Entregable 1

Alex Pérez

30 de octubre de 2024

Título del Proyecto

Machine Learning Applied for Cybersecurity of Energy Management Systems

(Aprendizaje Automático Aplicado a la Ciberseguridad del Manejo de Sistemas Energéticos)

Tutor: Felipe Grijalva

Autor: Alex Pérez

Relevancia y Justificación

La creciente adopción de fuentes de energía renovable, como paneles solares, ha ocasionado la necesidad de contar con sistemas avanzados de monitoreo y análisis para evitar la vulnerabilidad de los sistemas digitales de estas fuentes de energía a ciberataques.

La relevancia de este proyecto radica en, haciendo uso de modelos basados en redes neuronales recurrentes, garantizar la integridad de datos de generación y consumo de energía al identificar patrones anómalos para así evitar posibles manipulaciones o fraudes.

Propuesta del Proyecto

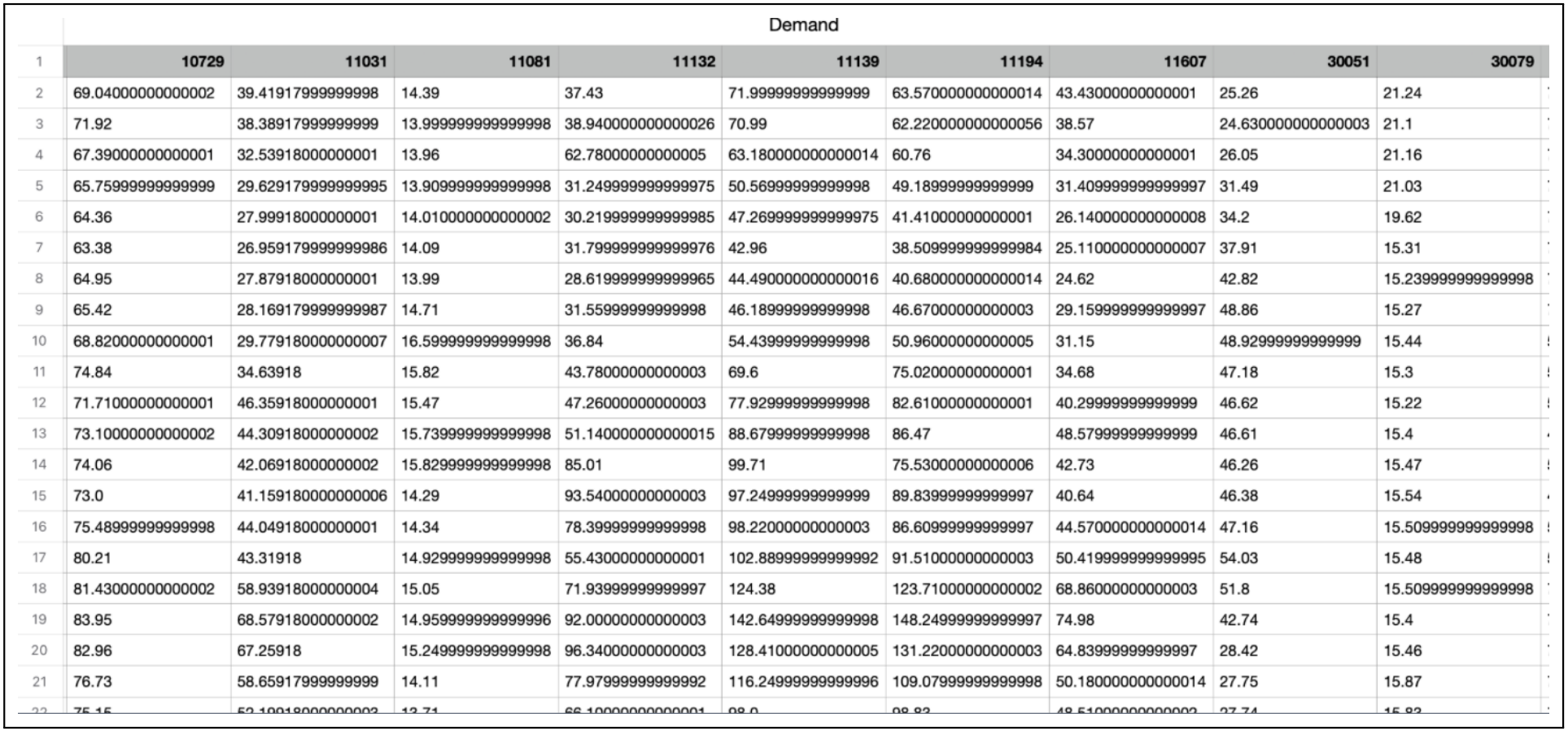
Objetivo:

- Desarrollar y evaluar modelos con distintas arquitecturas: LSTM, TCN, TCN+LSTM híbrido, y Transformers (por concluir).
- Configuraciones de modelos:
  - SISO (Single Input, Single Output).
  - MIMO (Multiple Input, Multiple Output)(por realizar).

Evaluación del Rendimiento:

- Análisis de Fourier y densidad espectral de potencia (PSD) para explorar características de la señal de demanda y generación energética.
- Métricas: Error Cuadrático Medio (MSE) y análisis de intervalos de confianza.

Visualización del Dataset Original



	Demand								
1	10729	11031	11081	11132	11139	11194	11607	30051	30079
2	69.04000000000002	39.41917999999998	14.39	37.43	71.99999999999999	63.570000000000014	43.43000000000001	25.26	21.24
3	71.92	38.38917999999999	13.999999999999998	38.940000000000026	70.99	62.220000000000056	38.57	24.630000000000003	21.1
4	67.39000000000001	32.53918000000001	13.96	62.78000000000005	63.180000000000014	60.76	34.30000000000001	26.05	21.16
5	65.75999999999999	29.629179999999995	13.909999999999998	31.249999999999975	50.56999999999998	49.18999999999999	31.409999999999997	31.49	21.03
6	64.36	27.99918000000001	14.010000000000002	30.219999999999985	47.269999999999975	41.41000000000001	26.140000000000008	34.2	19.62
7	63.38	26.959179999999986	14.09	31.799999999999976	42.96	38.509999999999984	25.110000000000007	37.91	15.31
8	64.95	27.87918000000001	13.99	28.619999999999965	44.490000000000016	40.680000000000014	24.62	42.82	15.239999999999998
9	65.42	28.169179999999987	14.71	31.55999999999998	46.18999999999998	46.67000000000003	29.159999999999997	48.86	15.27
10	68.82000000000001	29.779180000000007	16.599999999999998	36.84	54.43999999999998	50.96000000000005	31.15	48.92999999999999	15.44
11	74.84	34.63918	15.82	43.78000000000003	69.6	75.02000000000001	34.68	47.18	15.3
12	71.71000000000001	46.35918000000001	15.47	47.26000000000003	77.92999999999998	82.61000000000001	40.29999999999999	46.62	15.22
13	73.10000000000002	44.30918000000002	15.739999999999998	51.140000000000015	88.67999999999998	86.47	48.57999999999999	46.61	15.4
14	74.06	42.06918000000002	15.829999999999998	85.01	99.71	75.53000000000006	42.73	46.26	15.47
15	73.0	41.159180000000006	14.29	93.54000000000003	97.24999999999999	89.83999999999997	40.64	46.38	15.54
16	75.48999999999998	44.04918000000001	14.34	78.39999999999998	98.22000000000003	86.60999999999997	44.570000000000014	47.16	15.509999999999998
17	80.21	43.31918	14.929999999999998	55.43000000000001	102.88999999999992	91.51000000000003	50.419999999999995	54.03	15.48
18	81.43000000000002	58.93918000000004	15.05	71.93999999999997	124.38	123.71000000000002	68.86000000000003	51.8	15.509999999999998
19	83.95	68.57918000000002	14.959999999999996	92.00000000000003	142.64999999999998	148.24999999999997	74.98	42.74	15.4
20	82.96	67.25918	15.249999999999998	96.34000000000003	128.41000000000005	131.22000000000003	64.83999999999997	28.42	15.46
21	76.73	58.65917999999999	14.11	77.97999999999992	116.24999999999996	109.07999999999998	50.180000000000014	27.75	15.87

Figura: Dataset de Demanda

[9].

Visualización del Dataset Original

Generation

10292	10370	10729	11031	11081	11132	11139	11194	11607
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.01	0.060000000000000005	0.0	0.0	0.0	0.0	0.04	0.0	0.05
0.01	0.4199999999999999	0.04	0.0	0.0	0.01	0.29	0.13	1.03
0.31	1.4300000000000002	0.26	0.0	0.0	0.11	2.02	1.05	4.92
0.6200000000000001	1.81	0.4299999999999999	0.0	0.0	0.24	3.21	1.32	6.669999999999999
1.4	3.02	0.63	0.0	0.0	0.7199999999999999	4.63	2.9400000000000004	12.31
3.49	4.07	0.67	0.0	0.0	1.4500000000000002	3.81	4.82	18.18
0.69	1.48	0.13	0.0	0.0	0.46	1.02	1.38	8.520000000000001
0.0	0.1	0.01	0.0	0.0	0.0	0.05	0.02	0.3
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figura: Dataset de Generación

Dataset Utilizado

Datos de Entrenamiento:

- Cada transformador tiene 25,000 mediciones temporales.
- Dataset total: 17 transformadores.

Distribución de Datos:

- **Entrenamiento:** Datos 0 a 17,500.
- **Validación:** Datos 17,501 a 20,000.
- **Testing:** Datos 20,001 a 22,500.
- **Evaluación FDIA:** Últimos 2,500 datos.

Frecuencia de las Mediciones:

- Cada medición representa 1 hora de datos, lo que implica un total de 25,000 horas.

Herramientas Utilizadas

Entrenamiento y Gestión de Datos:

- **DataLoader** y **DataModule** de PyTorch Lightning.
  - Facilitan la carga y procesamiento concurrente de grandes cantidades de datos.
  - Permiten una administración modular y organizada de los conjuntos de datos.

Plataforma Weights and Biases (Wandb):

- Monitoreo detallado de los experimentos.
- Visualización de métricas en tiempo real y comparación de diferentes modelos.
- Ayuda en la detección de overfitting y underfitting.

Análisis de Fourier y Densidad Espectral

Transformada de Fourier:

- Se implementó en MATLAB para pasar la señal del dominio temporal al dominio de frecuencia para así identificar las frecuencias dominantes en las señales de demanda y generación.

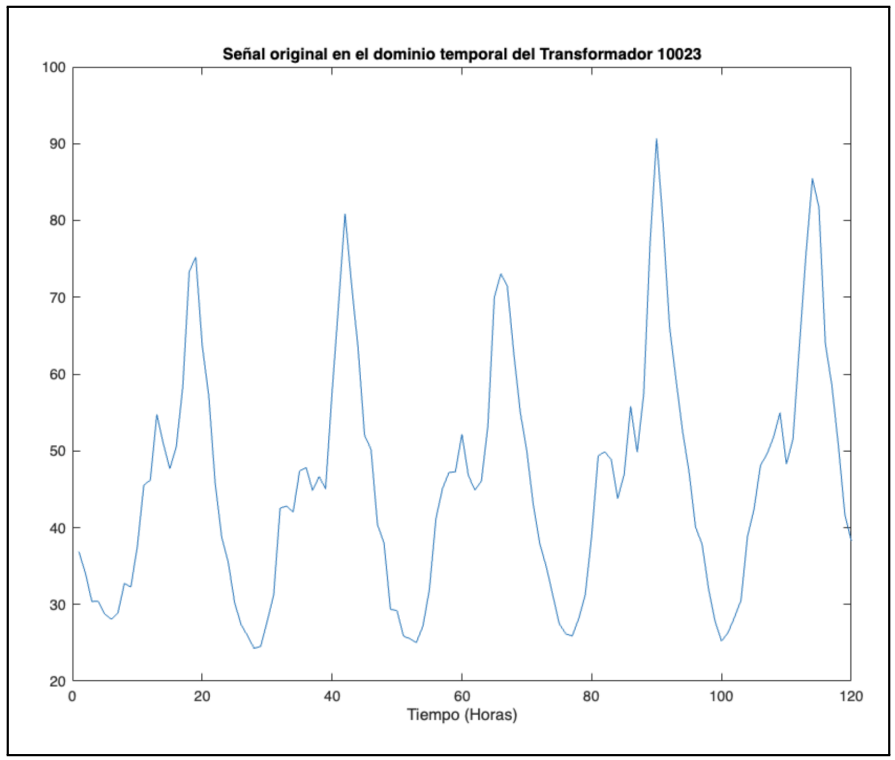


Figura: Señal Original en el Dominio Temporal para el Transformador 10023

Resultados del Análisis de Fourier

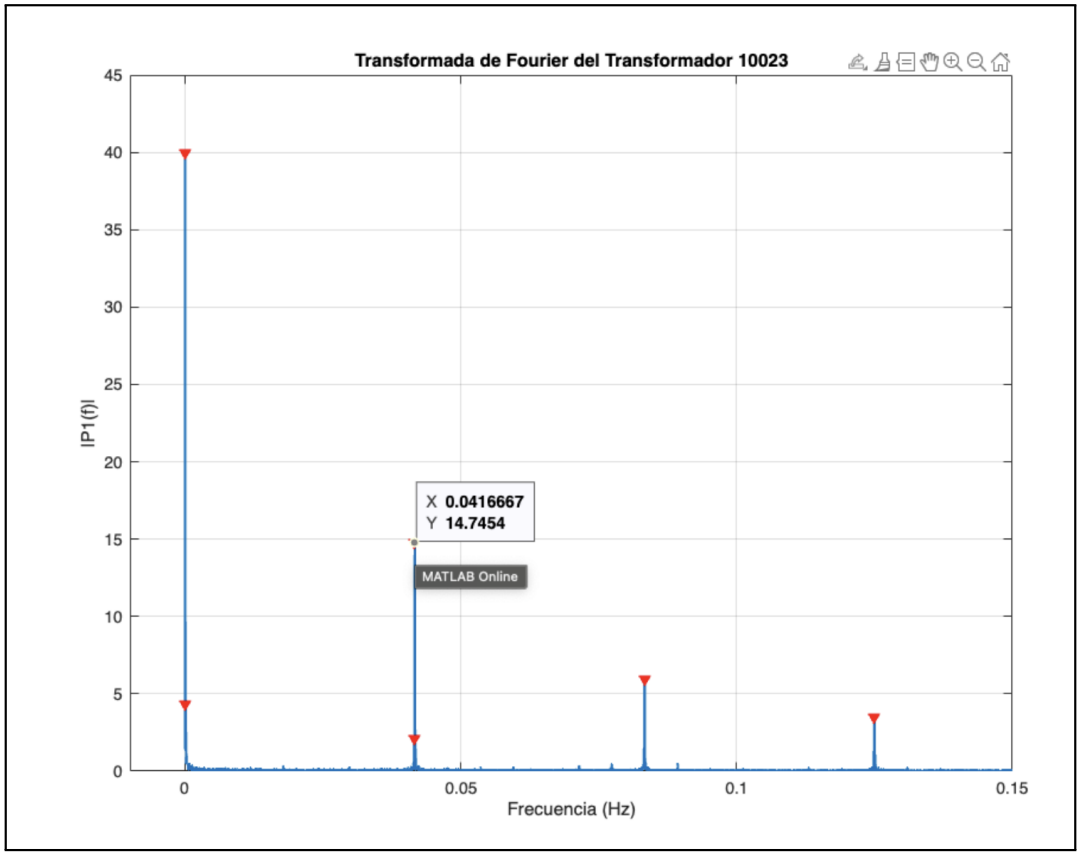


Figura: Transformada de Fourier (FFT) en el Dominio de Frecuencia para el Transformador 10023

Identificación de Frecuencias Dominantes

Frecuencias Dominantes:

- Al aplicar la Transformada de Fourier, se identificaron los primeros picos que representan los armónicos de la señal.
- Estos armónicos indican las frecuencias más relevantes que componen la señal de demanda y generación energética.

Densidad Espectral de Potencia (PSD):

- Se utilizó el método de Welch para calcular la PSD y explorar características de la señal.
- La PSD permite identificar las frecuencias con mayor potencia, destacando las dominantes en el comportamiento del sistema.

Identificación de Frecuencias Dominantes

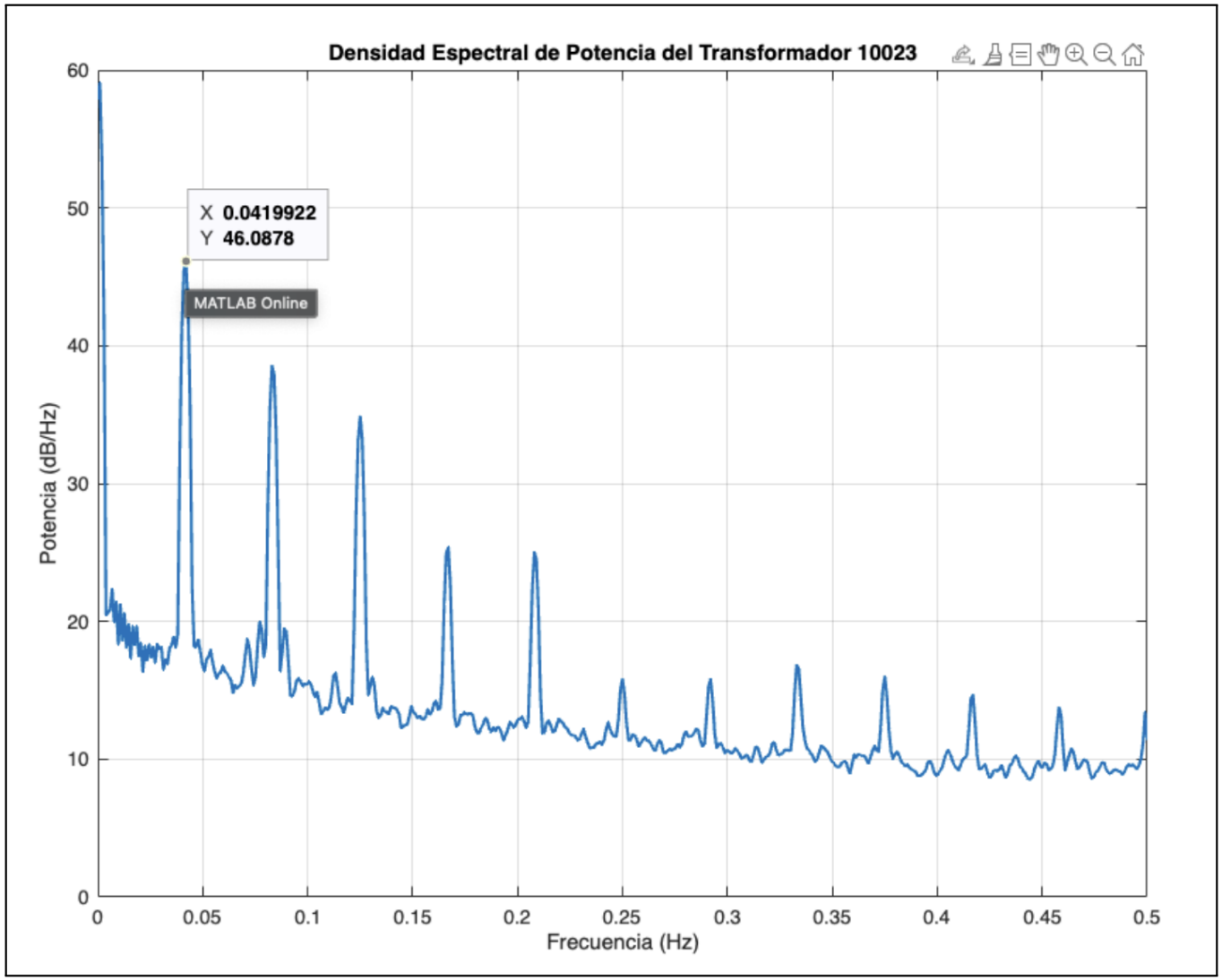


Figura: Densidad Espectral de Potencia (PSD) del Transformador 10023

Frecuencia Fundamental y Ciclo Diario

Frecuencia Fundamental:

- Frecuencia identificada: $f \approx 0.0418\,\text{Hz}$.
- Esta frecuencia corresponde a un ciclo completo cada 24 horas.

Relación Matemática:

$$T = \frac{1}{f} = \frac{1}{0.0418} \approx 24\,\text{horas} \tag{1}$$

Interpretación:

- El ciclo de 24 horas es consistente con patrones típicos en la demanda energética, la cual varía diariamente.
- La frecuencia fundamental es clave para determinar el tamaño adecuado de la ventana en modelos predictivos, permitiendo capturar la periodicidad de la señal.
- Relación con el tamaño de ventana: un ciclo completo captura la variabilidad cíclica diaria de la demanda energética.

Modelo LSTM

¿Qué es una LSTM?

- Long Short-Term Memory (LSTM) es una variante de las Redes Neuronales Recurrentes (RNN) diseñada para aprender dependencias a largo plazo en secuencias de datos.
- Especialmente útil para series temporales debido a su capacidad de retener información de eventos previos y aplicar ese conocimiento al realizar predicciones.
- En este caso, se entrenó para predecir la demanda energética de paneles solares.

Configuración del Modelo:

- Implementación en PyTorch Lightning para facilitar el entrenamiento modular.
- La arquitectura consta de un número variable de capas ocultas y celdas LSTM.

Arquitectura del Modelo LSTM

Componentes del Modelo LSTM:

- **Capa LSTM:** (`self.lstm`)
  - Configurada con `input_dim` (número de características por paso de tiempo) y `hidden_dim` (tamaño de la memoria oculta).
  - Se utilizan 2 capas LSTM para aumentar la capacidad de aprender dependencias temporales complejas.
- **Capa Lineal:** (`self.fc`)
  - Mapea el estado oculto final de la secuencia a la salida deseada.
  - `output_dim` es 1, ya que queremos predecir un único valor de demanda.
- **Forward Pass:**
  - El método `forward()` define cómo se procesan los datos en el modelo.
  - La salida de la capa LSTM se pasa por una capa lineal para obtener la predicción.

Implementación del Modelo LSTM

Código Simplificado:

```
1     # Modelo con LSTM
2     class LSTMModel(pl.LightningModule):
3       def __init__(self, input_dim, hidden_dim,
            num_layers, output_dim):
4         super(LSTMModel, self).__init__()
5         self.lstm = nn.LSTM(input_dim, hidden_dim,
              num_layers, batch_first=True)
6         self.fc = nn.Linear(hidden_dim, output_dim)
7
8       def forward(self, x):
9         lstm_out, _ = self.lstm(x)
10        output = self.fc(lstm_out[:, -1, :]) # Se usa
              la ultima salida de la secuencia
11        return output
```

Código 1: Modelo con LSTM

Componentes del Modelo LSTM

Capa LSTM:

- `num_layers`: Cantidad de capas LSTM utilizadas (en este caso, 2).
- Esta capa permite que la secuencia de datos fluya a través del tiempo y pueda aprender dependencias temporales a largo plazo.

Capa Lineal:

- `self.fc` toma como entrada el estado oculto final y lo transforma en la salida deseada.
- `output_dim` es 1, ya que queremos predecir un único valor de demanda.

Forward Pass y Evaluación del Modelo

Método `forward()`:

- Primero se aplica la capa LSTM sobre la secuencia de entrada.
- Luego, se pasa la salida oculta final (`lstm_out[:, -1, :]`) a la capa lineal (`fc`).
- Esto permite que el modelo tome en cuenta toda la información acumulada durante la secuencia para realizar una predicción precisa.

Evaluación del Tamaño de Ventana:

- Para determinar la mejor ventana de entrada, se evaluaron ventanas de tamaño 1 a 48 horas.
- Esto se hizo mediante la función `evaluate_window_size`, que genera un módulo de datos para entrenamiento, validación y prueba.

Código para Evaluación del Tamaño de Ventana

Código Simplificado:

```
# Evaluar el tama o de la ventana
def evaluate_window_size(train_data, val_data, test_data, window_size):
    data_module = DemandDataModule(train_data, val_data, test_data, window_size)
    input_dim = 1
    hidden_dim = 64
    num_layers = 2
    output_dim = 1
    model = LSTMModel(input_dim=input_dim, hidden_dim=hidden_dim, num_layers=num_layers, output_dim=output_dim)
    # Definir el entrenador
    trainer = pl.Trainer(max_epochs=10, logger=False, enable_checkpointing=False)
```

Código 2: Evaluar el tamaño de la ventana

Código para Evaluación del Tamaño de Ventana

Explicación del Código:

- El parámetro `hidden_dim` = 64 define la capacidad de las celdas de memoria internas.
- `num_layers` = 2 equilibra la capacidad de modelado y la eficiencia computacional.

Modelo TCN (Temporal Convolutional Network)

¿Qué es una TCN?

- Temporal Convolutional Network (TCN) es un tipo de red neuronal que utiliza convoluciones dilatadas para capturar dependencias temporales en datos de series temporales.
- A diferencia de las RNNs, las TCN pueden realizar operaciones en paralelo, aumentando la eficiencia del aprendizaje.

Arquitectura del Modelo TCN

Componentes del Modelo TCN:

- **Capas Convolucionales con Dilatación:**
  - Utilizan un `dilation_size` que se incrementa exponencialmente (2^i).
  - Permiten que el modelo capture dependencias a largo plazo en los datos.
- **Activación y Dropout:**
  - Se aplica una función de activación ReLU seguida de una capa de Dropout para evitar el sobreajuste.
- **Capa Lineal:**
  - Similar al LSTM, toma la salida de la última convolución y la convierte en la predicción.

Implementación del Modelo TCN

Código Simplificado:

```
1     class TCNModel(pl.LightningModule):
2       def __init__(self, input_dim, hidden_dim,
            output_dim, num_layers, kernel_size=2,
            dropout=0.2):
3         super(TCNModel, self).__init__()
4         layers = []
5         for i in range(num_layers):
6           in_channels = input_dim if i == 0 else
              hidden_dim
7           out_channels = hidden_dim
8           dilation_size = 2 ** i
```

Código 3: Modelo con TCN

Implementación del Modelo TCN

Código Simplificado:

```
1            padding_size = (kernel_size - 1) *
                 dilation_size
2            layers.append(nn.Conv1d(in_channels,
                 out_channels, kernel_size=kernel_size,
                 dilation=dilation_size, padding=
                 padding_size))
3            layers.append(nn.ReLU())
4            layers.append(nn.Dropout(dropout))
5        self.tcn = nn.Sequential(*layers)
6        self.fc = nn.Linear(hidden_dim, output_dim)
```

Código 4: Modelo con TCN

Componentes:

- Capas convolucionales definidas en un bucle.
- Cada capa tiene una convolución 1D con dilatación exponencial.

Capas Convolucionales con Dilatación

¿Qué es una Dilatación?

- La **dilatación** se refiere a la separación entre los valores sobre los que se aplica la convolución.
- Por ejemplo, una dilatación de 1 significa que cada valor es adyacente, mientras que una dilatación de 2 implica que se omite un valor.
- Esto permite capturar relaciones a largo plazo sin aumentar excesivamente el número de parámetros.

Ventaja:

- Proporciona una **ventana de percepción** más amplia sin un incremento significativo en el tamaño del modelo (kernel_size).

Activación y Dropout

Función de Activación:

- Se utiliza `nn.ReLU()` para introducir no linealidades en el modelo.
- Esto permite que el modelo aprenda relaciones complejas en los datos de series temporales.

Capa de Dropout:

- Dropout (`nn.Dropout(dropout)`) se aplica para evitar el sobreajuste.
- Desactiva aleatoriamente unidades durante el entrenamiento, haciendo que el modelo sea más robusto.
- El valor de `dropout` utilizado es 0.2.

Capa Lineal y Método Forward

Capa Lineal (`fc`):

- Convierte la salida de la última capa convolucional en un valor de predicción.
- `output_dim` es 1, ya que se predice la demanda energética para el próximo periodo.

Método Forward:

- Se realiza la transpuesta de la entrada antes de aplicar la TCN (`x = x.transpose(1, 2)`) para que tenga la forma correcta.
- La salida se transpone de nuevo antes de pasar por la capa lineal para obtener la predicción final.

Evaluación del Tamaño de la Ventana

Código para Evaluación del Tamaño de Ventana:

```
1     # Evaluar el tama o de la ventana
2     def evaluate_window_size(train_data, val_data,
          test_data, window_size):
3         data_module = DemandDataModule(train_data,
              val_data, test_data, window_size)
4         input_dim = 1
5         hidden_dim = 64
6         num_layers = 3
7         output_dim = 1
8         model = TCNModel(input_dim=input_dim,
              hidden_dim=hidden_dim, output_dim=
              output_dim, num_layers=num_layers)
```

Código 5: Evaluar el tamaño de la ventana en TCN

Aclaración y Conclusiones sobre el Modelo TCN

Parámetros Importantes:

- `hidden_dim` = 64: Define la capacidad de representación de las características ocultas.
- `num_layers` = 3: se tiene suficiente capacidad para captar patrones complejos en las series temporales (debido a la dilatación) sin sacrificar eficiencia y sin riesgo significativo de sobreajuste.

Ventajas de TCN:

- Operaciones en paralelo.
- Capturar dependencias a largo plazo sin los problemas de "desvanecimiento del gradienteçomunes en las RNN.

Uso de Convoluciones Dilatadas:

- Permiten ampliar la ventana de percepción del modelo sin aumentar drásticamente el número de parámetros.

Modelo Híbrido: TCN + LSTM

¿Qué es el Modelo Híbrido?

- El modelo híbrido combina una Red Convolucional Temporal (TCN) con una arquitectura Long Short-Term Memory (LSTM).
- Se busca aprovechar las ventajas de ambos enfoques en el análisis de series temporales.
  - **TCN:** Captura patrones temporales y relaciones a largo plazo mediante convoluciones dilatadas.
  - **LSTM:** Captura dependencias complejas a nivel secuencial utilizando celdas de memoria.

Arquitectura del Modelo Híbrido

Componentes del Modelo Híbrido:

- **Capas TCN:**
  - Convoluciones con dilatación exponencial para capturar relaciones a largo plazo.
  - Cada capa convolucional seguida por una función de activación ReLU y un Dropout para evitar sobreajuste.
- **Capa LSTM:**
  - Recibe la salida de la TCN y captura dependencias secuenciales complejas.
- **Capa Fully Connected (FC):**
  - Convierte la salida de la LSTM en la predicción de demanda energética.

Implementación del Modelo Híbrido (Parte 1)

Definición del Modelo:

```
1     class HybridModel(pl.LightningModule):
2       def __init__(self, input_dim, tcn_hidden_dim,
            lstm_hidden_dim, output_dim, num_tcn_layers
            =3, num_lstm_layers=1, kernel_size=2, dropout
            =0.2):
3         super(HybridModel, self).__init__()
4         # Capas convolucionales dilatadas (TCN)
5         tcn_layers = []
6         for i in range(num_tcn_layers):
7           in_channels = input_dim if i == 0 else
              tcn_hidden_dim
8           out_channels = tcn_hidden_dim
9           dilation_size = 2 ** i  # Dilataci n
              exponencial
```

Código 6: Modelo híbrido TCN + LSTM

Implementación del Modelo Híbrido (Parte 1)

```
1            padding_size = (kernel_size - 1) *
                 dilation_size
2            tcn_layers.append(nn.Conv1d(in_channels,
                 out_channels, kernel_size=kernel_size,
                 dilation=dilation_size, padding=
                 padding_size))
3            tcn_layers.append(nn.ReLU())
4            tcn_layers.append(nn.Dropout(dropout))
5        self.tcn = nn.Sequential(*tcn_layers)
```

Código 7: Modelo híbrido TCN + LSTM

- `tcn_layers`: Convoluciones con dilatación para capturar patrones a largo plazo.
- Funciones de activación ReLU y Dropout después de cada convolución.

Implementación del Modelo Híbrido (Parte 2)

Definición del Modelo:

```
        # Capa LSTM
        self.lstm = nn.LSTM(input_size=tcn_hidden_dim,
            hidden_size=lstm_hidden_dim, num_layers=
            num_lstm_layers, batch_first=True)

        # Capa de salida (fully connected)
        self.fc = nn.Linear(lstm_hidden_dim,
            output_dim)

    def forward(self, x):
        # TCN: Conv1D espera (batch_size, input_dim,
            sequence_length), as  que cambiamos las
            dimensiones
        x = x.transpose(1, 2)
        tcn_out = self.tcn(x)
```

Código 8: Modelo híbrido TCN + LSTM (continuación)

Implementación del Modelo Híbrido (Parte 2)

Definición del Modelo:

```
        # Cambiamos de nuevo a (batch_size,
            sequence_length, hidden_dim) para la LSTM
        tcn_out = tcn_out.transpose(1, 2)

        # LSTM
        lstm_out, _ = self.lstm(tcn_out)

        # Usamos la última salida de la secuencia
        output = self.fc(lstm_out[:, -1, :])
        return output
```

Código 9: Modelo híbrido TCN + LSTM (continuación)

Capa LSTM y Fully Connected:

- `self.lstm`: Procesa las características extraídas por la TCN.
- `self.fc`: Convierte la última salida de la LSTM en la predicción deseada.

Método Forward

Descripción del Método `forward()`:

- **Entrada:**
  - Los datos ingresan a la TCN, se realiza la convolución y se aplica ReLU y Dropout.
  - La salida de la TCN se transfiere a la LSTM.
- **Salida:**
  - Se obtiene la última salida de la LSTM y se pasa por la capa `fc` para generar la predicción final.

Nota: La transposición de dimensiones asegura que los datos estén en el formato correcto para cada operación.

Evaluación del Tamaño de Ventana

Función `evaluate_window_size`:

```
1   # Evaluar el tama o de la ventana
2   def evaluate_window_size(train_data, val_data,
        test_data, window_size):
3       data_module = DemandDataModule(train_data,
            val_data, test_data, window_size)
4       input_dim = 1
5       tcn_hidden_dim = 64
6       lstm_hidden_dim = 64
7       num_tcn_layers = 3
8       num_lstm_layers = 1
9       output_dim = 1
10      model = HybridModel(input_dim=input_dim,
            tcn_hidden_dim=tcn_hidden_dim,
            lstm_hidden_dim=lstm_hidden_dim, output_dim
            =output_dim, num_tcn_layers=num_tcn_layers)
```

Código 10: Evaluar el tamaño de la ventana en el modelo híbrido

Parámetros del Modelo Híbrido (Continuación)

Configuración del Modelo:

- `num_tcn_layers` = 3: Número de capas convolucionales. Se eligieron 3 capas para capturar patrones complejos y relaciones de largo alcance.
- `num_lstm_layers` = 1: Se utiliza una sola capa LSTM para mantener la simplicidad y capturar las dependencias temporales restantes.
- `output_dim` = 1: El modelo predice un solo valor que corresponde a la demanda energética futura.

Razón de Selección de Parámetros:

- Los valores de `hidden_dim` y `num_layers` fueron elegidos como un compromiso entre la capacidad de representación y la eficiencia computacional.
- El uso de 64 para las dimensiones ocultas permite suficiente capacidad para representar patrones temporales sin sobrecargar el entrenamiento.

Ventajas del Modelo Híbrido TCN + LSTM

- **Capacidad de Captura de Patrones:**
  - **TCN:** Eficiente en capturar dependencias a largo plazo y patrones temporales mediante convoluciones con dilatación.
  - **LSTM:** Ideal para capturar dependencias secuenciales y relaciones temporales complejas.
- **Complementariedad:**
  - La TCN aprende patrones de gran escala en los datos, mientras que la LSTM se enfoca en el procesamiento secuencial detallado.
  - El enfoque híbrido combina lo mejor de ambos métodos, ofreciendo un modelo robusto y capaz de aprender de manera profunda las variaciones en la demanda energética.

Resultados

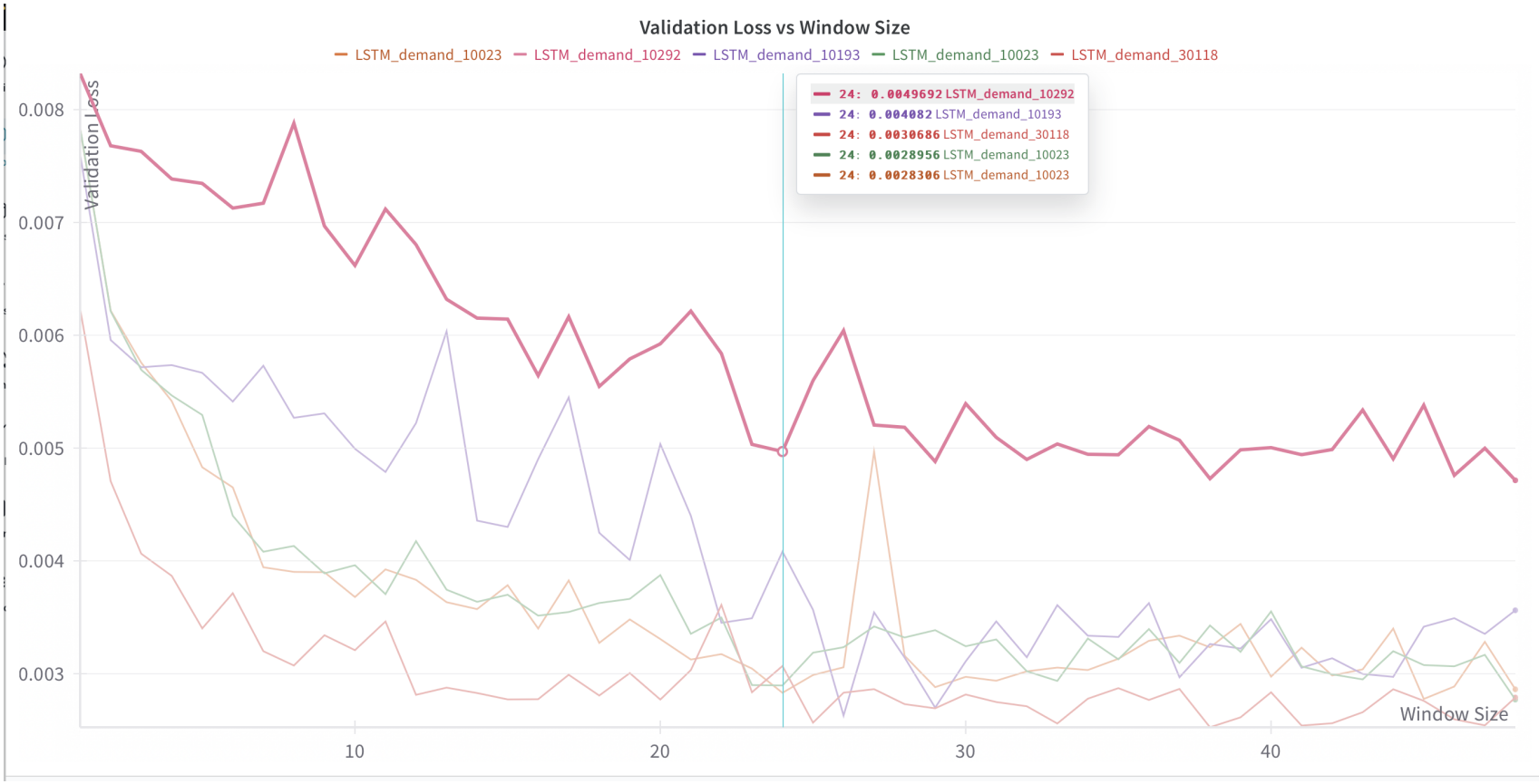


Figura: MSE en conjunto de validación - LSTM

Resultados

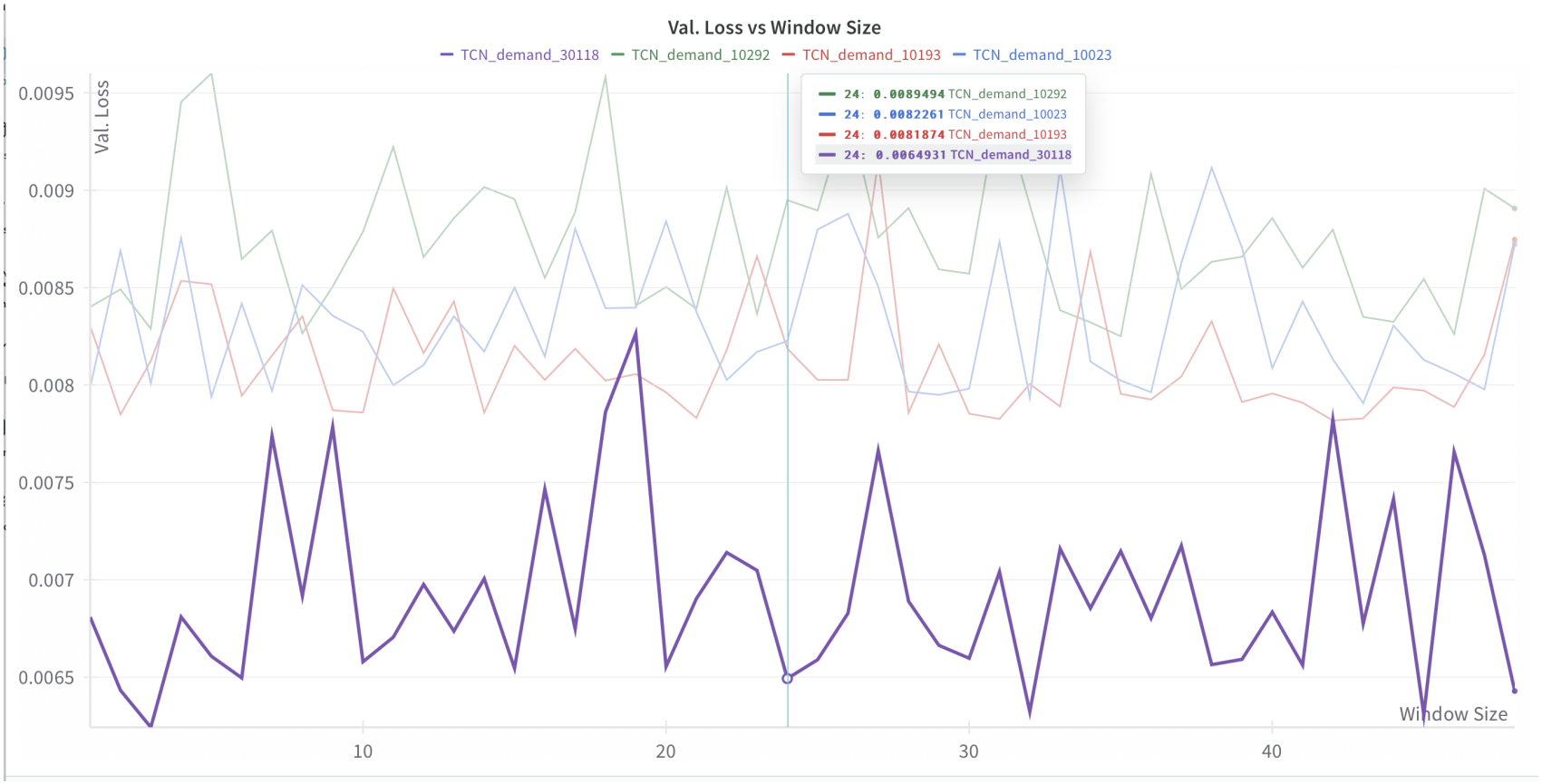


Figura: MSE en conjunto de validación - TCN

Resultados

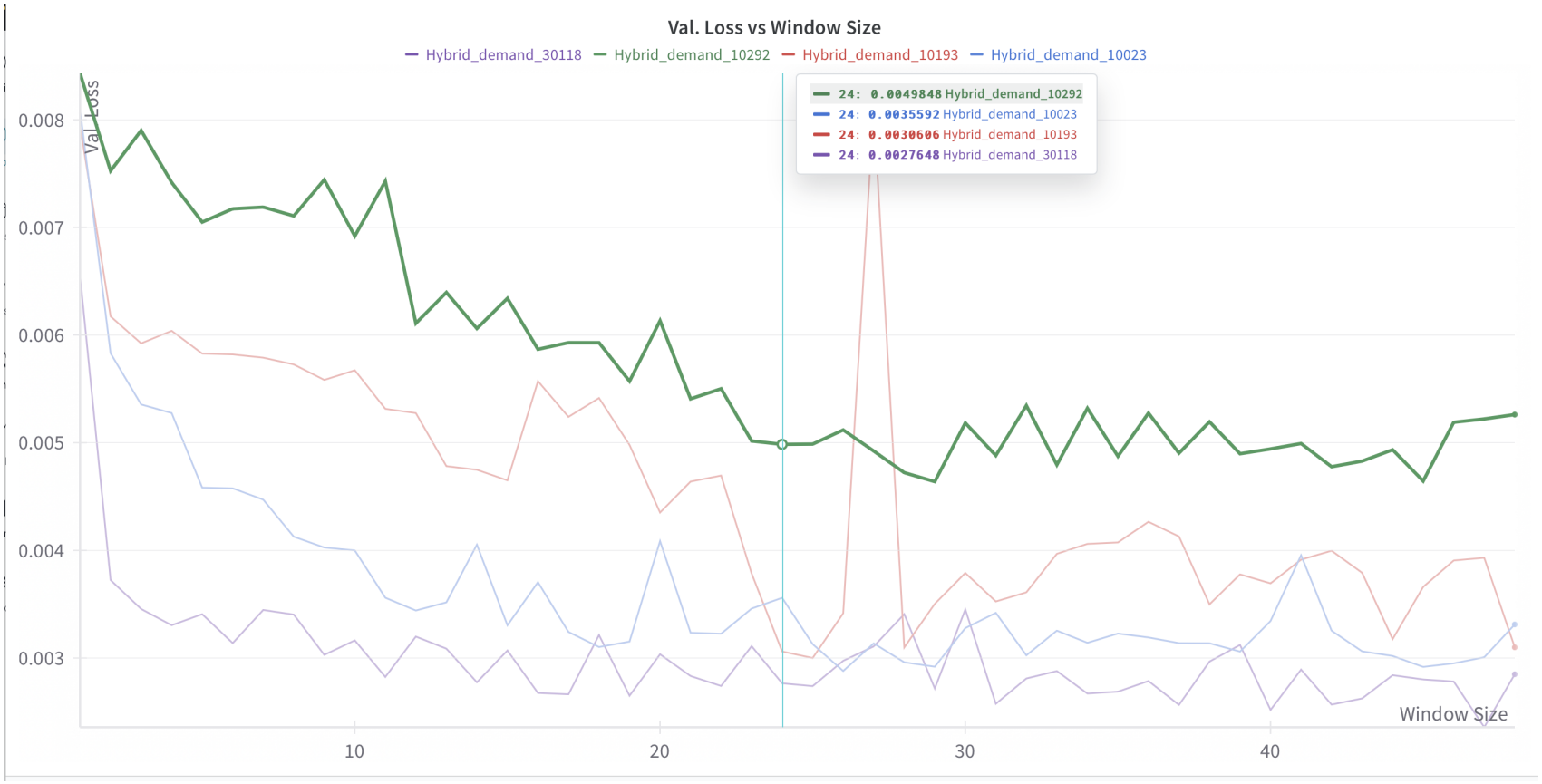


Figura: MSE en conjunto de validación - Híbrido

Referencias

[1] Zhang, Y., Wang, X., & Liu, J. (2020). Detecting False Data Injection Attacks in Smart Grids Using CNNs and LSTMs. *IEEE Transactions on Smart Grid*, 11(4), 3043-3051. https://arxiv.org/pdf/2006.11477.

[2] Stanford University. (2024). Recurrent Neural Networks cheatsheet. Recuperado de `https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks`.

[3] Cayci, S., & Eryilmaz, A. (2024). Convergence of Gradient Descent for Recurrent Neural Networks: A Nonasymptotic Analysis. *arXiv preprint arXiv:2402.12241*. Recuperado de `https://arxiv.org/abs/2402.12241`.

[4] Staudemeyer, R. C., & Morris, E. R. (2019). Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks. *arXiv preprint arXiv:1909.09586*. Recuperado de `https://arxiv.org/abs/1909.09586`.