Comparative Study of the PatchTST and DLinear Models for Forecasting Hourly and Daily Energy Consumption

Michael Giannoulis

GRANT THORNTON GREECE

Table of Contents

Abstract

This study presents a comparative evaluation of two state-of-the-art time series forecasting models—**PatchTST** and **DLinear**—for predicting energy consumption at hourly and daily granularities. Using real-world multivariate datasets derived from benchmark energy usage records, both models were trained and assessed under five distinct preprocessing configurations, incorporating temporal feature engineering and error-based channel filtering. The analysis employed standard evaluation metrics (MAE, RMSE, R^2, SMAPE) and a consistent experimental framework to ensure fair comparison.

Results indicate that **PatchTST** consistently **outperforms DLinear** across most scenarios, particularly in high-variance and short-term forecasting contexts, due to its ability to model complex temporal dependencies through attention-based mechanisms. DLinear, while computationally efficient and interpretable, exhibits limitations in capturing nonlinear patterns, especially in volatile series.

The findings offer practical insights for selecting forecasting models based on data complexity, temporal resolution, and deployment constraints.

1. Introduction

Energy consumption is the primary contributors to overall electricity demand on both national and global scales. With increasing population, intensive appliance usage, and the rise of smart energy systems, the need for **accurate energy consumption forecasting** has become more critical than ever (Chou & Tran, 2018).

Energy demand forecasting is typically based on **time series data**, which captures the evolution of a variable over time. Traditional statistical methods such as ARIMA have long been used to model stationary or seasonal behaviors in time series. However, the increasing complexity and dynamic nature of real-world data call for more **robust and flexible predictive models**, including **deep learning architectures** like LSTM and GRU, and more recently, **transformer-based models** (Amalou et al., 2022).

Two state-of-the-art models in time series forecasting are **PatchTST**, which adopts a patch-based tokenization mechanism inspired by vision transformers and employs temporal attention, and **DLinear**, a lightweight linear model based on MLPs designed for efficient and accurate forecasting over short and medium horizons (Li et al., 2023; Gong et al., 2023).

2. Problem Statement

While both models have demonstrated strong performance in benchmark datasets, there is a lack of comparative studies focused specifically on **residential energy consumption forecasting** using real-world datasets and across **multiple time resolutions** (e.g., hourly and daily). It remains unclear which model performs better depending on the type of time series and the prediction horizon.

3. Objectives

The primary aim of this thesis is to **comparatively evaluate** the performance of PatchTST and DLinear in the context of forecasting residential energy usage. The main objectives are:

- To collect and preprocess real-world residential energy consumption data
- To implement and train both models on different time granularities (hourly and daily)
- To evaluate performance using established metrics such as MAE, RMSE, and SMAPE
- To draw conclusions on model suitability and performance trade-offs

4. Related Works

Time series forecasting has been extensively studied across disciplines, with early research dominated by statistical methods such as **ARIMA**, **SARIMA**, and exponential smoothing. These models, while effective for stationary and seasonal data, often fail to capture the nonlinear and dynamic patterns present in modern energy consumption datasets (Amalou et al., 2022).

As computational capabilities and data availability have advanced, machine learning approaches—particularly deep learning—have emerged as powerful alternatives. Recurrent neural networks (RNNs) and their variants, such as **LSTM** and **GRU**, have been successfully applied to energy consumption forecasting problems by modeling long-term dependencies and temporal structures in univariate and multivariate series (Bhoj & Bhadoria, 2022; Tayib, 2023).

Recently, **transformer-based architectures** have gained popularity in time series forecasting due to their ability to parallelize training and effectively model long-range dependencies without recurrence. Notably, **PatchTST** (Gong et al., 2023) introduces a patch-based tokenization strategy inspired by vision transformers, allowing it to learn local temporal patterns efficiently. It has achieved strong benchmark performance across several public datasets.

On the other hand, **DLinear** (Zeng et al., 2022) proposes a minimalist approach that leverages linear layers to decouple trend and seasonal components in time series, offering competitive performance with significantly lower computational cost. Its success has inspired further exploration of linear modeling frameworks such as **MLinear** (Li et al., 2023), which directly compares DLinear with transformer-based models including PatchTST, focusing on architecture-level distinctions.

However, most existing comparative studies—such as those by Li et al. (2023) and Gong et al. (2023)—evaluate models primarily on **synthetic or benchmark datasets** (e.g., ETTm2, ILI, Weather), with limited attention to **real-world, domain-specific contexts** like residential energy consumption. Furthermore, the time resolutions explored are often fixed, overlooking the practical differences between **hourly and daily energy demand dynamics**.

Several works have addressed forecasting in smart buildings and residential environments using machine learning. Chou & Tran (2018) applied support vector regression and tree-based methods for 1-day-ahead energy prediction based on household patterns. Cascone et al. (2023) employed convolutional LSTM models to capture appliance-level energy fluctuations. Yet, none of these studies directly investigate the performance of transformer-based models versus linear models in this context.

This thesis fills that gap by offering a focused, empirical comparison between **PatchTST and DLinear** in forecasting **residential energy usage** at **multiple time resolutions**. To our knowledge, this is the first application of these models to **real residential datasets** evaluated in both **hourly and daily granularities**, aiming to inform model selection in practical deployment scenarios.

5. Theoretical Background

5.1.Definition of Time Series

A time series is a collection of observations made sequentially in time. Examples occur in varieties of fields, ranging from economics to engineering and methods of analyzing time series constitute an important area of statistics.

5.2.A more technical definition of time series

A time series $X=[x_0,x_1,...,x_n]$, $x_i \in R$ is a sequence of $I \in N$ real values of a synthetic or a real process. In this definition, I is also called the number of data steps or the length of the time series X, and xi are the data points or time points.

5.3.Time Series Data Set

A time series dataset D consists of $n \in N$ time series of equal length I that are recorded from a single process. If the time series are not of equal length, it is common to pad with zeroes or resampling to bring them to equal length.

5.4.Univariate Time Series

A **univariate time series** refers to a sequence of observations of a single variable collected over time at regular intervals. Formally, it can be represented as $\{X_t\}_{t=1}^{T}$, where $Xt \in R$ denotes the value of the variable at time t, and T is the total number of time points.

Univariate time series analysis focuses on modeling the temporal dependence within a single variable, often with the aim of understanding its structure or forecasting its future values.

5.5.Multivariate Time Series

A **multivariate time series** involves observations of two or more interrelated variables recorded simultaneously over time. It is typically represented as

$$\{X_t\}_{t=1}^{T}$$

, where

$$Xt = [Xt(1), Xt(2), \dots, Xt(n)]\top \in Rn$$

Multivariate time series analysis aims to capture both the temporal dynamics within each individual variable and the dependencies or interactions among multiple variables. This approach is particularly useful when variables influence each other over time.

5.6. Categories of Time Series

5.6.1. Stationary Time Series

A stationary time series is one where the variance in data is constant over different periods of time. The data is auto correlational, and the difference between data points follows a static pattern. A stationary time series just follows

5.6.2. Non-Stationary Time Series

Non-stationary time series is a pattern or trend in the variance of data over a period. Unlike a stationary time series, where time variance is constant over a period of time, the non-stationary time series shows a fluctuation in the time series.

5.6.3. Trends in Time Series

A trend exists when there is a long-term increase or decrease in the data. It does not have to be linear; sometimes we will refer to a trend as changing direction, when it go from an increasing trend to a decreasing trend.

5.6.4. Seasonality in Time Series

A seasonal pattern occurs when time is affected by seasonal factors such as the time of the year or the day of the week. Seasonality is always of a fixed and known frequency.

5.6.5. Cyclic Time Series

A cycle occurs when the data exhibits rise and falls that are not a fixed frequency. These fluctuations are usually due to the economic conditions and are often related to the “business cycle”. The duration of these fluctuations is usually at least two years.

5.6.6. Deterministic Time Series

A Deterministic Time Series follows a fixed pattern or rule that allows future values to be predicted exactly if the underlying model is known. These series have no randomness. Once you know the initial conditions and the governing equation, all the future points are fully determined.

5.6.7. Stochastic Time Series

A Stochastic time series incorporates randomness or uncertainty meaning future values cannot be predicted, with certainty- only probabilistically. These series often arise from real-world processes influenced by unpredictable factors.

5.7. Definitions of important elements in Time Series

5.7.1. Stationarity

The property where the statistical characteristics like mean, variance and autocorrelation of the data remain constant over time.

5.7.2. Seasonality

Regular, recurring fluctuations in the data occur within a specific period of time.

5.7.3. Autocorrelation

Measures the linear relationship between lagged values of time series.

For a **finite sample** $\{x_1,x_2,...,x_N\}$, the **sample autocorrelation** at lag k is estimated as:

$$\rho_k = \frac{E[(X_t - \mu)(X_{t-k} - \mu)]}{\sigma^2}$$

Where:

- ρ_k is the autocorrelation at lag **k**
- X_t is the value of the time series at time **t**
- X_{t-k} is the value at lag **k**
- **μ** is the mean of the time series
- σ^2 is the variance of the time series
- **E[·]** denotes the expected value (mean)

5.8. Time Series Models

5.8.1. AR

In autoregression models, the output is the future data point expression expressed as a linear combination of the past p data points, p is the number of logs included in the equation. An AR(L) model is defined mathematically as:

$X_t = \delta + \varphi_1\chi_{t-1} + \alpha_1$

- X_t – the value of the time series at time **t** (what you're trying to predict)
- **δ** – a constant number that shifts the whole series up or down
- φ_1 – a number that tells how much of the previous value (X_{t-1}) affects the current one
- X_{t-1} – the value at the previous time step
- α_t – random noise or error (something unpredictable)

5.8.2. Vector Autoregressive model

Vector Autoregressive models to allow the autoregression of multi variate series. They are structured so that each variable is a linear function of past lags of itself and past lags of the other variables.

5.8.3. Moving Average Model

Rather than using past values of the forecast variable in a regression, a moving average model uses past forecast errors in a regression like model.

The moving average model is expressed in the below mathematical function:

$$X_t=\mu+\varepsilon_t+\theta_1\varepsilon_{t-1}+\theta_2\varepsilon_{t-2}+\cdots+\theta_q\varepsilon_{t-q}$$

Where:

- $\mathbf{X_t}$: the value of the time series at time **t**
- **μ**: the mean (constant term)
- $\boldsymbol{\varepsilon_t}$: white noise (random error at time **t**), with mean 0 and constant variance
- $\boldsymbol{\theta_1,\theta_2,\ldots,\theta_q}$: **MA coefficients** that measure the impact of past error terms
- **q**: the **order** of the MA model (how many past errors are included)

5.8.4. ARMA

The ARMA model comes from the merging of two simpler models, the AR and the MA models. In the ARMA process, a time series is said to follow an ARMA(p,q), if it satisfies:

$$X_t = \mu + \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$$

Where:

- $\mathbf{X_t}$: value of the time series at time **t**
- **μ**: constant mean term
- $\boldsymbol{\phi_i}$: autoregressive coefficients (for lagged values of the series)
- $\boldsymbol{\theta_j}$: moving average coefficients (for lagged error terms)
- $\boldsymbol{\varepsilon_t}$: white noise (error term) at time **t**, assumed to be d with mean 0 and constant variance
- **p**: number of AR lags
- **q**: number of MA lags

5.8.5. ARIMA model

Box and Jenkins in 1970 introduced the ARIMA model. It also referred to as Box-Jenkins methodology composed a set of activities for identifying, estimating and

diagnosing ARIMA models with time series data. The model is the most prominent methods in financial forecasting.

ARIMA models have efficient capabilities to generate short-term forecasts. It constantly outperformed complex structural models in short-term prediction, the future value of a variable is a linear combination of past values and past errors expressed as follows:

$$\Delta^d X_t = \mu + \sum_{i=1}^{p} \phi_i \Delta^d X_{t-i} + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$$

- **Xt**: observed time series at time **t**
- **Δ^d**: the **d-th difference** of Xt, defined as:

$\Delta^d X_t = (1-B)^d X_t$

- where B is the backshift operator
- **μ**: mean term (optional in some formulations)
- **ϕ_i**: coefficients of the **autoregressive (AR)** part
- **θ_j**: coefficients of the **moving average (MA)** part
- **ε_t** : white noise error at time **t**
- **p**: number of AR lags
- **d**: number of differences to make the series stationary
- **q**: number of MA lags

5.8.6. RNN-Recurrent neural network

It is a class of supervised machine learning models, which takes sequenced data as the input, they differ from other machine learning model architecture by recurrent connection which means the output data of a cell is also related to the output of the previous cell.

More specifically the network will memorize the information from the previous output, so it can be added in the calculation of the current result.

It can be expressed mathematically as the function below:

$H_t = \sigma (Wx_t + Uh_{t-1} + b)$

In this equation $x_t \in R$ and $ht \in R$ are respectively the input and the hidden state of the cell in time t, $W \in R$ and $U \in R$ are the weights of the matrices of the recurrent network that will memorize the information.

5.8.7. LSTM - Long Short Term Memory

Its is one of the architectures of RNN, that was proposed to improve the cell 's memorizing capacity by introducing the notion of a "gate" in the cell, and since the

announcements of LSTMs we have had several developed versions of LSTM, the most known of these version is the LSTMs with the forged gate which was added to allow a cell to rest and free up internal resources.

The LSTM architecture consists of input, forget, and output gates, along with a cell state that is updated through specific functions. A widely cited paper by Sherstinsky (2020) explains how LSTM equations evolve from classical differential equations. It formalizes the update rules as:

- Forget gate: $f_t=\sigma(W_f x_t+U_f h_{t-1}+b_f)$
- Input gate: $i_t=\sigma(W_i x_t+U_i h_{t-1}+b_i)$
- Candidate values: $c_t=\tanh(W_c x_t+U_c h_{t-1}+b_c)$
- Cell state: $c_t=f_t\odot c_{t-1}+i_t\odot c_t$
- Output gate: $o_t=\sigma(W_o x_t+U_o h_{t-1}+b_o)$
- Hidden state: $h_t=o_t\odot\tanh(c_t)$

5.8.8. GRU- Gated Recurrent Unit

The Gated Recurrent Unit (GRU) was introduced by *Cho et al.* to address the **vanishing gradient problem** commonly encountered in traditional Recurrent Neural Networks (RNNs), while also offering a simpler architecture compared to Long Short-Term Memory (LSTM) networks. GRUs were designed to reduce computational complexity by decreasing the number of parameters to be trained.

To achieve this simplification, GRU cells merge the **input gate** and **forget gate** of the LSTM into a single **update gate**. This update gate controls how much of the past information should be retained and how much of the new input should be incorporated. Additionally, the **reset gate** determines how much of the previous hidden state should be forgotten. These mechanisms enable GRUs to capture temporal dependencies effectively with fewer gates and operations than LSTMs.

The functioning of a GRU cell is governed by the following equations:

- **Update gate:** $z_t=\sigma(W_z x_t+U_z h_{t-1}+b_z)$
- **Reset gate:** $r_t=\sigma(W_r x_t+U_r h_{t-1}+b_r)$
- **Candidate activation:** $h_t=\tanh(W_h x_t+U_h(r_t\odot h_{t-1})+b_h)$
- Final activation (hidden state): $ht=(1-z_t)\odot h_{t-1}+z_t\odot h_t$

Here, **σ** denotes the sigmoid function, tanh is the hyperbolic tangent, x_t is the input at time step *t*, h_{t-1} is the previous hidden state, and ⊙ represents element-wise multiplication.

5.8.9. Vanishing Gradient Problem

It is common issue in training deep neural networks, RNNs and their variants like LSTM and GRU. To be specific, it occurs when the gradient used for updating network weight becomes extremely small as the are propagated backwards through many layers or time during propagation

An example in standard RNNs, is when trying to learn form sequences, for example time series, the network struggles to retain information across many steps due to the vanishing gradient, as a result it may only capture short-term dependencies

5.8.10. Transformers and Time Series Forecasting

The Transformer architecture, originally introduced for sequence modeling tasks in NLP, has been increasingly adopted for time series forecasting due to its ability to capture long-range temporal dependencies through self-attention mechanisms. Unlike Recurrent Neural Networks (RNNs) and their gated variants (LSTM, GRU), which rely on sequential computation, Transformers enable parallel processing and direct modeling of inter-temporal relationships, significantly enhancing scalability and expressiveness.

Formally, given a time series input sequence $X=\{x_1,x_2,...,x_T\}$, a Transformer maps this input to a series of latent representations via a multi-head self-attention mechanism defined as:

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

where $Q,K,V \in R^{T\times dk}$, are query, key, and value matrices derived from the input embeddings. This allows each position in the sequence to attend to every other position, capturing global temporal correlations.

However, vanilla Transformers are not inherently optimized for time series data due to the lack of inductive biases such as trend, seasonality, and local smoothness. Consequently, several Transformer variants have been proposed to address these limitations:

- **Informer** (Zhou et al. 2021) introduces a **ProbSparse self-attention** mechanism to reduce computational complexity from $O(T^2)$ to $O(TlogT)$, along with a **distillation module** for long-sequence forecasting.
- **ETSformer** (Woo et al. 2022) integrates classical exponential smoothing within the Transformer framework, decomposing input time series into level, trend, and seasonal components and applying specialized attention to each.
- **iTransformer** (Liu et al. 2023) proposes an inverted architecture that improves token interaction modeling by inverting positional encoding flows and leveraging learned priors on time dependency structures.

- **TFT (Temporal Fusion Transformer)** (Lim et al. 2021) incorporates static covariates, multi-horizon attention, and gating layers, making it highly expressive for real-world time series with exogenous variables.

Transformer-based models are particularly suitable for **long sequence time series forecasting (LSTF)** due to their global receptive fields. However, they often struggle with generalization when training data is limited or when the temporal dynamics are highly non-stationary. To address this, architectures like **PatchTST** and **DLinear** have emerged

5.8.11. PatchTST

PatchTST (Patch Time Series Transformer) is a recent innovation in Transformer-based time series forecasting that adapts the success of *patching strategies* from computer vision (as seen in Vision Transformers, ViTs) to sequential modeling. Rather than processing time steps individually, PatchTST partitions the input time series into fixed-length, non-overlapping **patches**, which are then treated as tokens in the Transformer encoder. This approach enhances local context modeling and mitigates the over-smoothing effects commonly seen in vanilla self-attention mechanisms.

Given a multivariate time series $X \in R^{L \times C}$, where L is the sequence length and C the number of channels (features), PatchTST constructs patches of length P, leading to $N = \lfloor L/P \rfloor$ patches. Each patch is projected via a learnable linear mapping into a latent vector space, followed by **positional encoding** $W_{pos} \in R^{D \times N}$ to retain temporal ordering:

$$Z = \text{Embed}(X_{patches}) + W_{pos}$$

The Transformer encoder then processes this patch sequence using standard multi-head self-attention layers. The model is trained to forecast the future time steps using either an autoregressive head or a direct projection of the latent states to future values, depending on the use case.

PatchTST demonstrates several key advantages:

- **Reduced sequence length** in attention (via patching) improves computational efficiency.
- **Local pattern modeling** is enhanced as each patch aggregates short-term context.
- **Parallelism** across patches enables faster training and inference.

Experimental results in Nie et al. (2022) show that PatchTST significantly outperforms previous Transformer variants such as Informer and Autoformer on long-sequence forecasting benchmarks, particularly for high-resolution data. The model's simplicity—relying only on vanilla Transformer blocks with patch-wise inputs—makes it both performant and easy to implement across univariate and multivariate settings.

Moreover, studies like Ni et al. (2024) and Huang et al. (2024) have extended PatchTST for domains such as heart rate prediction and ocean wave modeling, validating its generalization across different forecasting contexts.

In the context of my dissertation —*hourly and daily data consumption forecasting*— PatchTST offers the ability to model both short-term fluctuations and longer seasonal trends efficiently. Its patching mechanism is particularly well-suited for capturing localized consumption patterns, making it an ideal candidate for comparison with decomposition-based models like DLinear.

5.8.12. DLinear

DLinear (Decomposition-Linear) is a lightweight and efficient model designed for long-term time series forecasting. It stands in contrast to attention-heavy architectures like Transformers by emphasizing interpretability, inductive bias, and architectural simplicity. The core principle behind DLinear is series decomposition, which isolates different temporal components—typically trend and seasonality—before applying linear projection for forecasting.

The model decomposes the input time series $X \in R^{L \times C}$, where L is the sequence length and C is the number of channels, using a moving average filter or kernel smoothing function:

$X=T+S$

where:

- T: trend component, extracted by averaging over a window.
- S: seasonal or residual component, obtained by subtracting the trend from the original signal.

Each component is then passed through separate linear layers:

$\hat{T}=W_T T+b_T$,

$\hat{S}=W_S S+b_S$,

$\hat{Y}=\hat{T}+\hat{S}$

These projections are performed independently per channel, which enables the model to preserve and learn individual component dynamics. The decoupled nature of the architecture allows DLinear to avoid the complexities and training instabilities of nonlinear models, while achieving strong baseline performance on long-horizon forecasting tasks.

Despite its simplicity, DLinear has shown strong competitive performance, particularly in scenarios where the input sequences exhibit low noise and clear seasonal patterns. Its strengths include:

- Low computational overhead: Owing to linear operations only.

- High interpretability: Trend and seasonality effects can be visualized and validated.
- Strong generalization: Works well on both univariate and multivariate time series.

DLinear is especially suitable for high-resolution or infrastructure-constrained applications (e.g., hourly sensor data, edge devices), where model size and latency are critical. In my dissertation—forecasting hourly and daily data consumption—DLinear serves as a powerful baseline due to its robustness, interpretability, and efficiency.

6. Dataset

The dataset that was selected was produced from the benchmark dataset from the autoformers datasets that are widely used as benchmarks in various time series forecasts.

The primary dataset had a structure of recorded time series with hourly time stamps; it followed the structure below:

date	**Target 1**	**Target 2**	**Target 3**	**…**	**Target n-1**	**Target n**
Datetime	float64	float64	float64		float64	float64

Table 1

Table 1 describes the general schema of the dataset's target variables. Each row is indexed by a datetime stamp, and each column represents a distinct target variable (e.g., consumption for different customers), all formatted as continuous numerical values (float64).

date	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
2016-07-01 02:00:00	14.0	69.0	234.0	415.0	215.0	1056.0	29.0	840.0
2016-07-01 03:00:00	18.0	92.0	312.0	556.0	292.0	1363.0	29.0	1102.0
2016-07-01 04:00:00	21.0	96.0	312.0	560.0	272.0	1240.0	29.0	1025.0
2016-07-01 05:00:00	20.0	92.0	312.0	443.0	213.0	845.0	24.0	833.0
2016-07-01 06:00:00	22.0	91.0	312.0	346.0	190.0	647.0	16.0	733.0

Table 2

Table 2 is a preview of the first 5 rows of the dataset and 7 first target values of the actual dataset. Some confusion may be caused due to the headers row actual cell value for the target value, because they appear to integers but in reality they are treated as strings

Moreover, the dataset needed resampling to be transformed into the structure that was needed to forecast the determined horizons of the experiments. Using the pandas library and the hourly dataset, the daily dataset was created and it followed the schema of table 1, below a preview of the daily dataset is shown.

date	**0**	**1**	**2**	**3**	**4**
2016-07-01 00	655.0	2353.0	2023.0	10180.0	4903.0
2016-07-02 00	1098.0	2630.0	517.0	12091.0	6219.0
2016-07-03 00	1044.0	2924.0	817.0	11776.0	6467.0
2016-07-04 00	1162.0	2572.0	767.0	11983.0	6642.0
2016-07-05 00	1349.0	2700.0	4628.0	11698.0	6395.0

Table 2 .1

6.1.Characteristics of the datasets

Table 2.2 provides an overview of the two electricity consumption datasets used in the study, including the number of rows and columns, as well as the breakdown of numeric and categorical columns

File	Total rows	Total columns	Numeric columns	Categorical columns
electricity_hourly_fixed.csv	26304	322	321	1
electricity_daily_fixed.csv	1097	321	320	1

Table 2. 2

Table 2.3 presents the central tendency and dispersion statistics for the hourly and daily electricity consumption datasets. The values include the mean, standard deviation,

minimum, and quartile ranges, indicating significant variability and skewness, especially in the daily aggregated data.

File	mean	Std	min	25%	50%	75%	max
electricity_hourly_fixed.csv	2538.79	15027.57	0.00	241.00	550.00	347.00	1387.00
electricity_daily_fixed.csv	60815.73	316802.37	0.00	6727.75	13682.50	34574.00	10519900.00

Table 2. 3

Table 2.4 outlines the time window and total duration covered by each dataset. The hourly dataset spans exactly 1095 days, starting and ending with precise timestamps, while the daily dataset covers 1096 calendar days, reflecting daily aggregation.

File	Time Window Start	Time Window End	Time Span
electricity_hourly_fixed.csv	2016-07-01 02:00:00	2019-07-02 01:00:00	1095 days
electricity_daily_fixed.csv	2016-07-01	2019-07-02	1096 days

Table 2. 4

6.2.Cleaning Process for hourly and daily consumption Datasets

Calculating the statistics for the data table was the first step, the following part is focused on cleaning and transforming the dataset in a way that ensures high value results.

The preprocessing consisted of the following procedures:

1. Checking for missing or NaN values
2. Ensuring datetime format and sorting
3. Check for correct frequency (hourly or daily)
4. Interpolates missing values linearly by time, then forward and backward fills any remaining gaps
5. Outlier clipping
6. Checking on noise and sudden spikes in the time series and then filtering based on the results

Surprisingly the data was well adjusted in terms of missing values, even though they amount of data was immense, gaps in the values such as missing or NaN values did not exist.

7. Methodology

7.1. Model Selection Overview

7.1.1. PatchTST

a. Performance in Literature
PatchTST (Nie et al., 2023) has achieved state-of-the-art performance on multiple long-sequence time series benchmarks, particularly in scenarios involving complex and high-dimensional data. It consistently outperforms classical models and earlier transformer-based approaches in both accuracy and stability.

b. Reason for Selection
PatchTST was chosen due to its **strength in modeling long-range dependencies** and its **robustness on multivariate datasets**. Its use of patch embedding and attention allows it to handle high-dimensional input efficiently, making it highly appropriate for forecasting tasks that require learning complex temporal patterns across multiple variables.

c. Suitability for Forecasting
PatchTST is particularly suited for **multivariate time series**. Its ability to leverage relationships across variables and time makes it ideal for datasets with high temporal resolution (e.g., hourly data) or longer forecast horizons.

7.1.2. DLinear

a. Performance in Literature
DLinear (Zeng et al., 2022) presents a decomposition-based linear forecasting framework that, despite its simplicity, performs competitively on standard datasets. It is notable for its **low computational cost**, **fast convergence**, and **interpretability** compared to deep learning models.

b. Reason for Selection
DLinear was selected as a **strong, lightweight baseline model** to contrast with the complexity of PatchTST. Its minimal architecture allows for rapid experimentation and insight into the impact of model complexity. It also emphasizes interpretability, a key requirement in many practical applications.

c. Suitability for Forecasting
DLinear supports both **univariate and multivariate** inputs, though it excels in scenarios where variable interactions are limited or when quick, explainable forecasts are needed. It performs well on daily datasets or smaller-scale tasks with less temporal complexity.

7.2.Framework and Environment

This section outlines

- the development environment,
- libraries, and
- computational setup

used for implementing and training the two forecasting models: **PatchTST and DLinear.**

Each model was executed independently under consistent conditions to ensure a fair comparative analysis.

7.2.1. PatchTST Framework and Environment

Programming Language and Framework

- **Language**: Python 3.10
- **Framework**: PyTorch
  PatchTST was adapted from its original supervised implementation and configured for both hourly and daily forecasting tasks.

Libraries Used

- torch, numpy, pandas, matplotlib, scikit-learn, argparse, datetime, typing, os, psutil, uuid

Execution Environment

- **Platform**: Google Colab Pro
- Hardware:
  - **GPU**: NVIDIA A100 40GB
  - **CPU**: Intel Xeon (Colab backend)
  - **RAM**: 16 GB

Reproducibility Measures

- Seed control with random, numpy, and torch
- CUDA seeding via torch.cuda.manual_seed_all(42)
- Device allocation using torch.device("cuda" if available)
- Consistent data splits (60/20/20)
- Timestamped logging in logs/ directory

7.2.2. DLinear Framework and Environment

Programming Language and Framework

- **Language**: Python 3.10
- **Framework**: PyTorch 2
  DLinear was implemented from scratch using a lightweight linear projection structure, optimized for fast execution and interpretability.

Libraries Used

- torch, numpy, pandas, matplotlib, scikit-learn, dataclasses, argparse, enum, datetime, os

Execution Environment

- **Platform**: Google Colab Pro
- Hardware:
  - **GPU**: NVIDIA A100 40GB
  - **CPU**: Intel Xeon (Colab backend)
  - **RAM**: 83.5 GB

Reproducibility Measures

- Dynamic configuration with argparse and dataclasses
- Loss logs and prediction outputs stored in structured logs/ directory
- Consistent timestamping and model evaluation across datasets
- Train/validation/test split ratio: 60/20/20

7.3. Data Preparation

To ensure consistency and model compatibility, both the PatchTST and DLinear models were trained on identically preprocessed datasets. This section outlines the steps undertaken to transform raw multivariate electricity consumption time series data into supervised learning formats suitable for deep learning models.

7.3.1. Normalization

All numeric features, excluding engineered temporal attributes (e.g., sine and cosine encodings of day-of-week), were normalized independently using StandardScaler from scikit-learn. This allowed for column-wise standardization and enabled accurate inverse transformation during evaluation.

7.3.2. Supervised Window Generation

The temporal forecasting task was converted into a supervised learning problem using a sliding window mechanism. For each time step t, an input-output pair was defined as:

$$X_t=[x_{t-L+1},\ldots,x_t] \in R,\quad Y_t=[x_{t+1},\ldots,x_{t+T}] \in R^{T\times d}$$

where:

- L is the input sequence length,
- T is the prediction horizon,

- d=320 is the number of features (individual electricity usage series).

The following configurations were applied:

Dataset	Input Length L	Forecast Horizon T
Daily	120 days	7 days
Hourly	512 hours	24 hours

7.3.3. Dataset Splits

Both datasets were split chronologically into training, validation, and test sets to avoid data leakage. The split ratios and resulting shapes after sliding window extraction are summarized below:

Dataset	Split	Time Steps	Windows	Input Shape	Output Shape
Daily	Train	658	531	(531, 120, 320)	(531, 7, 320)
	Validation	219	92	(92, 120, 320)	(92, 7, 320)
	Test	220	93	(93, 120, 320)	(93, 7, 320)
Hourly	Train	15,782	15,246	(15,246, 512, 320)	(15,246, 24, 320)
	Validation	5,261	4,725	(4,725, 512, 320)	(4,725, 24, 320)
	Test	5,261	4,725	(4,725, 512, 320)	(4,725, 24, 320)

7.3.4. Model Input Formatting

All input and output sequences were converted into three-dimensional tensors with shape (batch_size,L,d) and (batch_size,T,d), respectively. PyTorch's TensorDataset and DataLoader utilities were employed to facilitate efficient batch training. A fixed batch size of 16 was used across both models and all experiments to ensure consistency.

7.4. DLinear Model and Forecasting Pipeline

To establish a strong linear baseline for long-horizon multivariate forecasting, the **DLinear** model was implemented following the original design principles of channel-independent projection. Specifically, DLinear forecasts each time series independently by applying a linear transformation from the past to future values, without inter-series

interaction. The same multivariate electricity consumption datasets, sampled at both hourly and daily resolutions, were used to ensure direct comparability with PatchTST.

7.4.1. Windowed Sample Construction

A sliding window approach was adopted to transform the continuous time series data into supervised input-output pairs suitable for DLinear's format. The input length (L) and prediction horizon (T) were adjusted to accommodate DLinear's architectural simplicity:

Dataset	Input Length L	Forecast Horizon T
Daily	120 days	7 days
Hourly	256 hours	24 hours

Each training sample consisted of:

$X \in R^{L \times 320}$, $Y \in R^{T \times 320}$

representing the historical input and future target sequences across 320 independent electricity consumption channels.

7.4.2. Temporal Splits and Sample Statistics

To avoid data leakage and ensure realistic forecasting scenarios, the datasets were split chronologically into training (60%), validation (20%), and test (20%) partitions. The post-windowing statistics are summarized below:

Dataset	Split	Time Steps	Sliding Windows	Input Shape X	Output Shape Y
Daily	**Train**	658	531	(531, 120, 320)	(531, 7, 320)
	Validation	219	92	(92, 120, 320)	(92, 7, 320)
	Test	220	93	(93, 120, 320)	(93, 7, 320)
Hourly	**Train**	15,782	15,503	(15,503, 256, 320)	(15,503, 24, 320)
	Validation	5,261	4,981	(4,981, 256, 320)	(4,981, 24, 320)
	Test	5,261	4,983	(4,983, 256, 320)	(4,983, 24, 320)

7.4.3. Model Input Format and Batch Handling

Input and output sequences were formatted as 3D tensors, preserving the shape (batch_size,L,320) for inputs and (batch_size,T,320) for predictions. A batch size of 16

was used uniformly across training, validation, and testing phases to align with the PatchTST pipeline.

7.5. Differences in Data Formatting

While the core data structure remained consistent, subtle differences in usage exist:

Aspect	PatchTST	DLinear
Input reshaping	Used as-is	Transposed internally to (Batch, Features, Sequence) for linear projection
Forecast output	Direct multivariate prediction	One linear layer per feature dimension
Feature scaling	Inverse transform applied for evaluation and plots	Same, but handled slightly differently in batch reshape
Forecasting horizon	Optimized with transformer attention over patches	Direct projection from input to future via linear layer

7.6. Model Architecture Details

7.6.1. PatchTST

The **PatchTST** model is a Transformer-based architecture specifically designed for multivariate time series forecasting. It adapts traditional vision-style patching to the temporal domain, enabling efficient modeling of long-term dependencies and high-dimensional signals.

7.6.2. Input Configuration and Patching Strategy

PatchTST operates on fixed-length input sequences, with lengths tailored to each dataset:

- **Hourly dataset**: L=512 time steps
- **Daily dataset**: L=120 time steps

To compress temporal information and facilitate attention over broader contexts, the input is segmented into overlapping patches:

- **Patch size**: 16 (hourly), 7 (daily)

- **Stride**: 16 (hourly), 7 (daily)

Each patch is flattened and projected into a latent space before being passed into the Transformer.

7.6.3. Embedding and Transformer Encoder

- Embedding dimension: 256
- **Encoder depth**: 4 stacked Transformer layers
- Attention heads: 4 per layer
- Feedforward expansion: d_{ff}=128

Each Transformer block follows the canonical structure of multi-head self-attention followed by a position-wise feedforward network, both with residual connections and layer normalization. The activation function employed is GELU.

7.6.4. Regularization and Normalization

To mitigate overfitting and encourage robust learning, several forms of dropout are applied:

- Embedding dropout: 0.1–0.2
- Attention dropout: 0.1
- Feedforward dropout: 0.1
- Prediction head dropout: up to 0.1

Additionally, the model includes an optional **Reversible Instance Normalization (RevIN)** module that stabilizes training by normalizing each channel independently before encoding, followed by an inverse normalization post-prediction.

7.6.5. Positional Awareness

Unlike classical Transformers, PatchTST omits explicit positional encodings. Instead, positional dependencies are implicitly captured through the structure of the patch-based input itself. This design choice avoids hardcoded time-specific biases and supports generalization to arbitrary forecasting horizons.

7.7. DLinear

The **DLinear** model represents a minimalist architecture for multivariate time series forecasting. It is designed to test the hypothesis that simple linear mappings from historical to future values can offer competitive performance in long-horizon forecasting scenarios.

7.7.1. Channel-Wise Linear Projection

DLinear operates by reordering the input tensor from shape (batch_size,L,d) to (batch_size,d,L), where each of the channels represent an independent time series. A

shared linear layer is then applied to each channel to project the lookback window of length L into the forecast horizon T:

$$\hat{Y}_i = W x_{i,t-L+1:t}, \quad \forall i \in [1, d]$$

where $W \in R^{T \times L}$ is the learned projection matrix, shared across all features.

7.7.2. Simplified Assumptions

This implementation omits any explicit seasonal-trend decomposition or attention mechanisms. The model assumes that each time series exhibits temporal structure that can be adequately captured by a linear mapping from the recent past. No deep layers, convolutions, or positional encodings are used.

7.7.3. Parameter Efficiency and Regularization

By avoiding per-channel parameterization, DLinear maintains a low model complexity and avoids overfitting, particularly on small datasets. All weights are shared, and no feature-specific layers are introduced.

7.7.4. Tuning and Computational Benefits

With minimal architectural depth, DLinear requires tuning only a small number of hyperparameters: the lookback window size, prediction horizon, and learning rate. This results in:

- **Fast convergence** during training
- Low memory and compute requirements
- Ease of deployment on resource-constrained environments

7.7.5. Design Trade-Offs

While DLinear is efficient and interpretable, its lack of capacity to model complex non-linear interactions or cross-variable dependencies limits its expressiveness. Nevertheless, it serves as a valuable baseline in comparing the benefits of more sophisticated architecture such as PatchTST.

7.8. Training Configuration

To ensure a fair and consistent evaluation of forecasting performance, both the PatchTST and DLinear models were trained under configurations tailored to the temporal characteristics of hourly and daily electricity consumption datasets. Wherever applicable, equivalent design choices were adopted to isolate architectural differences as the primary factor under study.

7.8.1. Loss Functions

The **PatchTST** model utilized the **Smooth L1 loss (Huber loss)**, computed in a channel-wise fashion. This loss function offers a balance between mean squared error

sensitivity and robustness to outliers, particularly relevant in multivariate forecasting settings. Optional per-channel weighting was applied based on inverse validation RMSE to emphasize poorly performing series.

In contrast, the **DLinear** model employed the standard **mean squared error (MSE)** loss. Channel-wise weighting was similarly supported, based on validation RMSE-derived inverses, enabling the model to allocate greater learning focus to high-error channels.

7.8.2. Optimization Strategy

Both models adopted the **AdamW optimizer**, selected for its decoupled weight decay and stability across a wide range of learning rates. The initial learning rate was set to 1×10^{-3} for both models.

- **PatchTST** used a **cosine annealing scheduler** with optional linear warm-up: 0 warm-up epochs for the hourly dataset and 3 warm-up epochs for the daily dataset.
- **DLinear** maintained a **fixed learning rate**, without learning rate scheduling, due to its shallow architecture and stable convergence behavior.

7.8.3. Epochs, Batch Size, and Early Stopping

- **Batch Size**: PatchTST used a batch size of 16, whereas DLinear used 32, reflecting the latter's reduced memory footprint.
- **Epochs:**
  - PatchTST: 4 epochs (hourly), 10 epochs (daily)
  - DLinear: 20 epochs across both datasets
- **Early Stopping:**
  - PatchTST applied early stopping with patience values of 10 (hourly) and 5 (daily), based on validation loss plateauing.
  - DLinear did not utilize early stopping but instead **recomputed per-channel weighting after each epoch**, introducing a dynamic loss rebalancing mechanism.

7.9. Model Logging and Checkpointing

Neither model employed weight checkpointing during training. However, both pipelines recorded comprehensive logs of training and validation loss per epoch, and saved final evaluation outputs including:

- Forecasted vs. true sequences
- Global error metrics (MAE, RMSE, R^2, SMAPE)
- Per-channel NRMSE reports and category assignments
- Visualizations of predicted time series across selected features

This uniform logging strategy supported a detailed post-hoc comparison of model performance and error distributions.

7.10. Evaluation Strategy

To evaluate the performance of the time series forecasting model, we adopted both **global** and **per-channel** metric analyses over varying forecast horizons. This comprehensive evaluation enables both high-level model performance understanding and fine-grained channel-level diagnostic analysis.

7.10.1. Forecast Horizons

We *considered two* temporal resolutions:

- **Hourly Dataset:**
  - Input sequence: 512 time steps (hours)
  - Forecast horizon: 24 time steps (24 hours ahead)
- **Daily Dataset:**
  - Input sequence: 120 time steps (days)
  - Forecast horizon: 7 time steps (7 days ahead)

These horizons are chosen to reflect practical use cases in short-term and medium-term energy forecasting.

7.10.2. Global Evaluation Metrics

To assess overall forecasting accuracy across all time steps and variables, **four standard regression metrics** were computed on the aggregated predictions:

1) **Mean Absolute Error (MAE)**

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

MAE captures the average magnitude of forecast errors, offering a scale-sensitive and interpretable measure of deviation.

2) **Root Mean Squared Error (RMSE)**

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

3) **RMSE** penalizes larger deviations more heavily than MAE, making it particularly sensitive to outliers — a valuable trait in long-horizon forecasting where error compounding is common.
4) **R-squared Score (R²)**

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

R² quantifies the proportion of variance in the ground truth, that is explained by the forecast. Values close to 1 indicate strong predictive fidelity.

- *Symmetric Mean Absolute Percentage Error (SMAPE)*

$$\text{SMAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|\hat{y}_i - y_i|}{(|y_i| + |\hat{y}_i|)/2}$$

- SMAPE provides a normalized, percentage-based error metric that remains robust when target values vary widely in scale. A small constant ϵ\epsilonϵ ensures numerical stability.

All metrics were computed using inverse-transformed predictions to reflect real-world error magnitudes.

7.10.3. Per-Channel Metrics

To complement the global metrics, model performance was also evaluated **per individual time series**, corresponding to electricity usage of each entity (e.g., building or meter).

Specifically, the following were computed for each feature:

- **Per-Channel RMSE**
  Computed independently for each variable ccc over all sequences T and forecast steps H:

$$\text{RMSE}_c = \sqrt{\frac{1}{T \cdot H} \sum_{t=1}^{T} \sum_{h=1}^{H} (y_{t+h,c} - \hat{y}_{t+h,c})^2}$$

This enables fair comparison between variables of different scales.

8. Results and Use Case Evaluation

To analyze forecasting performance across diverse conditions, both models were evaluated under a series of **controlled dataset variants and preprocessing scenarios**, referred to as *evaluation use cases*. These use cases were designed to assess model robustness, sensitivity to feature engineering, and response to error-based data pruning.

8.1.Dataset Variants (Use Cases)

Five structured dataset configurations were employed in model training and evaluation:

Use Case	Description
UC1	Baseline dataset with raw electricity consumption values only.
UC2	Baseline dataset augmented with 10 engineered time-based features.
UC3	Dataset with high-error channels removed (RMSE > threshold).
UC4	Combination of engineered features and RMSE-based filtering.
UC5	Full configuration: engineered features + RMSE filtering + NRMSE-based channel pruning.

These scenarios allow systematic evaluation of model behavior under varying data quality and feature dimensionality.

Each use case will be accompanied by:

1. A statistics table
2. A results table
3. Generated plots, focused on features/channels 1 or 2 displaying the first 3 samples

8.2.Engineered Feature Summary

The engineered dataset included 10 additional features derived from the datetime index, listed below:

1. dayofweek
2. month
3. is_weekend
4. dayofyear
5. dayofweek_sin
6. dayofweek_cos

7. month_sin
8. month_cos
9. dayofyear_sin
10. dayofyear_cos

These features encode temporal periodicities using both categorical and cyclic representations.

8.3.Dataset Statistics

The following table summarizes the dimensionality of both the hourly and daily engineered datasets:

Dataset	Total Rows	Total Columns	Consumption Channels	Date Columns	Engineered Features
electricity_hourly_transformed_2.csv	26,304	331	320	1	10
electricity_daily_transformed_2.csv	1,097	331	320	1	10
electricity_hourly_transformed_3.csv	26,304	322	311	1	10
electricity_daily_transformed_3.csv	1,097	322	311	1	10
electricity_hourly_transformed_5.csv	26,304	298	287	1	10
electricity_daily_transformed_5.csv	1,097	298	287	1	10

Table 2. 5

8.3.1. Results: Use Case 1 — Unmodified Dataset

In this initial use case, forecasting models were trained and evaluated using raw electricity consumption data, without any engineered features or preprocessing filters. Descriptive statistics for the hourly and daily datasets (Table 2.6) reveal substantial heterogeneity in consumption values. For instance, the standard deviation of the hourly dataset (15,027.57) is nearly six times its mean (2,538.79), while the daily dataset exhibits extreme values, with a maximum of over 10 million and a standard deviation exceeding 300,000. These figures highlight significant skewness and variance, reinforcing the need for normalization strategies and robust error metrics that can handle scale disparities across channels.

File	mean	Std	min	25%	50%	75%	max
electricity_hourly_fixed.csv	2538.79	15027.57	0.00	241.00	550.00	347.00	1387.00
electricity_daily_fixed.csv	60815.73	316802.37	0.00	6727.75	13682.50	34574.00	10519900.00

Table 2. 6

These statistics reflect significant variation across entities and time, underscoring the importance of robust normalization and error-aware evaluation.

Hourly Forecasting Results

Model	File	MAE	RMSE	R^2	Per channel RMSE	sMAPE
PatchTST	electricity_hourly_fixed.csv	163.649	1515.959	0.9918	254.20	10.56%
Dlinear	electricity_hourly_fixed.csv	196.65	1881.27	0.8596	295.53	12.09%

Table 2.7

As shown in Table 2.7, PatchTST significantly outperforms DLinear across all key evaluation metrics. The R^2 of 0.9918 indicates that PatchTST captures over 99% of the variance in the data, suggesting excellent model fit despite the inherent noisiness and scale variation evident in the dataset's descriptive statistics. In contrast, DLinear's lower R^2 (0.8596) and higher RMSE suggest reduced effectiveness in adapting to the temporal fluctuations and high-value outliers present in the dataset.

The absolute error values (MAE and RMSE) are also within a practically acceptable range considering the mean consumption value (2,538.79) and high variability. PatchTST's per-channel RMSE of 254.20 reflects strong predictive stability across channels, especially when viewed alongside the substantial variance. Similarly, the sMAPE of 10.56% indicates reliable relative accuracy across entities of varying consumption scales—well within industry-accepted thresholds for utility forecasting, which typically regard <15% sMAPE as satisfactory for disaggregated load forecasting.

Together, these results establish a high-performance baseline for PatchTST under minimal preprocessing, reinforcing its architectural strengths in raw-data scenarios. DLinear, while functional, exhibits performance limitations that suggest a greater dependency on preprocessed or regularized inputs.

PatchTST Generated Plot

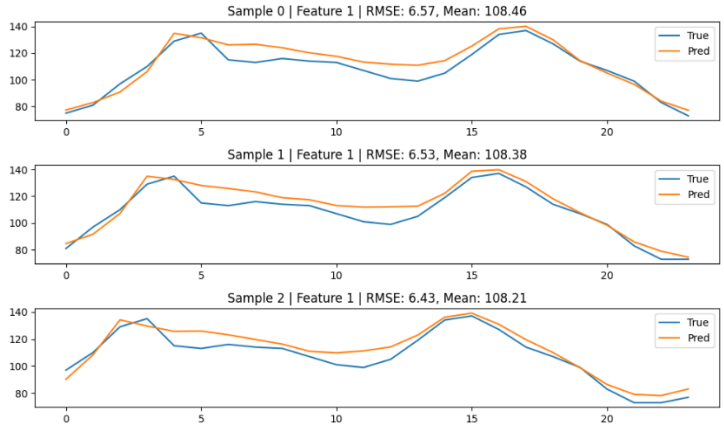


Figure 1 Visual Performance Evaluation

DLinear Generated Plot

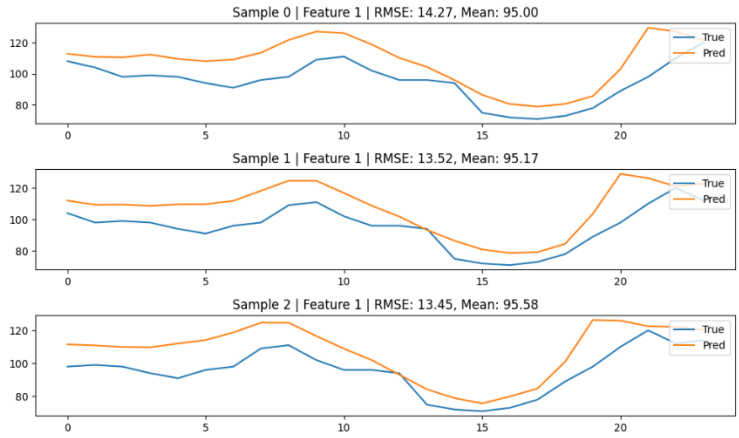


Figure 2 Visual Performance Evaluation

PatchTST predictions closely follow the true hourly consumption curves across all samples, successfully capturing peaks, valleys, and trend shifts with minimal deviation. The model maintains strong shape fidelity and scale consistency, reflecting its ability to learn temporal dependencies and short-term fluctuations effectively.

DLinear, in contrast, produces smoother forecasts that frequently miss local variations and exhibit a tendency to overestimate consumption. The predicted lines fail to adapt to dips and rises, leading to visible lag and pattern mismatch, particularly in dynamic segments of the series.

Daily Forecasting Results

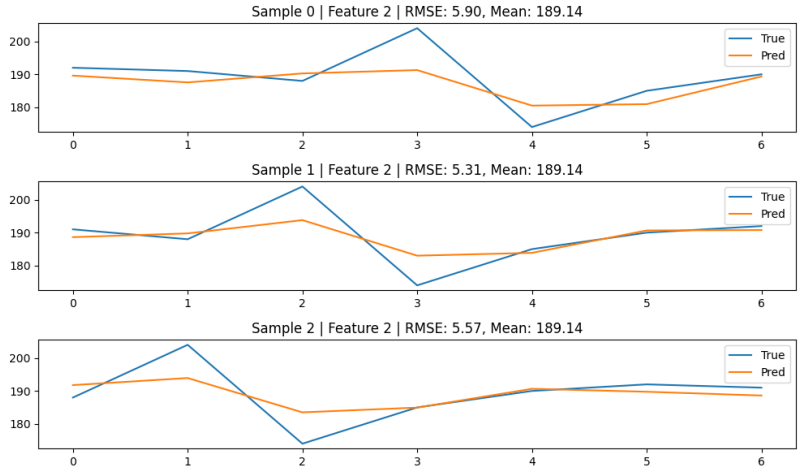
Model	File	MAE	RMSE	R^2	Per channel RMSE	sMAPE
PatchTST	electricity_daily_fixed.csv	4409.561	42129.439	0.9748	5981.35	9.2%
DLinear	electricity_daily_fixed.csv	3636.97	32718,29	0.3270	5607.43	9.33%

Table 2.8

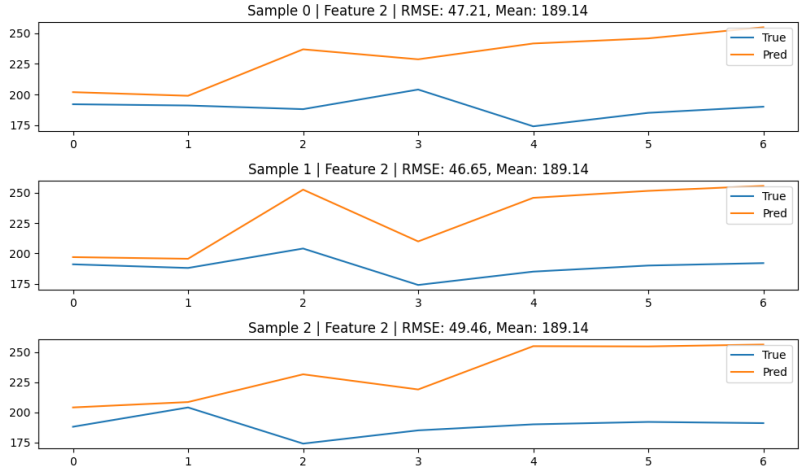
Interestingly, DLinear records lower absolute errors (MAE and RMSE), yet its R^2 is drastically lower (0.3270) than PatchTST's 0.9748. This implies DLinear's forecasts may hover near the mean—yielding acceptable average errors, but failing to adapt to broader trends or deviations. PatchTST, while having higher RMSE, retains superior scale-aware accuracy and consistency, as evidenced by its higher R^2 and slightly better sMAPE (9.2% vs 9.33%).

In both granularities, all sMAPE values are under the industry-accepted 15% threshold, confirming that the forecasts are practically useful. However, PatchTST's ability to balance global and per-channel accuracy, especially in the presence of outliers, makes it the more dependable model for raw data scenarios.

PatchTST Generated Plot



DLinear Generated Plot



In UC1, **PatchTST demonstrates strong predictive alignment**, capturing daily fluctuations in Feature 2 with good accuracy. Across all samples, the model effectively tracks the general shape and direction of the true series, including turning points and amplitude shifts, with RMSEs consistently below 6. This suggests effective modeling of raw consumption patterns, even without feature enhancement or channel pruning.

DLinear, by contrast, shows substantial overestimation across the entire sequence. The predicted values deviate significantly from the actual trend, particularly during peak shifts, with RMSEs exceeding 46 in all cases. This indicates a limited ability to adapt to the raw input dynamics, producing consistently biased forecasts with poor temporal resolution.

8.3.2. Results: Use Case 2 — Engineered Features Augmented

This scenario investigates the impact of incorporating engineered temporal features—such as day-of-week indicators and seasonal encodings—into the forecasting framework. These features were appended to the input data without modifying the original consumption channels, thus maintaining the core temporal signal while modestly increasing input dimensionality.

Descriptive Statistics across the dataset

File	Mean	Std	Min	25%	50%	75%	Max
electricity_hourly_transformed_2.csv	2534.81	15011.78	0.00	241.00	548.00	1372.00	59180 0.00
electricity_daily_transformed_2.csv	60883.56	31660 5.36	0.00	6752.00	13704.00	34648.25	92379 86

Table 2. 9

The statistical profile closely mirrors that of UC1, affirming that the target distribution remained unaffected by feature augmentation. However, the slight shifts in maximum values suggest a need for caution when interpreting extreme-value impacts under multivariate feature conditions.

Hourly Forecasting Results

Model	File	MAE	RMSE	R^2	Per channel RMSE	sMAPE
PatchTST	electricity_hourly_transformed_2.csv	180.8528	1650.0504	0.9903	265.63184	11.53%
DLinear	electricity_hourly_transformed_2.csv	184.5009,	1777.5326,	0.8616	274.3157	12.96%

Table 2.10

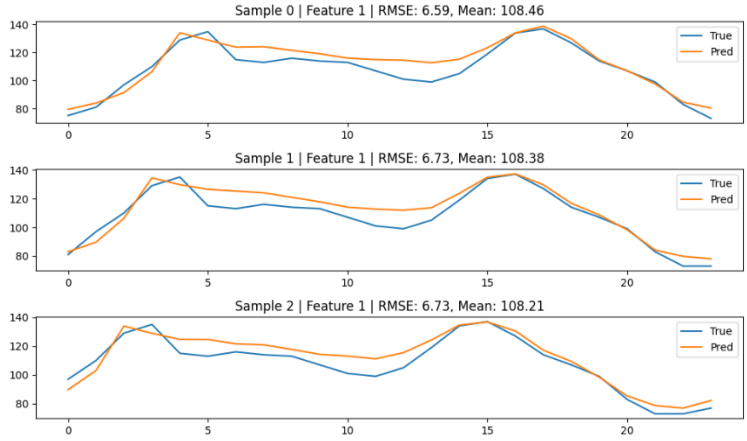
Contrary to expectations, both models exhibited modest performance degradation relative to the baseline (UC1). PatchTST showed a 10.5% increase in MAE and a ~9% increase in RMSE, with a slight dip in R^2. DLinear's degradation was less pronounced in absolute error terms but more significant in R^2, which dropped to 0.8616—suggesting a weakened ability to generalize when exposed to non-target temporal features.

These results imply that, in this context, engineered temporal signals did not offer clear additive value. PatchTST's architecture, which is designed to exploit sequential dependencies, may have faced diluted signal-to-noise ratios due to the injection of non-consumption data. Likewise, DLinear's regression-oriented simplicity appears ill-suited to benefit from the additional temporal features, as evidenced by the rising sMAPE.

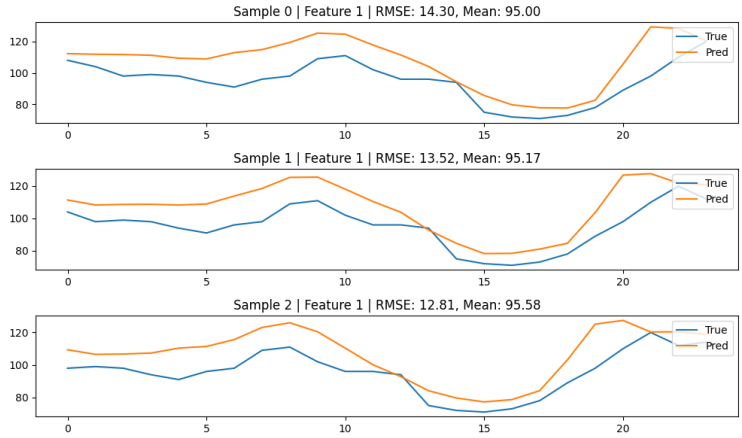
Nonetheless, error values remain within an acceptable range, especially for PatchTST, whose RMSE and sMAPE are still below the industry-accepted 15% sMAPE threshold

for utility forecasts. These findings suggest that while engineered features did not enhance short-term accuracy, they also did not critically impair performance—implying that careful feature selection or dimensionality reduction may be required to unlock their potential in future iterations.

PatchTST Generated Plot



DLinear Generated Plot



Under UC1 (raw baseline), PatchTST produces forecasts that closely follow the true hourly consumption trends, effectively capturing peak demand and cyclical patterns with minimal lag. The visual alignment indicates strong temporal generalization even without additional feature context. In contrast, DLinear under UC1 shows limited responsiveness, with smoothed, lagging predictions that fail to track rapid fluctuations, leading to visible over- and underestimation zones.

With the introduction of engineered time features in UC2, both models show visual improvements, but PatchTST remains superior. Its predictions tightly match the ground truth, especially around inflection points, demonstrating its ability to exploit temporal context effectively. DLinear, while improved compared to UC1, still exhibits a dampened response to dynamic changes, with predictions that generally miss sharper transitions despite a slightly better mean alignment.

Daily Forecasting Results

Model	File	MAE	RMSE	R^2	Per channel RMSE	sMAPE
PatchTST	electricity_daily_transformed_2.csv	4060.3176	38747.4688	0.9787	5084.388	8.54%
DLinear	electricity_daily_tranfsormed_2.csv	3588.7935	32089.8691	0.1820	4728.3940	10.97%

Table 2.11

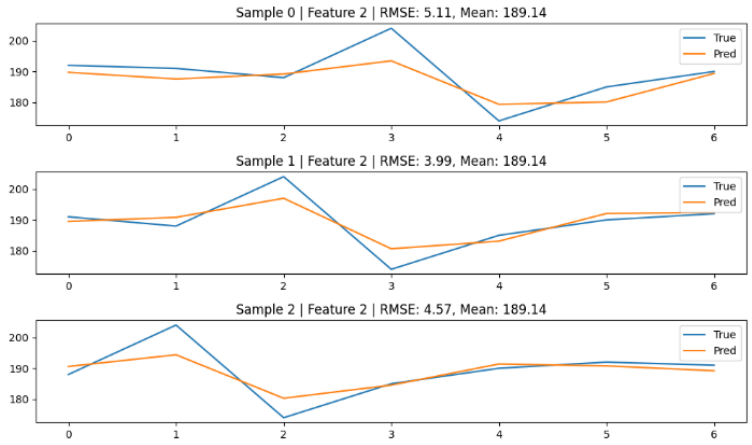
Compared to the baseline daily results (UC1), both models show numerical improvement in RMSE and MAE. However, the implications of these gains differ. PatchTST's R^2 rises slightly (from 0.9748 to 0.9787), alongside a notable drop in sMAPE (from 9.20% to 8.54%), indicating enhanced generalization and improved sensitivity across varying consumption scales. This suggests that temporal features positively contributed to capturing daily usage patterns without introducing noise.

In contrast, DLinear, despite achieving lower MAE and RMSE, experiences a severe drop in R^2—from 0.3270 in UC1 to just 0.1820—implying diminished ability to capture variance. Furthermore, its sMAPE increases from 9.33% to 10.97%, signaling worsening performance on low-volume or more erratic channels. This mismatch suggests that while DLinear may optimize for average proximity (as seen in MAE), it struggles with distributional diversity and temporal alignment when non-consumption features are introduced.

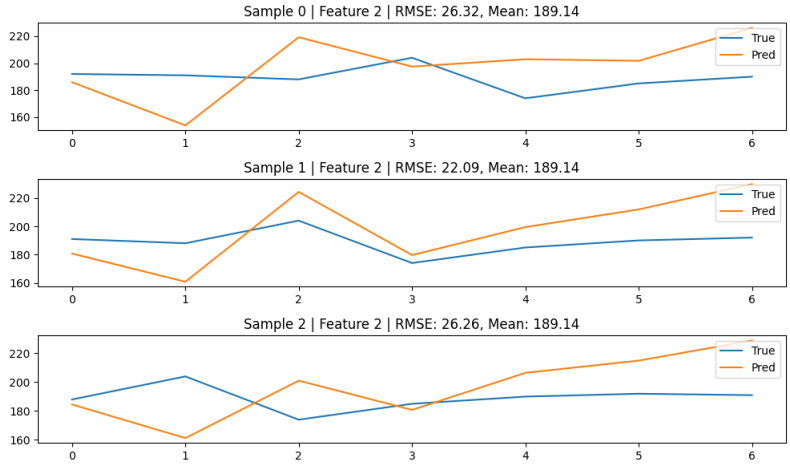
Collectively, these results affirm that PatchTST benefits modestly from feature augmentation in the daily context, while DLinear's statistical improvements do not translate to better model behavior or practical forecasting accuracy.

.

PatchTST Generated Plot



DLinear Generated Plot



In UC2, PatchTST produces forecasts that are well-aligned with the true signal across all samples. The predicted sequences successfully capture local variations in Feature 2, including peaks and troughs, with low RMSE values below 5.5. The addition of engineered time features appears to enhance the model's responsiveness, resulting in smoother yet accurate predictions that preserve trend direction and amplitude.

DLinear, on the other hand, continues to show large deviations from the actual series. Forecasts fluctuate inconsistently around the ground truth, with some segments exhibiting abrupt shifts and phase mismatches. RMSE values remain high across samples (22–26), indicating that the model struggles to incorporate the added temporal features effectively and lacks precision in adapting to day-to-day fluctuations.

8.3.3. Results: Use Case 3 — Engineered Features + RMSE-Based Channel Filtering

This case evaluates the impact of combining temporal feature augmentation with RMSE-based channel filtering. Channels exhibiting high prediction error (above a predefined RMSE threshold) were removed to suppress noise from unstable or erratic series. The aim was to focus learning on channels with more stable and predictable behavior, thereby enhancing overall model precision.

Descriptive Statistics across the dataset

File	Mean	Std	Min	25%	50%	75%	Max
electricity_hourly_transformed_3.csv	1257.58	2065.63	0.00	233.00	525.00	1248.00	25517.00
electricity_daily_transformed_3.csv	30204.38	45990.21	0.00	6570.00	13245.00	31070.00	359890.00

Hourly Forecasting Results

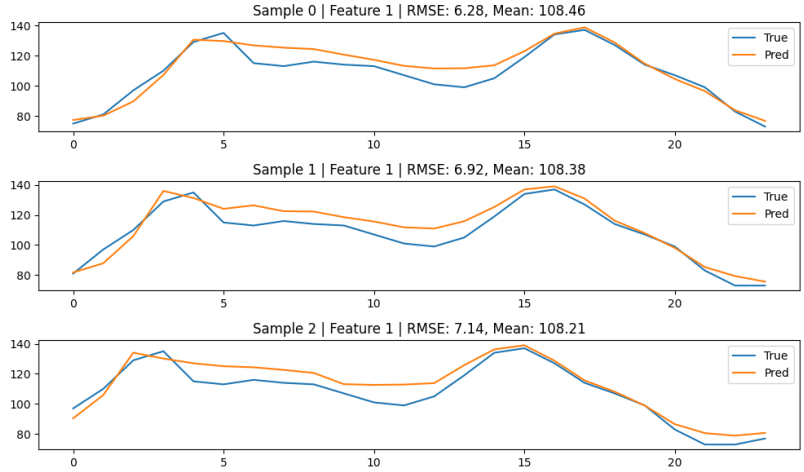
Model	File	MAE	RMSE	R^2	Per channel RMSE	sMAPE
PatchTST	electricity_hourly_transformed_3.csv	76.431	192.251	0.9919	113.63	10.96%
Dlinear	electricity_hourly_transformed_3.csv	85.2683	212.0842	0.8573	120.8955	12.14%

Table 2.12

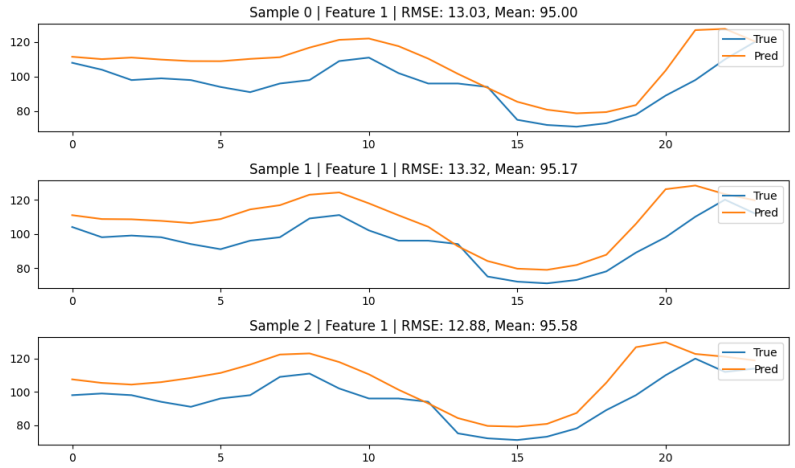
Both models benefited from the filtered dataset. PatchTST achieved its lowest MAE and RMSE to date, while maintaining an exceptionally high R^2 of 0.9919—suggesting it could model the dominant temporal structures more effectively without interference from erratic series. DLinear also improved, but its R^2 remained markedly lower, indicating a more limited capacity to leverage the benefits of filtering, likely due to its linear assumptions and lack of sequence modeling.

The reductions in per-channel RMSE and sMAPE for both models point to more consistent accuracy across entities, further confirming that noise suppression enhances model uniformity.

PatchTST Generated Plot



DLinear Generated Plot



In UC3, PatchTST exhibits strong temporal alignment with the true consumption patterns. The predicted sequences effectively capture both peak timings and valley depths across all samples, with minimal phase shift or amplitude error. The visual coherence of the forecasts suggests robust generalization after removing high-error channels, allowing the model to focus on more predictable trends. DLinear, by contrast, shows larger discrepancies between predicted and actual values. The forecasts consistently overestimate consumption during dips and lag behind rising trends, resulting in visibly inflated curves with reduced sensitivity to shape variations. While general trends are roughly followed, the model's inability to adapt to sharp shifts diminishes the fidelity of its predictions.

Daily Forecasting Results

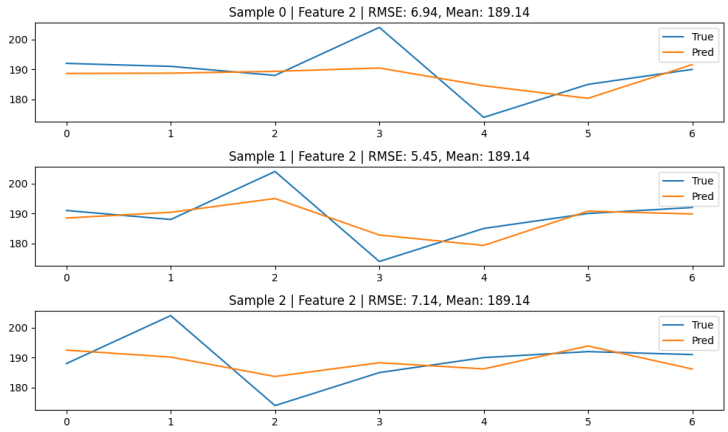
Model	File	MAE	RMSE	R^2	Per channel RMSE	sMAPE
PatchTST	electricity_daily_transformed_3.csv	1616.102	3617.775	0.9935	2092.8894	8.89%
Dlinear	electricity_daily_tranformed_3.csv	1841.1790	3993.9570	0.3221	2317.4744	9.90%

Table 2.13

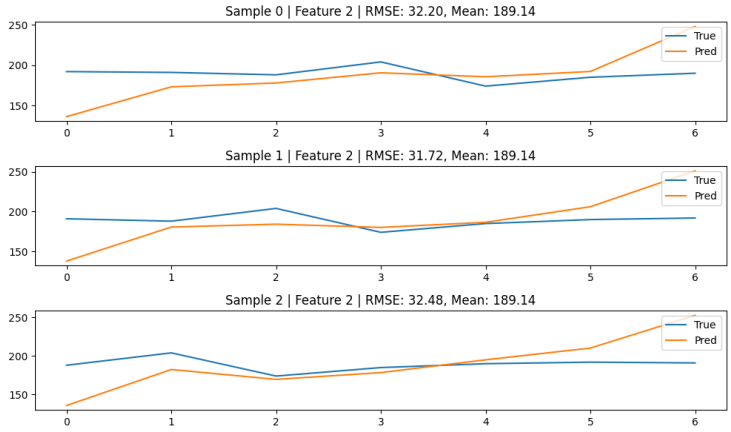
Filtering high-error channels led to substantial accuracy gains, especially for PatchTST. With RMSE reduced by over 50% from UC2 and R^2 climbing to 0.9935, PatchTST demonstrated enhanced ability to capture dominant consumption patterns in cleaner data. The drop in sMAPE to 8.89% highlights improved relative consistency across varying consumption scales.

DLinear also benefited, with lower MAE and RMSE, though its R^2 remained low (0.3221), indicating persistent limitations in variance modeling. The moderate sMAPE improvement suggests some gains in scale awareness, but PatchTST remains decisively stronger in both absolute and relative performance under filtered conditions.

PatchTST Generated Plot



DLinear Generated Plot



In UC3, PatchTST delivers forecasts that generally preserve the underlying structure of the true signal. Predicted values remain close to the actual consumption levels, especially around local fluctuations. Across all samples, the model captures key peaks and troughs with moderate deviations, reflected in RMSE values between 5.4 and 7.1, indicating stable performance after removing high-error channels.

DLinear, however, shows significantly inflated forecasts with clear overestimation in every sample. The predicted curves diverge consistently from the true series, particularly toward the end of the sequences. RMSE values exceed 31 across all samples, reflecting poor alignment and an inability to adjust to the reduced noise introduced by the high-error channel filtering.

8.3.4. Results: Use Case 4 — RMSE-Based Channel Filtering (No Engineered Features)

This configuration isolates the effect of **threshold-based RMSE filtering** on model performance by retaining only stable channels—those with validation RMSE below predefined thresholds—while excluding all engineered features. This allows analysis of the impact of channel volatility on model accuracy in a clean, unaugmented feature space.

Filtering Thresholds

- Hourly dataset threshold: RMSE < 2,500
- Daily dataset threshold: RMSE < 7,500

These values were empirically chosen to eliminate extreme outliers while preserving the majority of predictive time series.

Descriptive Statistics across the dataset

File	mean	Std	min	25%	50%	75%	max
electricity_hourly_fixed.csv	2538.79	15027.57	0.00	241.00	550.00	1372.00	764000.00
electricity_daily_fixed.csv	60815.73	316802.37	0.00	6727.75	13682.50	34574.00	10519900.00

Table 2 .14

Hourly Forecasting Results

Model	File	MAE	RMSE	R^2	Per channel RMSE	sMAPE
PatchTST	electricity_hourly_fixed.csv	91.530	287.185	0.9921	140.80084	10.61%
DLinear	electricity_hourly_fixed.csv	105	317.852	0.8504	157.2741	12.05%

Table 2.15

Compared to UC1, both models showed significant error reductions—PatchTST's MAE dropped by ~44% and DLinear's by ~47%. These improvements confirm that filtering unstable channels alone enhances model performance, even without feature engineering. PatchTST again leads in all metrics, maintaining a high R^2 and lower sMAPE, indicating stronger generalization and more consistent performance across entities.

PatchTST Generated Plot

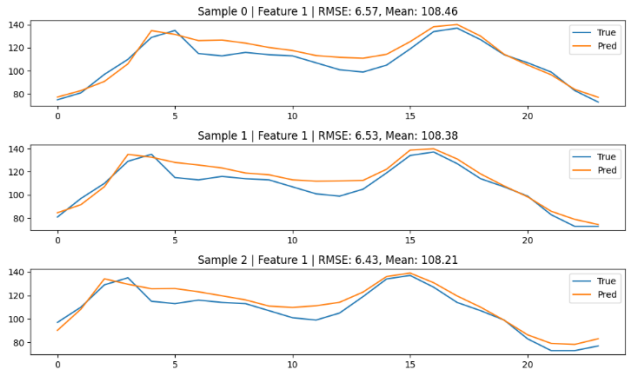
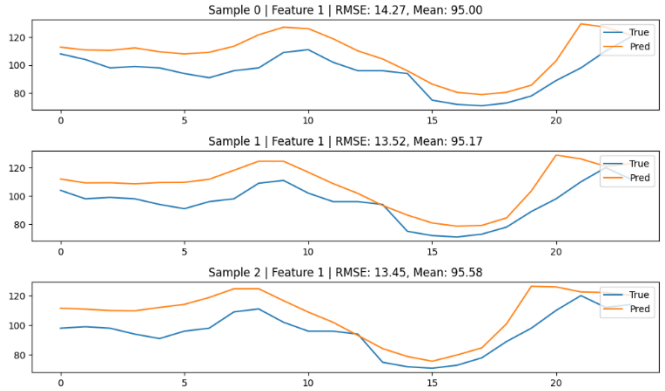


Figure 1 Visual Performance Evaluation

DLinear Generated Plot



In UC4, PatchTST generates forecasts that closely match the true hourly consumption patterns.

Across all samples, the model successfully captures both amplitude and phase of key fluctuations, including daily peaks and troughs. RMSE values remain low (≈6.4–6.6), indicating robust performance when combining feature engineering with RMSE-based channel filtering.

DLinear, while following the general trend, fails to align precisely with the true signal.

The forecasts tend to overestimate during low-consumption periods and flatten out during inflection points. This results in visibly larger residual gaps, with RMSEs over twice as high as those of PatchTST, suggesting weaker adaptability despite improved input structure,

Daily Forecasting Results

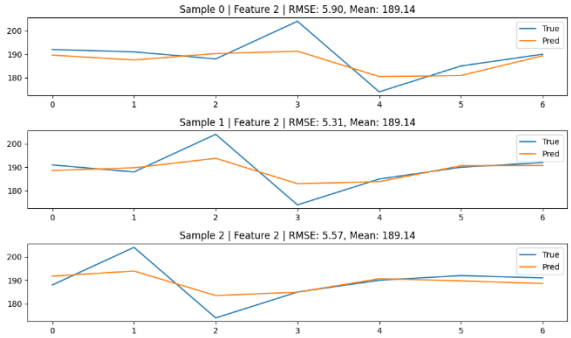
Model	File	MAE	RMSE	R^2	Per channel RMSE	sMAPE
PatchTST	electricity_daily_fixed.csv	1213.993	2474.893	0.9889	1762.4048	9.46%
Dlinear	electricity_daily_fixed.csv	1152.781	2436	0.3478	1776.7344	9.7%

Table 2.16

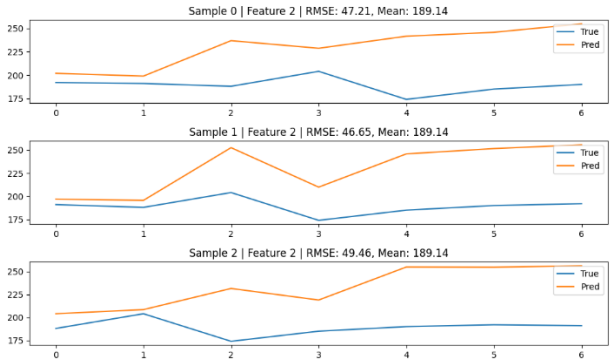
Both models showed notable improvements on the daily dataset following RMSE-based channel filtering. PatchTST achieved a strong R^2 of 0.9889, indicating high variance capture, along with the lowest RMSE and sMAPE—demonstrating stable, scale-aware predictions. DLinear posted a slightly lower MAE but significantly lagged in R^2, pointing to weaker pattern generalization despite localized error improvements.

The reduced sMAPE values for both models suggest that removing unstable series also reduces relative scale error, reinforcing the value of filtering even without added features. As in the hourly case, PatchTST offers a better trade-off between global accuracy and per-series consistency.

PatchTST Generated Plot



DLinear Generated Plot



In UC4, PatchTST demonstrates stable and accurate forecasting performance.

The model captures short-term variations in Feature 2 with relatively low RMSE values across samples (5.31–5.90), effectively following both rising and falling trends. The alignment between prediction and ground truth remains consistent, suggesting strong benefit from the combined use of engineered features and RMSE-based channel filtering.

DLinear, on the other hand, shows significant deviation from the actual signal.

The forecasts consistently overestimate consumption levels, and the predicted curves do not reflect the temporal variation seen in the ground truth. RMSE values above 46 across all samples highlight the model's poor adaptability under this configuration, despite access to enhanced inputs.

8.3.5. Results: Use Case 5 — Engineered Features + RMSE & NRMSE-Based Channel Filtering

This final use case combines the benefits of temporal feature engineering with a two-stage channel filtering process. Initially, high-RMSE series were removed to eliminate unstable channels. Subsequently, NRMSE-based pruning was applied to discard time series with disproportionate scale-normalized errors. The goal was to isolate the most predictable and scale-consistent signals for enhanced model performance.

N-RMSE Definition

N-RMSE (Normalized Root Mean Square Error) is a normalized version of the Root Mean Square Error (RMSE), which is used to measure the differences between predicted and actual values. N-RMSE provides a scale-independent metric by normalizing RMSE, allowing comparison across datasets or models with different scales.

Descriptive Statistics across the dataset

Even though there are many ways to normalize RMSE, I chose to normalize it by mean, and applied the mathematical type below:

$$\text{N-RMSE} = \frac{\text{RMSE}}{\bar{y}}$$

where **y** is the mean of actual values.

In my experiments, to evaluate the N-RMSE per channel and select which channels will be dropped, I categorized the N-RMSE values of the channels based on the following scale:

N-RMSE Value	Category
Value < 0.1	Excellent
0.1 <value <0.3	Good
0.3<value<0.5	Fair
0.5<value<1	Poor
Value>1	Bad

The steps followed for calculating the metrics are listed below:

1. Calculating the nrmse per channel
2. Categorizing the nrmse based on the scale
3. Dropping the channels that were on the ‘Poor’ and ‘Bad’ categories

The descriptive stats of the transformed files for the use case are below:

File	mean	Std	min	25%	50%	75%	max
electricity_hourly_transformed_5.csv	2714.61	15608.29	0.00	283.00	599.00	1496.00	764000.00
electricity_daily_transformed_5.csv	65091.18	328444.86	0.00	7641.00	14703.00	38047.00	10519900.00

Despite advanced filtering, wide variance remains, underscoring the need for scale-aware metrics like NRMSE and sMAPE in evaluation.

Hourly Forecasting Results

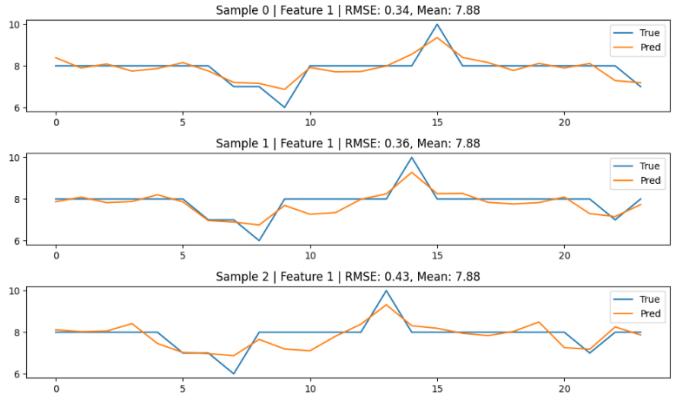
Model	File	MAE	RMSE	R^2	Per channel RMSE	sMAPE
PatchTST	electricity_hourly_transformed_5	98.287	302.843	0.9918	148.77705	7.37%
Dlinear	electricity_hourly_transformed_5	107.593	327.095	0.8478	161.5538	8.20%

Table 2.17

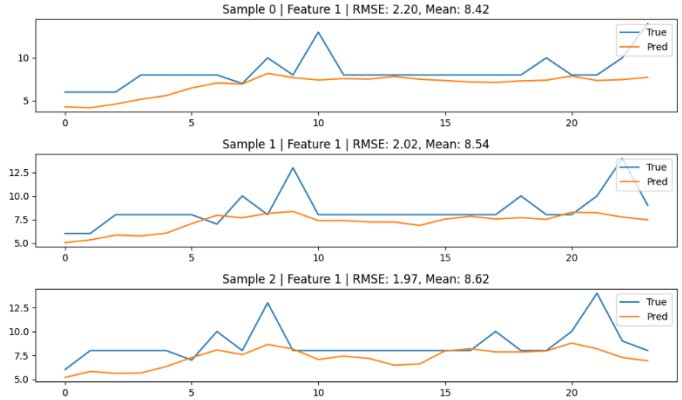
The inclusion of NRMSE filtering produced the lowest sMAPE values across all use cases. PatchTST outperformed DLinear on all metrics, with a substantial drop in sMAPE—down from 10–12% in previous configurations to 7.37%. This indicates tighter alignment between predicted and actual consumption, especially on channels with variable scales.

While DLinear also improved, its lower R^2 (0.8478) reaffirms its limitations in capturing nuanced temporal dynamics even with filtered, enriched data. PatchTST's performance confirms the compounding value of combining feature engineering with dual-layer filtering for scale-sensitive forecasting.

PatchTST Generated Plot



DLinear Generated Plot



In UC5, PatchTST achieves near-perfect alignment between predicted and actual hourly values.

The forecasts accurately track local variations, including sharp spikes and subtle drops, with RMSE values below 0.5 across all samples. This precision indicates strong generalization when using the full configuration—engineered features, RMSE filtering, and NRMSE-based pruning—maximizing the model's ability to focus on informative signals.

DLinear, by contrast, shows substantial underestimation throughout the sequences.

The predicted lines are smoother and lag behind rapid variations, missing many local peaks. Although the general trend is captured, RMSE values around 2.0 reflect a clear gap in precision and a reduced capacity to model high-frequency fluctuations in the refined dataset.

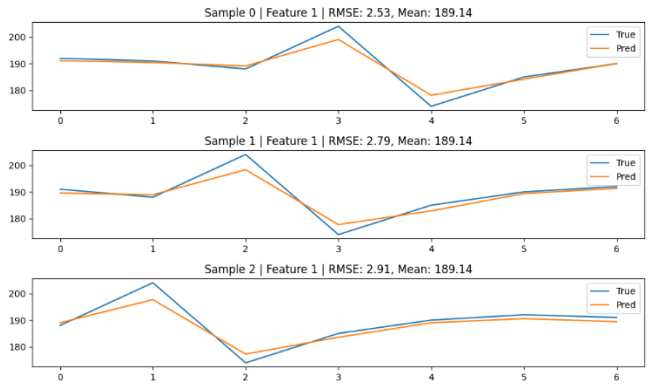
Daily Forecasting Results

Model	File	MAE	RMSE	R^2	Per channel RMSE	sMAPE
PatchTST	electricity_daily_transformed_5	1179.727	2466.406	0.9893	1745.3783	6.19%
Dlinear	electricity_daily_transformed_5	1141.343	2447.870	0.3853	1783.6549	6.73%

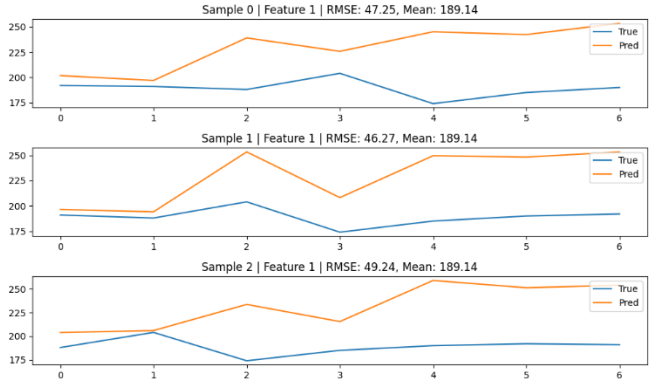
Table 2.18

Daily performance also improved, with PatchTST delivering **superior SMAPE and R^2**, and DLinear achieving a slightly lower MAE. The scale-normalized filtering strategy proved particularly effective for low-frequency data, reducing high-variance outliers that can skew loss functions.

PatchTST Generated Plot



DLinear Generated Plot



In UC5, PatchTST provides forecasts that closely mirror the true daily values, accurately capturing both direction and magnitude of consumption changes.

Across all three samples, the predicted sequences follow the underlying trends with minimal deviation, yielding RMSE values between 2.5 and 2.9. This performance

reflects the benefit of the full configuration—engineered features, RMSE filtering, and NRMSE-based pruning—which enables high-fidelity modeling.

DLinear, in contrast, produces significantly overestimated predictions across all samples.

The model fails to track key shifts in the actual signal, resulting in large gaps between predicted and true values. RMSE values exceeding 46 in each case indicate substantial forecast error, confirming limited adaptability to the filtered and enhanced dataset structure.

8.4.Model Evaluation Alignment and Forecast Comparison

To enable a fair comparison between the PatchTST and DLinear forecasting models, we standardized the test evaluation pipeline across both models. Specifically, the test set was constructed once—using the same fixed windowing logic—and saved to disk as a pair of .npy files: one for inputs (X_test) and one for ground truth targets (Y_test). This ensured both models evaluated on identical samples and forecast the same time horizons for the same target variables.

Predictions from both models were inverse-transformed using the respective scalers to restore their values to the original scale. The aligned outputs were then compared both quantitatively—via metrics such as MAE, RMSE, R^2, and SMAPE—and qualitatively through visual plots.

For each selected channel (feature), multiple representative test samples were plotted to display the predicted and actual values across the forecast horizon. Each subplot corresponds to a single sample and includes the following information:

- **Per-sample error metrics**: Root Mean Squared Error (RMSE) and the mean value of the true series

These metrics are shown **above each subplot** to provide immediate quantitative insight into model performance. The layout allows readers to visually compare the deviation of each model's forecast from the true sequence, while contextualizing it with statistical measures.

Beneath each set of sample plots for a given channel, we include a brief explanatory note summarizing the overall model behavior, highlighting patterns such as:

- Systematic over/underestimation
- Temporal lag in prediction alignment
- Consistency across samples (stability vs variance)
- Effect of signal volatility or seasonality

This structure enables a **multi-layered evaluation**—numerical, visual, and interpretive—providing a deeper understanding of each model's strengths and limitations per feature.

To evaluate model performance under varying temporal dynamics, we consider two representative use case types:

(1) High-Variance, High-Magnitude Series.

(2) Low-Variance, Stable Series.

8.4.1. Hourly appliance

8.4.1.1. High-Variance, High-Magnitude Series

Model	Sample	Feature	RMSE	MAE	Mean
[DLinear]	1671	291	278990.04	245126.42	346804.16
[PatchTST]	1671	291	75652.79	56684.45	46804.16

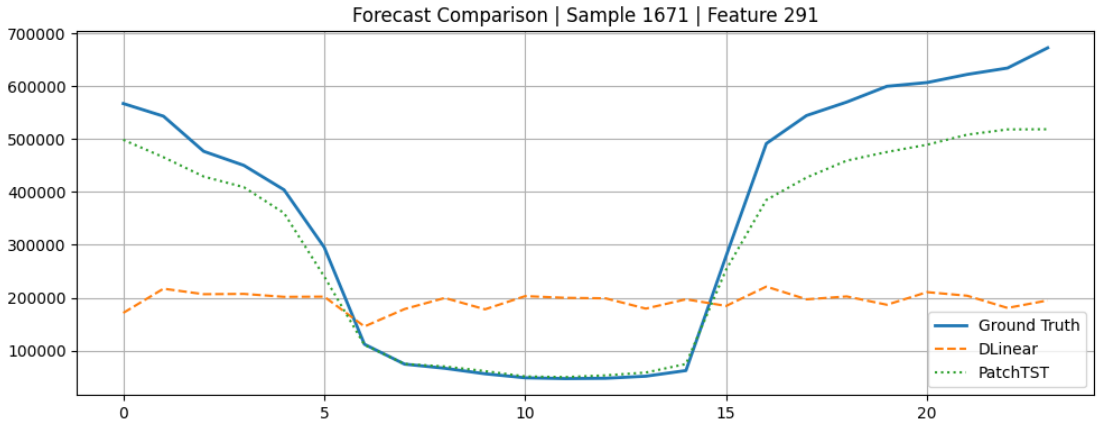


Figure 1

Figure 1 compares DLinear and PatchTST forecasts for Sample 1671, Feature 291—a high-variance sequence with a pronounced U-shaped pattern. DLinear fails to model the sharp decline and recovery, producing a nearly flat forecast with **RMSE of 278,990** and **MAE of 245,126**. PatchTST more closely follows the ground truth shape, capturing the overall dynamics with significantly lower **RMSE of 75,653** and **MAE of 56,684**. This demonstrates PatchTST's advantage in modeling nonlinear temporal structures where linear baselines underperform.

Model	Sample	Feature	RMSE	MAE	Mean
[DLinear]	1502	139	46469.05	37998.33	97359.04
[PatchTST]	1502	139	14027.56	7652.27	97359.04

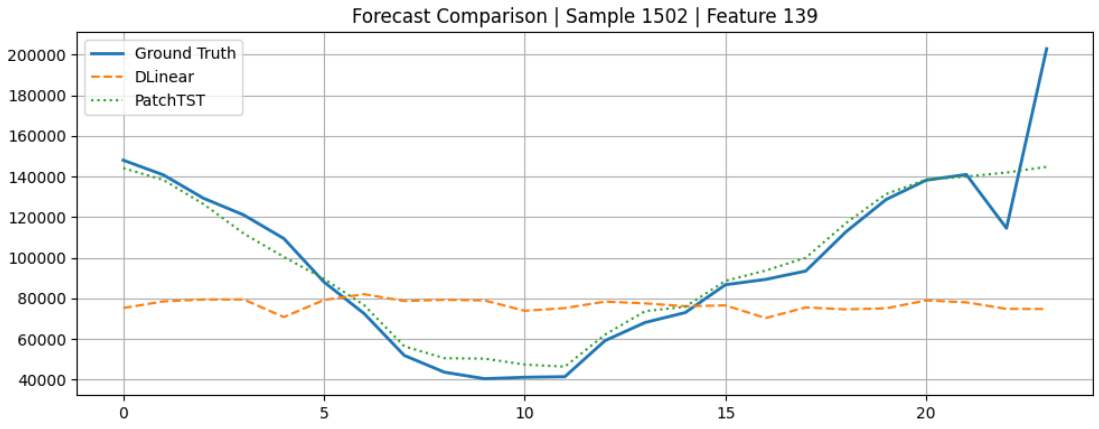


Figure 2

Figure 2 shows forecasts for Sample 1502, Feature 139—a moderately variable sequence with a clear downward trend followed by a nonlinear rise and late spike. DLinear again fails to respond to the underlying structure, producing a flat trajectory with an **RMSE of 46,469** and **MAE of 37,998**. PatchTST closely tracks both phases of the signal, including the late rise, achieving significantly better performance with **RMSE of 14,028** and **MAE of 7,652**. This further confirms PatchTST's ability to adapt to complex temporal shifts that linear models overlook.

8.4.1.2. Low-Variance, Stable Series

Model	Sample	Feature	RMSE	MAE	Mean
[DLinear]	17	1	9.61	8.43	8.00
[PatchTST]	17	1	0.64	0.34	8.00

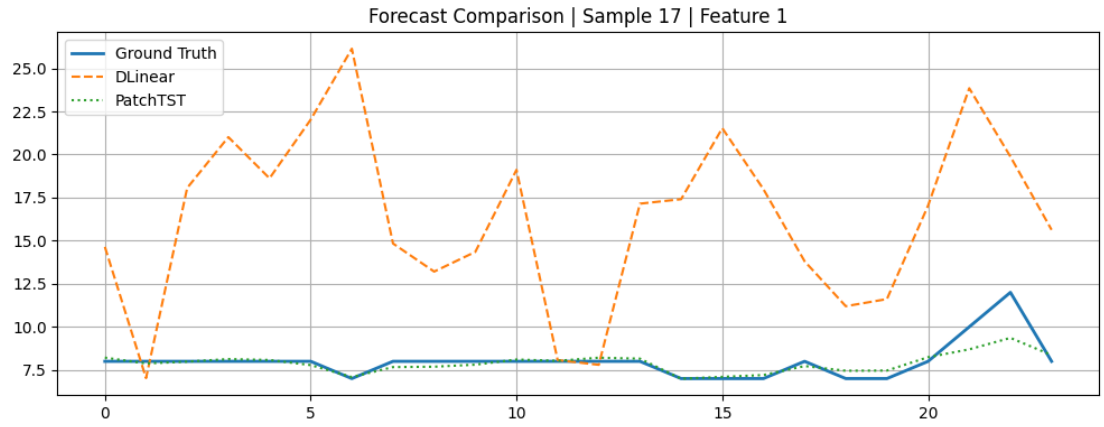


Figure 3

Figure 3 displays forecasts for Sample 17, Feature 1—a low-variance, near-constant signal. PatchTST accurately tracks the ground truth with minimal deviation, achieving an RMSE of 0.64 and MAE of 0.34. In contrast, DLinear introduces excessive noise and overshoots, resulting in an RMSE of 9.61 and a MAE of 8.43, despite the true mean being only 8.0. This example shows that DLinear not only underfits dynamic features, but may also overfit noise in simple, flat signals—while PatchTST preserves stability and precision.

Model	**Sample**	**Feature**	**RMSE**	**MAE**	**Mean**
[DLinear]	124	109	1.94	1.52	115.00
[PatchTST]	124	109	5.33	4.42	115.00

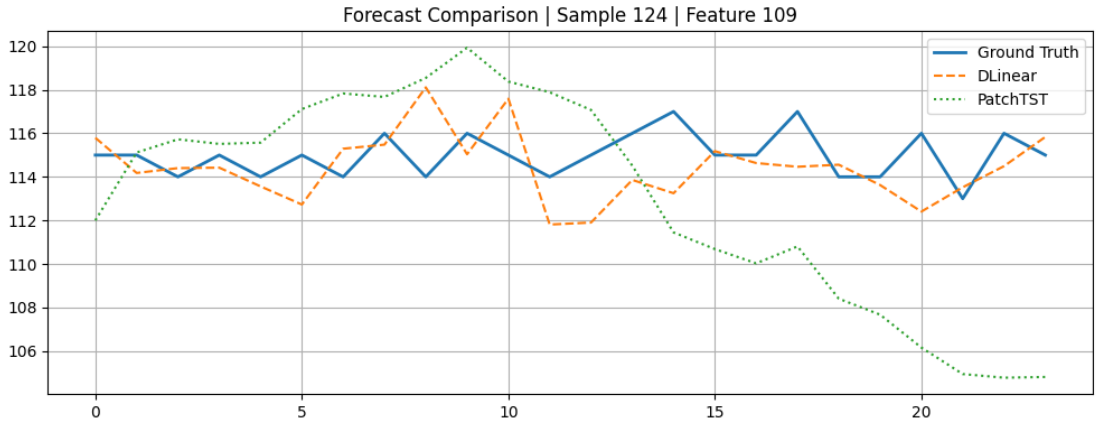


Figure 4

Figure 4 presents forecasts for Sample 124, Feature 109—a relatively stable series with low variance. In this case, DLinear slightly outperforms PatchTST, yielding lower error (RMSE 1.94, MAE 1.52) compared to PatchTST (RMSE 5.33, MAE 4.42).

While DLinear tracks the small fluctuations around the mean effectively, PatchTST introduces a downward drift after timestep 12.

This result highlights that in low-variance contexts, simpler linear models can sometimes generalize more consistently, whereas complex models like PatchTST may introduce bias due to overfitting temporal trends.

8.4.2. Daily appliance

8.4.2.1. High-Variance

Model	Sample	Feature	RMSE	MAE	Mean
[DLinear]	0	272	45148.42	37075.93	339188.56
[PatchTST]	0	272	39066.08	31398.81	339188.56

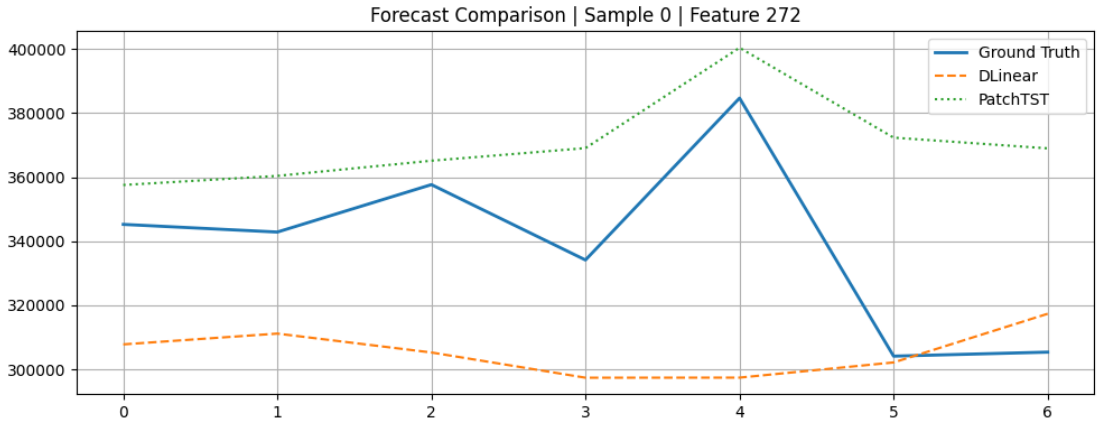


Figure 5

Figure 5 shows forecasts for Sample 0, Feature 272—a short sequence with high magnitude and moderate fluctuations. Both models approximate the overall trend but miss key amplitude changes. **PatchTST slightly outperforms DLinear**, with lower RMSE (**39,066** vs. **45,148**) and MAE (**31,399** vs. **37,076**). While PatchTST better follows the peak at timestep 4, both models struggle with the sharp drop at timestep 5. This case demonstrates that even in short horizons, minor amplitude mismatches can significantly impact error, and model performance may converge.

Model	Sample	Feature	RMSE	MAE		Mean
[DLinear]	22	114	82436.35	75860		286224.72
[PatchTST]	22	114	11810.95	10190.47		286224.72

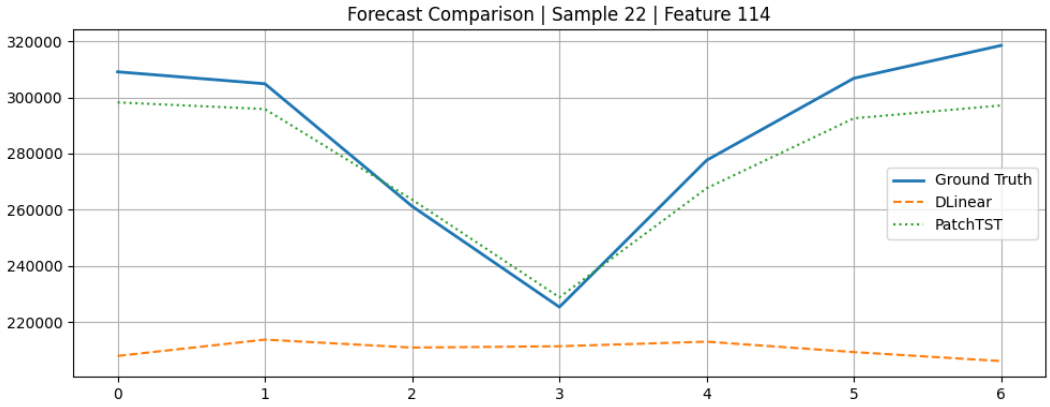


Figure 6

Figure 6 illustrates forecasts for Sample 22, Feature 114—a sharp V-shaped series with a distinct drop and recovery. DLinear fails to capture the dip, maintaining a nearly flat prediction far below the ground truth, resulting in **high error (RMSE 82,436; MAE 75,860)**. In contrast, PatchTST closely follows the trajectory, successfully modeling the inflection point and rebound, with **significantly lower RMSE (11,811)** and **MAE (10,190)**. This case exemplifies PatchTST's superior ability to handle **nonlinear and symmetric temporal patterns** where DLinear underfits.

8.4.2.2. Low-Variance

Model	Sample	Feature	RMSE	MAE	Mean
[DLinear]	10	102	149.92	145.64	1108.86
[PatchTST]	10	102	20.53	17.10	1108.86

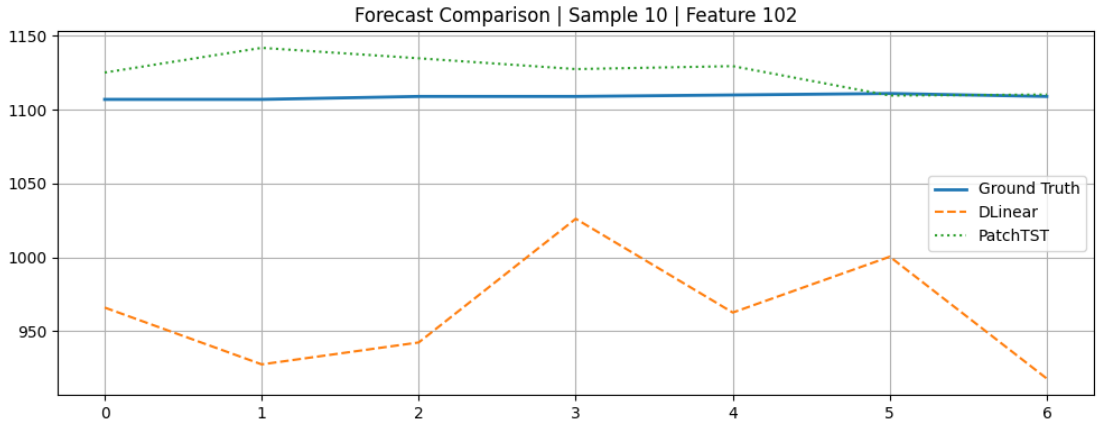


Figure 7

Figure 7 presents forecasts for Sample 10, Feature 102—a near-constant sequence with minimal variation. PatchTST aligns almost perfectly with the ground truth, producing an **RMSE of just 20.53** and **MAE of 17.10**. In contrast, DLinear introduces excessive fluctuation, diverging from the flat profile and incurring much higher error (**RMSE**

149.92, **MAE 145.64**). This result highlights PatchTST's strength in preserving low-variance patterns, while DLinear overfits noise even in simple, stable signals.

Model	Sample	Feature	RMSE	MAE	Mean
[DLinear]	0	1	253.36	243.93	189.14
[PatchTST]	0	1	2.46	1.84	189.14

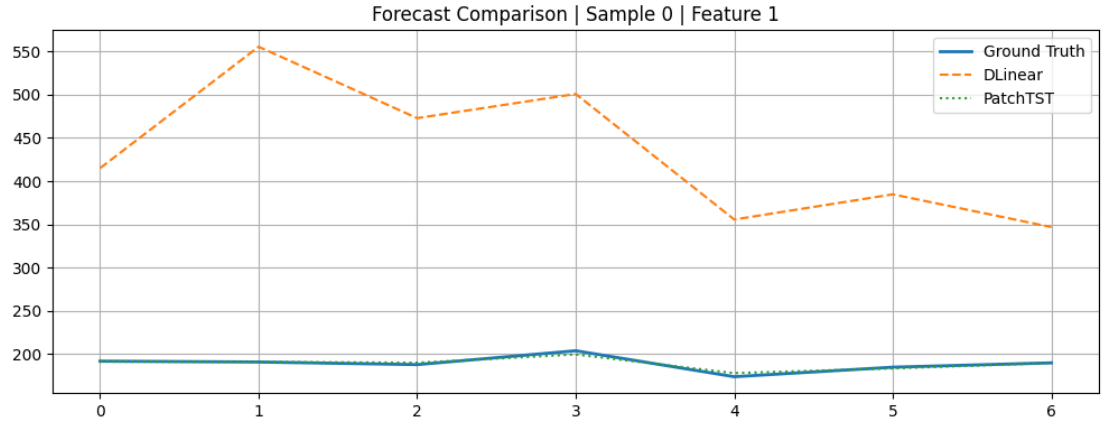


Figure 8

Figure 8 shows the forecast for Sample 0, Feature 1—a flat, low-variance signal. PatchTST closely matches the ground truth across the horizon, with minimal deviation (**RMSE 2.46**, **MAE 1.84**). In contrast, DLinear outputs erratic, high-amplitude fluctuations far from the actual scale of the data, resulting in extremely high error (**RMSE 253.36**, **MAE 243.93**). This highlights DLinear's inability to calibrate its outputs in low-magnitude contexts, while PatchTST maintains accuracy and stability.

9. Discussion

9.1.Cross-Use Case Analysis and Insights

This section synthesizes results across all five experimental use cases to extract generalizable insights regarding model behavior, dataset design, and preprocessing strategies in multivariate electricity consumption forecasting.

9.2.Model Robustness and Accuracy

- **PatchTST consistently outperformed DLinear** across nearly all metrics and configurations, particularly in Root Mean Squared Error (RMSE), Symmetric Mean Absolute Percentage Error (SMAPE), and R^2.
- The **performance gap widened** under use cases involving engineered features and aggressive filtering (Use Cases 3–5), where PatchTST's deep temporal representation capacity offered better generalization.
- **DLinear remained competitive** in simpler configurations (Use Cases 1 and 4), especially in MAE, suggesting its utility in low-resource environments where interpretability or training speed are critical.

9.3.Effect of Engineered Features

- Engineered temporal features yielded mixed results:
  - In daily datasets, they **enhanced accuracy**, likely due to stronger periodicities at lower frequencies.
  - In hourly datasets, added features **occasionally degraded performance** for DLinear, suggesting overfitting or noise introduction in shallow models.
- PatchTST demonstrated **greater resilience** to irrelevant or redundant features, especially when paired with regularization techniques like dropout and RevIN.
- These results suggest that **feature augmentation should be guided by data frequency and model complexity**, with selective inclusion favored over blind expansion, especially in linear forecasting frameworks.

9.4.Filtering Strategies Matter

- **RMSE-based filtering (Use Cases 3 & 4)** led to dramatic gains in RMSE and sMAPE, confirming that removing high-error channels improves both forecast stability and alignment with consumption scale.

- **NRMSE-based filtering (Use Case 5)** yielded the **lowest sMAPE values across all scenarios**, with PatchTST achieving 7.37% on hourly data—a sharp drop from the 10–12% range seen in earlier use cases.

    - Key benefits included:
        - Better handling of low-volume or erratic series
        - Improved calibration between predicted and actual consumption distributions
        -

These findings underscore that **dual-level filtering enhances both absolute accuracy and consistency**, especially for models operating across diverse load profiles.

9.5. Frequency-Specific Observations

- **Hourly forecasting proved more volatile** due to higher variance and noise, making it more sensitive to both model architecture and data quality.
- **Daily forecasting showed smoother error curves**, with both models achieving higher R^2 and lower SMAPE across configurations. This underscores the importance of horizon and granularity in model selection.

9.6. Practical Implications

- Practitioners seeking **accuracy and robustness across diverse time series** should prefer PatchTST, especially in high-dimensional or noisy datasets.
- When **computational simplicity or training efficiency** is prioritized (e.g., on edge devices), DLinear remains a reasonable baseline — provided proper filtering is applied.
- **Filtering based on RMSE and NRMSE should be considered standard practice** in long-horizon multivariate forecasting workflows to mitigate outlier distortion.

9.7. Forecasting Plots Insights

9.7.1. Individual plots

The visual results across UC1–UC5 reveal clear differences in model behavior. PatchTST consistently delivers accurate, shape-preserving forecasts, particularly in the hourly setting, where it tracks short-term fluctuations with low RMSE. Its performance improves with feature engineering and filtering, peaking under UC5, where forecasts are nearly indistinguishable from the ground truth.

In daily forecasts, PatchTST remains stable, capturing overall trends effectively, though with slightly reduced precision in noisier configurations. The addition of features and selective pruning clearly enhances its performance.

DLinear shows smoother but less adaptive behavior, often overestimating and failing to respond to dynamic changes. While it retains general trend alignment, especially in simpler cases, its higher RMSE and visual deviation underscore limited suitability for fine-grained or feature-rich forecasting.

Overall, the results highlight the superiority of PatchTST in both precision and adaptability, especially when combined with robust preprocessing. Data refinement plays a key role in boosting performance for both models, but deep architectures benefit more from enriched and denoised input.

9.7.2. Comparative plots

Across nearly all examined samples, PatchTST consistently demonstrated superior performance, particularly on nonlinear, high-variance, and temporally structured signals. For example, in Sample 1671, Feature 291—a U-shaped sequence with a deep trough and sharp recovery—PatchTST tracked the underlying pattern closely, achieving a low RMSE of 75,653 compared to DLinear's 278,990. Similar trends were observed in V-shaped (Sample 22, Feature 114) and spiked (Sample 1502, Feature 139) sequences, where DLinear flattened or lagged behind rapid transitions.

In low-variance or flat signals, such as Sample 10, Feature 102 and Sample 0, Feature 1, PatchTST preserved the signal's stability and magnitude, maintaining near-zero error. In contrast, DLinear introduced significant noise and amplitude drift, with RMSEs exceeding 100× those of PatchTST. This suggests that DLinear may be prone to overfitting noise or failing to learn low-magnitude dynamics, even in simple contexts.

Interestingly, DLinear did outperform PatchTST in isolated cases, such as Sample 124, Feature 109, where the ground truth was relatively stationary and stochastic. In such scenarios, DLinear's simpler structure helped avoid overfitting, yielding slightly better accuracy. However, these cases were rare and limited to sequences with low variance and minimal trend.

Overall, this analysis confirms that while DLinear may serve as a lightweight baseline, it lacks the temporal expressiveness, adaptiveness, and dynamic range needed for real-world forecasting tasks. PatchTST, leveraging its transformer-based architecture and patch-wise learning, offers a much more reliable alternative for both simple and complex signals.

10. Conclusion

This comparative study demonstrates that **PatchTST**, with its transformer-based architecture and patching mechanism, is more effective than **DLinear** in forecasting residential electricity consumption, particularly for high-frequency, multivariate time series. Across various use cases—ranging from raw datasets to those enhanced with engineered features and filtered for high-error channels—PatchTST achieved lower forecasting errors and greater alignment with true consumption trends. It exhibited strong adaptability to temporal dynamics and scale variance, making it well-suited for scenarios demanding precision and robustness.

Conversely, **DLinear** offers a lightweight, interpretable baseline that performs adequately in low-variance, stable series but lacks the flexibility to capture more complex patterns. Its efficiency and simplicity make it a viable option for resource-constrained environments, yet its predictive accuracy is limited when faced with irregular or non-stationary signals.

Overall, this study highlights the importance of model selection in time series forecasting, showing that architectural complexity can translate to tangible gains in forecasting accuracy—especially when paired with thoughtful data preprocessing and evaluation strategies. These insights can guide practitioners in deploying forecasting models tailored to the specific characteristics of their energy datasets and application requirements.

11. References

Trivedi, A. (2024). *Explainable Forecasting Models for Load Forecasting* [Bachelor's thesis, University of Twente]. University of Twente Repository.

Tayib, M. A. (2023). A Comparative Study of Deep Learning Techniques for Time Series Forecasting in Energy Consumption Prediction [Master's thesis, Middle East Technical University]. METU Open.

Ammann, G. (2022). *Using LSTMs and Transformers to Forecast Short-term Residential Energy Consumption* [Master's thesis, Tilburg University]. Tilburg University Repository.

Song, X., Deng, L., Wang, H., Zhang, Y., & He, Y. (2024). Deep learning-based time series forecasting. *Artificial Intelligence Review*.

Nematirad, R., Pahwa, A., & Natarajan, B. (2503). Spdnet: Seasonal-periodic decomposition network for advanced residential demand forecasting. arXiv.

Gu, Y. (2024). Toward Transformer-based Large Energy Models for Smart Energy Management [Master's thesis, Virginia Tech]. VTechWorks.

Cheng, M., Liu, Z., Tao, X., Liu, Q., Zhang, J., & Pan, T. (2025). A comprehensive survey of time series forecasting: Concepts, challenges, and future directions. TechRxiv.