Project Report On

“Sustainable Energy Balance Forecasting”

Under the guidance of Dr. Anshul Sir

Department of Computer Science & Engineering

NATIONAL INSTITUTE OF TECHNOLOGY PATNA

University Campus, Bihta - 801103

Submitted by:

Name	Roll No.
Omkar Kumar	2206010
Raju Kumar	2206012
Ravi Kumar	2206003

Course Code: CS64123

Table of Contents

1. Problem Statement

With the rise of renewable energy adoption in buildings—especially rooftop solar—managing energy efficiency requires accurate forecasting of net energy demand. This project presents a **smart building load forecasting framework** that predicts net energy demand, defined as the total building energy consumption minus on-site solar energy generation. Using historical energy data, solar generation records, and weather features, a machine learning-based model is developed to enable better grid integration, demand-side management, and operational planning for energy-efficient buildings.

2. Motivation

Rising rooftop solar adoption makes net energy demand (building consumption minus solar generation) harder to predict. Accurate forecasts are vital for grid stability, demand-side management, and cost-efficient building operations. *´Sustainability isn't a trend—it's the future. Those who solve these problems today will lead tomorrow.'* Every kilowatt-hour saved or optimally used delays the need for fossil fuels, directly reducing CO_2 emissions. This project develops a machine learning model using historical energy data, solar generation, and weather inputs to improve net demand predictions, enabling smarter energy use and better integration of renewables into the grid.

3. Introduction

The integration of solar photovoltaic (PV) systems in buildings presents both opportunities and challenges in energy management. While on-site generation reduces dependence on grid power, the variability in solar output

necessitates accurate net energy demand forecasting for optimizing building operations and supporting the smart grid. This project addresses the need for intelligent forecasting systems capable of accounting for both consumption and renewable generation dynamics.

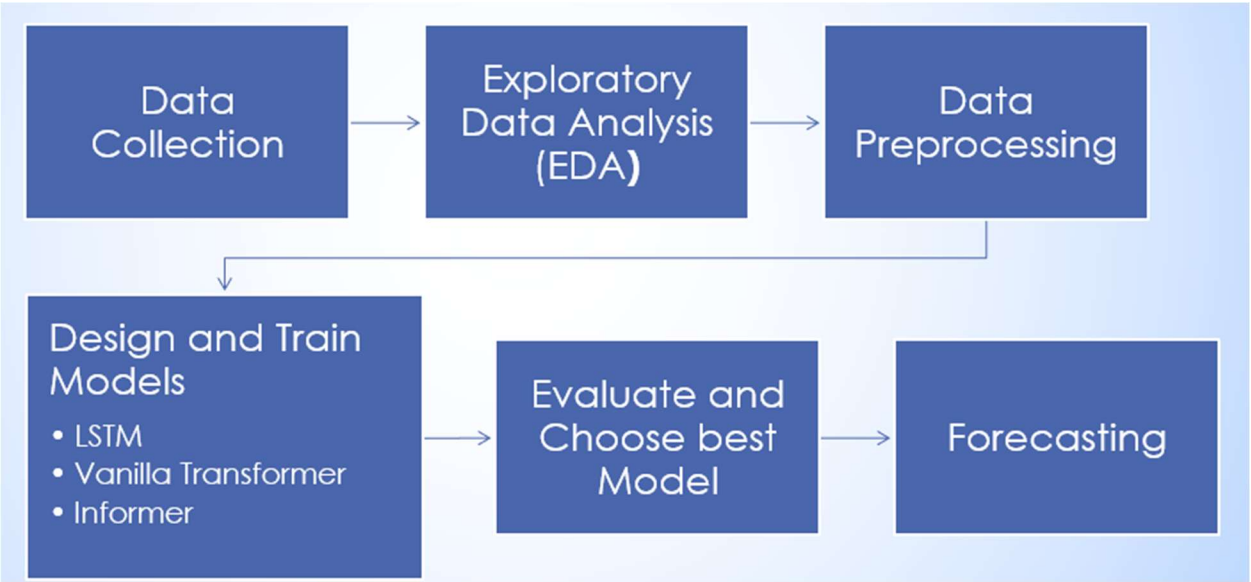
Traditional load forecasting focuses solely on consumption, neglecting the impact of renewable sources like solar. The goal of this project is to predict net energy demand:

Net Demand = Total Consumption - Solar Generation

This forecasting helps in:

- Efficient scheduling of HVAC and battery storage systems.
- Minimizing grid import/export costs.
- Enhancing demand response strategies.

4. Proposed Methodology



4.1. Data Collection

- The data was collected from a detached house near Tallinn University.
- It is a two-story building with a total area of 130 m^2 that is used as a main residence by a single family with two adults and two children.

	Time	Total Load	Total PV Gen
0	2023-01-01 00:00:10	3.280	0.003
1	2023-01-01 00:00:20	3.284	0.003
2	2023-01-01 00:00:30	3.285	0.003
3	2023-01-01 00:00:40	3.332	0.003
4	2023-01-01 00:00:50	3.416	0.003

- Data collection period: Full year of 2023.
- Resolution: Every 10 seconds.
- Includes power usage, solar PV generation and timestamp.

4.2. Exploratory Data Analysis

- Created a pandas data frame.
- Get the shape of data , in our case (3153600,3).
- Get the statistics like mean, standard deviation, quartiles, etc.
- Check if null values present, no null values in our case.
- Check if data is skewed, not skewed in our case.

	Total Load	Total PV Gen
count	3.153600e+06	3.153600e+06
mean	1.726341e+00	5.593550e-01
std	1.497246e+00	1.084651e+00
min	3.200000e-02	-3.100000e-02
25%	2.760000e-01	2.000000e-03
50%	1.657000e+00	3.000000e-03
75%	2.431000e+00	5.230000e-01
max	1.294600e+01	4.266000e+00

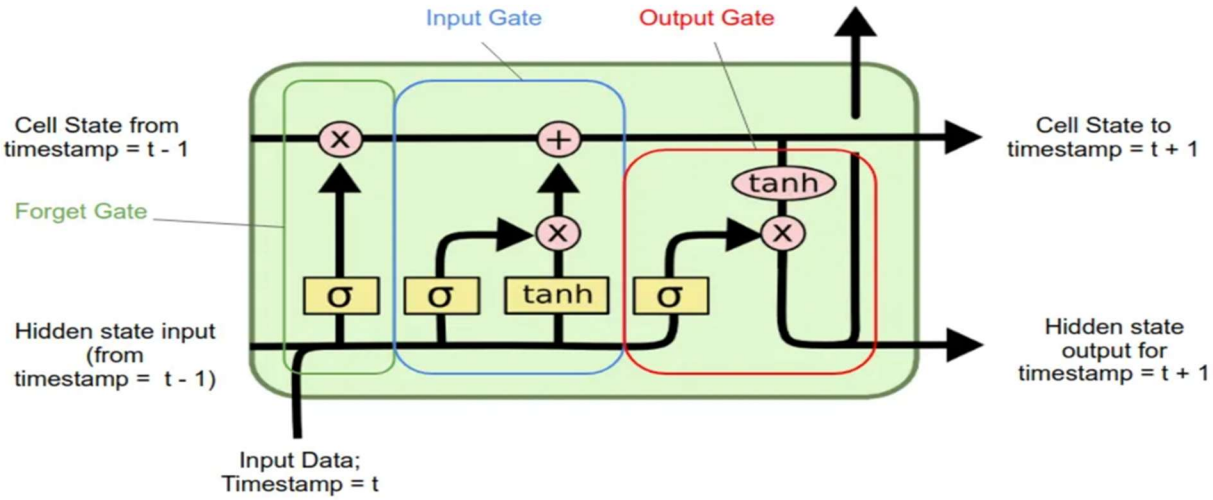
4.3. Data Preprocessing

- Added a Column ‘Net Load’ = ‘Total Load’ – ‘Total PV generated’

- Normalization (Min-Max normalization is used).
- Transform raw time-series data to proper format to make it compatible of feeding to models
- Train-Test split

4.4. Modeling Techniques

4.4.1 LSTM



1. LSTM architecture has a chain structure that contains four neural networks and different memory blocks called cells.
2. Information is retained by the cells and the memory manipulations are done by the gates.
3. The hidden state is the observable output of the LSTM at each time step.

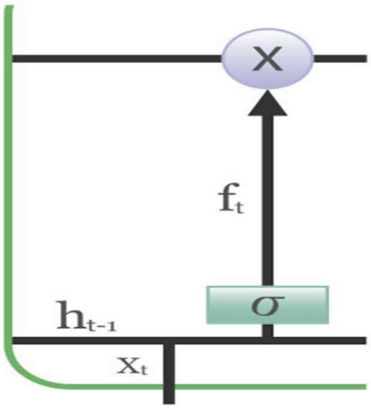
Long Short-Term Memory Networks(LSTMs)

Long Short-Term Memory Networks introduce a memory mechanism to overcome the vanishing gradient problem. Each LSTM cell has three gates:

- Input Gate: Controls how much new information should be added to the cell state.
- Forget Gate: Decides what past information should be discarded.
- Output Gate: Regulates what information should be output at the current step. This selective memory enables LSTMs to handle long-term dependencies, making them ideal for tasks where earlier context is critical.

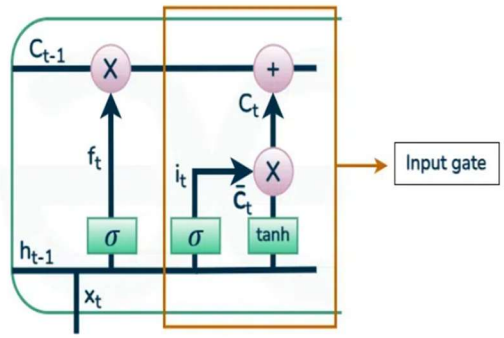
Working of LSTM- Forget Gate

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$



- The information that is no longer useful in the cell state is removed with the forget gate.
- Two inputs x_t (input at the particular time) and h_{t-1} (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias.
- Two inputs x_t (input at the particular time) and h_{t-1} (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias.
- If for a particular cell state the output is 0, the piece of information is forgotten and for output 1, the information is retained for future use.

Working of LSTM Input Gate



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

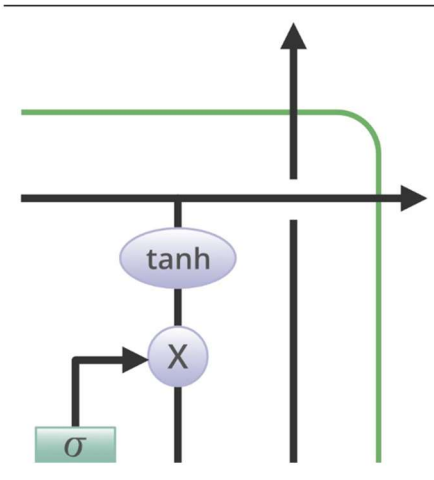
$$\hat{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t$$

- The addition of useful information to the cell state is done by the input gate.
- Filter the values to be remembered similar to the forget gate using inputs h_{t-1} and x_t. .
- Then, a vector is created using *tanh* function that gives an output from -1 to +1, which contains all the possible values from h_{t-1} and x_t.

- At last, the values of the vector and the regulated values are multiplied to obtain the useful information.
- We multiply the previous state by f_t, disregarding the information we had previously chosen to ignore. Next, we include $i_t * C_t$.

Working of LSTM Output Gate



$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

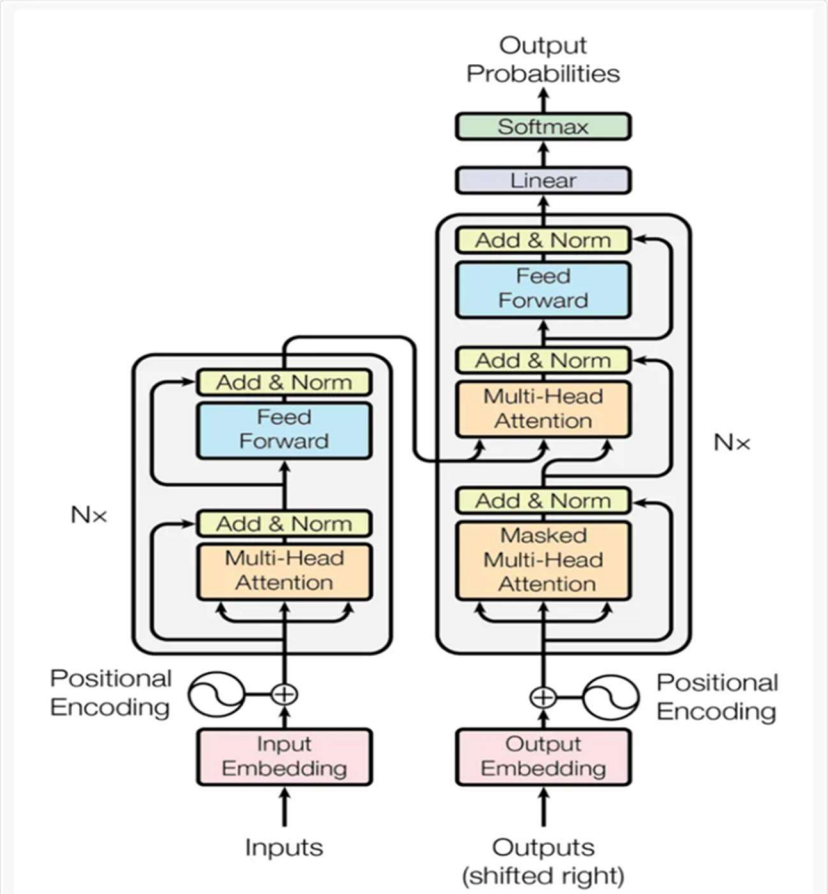
- The task of extracting useful information from the current cell state to be presented as output is done by the output gate.
- First, a vector is generated by applying tanh function on the cell.
- Then, the information is regulated using the sigmoid function and filter by the values to be remembered using inputs h_{t-1} and x_t.

- At last, the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell.

Why Use LSTM?

- Handles long-term dependencies better than traditional RNNs.
- Useful for tasks like speech recognition, language modeling, machine translation, and time-series forecasting.
- Solves the Vanishing Gradient Problem.
- Better Memory Retention.
- Selective Information Flow.
- Handles Long Sequences Efficiently.
- Reduces Training Time and Improves Stability.
- Improved Context Understanding

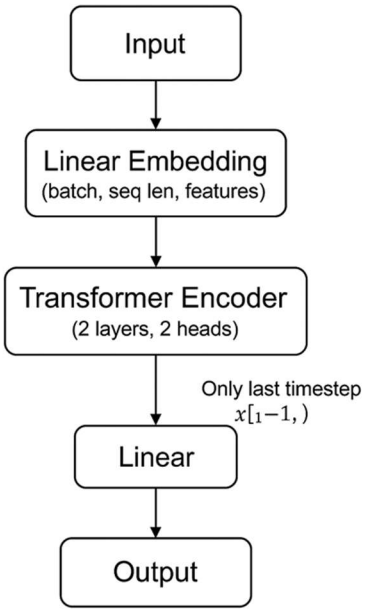
4.4.2 Vanilla Transformer



The encoder-decoder structure of the Transformer architecture
Taken from "Attention Is All You Need"

1. Uses an encoder-decoder structure for sequence-to-sequence tasks.
2. Self-attention mechanism allows each token to focus on all others.
3. Multi-head attention captures different types of relationships in parallel.
4. Positional encoding injects order information into input embeddings.
5. Feedforward layers apply transformations after attention blocks.

Design



1. **Linear Embedding :** Projects each timestep's scalar feature (like a single value per time) into a higher-dimensional space.
2. **Transformer Encoder :** A stack of encoder layers (in this case, 2 layers, 2 self attention heads) that use:
    a. Multi-head attention to find dependencies in the sequence.
    b. Feedforward neural nets to enhance representation power.
    c. Positional encoding (implicitly handled in PyTorch or can be added) to inform order in the sequence.
3. **Linear :** Takes the d_model dimensional vector from the last timestep and maps it to a single output value.

Working of Vanilla Transformer Architecture

1.Encoder:

- Each encoder layer consists of two main sub-layers:
  - Multi-Head Self-Attention Mechanism
  - Feedforward Neural Network (FFN)
- Each sub-layer is followed by:
  - Residual Connection: Adds the original input to the output of the sub-layer.
  - Layer Normalization

2. Decoder:

- The decoder generates the output sequence, one token at a time, using the encoded input representation and previously generated tokens.
- Each **decoder layer** contains three sub-layers:
  - **Masked Multi-Head Self-Attention**: Prevents the model from attending to future tokens.
  - **Encoder-Decoder Attention**: Attends to the encoder's output.
  - **Feedforward Neural Network (FFN)**
- Residual connections and layer normalization are also used here.

3. Positional Encoding

- Since the Transformer does not use recurrence or convolutions, it requires positional encodings to inject information about the order of the sequence.

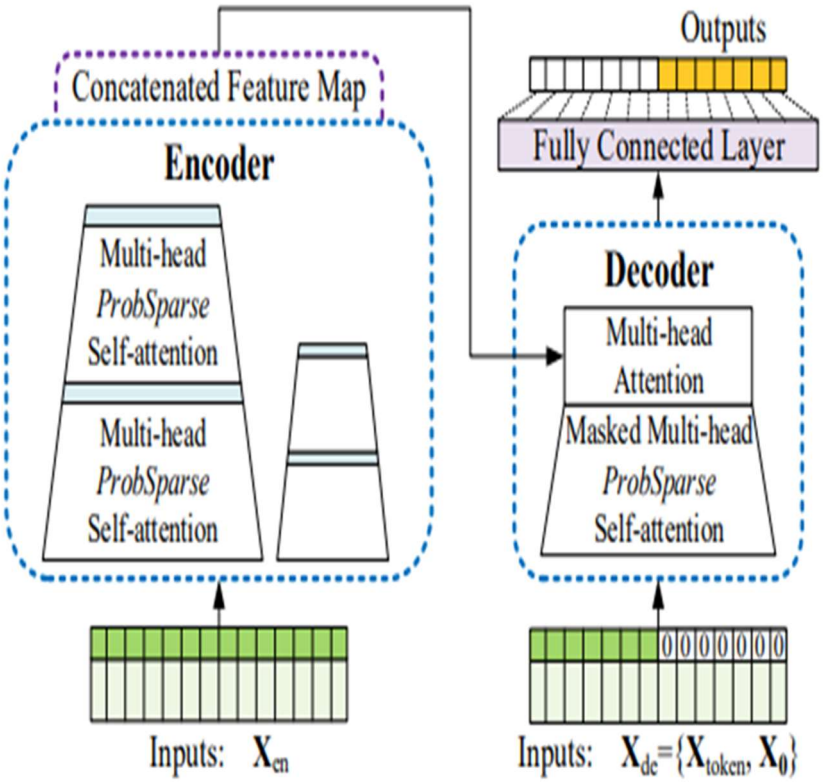
4. Output

- The final decoder output is passed through a **linear layer** followed by a **softmax** to generate the probability distribution over the vocabulary.

Why use Vanilla Transformer?

- **Captures Long-Term Dependencies** – Self-attention allows modeling of distant time relationships better than LSTM.
- **Parallel Processing** – Processes all time steps at once, speeding up training and inference.
- **Handles Multiple Inputs** – Easily incorporates weather, solar, and consumption data together.
- **No Recurrence Needed** – Avoids the limitations of sequential models like RNNs and LSTMs.
- **Scalable Architecture** – Easy to scale with more layers or features as data grows.
- **Flexible with Sequence Lengths** – Can handle varying time series lengths without performance loss.
- **Better Feature Attention** – Multi-head attention focuses on different aspects of the input simultaneously.
- **Proven Performance** – Outperforms traditional models in many real-world forecasting tasks.

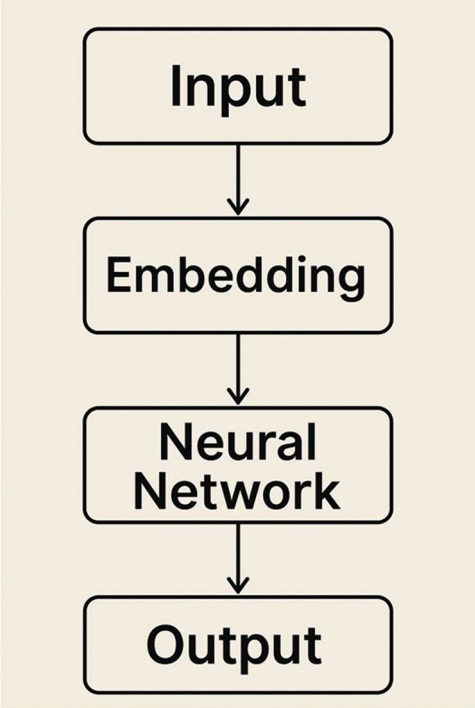
4.4.3. Informer



- **Encoder:** The encoder structure includes multiple layers of multi-head attention, which helps the model to understand complex patterns in the data.
- **Multi-head Attention:** Captures various aspects of the relationships in the data.
- **ProbSparse Attention:** Efficient attention mechanism for long sequences.
- **Self Attention:** Capture dependencies in the input.
- **Concatenated Feature Map:** It strengthens the model's ability to capture and utilize the context learned during the encoding phase.

- **Decoder:** The decoder is responsible for generating the output sequence based on the processed information from the encoder.
- **Masked multi head:** Ensures that during training, the decoder cannot access future tokens, which is essential for sequence generation.
- **Fully Connected Layer:** This layer takes the output from the decoder and transforms it into the final output format.

Design



1. **Input :** This is the raw data fed into the model.
2. **Embedding :** Converts input into dense vector representations.
3. **Neural Network :** core computation engine.
4. **Output :** Final prediction or result of the network.

Working of Informer Architecture

1. Encoder:

The encoder processes historical input data and extracts long-term temporal patterns using:

- **ProbSparse Self-Attention:** Reduces computational complexity by selecting only the most "informative" queries based on their importance. This dramatically lowers time and space requirements without losing key temporal information.
- **Self-Attention Distillation:** Progressively compresses long sequence representations into shorter sequences by selecting important time steps. This reduces redundancy while preserving critical features.

2. Decoder:

The decoder generates future predictions using:

- **Full Attention:** Focuses on relevant encoder outputs while maintaining autoregressive capabilities.
- **Generative Inference:** Allows direct multi-step forecasting without requiring iterative predictions, improving speed and stability for long horizons.

3. Embedding Layer:

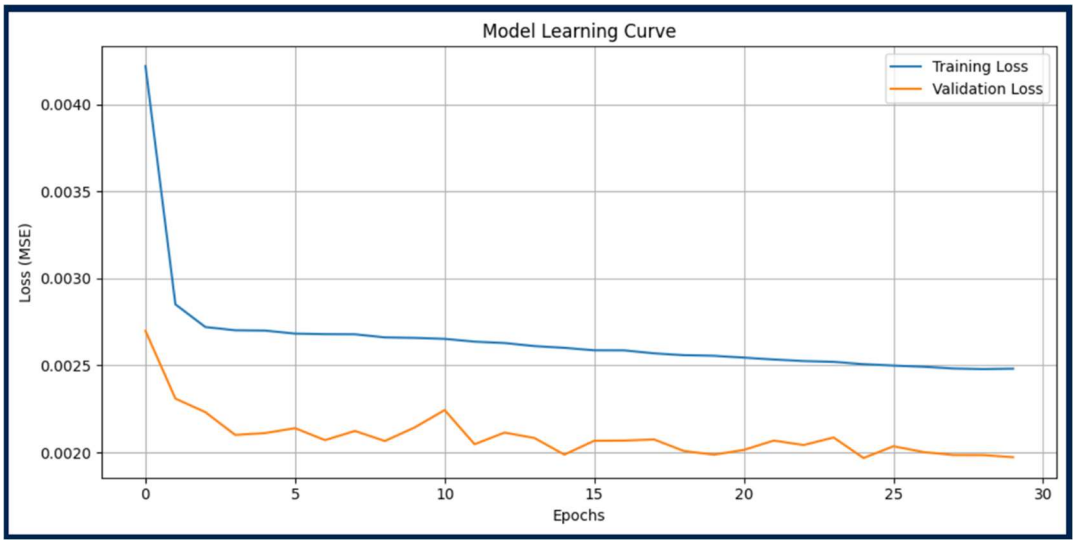
- Time features like *timestamp*, *day of the week*, *holiday flags* are embedded along with input features.
- Positional encoding helps maintain temporal order.

Why use Informer?

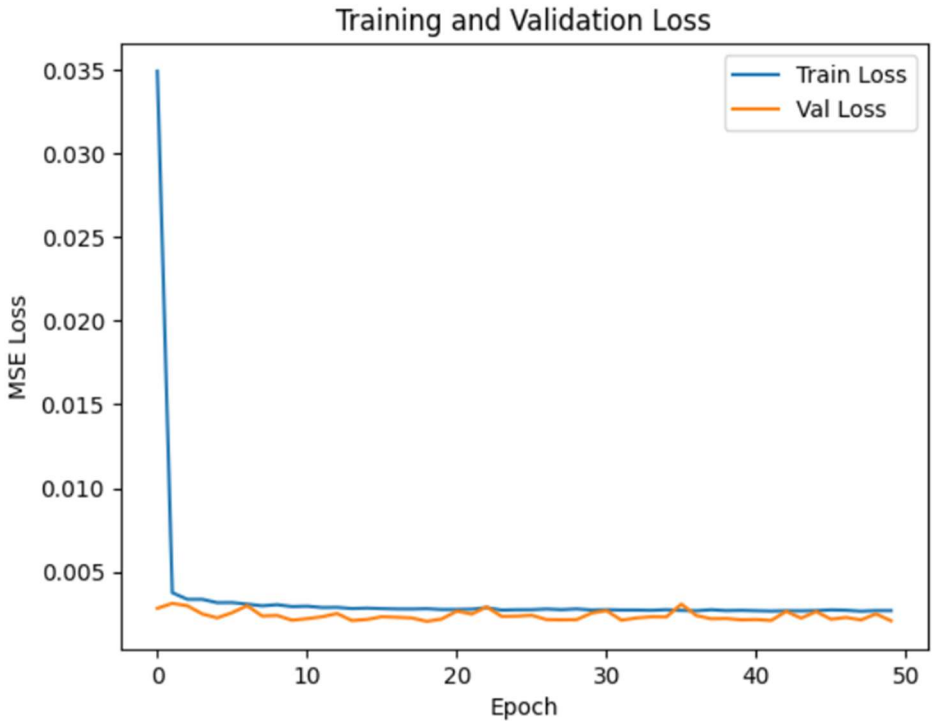
- **Efficient Long-Term Forecasting:** Informer handles very long time-series sequences with lower memory and computational cost than vanilla Transformers.
- **ProbSparse Attention:** Reduces attention complexity from $O(n^2)$ to $O(n \log n)$, making it scalable for large datasets.
- **Self-Attention Distillation:** Compresses input sequences by removing redundancy, helping the model focus on key patterns.
- **Direct Multi-Step Prediction:** Generates future time steps all at once instead of one-by-one, improving speed and accuracy for long-term forecasting.
- **Better Temporal Pattern Recognition:** Captures both short- and long-range dependencies—ideal for modeling seasonal or daily energy trends.
- **High Forecasting Accuracy:** Outperforms RNN, LSTM, and even vanilla Transformer models on many time-series benchmarks.
- **Supports Multivariate Inputs:** Handles multiple correlated features like energy consumption, solar power, and weather data effectively.
- **Practical for Real-Time Systems:** Faster inference and low overhead make it suitable for deployment in smart building energy management systems.

5. Results

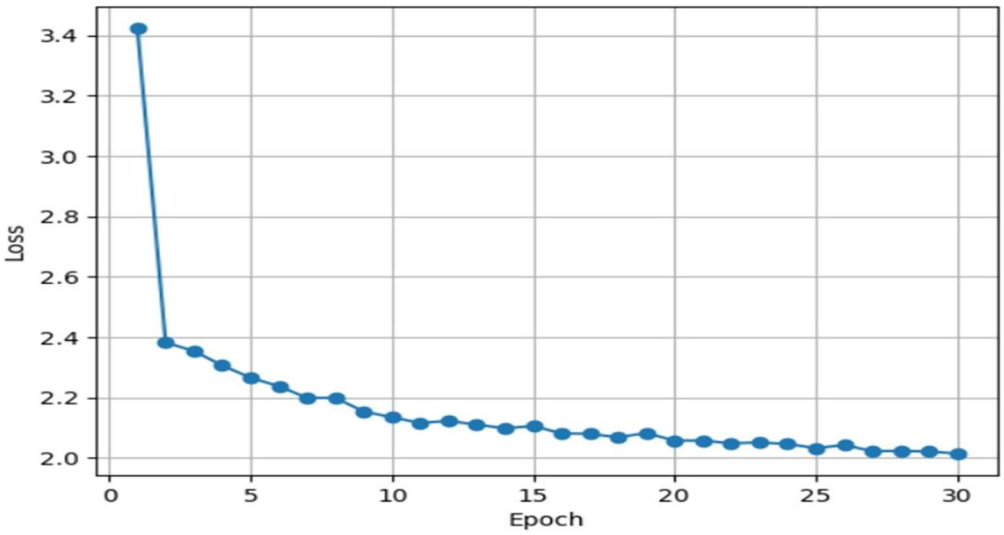
- **Learning Curve:**
  - **LSTM**



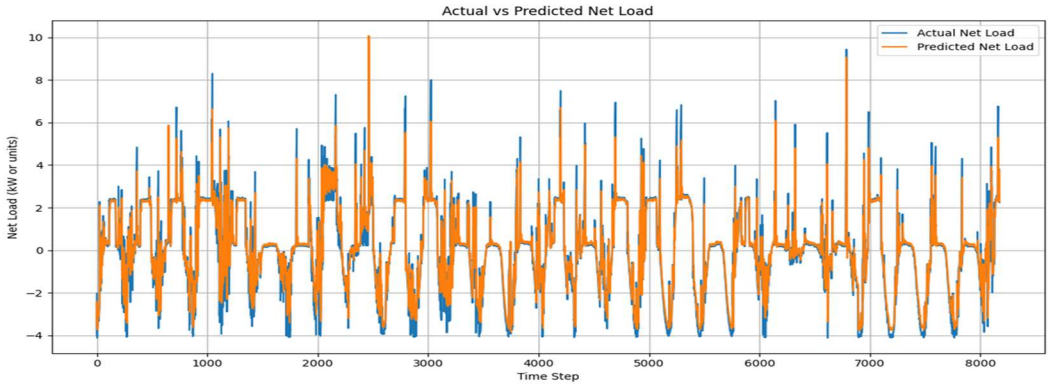
  - **Vanilla Transformer**



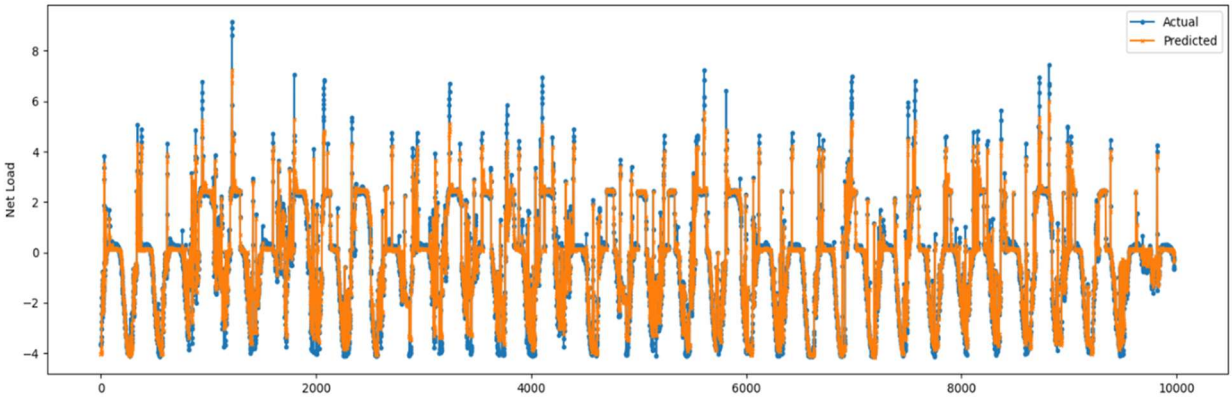
- **Informer**



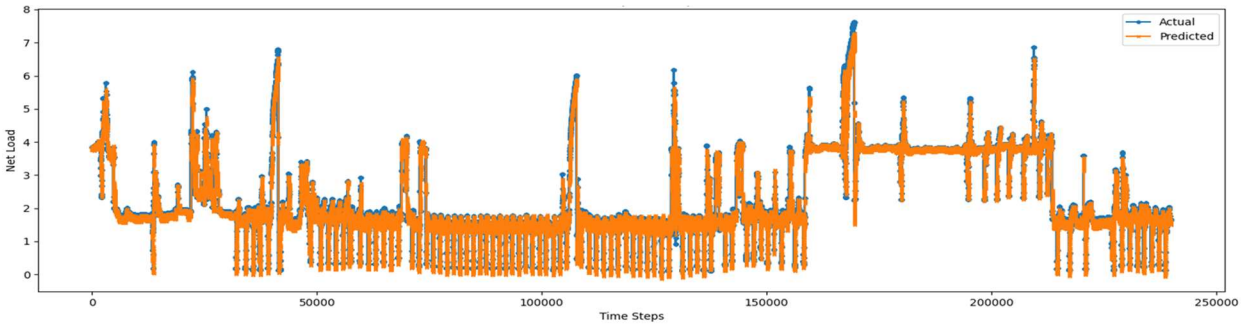
- **Model evaluation**
  - **LSTM**



  - **Vanilla Transformer**



- **Informer**



- **Comparisons**

Metric	LSTM	Vanilla Transfer	Informer
MAE	0.39	0.3814	0.2140
MSE	0.72	0.5483	0.1795
RMSE	0.84	0.7405	0.4326
R^2 Score	0.8729	0.8731	0.8829

6. Discussion

The forecasting of net energy demand in smart buildings—defined as the difference between energy consumption and solar generation—presents several unique challenges and opportunities. This project explored the use of Transformer-based models, including the vanilla Transformer and the Informer, for addressing these challenges effectively.

- **Model Performance and Insights:**
  - The use of Transformer models showed promising results in terms of accuracy and flexibility. Their self-attention mechanism allowed the model to dynamically focus on relevant time steps and features, such as daily consumption cycles or weather-influenced solar variability. Informer further improved performance by handling longer input sequences efficiently and compressing irrelevant temporal information through attention distillation.
  - The results indicated that these models are highly suitable for multi-step forecasting, which is essential for real-time energy management. In particular, the Informer's ability to process long sequences with low latency makes it practical for deployment in systems that rely on continuous data streams from smart meters and IoT sensors.

- **Challenges Encountered:**
  - Data Quality and Availability: Accurate forecasting depends heavily on high-quality, time-aligned data from consumption logs, solar production, and weather sensors. Missing values or inconsistent timestamps required careful preprocessing.
  - Hyperparameter Tuning: Transformer models are sensitive to architecture choices (e.g., number of layers, attention

heads), and extensive experimentation was needed to find the optimal configuration.

- Computational Resources: Training deep learning models on large time-series datasets requires substantial compute power, which may be a constraint for real-time deployment in resource-limited environments.

- **Impact of Feature Engineering:**
  - The inclusion of additional features such as time-of-day, day-of-week, and weather data significantly improved the model's ability to understand and predict variations in energy usage and solar generation. This highlights the importance of contextual information in time-series forecasting for smart buildings.

This enables smarter, greener buildings that contribute to sustainability goals while minimizing operational costs.

7. References

- Solar PV Generation and Consumption Dataset of an Estonian Residential Dwelling | Scientific Data
- Attention is all you need
- https://nixtlaverse.nixtla.io/neuralforecast/models.informer.html
- https://nixtlaverse.nixtla.io/neuralforecast/models.vanillatransformer.html