applications is facilitated.

The framework also optimally utilizes GPU, CPU, and memory resources, resulting in low computational resource requirements and reduced execution time while delivering high performance in time series prediction tasks.

5. PRACTICAL APPLICATIONS AND PERFORMANCE ANALYSIS

The performance of the developed software framework was evaluated through its practical applications in two scenarios. The first is for stock price prediction on the financial data forecasting platform, *cashspeeder.com*; the second, prediction of earthquake occurence using time series data from the Electrostatic Rock Stress (ERS) monitoring method within the context of the earthquake prediction project at Istanbul Technical University (*deprem.itu.edu.tr*) [27,28].

In the evaluation of prediction results, the metrics *Root Mean Squared Error (RMSE)* and *Coefficient of Determination* (R^2), whose formulas are specified in Table 5.1, have been used.

Table 5.1 : Formulas of the Evaluation Metrics.

Metric	Formula
R^2	$1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$
$RMSE$	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$

The PECNET framework's predictive performance has also been comparatively evaluated using Long Short-Term Memory (LSTM) units, which has emerged as a nearly de facto standard for time series forecasting [29,30]. LSTM units are an advanced type of Recurrent Neural Network (RNN) designed to overcome the limitations of traditional RNNs, particularly the vanishing gradient problem. LSTMs achieve this through the implementation of a complex architecture consisting of multiple gates (input, output, and forget gates), which regulate the flow of information. This structure enables the network to maintain long-term dependencies and efficiently process sequential data, making it highly effective for tasks that require the understanding of temporal dynamics, such as time series forecasting [31]. Therefore, it is selected as the baseline model for this study.

5.1 Comparative Experiment with LSTM on Apple Stock Data

Financial data is inherently time series in nature, characterized by its sequential order and temporal dependencies. It is in a stochastic, non-linear, and volatile form, shaped by countless micro and macroeconomic factors, as well as complexities and uncertainties stemming from human behaviors. This makes it an ideal candidate for demonstrating the efficacy of time-series algorithms. For this purpose, one of the most popular financial datasets, Apple stock prices, was used in a comparative experiment with PECNET and LSTM. Additional details regarding the data, network architectures, and performance results of this experiment are given in the subsections that follow.

5.1.1 Data overview and preprocessing

The Apple stock data to be used in the models is the historical time series of the adjusted close prices at a daily frequency, covering the period from the beginning of 2000 to September 2023.

Before being fed into models, the data is sampled using the *mean* for sampling periods of 1 and 5, followed by converting each resulting data series into sequences of 4. Each sequence is normalized internally through *average subtraction* and subsequently subjected to a maximum-level *Haar wavelet transformation*. The first zero wavelet coefficient obtained is removed, and the remaining three coefficients in the sequence are provided for the training process. In PECNET, wavelet coefficients at two separate frequencies are arranged in a hierarchical structure, and error data is processed in parallel through the same procedures and sequenced into groups of 4. For LSTM, the original sampled and sequenced data is first min-max normalized and then the one at the lower frequency is concatenated with the other one at the higher frequency. Or, as another option, both frequency bands can be given to the LSTM at the same time as separate features. Lastly, the train-test data split is performed at a ratio of 0.97.

5.1.2 Architecture and configuration of LSTM and PECNET

The LSTM model was constructed using the *Sequential()* object from *Keras* library version 2.13, featuring an architecture with one LSTM layer containing 8 neurons

followed by a hidden layer with 12 neurons that connects the input and output layers. Similarly, for the PECNET model, the current framework was used to build a network structure that comprises two hidden layers with 64 and 32 neurons, respectively. Both models were fine-tuned, using the *He* initialization method, *Adam* optimizer with a learning rate of 0.001, mean squared error as the loss function, *GELU* for activation fuction, an epoch size of 500, and a batch size of 64.

5.1.3 Evaluation metrics and comparative results

After processing the data as specified and completing the training on the established networks, the graphs of the prediction results obtained with approximately six months of test data can be shown as in Figure 5.1.

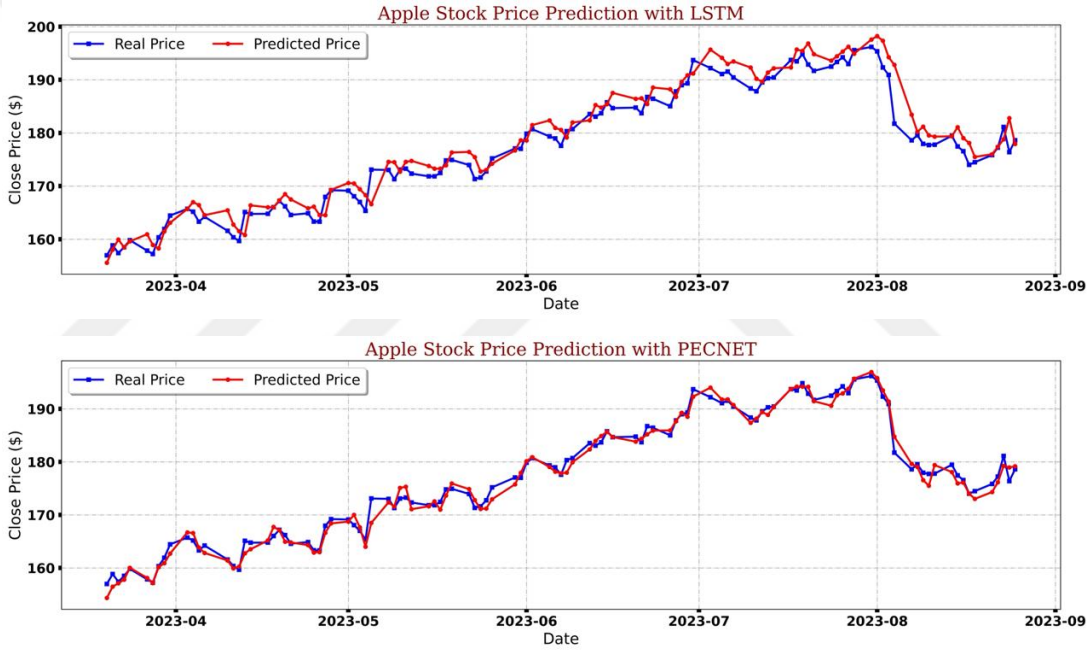

Figure 5.1 : Predictions of Apple Stock Prices with LSTM and PECNET.

The metric-based evaluation results regarding these predictions are approximately as in Table 5.2 as well.

Table 5.2 : Performance Comparison of PECNET and LSTM on Apple Stock Prices.

Metric	LSTM	PECNET
R2	0.94	0.98
RMSE	2.55$	1.24$

5.2 Comparative Experiment with LSTM on Electric Field Data

Natural phenomena data, due to its chaotic structure, complex interactions among variables, and exposure to external influences, represents a challenging type of time series data for forecasting. So, it serves as a good option for measuring the performance of prediction algorithms.

Electric Field Data (EFD) is a type of natural phenomenon time series that resembles an electrocardiogram (ECG) of the Earth and is believed to be related to rock stress. The EFD content can be used as a type of precursor for earthquakes [27].

In the subsequent sections, we will share details regarding the comparative study conducted for earthquake prediction using LSTM and PECNET, focusing on the data characteristics, network structures, and prediction outcomes.

5.2.1 Data overview and preprocessing

EFD (shortly EF) has been collected from various seismic stations across the Marmara Region for over 20 years as part of seismic risk monitoring efforts. The data subject to this study was gathered between 2002 and 2013 from the Sakarya and Yeşilköy stations, with a sampling frequency of 1 Hz. The raw appearance of the data from the Sakarya station is as shown in Figure 5.2.

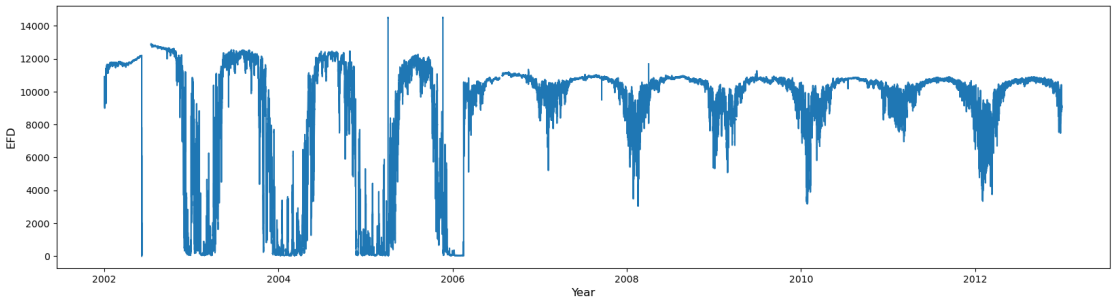

Figure 5.2 : Raw Form of Electric Field Data.

The year range and stations with the least amount of data loss have been chosen for training because EFD often contains gaps caused by station faults. In the processing of this data, initially, sampling is performed using the mean and standard deviation for periods of 22.5 minutes, 3 hours, and 1 day, which are then segmented into sequences of eight, as depicted in Figure 5.3 with T=1, to create six distinct sampled data.

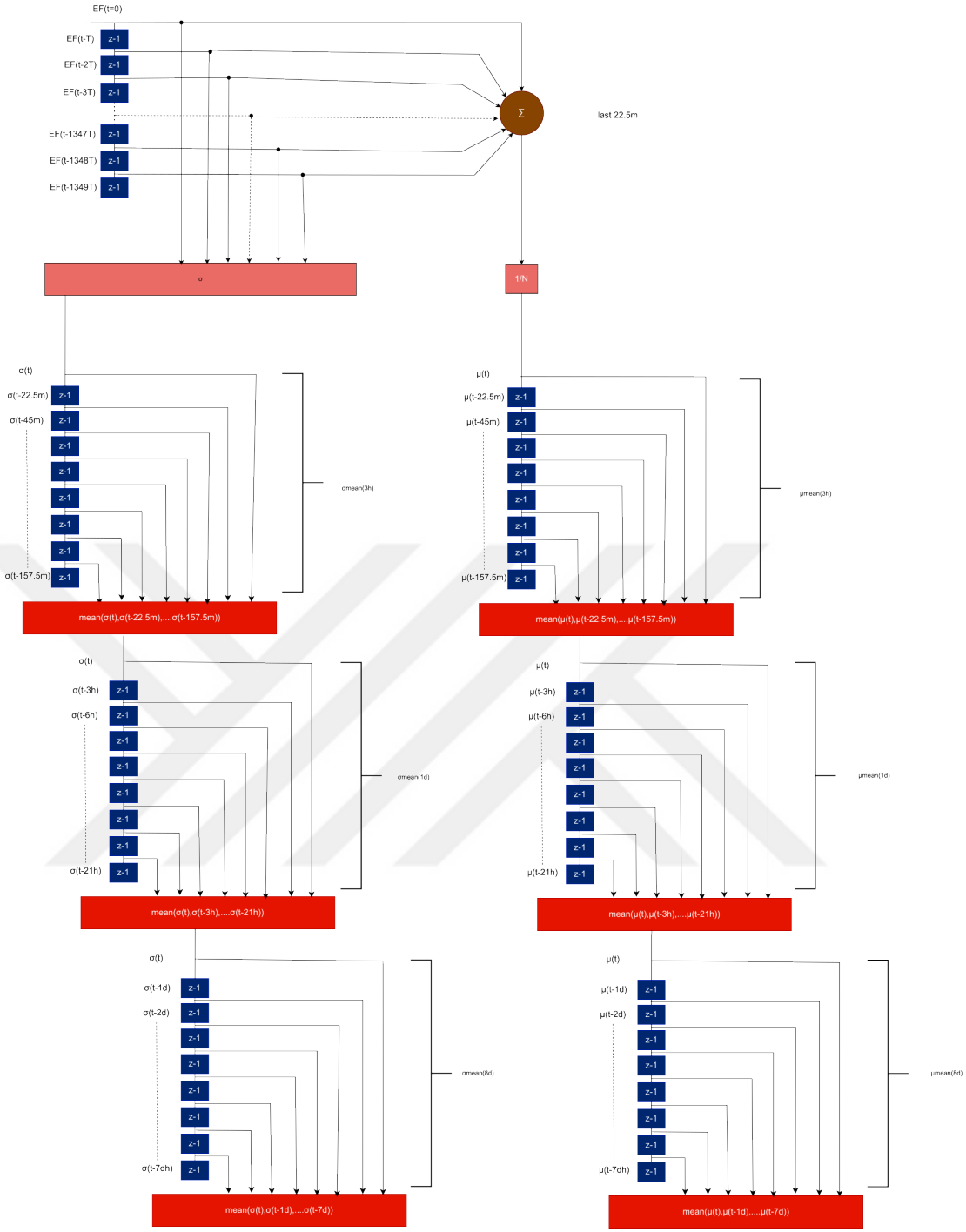

Figure 5.3 : Sampling of Electric Field Data.

For LSTM, each data sequence is min-max normalized and simultaneously fed into the network as different features. For PECNET, each sampled data sequence is internally normalized by subtracting the mean and then subjected to a maximum-level Haar wavelet transformation. The first zero wavelet coefficient is removed, and

the remaining three coefficients in the sequence are used for the training process. Subsequently, all six sampled data hierarchically organized from the lowest to the highest frequency in a pattern of mean-standard deviation, mean-standard deviation, and so forth. Similar preprocessing steps are also performed for the error data. In the final step, the train-test data split is conducted at a ratio of 0.8.

Processed EFD is used as input data in training phase. As for target data, it is derived from an earthquake catalog constructed based on the energy (E) released by earthquakes with a magnitude (M) greater than 1.5 within the Marmara Region during the years 2002-2013. Due to the substantial amount of energy released during an earthquake, to avoid excessively large exponential values during learning, the following formula in (5.1) has been used in place of the original formula for calculating energy [11,32,33]:

$$E = 2^{5.24+1.44M} \tag{5.1}$$

Due to the discrete nature of the target data, which includes many zero values, the contained energy values are not directly utilized. Instead, a transformation is applied using a weekly moving average calculated over a window size of seven units. The target data obtained after this process is depicted in Figure 5.4.

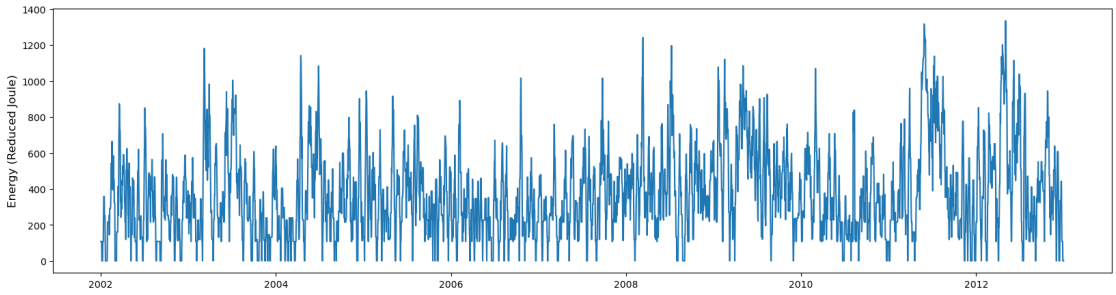

Figure 5.4 : Seismic Energy Release Data.

Consequently, we will attempt to predict the average energy for the upcoming week using six separate EFD datasets, which together cover a maximum of eight days.

5.2.2 Architecture and configuration of LSTM and PECNET

The LSTM architecture for Electric Field (EF) - Energy Release (ER) prediction task consists of four stacked LSTM layers placed between the input and output layers, with

32, 64, 128, and 64 neurons, respectively. Each LSTM layer is followed by a dropout layer, with the first having a dropout rate of 0.1 and the subsequent three layers each having a dropout rate of 0.2. The dropout layers are employed to enhance the model's generalization capabilities. For the same task, the PECNET model is configured with two hidden layers, containing 64 and 32 neurons, respectively, placed between the input and output layers. For both models, the hyperparameters are kept constant with those used in financial data prediction task.

After completing the data processing steps and the configuration of the network system, the structure of the resulting PECNET model can be illustrated as shown in Figure 5.5 [11].

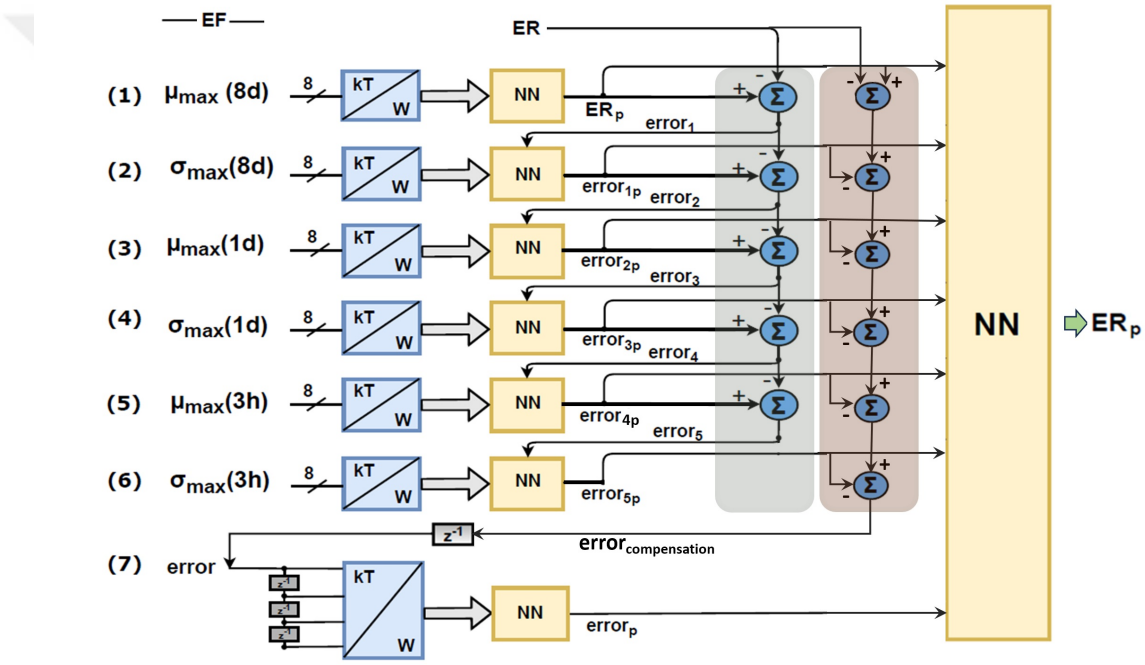

Figure 5.5 : PECNET Configuration for EF-ER Prediction.

5.2.3 Evaluation metrics and comparative results

The LSTM and PECNET models, constructed to predict ER using EF with the aforementioned architectures, yield results approximately as shown in Table 5.3 in terms of R2 and RMSE evaluation metrics for the Sakarya and Yeşilyurt test data spanning over two years.

Table 5.3 : Performance Comparison of PECNET and LSTM on EF-ER Prediction.

Station	PECNET		LSTM	
	R2	RMSE	R2	RMSE
Sakarya	0.50 - 0.54	130 - 145J	0.29	300 - 310J
Yeşilyurt	0.50 - 0.52	135 - 150J	0.21	385 - 400J

The graphical representation of the predictions made by both models on this test data is also shown in Figure 5.6 for Sakarya station.

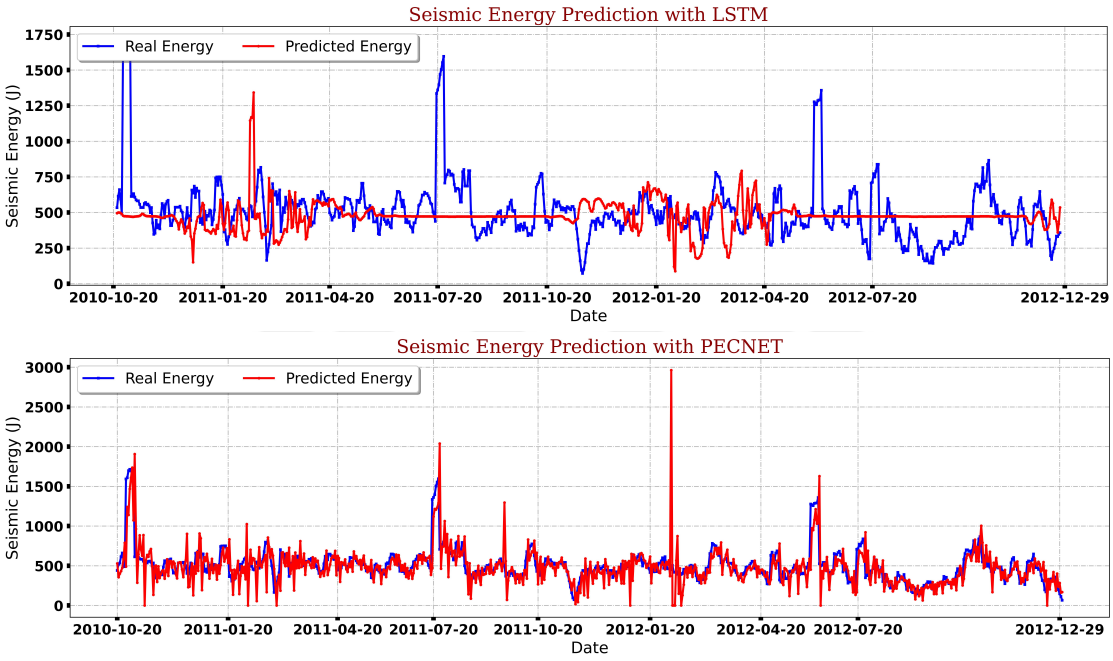

Figure 5.6 : Predictions of Energy Release (ER) with LSTM and PECNET.

In addition, specific to PECNET, the results of enhancing the existing network by incorporating the target data as input and by connecting the Sakarya-Yeşilyurt stations' data via data fusion at the end of the network are also presented in Table 5.4.

Table 5.4 : Evaluation of Data Fusion Results in PECNET.

Metric	PECNET	
	EF+ER	EF+EF
R2	0.61	0.55
RMSE	125- 138J	128 - 142J

5.3 Discussion and Interpretation of Results

In the context of financial forecasting, although both models are considered successful, PECNET has produced a higher RMSE and R2 score and has more accurately identified the downward and upward shifts in the overall trend.

In the context of seismic energy prediction, which requires more extensive processing and configuration than financial forecasting, the PECNET framework successfully trained cascaded networks on a single data type in three different time windows using two statistical sampling methods (mean and standard deviation). Furthermore, it made it easier to train cascaded networks by including another data type, whether with the same or different properties (data fusion). The results showed that the optimised LSTM model could not outperform an R2 level of 0.3 on the same dataset. In contrast, the architecture built with the PECNET framework produced performance levels ranging from 0.5 to 0.60 R2. Additionally, as the data characteristics become multivariate or as the volume of data increases, the models exhibit signs of overfitting but by adding the same or different types of data to the end of PECNET's cascaded networks using the framework, it was observed that data fusion resulted in an improvement in prediction performance.

The superior performance of the PECNET could be attributed to its ability to capture both temporal patterns and frequency-domain characteristics through its wavelet transformation and also to its capability of refining the predictions based on error analysis iteratively. Additionally, the use of adaptive normalization and a hierarchical iterative learning method with data at various frequencies helps it effectively handle complex relationships and non-stationarity in time series data.

6. CONCLUSIONS

The PECNET model has introduced solutions to challenges faced by existing models in time series predictions. The operational challenges arising from the interaction of different network structures and data flows within PECNET have been addressed by the developed framework. This framework has been tested on various types of data and demonstrated high performance. Subsequently, it has been seamlessly integrated into real-time web applications. Finally, to facilitate its application and testing in diverse problems, the application codes have been shared with other users via GitHub, and the library version is available on Python's pip repository. (see links at Appendix A)

6.1 Implications of the Study

The study has successfully demonstrated that PECNET, when encapsulated within a dedicated framework, can overcome its inherent complexities and operational challenges. Some important results are that the framework can make it easier to set up cascaded networks, manage data transfers and preprocessing efficiently, synchronise time shifts accurately, and integrate data fusion mechanisms easily. The framework has also proven to be effective in reducing computational power and time costs while maintaining high prediction accuracy, making it an important tool for practitioners with different levels of programming expertise.

For the first time in this study, PECNET has been turned into ready-to-use framework software that makes it easier to use for different types of time series predictions. This means that other developers can use this robust model against over-fitting in real-time machine learning systems without the need for specific coding knowledge of PECNET. The ability to adapt to different data sets without code-level changes is a significant distinction because making architectural changes and designing data flows at the code level in systems with high complexity ratios not only significantly increases testing and debugging time but also reduces reliability.

The developed framework has also been integrated into the earthquake prediction research project portal (`http://deprem.itu.edu.tr`) for real-time operation, enabling continuous learning-based predictions for seismic risk assessment. A similar integration has been implemented for online stock price prediction in markets such as BIST and NASDAQ, and it is being shared with users through the website `cashspeeder.com`.

6.2 Limitations and Future Research Directions

One of the primary limitations of the PECNET framework is that it needs past data with a certain number of samples at different frequency levels to make a prediction. Discontinuities within these sample sets can adversely impact prediction accuracy. To address this, the development of an advanced "Imputators" class is planned to fill small gaps in the data without distorting its intrinsic characteristics.

The PECNET framework currently operates on a two-layer neural network architecture, prioritising simplicity and performance. However, in scenarios where resource constraints are not a limiting factor and capturing long-range dependencies in large data sets is crucial, integrating a sample transformer-based machine learning model into the PECNET framework's model options is contemplated. This update would offer an alternative approach for generating superior outputs in such cases.

The PECNET framework significantly simplifies the data fusion process, but the decision on when and which data to fuse is left to the user's decision based on correlation calculations. A future enhancement involves making this selection process functional and automatic, enabling the framework to determine the order and amount of data to be fused.

The data preprocessing module of PECNET framework will be expanded to include a wider range of selectable statistical parameters for sampling.

Lastly, the PECNET framework will be benchmarked against newly developed state-of-the-art models such as TSMixer, AutoTS, TimeGPT, and Temporal Fusion

Transformers using current benchmark datasets. This comparison will showcase PECNET's high performance and establish its place in the field.

REFERENCES

[1] **Hyndman, R.J. and Athanasopoulos, G.** (2020). *Forecasting: principles and practice*, OTexts, 3. edition, https://otexts.com/fpp3/.

[2] **Ustundag, B.B. and Kulaglic, A.** (2020). High-Performance Time Series Prediction with Predictive Error Compensated Neural Networks, *IEEE Access*, *8*.

[3] **Brunton, S.L.**, **Proctor, J.L. and Kutz, J.N.** (2016). Discovering governing equations from data: Sparse identification of nonlinear dynamical systems, *Proceedings of the National Academy of Sciences*, *113*(15), 3932–3937.

[4] **Goodfellow, I.J.**, **Bengio, Y. and Courville, A.** (2016). Deep Learning, chapter: Machine Learning Basics, MIT Press, http://www.deeplearningbook.org.

[5] **Gurrapu, S. and Yadav, S.** (2020). Limitations and Challenges of Facebook Prophet for Time Series Forecasting, *International Journal of Computer Applications*, *179*(43), 15–21.

[6] **Ahmed, R. and Mahmoud, M.** (2016). The Limitations of ARIMA Models in Time Series Forecasting: A Comprehensive Review, *International Journal of Advanced Computer Science and Applications*, *7*(3), 256–265.

[7] **Capone, V.**, **Iannuzzo, G. and Camastra, F.** (2023). Deep Learning for Time Series Forecasting: Advances and Open Problems, *Information*, *14*(11), 598.

[8] **Jain, A. and Mao, J.** (1996). Artificial Neural Networks: A Tutorial, *Computer*, *29*.

[9] **Kulaglic, A. and Ustundag, B.B.** (2021). Predictive Error Compensating Wavelet Neural Network Model for Multivariable Time Series Prediction, *TEM Journal*, *10*.

[10] **Kulaglic, A. and Ustundag, B.B.** (2021). Stock Price Prediction Using Predictive Error Compensation Wavelet Neural Networks, *Computers, Materials Continua*.

[11] **Macit, S. and Ustundag, B.B.** (2023). Performance Improvement in Time Series Prediction through PECNET Framework, *International Conference on Machine Learning and Applications*, Florida, USA.

[12] **Huang, L.** *et al.* (2023). Normalization Techniques in Training DNNs: Methodology, Analysis and Application, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *45*(8), 10173–10196.

[13] **Ogasawara, E.** *et al.* (2010). Adaptive Normalization: A novel data normalization approach for non-stationary time series, *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pp.1–8.

[14] **Tanaka, T.** *et al.* (2022). Sliding-Window Normalization to Improve the Performance of Machine-Learning Models for Real-Time Motion Prediction Using Electromyography, *Sensors*, *22*(13), `https://www.mdpi.com/1424-8220/22/13/5005`.

[15] **Chun-Lin, L.** (2010). A tutorial of the wavelet transform, *NTUEE*.

[16] **Majid, F.** `https://users.math.yale.edu/pub/wavelets/software/xwpl/html/manual/node28.html`, date retrieved: 15.04.2024.

[17] **Meng, T.** *et al.* (2020). A survey on machine learning for data fusion, *Information Fusion*, *57*, 115–129.

[18] **Edwin, N.M.** (2014). Software frameworks, architectural and design patterns, *Journal of Software Engineering and Applications*, *2014*.

[19] **Washizaki, H.** *et al.* (2020). Machine learning architecture and design patterns, *IEEE Software*, *8*, 2020.

[20] **Mallat, S.G.** (1989). A theory for multiresolution signal decomposition: the wavelet representation, *IEEE transactions on pattern analysis and machine intelligence*, *11*(7), 674–693.

[21] **Hendrycks, D. and Gimpel, K.** (2016). Gaussian error linear units (gelus), *arXiv preprint arXiv:1606.08415*.

[22] **He, K.** *et al.* (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, *Proceedings of the IEEE international conference on computer vision*, pp.1026–1034.

[23] **Kingma, D.P. and Ba, J.** (2015). Adam: A Method for Stochastic Optimization, *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*.

[24] **comp.ai.neural-nets**. `http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-9.html`, date retrieved: 15.03.2024.

[25] **Patterson, D.W.** (1998). *Artificial neural networks: theory and applications*, Prentice Hall PTR.

[26] **Heaton, J.** `https://www.heatonresearch.com/2017/06/01/hidden-layers.html`, date retrieved: 15.03.2024.

[27] **Ustundag, B.B.** *et al.* (2005). Multilayer capacitor model of the Earth's Upper Crust, *Elektrik - Turkish Journal of Electrical Engineering and Computer Sciences*, *13*(1), 163–173.

[28] **Bagis, S. and Ustundag, B.B.** (2009). *Real Time Earthquake Risk Computation*.

[29] **Sezer, O.B.**, **Gudelek, M.U. and Ozbayoglu, A.M.** (2020). Financial time series forecasting with deep learning: A systematic literature review: 2005–2019, *Applied soft computing*, *90*, 106181.

[30] **Chandra, R.** *et al.* (2021). Evaluation of Deep Learning Models for Multi-Step Ahead Time Series Prediction, *IEEE Access*, *9*, 83105–83123.

[31] **Hochreiter, S. and Schmidhuber, J.** (1997). Long Short-Term Memory, *Neural Computation*, *9*(8), 1735–1780.

[32] **Gutenberg, B. and Richter, C.** (1942). Earthquake Magnitude, Intensity, Energy, and Acceleration (First Paper), *Bulletin of the Seismological Society of America*, *32*(3).

[33] **Gutenberg, B. and Richter, C.** (1944). Earthquake Magnitude, Intensity, Energy, and Acceleration (Second Paper), *Bulletin of the Seismological Society of America*, *46*(2).

APPENDICES

APPENDIX A : Useful PECNET Framework Links

APPENDIX A : Useful PECNET Framework Links

1. **`https://github.com/pecnet/pecnetframework` :** The PECNET framework's GitHub repository link provides access to all code files, related tutorials, application-level issues and their solutions, current merge requests, and more.

2. **`https://pypi.org/project/pecnet` :** This is the Python Package Index website that hosts the ready-to-install package of the framework. It contains the appropriate file type of the framework to be installed according to the installation environment, previous versions of the framework, necessary dependencies for installation, and important notes, alerts, and other information.

3. **`https://pecnet.readthedocs.io` :** This is the website where you can find the general reference documentation for the classes and functions within the framework.

4. **`https://pywavelets.readthedocs.io/en/latest/regression/wavelet.html` :** The PECNET framework utilizes the PyWavelet library for wavelet transformation. This link summarizes all supported types and parametric uses for the wavelet transformation.

5. **`https://pytorch.org/get-started/locally/` :** This website provides basic information on how to configure and manually install the PyTorch deep learning library according to the operating system, package manager and GPU platform being used.

6. **`https://cashspeeder.com` :** This is the website where daily price predictions for various global stocks are made and presented using the PECNET framework.

7. **`https://yerdurumu.com` :** It is a website where various data about the Earth, such as EFD, are collected in real time from stations installed in various locations in Turkey, visualized and interpreted by PECNET for earthquake prediction efforts.

CURRICULUM VITAE

Name SURNAME: Serkan MACİT

EDUCATION:

- **B.Sc.:** 2012, Kocaeli University, Faculty of Engineering, Department of Computer Engineering
- **M.Sc.:** Candidate for 2024, Istanbul Technical University, Faculty of Computer and Informatics Engineering, Department of Computer Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2012-2022 Governmental Institutions & Companies as a Software Developer & Researcher
- 2022- National Software and Certification Central at the Istanbul Technical University.

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Macit S.**, Üstündağ B.B. (2023). Performance Improvement in Time Series Prediction through PECNET Framework. *International Conference on Machine Learning and Applications*, December 15-17, 2023 Florida, USA.

OTHER PUBLICATIONS, PRESENTATIONS AND PATENTS:

- Bostanci R., Erguder L., **Macit S.**, Sayar A. (2011). An Implementation of Image Processing Web-Services for Distributed Systems. *International Science and Technology Conference (ISTEC)*, December 7-9, 2011 Istanbul, TURKEY.