ENANDES and ENANDES+ Projects

Enhancing Adaptive Capacity of Andean Communities through Climate Services (ENANDES) in Chile, Colombia, Peru, and ENANDES+ in Argentina, Bolivia, Ecuador

Activity: Energy mini-project on AI-based performance evaluation and correction of solar radiation forecasts from the Weather Research and Forecasting (WRF) numerical prediction model (Ref. 6455259-2025/GS/PEX)

Deliverable N°2: Operational Manual on AI-based performance evaluation and correction of solar radiation forecasts from the Weather and Research Forecasting (NWP) numerical prediction

History of draft preparation and review

V1.0- Initial draft by Germán Ariel Salazar and Rubén Darío Ledesma (UNSa-INENCO-CONICET)	27/03/2026
V2.0 – Revised based on review and comments by Silvina Righetti and Cynthia Matsudo (SMN), and Hamid Bastani (WMO)	08/04/2026
V3.0 – Revised based on review and comments by Silvina Righetti and Cynthia Matsudo (SMN), and Hamid Bastani (WMO)	30/04/2026
V4.0 - Revised based on review and comments by Silvina Righetti and Cynthia Matsudo (SMN), Anahi Lanson and María José Denegri (UNLu), Hamid Bastani (WMO)	12/05/2026

Note:

This deliverable (N°4) is composed of the following three main components:

1. The present technical document, developed as an OPERATIONAL MANUAL on AI-based performance evaluation and correction of solar radiation forecasts from the Weather Research and Forecasting (WRF) numerical prediction model.
2. The open-source Jupyter Notebook codes of the model, aligned with the present technical publication, available at: https://github.com/rdledesma/SMN-WRF-Site-Adaption/blob/main/Presentacion_WRF_SMN_MLP.ipynb
3. A recorded training on the methodology and the codes, aligned with the present technical publication and the Jupyter Notebook codes, available at:https://wmoomm-my.sharepoint.com/:f:/g/personal/hbastani_wmo_int/IgBYbDdmC8TeRaoVHZzTOsuHATtDM_2DWe3IIt--EexKBEs?e=CRlglG

Nota:

Este entregable (N°4) se compone de los siguientes tres componentes principales:

1. El presente documento técnico, elaborado como MANUAL OPERATIVO sobre la evaluación y corrección del desempeño, basada en inteligencia artificial, de los pronósticos de radiación solar generados por el modelo numérico Weather Research and Forecasting (WRF).
2. Los códigos de acceso abierto en Jupyter Notebook del modelo, alineados con la presente publicación técnica, disponibles en: https://github.com/rdledesma/SMN-WRF-Site-Adaption/blob/main/Presentacion_WRF_SMN_MLP.ipynb
3. Una capacitación grabada sobre la metodología y los códigos, alineada con la presente publicación técnica y los códigos en Jupyter Notebook, disponible https://wmoomm-my.sharepoint.com/:f:/g/personal/hbastani_wmo_int/IgBYbDdmC8TeRaoVHZzTOsuHATtDM_2DWe3IIt--EexKBEs?e=CRlglG

MANUAL OPERATIVO

Evaluación y corrección del desempeño, basada en inteligencia artificial, de los pronósticos de radiación solar generados por el modelo numérico Weather Research and Forecasting (WRF)

Autores: Dr. Germán Ariel Salazar y Lic. Rubén Darío Ledesma (UNSa-INENCO-CONICET)

Colaboradores: Cynthia Matsudo (SMN), Silvina Righetti (SMN), Anahi Lanson (UNLu), Maria Jose Denegri (UNLu), Hamid Bastani (WMO)

Coordinación técnica de la OMM: Hamid Bastani

Financiamiento: Esta publicación se ha elaborado en el marco del proyecto ENANDES (Enhancing Adaptive Capacity of Andean Communities through Climate Services), financiado por el Fondo de Adaptación, e implementado por la Organización Meteorológica Mundial (OMM). El proyecto apoya a los países participantes de ENANDES y ENANDES+, incluidos Chile, Colombia, Perú, Argentina, Bolivia y Ecuador, así como a otros países dentro y fuera de la región. La publicación es el resultado de un mini proyecto de energía desarrollado en estrecha colaboración entre la Oficina Regional de la OMM para las Américas (RAM), el Servicio Meteorológico Nacional (SMN) de Argentina y la Universidad Nacional de Salta.

Tabla de contenidos

3. Recomendaciones del consultor

3.1. Limitaciones e incertidumbres del modelo.

3.1.1. Sesgo sistemático de sobreestimación.

3.1.2. Robustez del MLP con el tiempo.

3.1.3. Desempeño crítico en condiciones de cielo cubierto.

3.2. Desarrollo futuro y líneas de investigación.

3.2.1. Incorporación de variables de nubosidad satelital.
3.2.2. Evaluación ante la variabilidad climática interanual.
3.2.3. Análisis de la importancia de las variables (feature importance).
3.2.4. Arquitecturas alternativas de ML.
3.2.5. Integración operativa con el SMN.

4. Anexos

4.1. Script de los códigos de Jupyter Notebook.

4.2. Script Referencia Cruzada.

1. Alcance y Estructura del Documento

Este documento se ha elaborado como un entregable del proyecto ENANDES[1] (Enhancing Adaptive Capacity of Andean Communities through Climate Services) de la Organización Meteorológica Mundial (OMM) (versión completa), financiado por el Fondo de Adaptación.

La actividad se define en el marco de la iniciativa de miniproyectos de energía de la OMM[2], en la que se desarrolla un paquete final estandarizado que incluye las presentes directrices técnicas, códigos de acceso abierto en Jupyter Notebook y un video de formación sobre cómo ejecutar los códigos, basados en el manual técnico.

En este contexto, el objetivo es no solo responder a las necesidades del Servicio Meteorológico Nacional de Argentina (SMN), sino también proporcionar una metodología escalable para apoyar a otros países participantes de ENANDES y ENANDES+, como Chile, Perú, Colombia, Ecuador y Bolivia, así como a otros países dentro y fuera de la región, permitiéndoles adoptar y adaptar esta metodología a necesidades similares. A partir de estos antecedentes, este manual se concibe como una herramienta operativa orientada a convertir los resultados metodológicos en una guía técnica reproducible. Su propósito no es solo documentar el estudio de caso, sino también facilitar que el personal técnico de los Servicios Meteorológicos e Hidrológicos Nacionales (SMHN) y otros usuarios especializados puedan replicar el procedimiento, adaptar la metodología a sus propios datos y utilizar herramientas abiertas para validar y corregir las salidas del modelo. En línea con este objetivo, la disponibilidad de pronósticos meteorológicos confiables constituye un insumo estratégico para la gestión operativa y la toma de decisiones en sectores sensibles a la variabilidad atmosférica.

En particular, la radiación solar global horizontal (GHI, por sus siglas en inglés) es una variable de gran interés para aplicaciones relacionadas con la energía, el clima y el monitoreo ambiental. En este contexto, el modelo numérico Weather Research and Forecasting v4.0 del Servicio Meteorológico Nacional argentino (WRF-SMN) genera pronósticos meteorológicos horarios con horizontes de 1 a 72 horas, a partir de corridas inicializadas a las 00, 06, 12 y 18 UTC, sobre una retícula de alta resolución espacial (4 km) del territorio argentino (sin considerar la Antártida). Sin embargo, la calidad de sus salidas puede variar según la variable analizada, el horizonte de pronóstico, la región geográfica y la disponibilidad y la calidad de los datos observacionales empleados para su validación.

En el caso de la GHI, la evaluación requiere especial cuidado porque se trata de una variable fuertemente modulada por la geometría solar, la nubosidad y las condiciones

[1] https://wmo.int/activities/projects/project-portfolio/enandes-building-regional-adaptive-capacity-and-resilience-climate-variability-and-change-vulnerable
[2] https://energymeteorology.info/projects/

atmosféricas locales, y que además presenta una marcada variabilidad diurna y estacional. Por ello, la comparación entre pronósticos y observaciones exige procedimientos consistentes de control de calidad, integración temporal, asignación de tiempos válidos y selección de métricas de desempeño.

La evaluación, realizada con datos de una estación radiométrica en Argentina, comparó los pronósticos horarios de GHI del WRF-SMN con mediciones simultáneas en superficie en Buenos Aires (BA). Para ello, se definió una metodología paso a paso que incluye: control de calidad de los datos minutales de GHI, integración horaria, extracción y preprocesamiento de pronósticos, conversión de variables radiativas acumuladas a irradiancia horaria, comparación mediante métricas como rMBE y rRMSE, una etapa posterior de adaptación al sitio mediante una arquitectura de aprendizaje automático basada en un Multilayer Perceptron (MLP).

Así, el objetivo general de este manual es proporcionar una guía técnica, clara y reproducible para evaluar el desempeño del modelo WRF-SMN en la predicción de GHI y aplicar un procedimiento de mejora de sus salidas mediante la adaptación al sitio con redes neuronales del tipo MLP, utilizando datos observacionales y herramientas de código abierto.

De manera específica, el manual busca:

- Describir el problema técnico abordado, es decir, la necesidad de validar los pronósticos horarios de GHI del WRF-SMN frente a mediciones en superficie y cuantificar sus errores en distintos horizontes de predicción.
- Presentar un procedimiento paso a paso para la adquisición, el control de calidad, la integración temporal y el preprocesamiento de datos observacionales y simulados, de modo que puedan compararse de forma consistente.
- Explicar el uso de métricas de evaluación, como MBE y RMSE, y sus versiones relativas, para caracterizar el sesgo y el error de los pronósticos del modelo numérico.
- Mostrar cómo construir y aplicar un modelo MLP para reducir las diferencias entre los valores pronosticados y los valores medidos de GHI, incluyendo la selección de regresores, el escalado de variables, la división de los datos y el seguimiento del entrenamiento.
- Facilitar la replicación de la metodología mediante el uso de código abierto, ejemplos de datos y materiales complementarios, de modo que pueda ser reutilizada y adaptada por los equipos técnicos de los Servicios Meteorológicos Nacionales.
- Servir de puente entre el estudio de caso y su aplicación operativa, favoreciendo la transferencia metodológica y el fortalecimiento de las capacidades institucionales para la validación y corrección de productos numéricos de pronóstico.

Este documento no pretende reemplazar la documentación oficial del WRF ni abarcar todas las variables meteorológicas del sistema, sino centrarse en la GHI como variable objetivo del estudio y en un procedimiento transferible a otros contextos.

2. Metodología y Aplicación

2.1. Marco Conceptual y Descripción General

Esta sección presenta el marco conceptual del procedimiento empleado para evaluar el desempeño del modelo numérico WRF-SMN en el pronóstico de GHI y para reducir su error mediante una estrategia de adaptación al sitio basada en el aprendizaje automático. El enfoque combina validación observacional, control de calidad radiométrico, análisis estadístico del error y calibración empírica mediante un MLP, con el objetivo de reducir las diferencias entre los pronósticos horarios del modelo y las mediciones en superficie. El procedimiento fue desarrollado y aplicado en el Entregable 1 para tres sitios de la región central de Argentina.

Desde el punto de vista conceptual, la metodología parte de una premisa central: las salidas de un modelo de predicción numérica, aun cuando representen razonablemente la variabilidad meteorológica regional, pueden presentar sesgos sistemáticos y errores dependientes del sitio al compararlas con observaciones locales. En el caso de la GHI, esas diferencias están asociadas tanto a la representación de la nubosidad y de los procesos atmosféricos en el modelo como a cuestiones de escala espacial, calidad de los datos medidos, sincronización temporal y características micro-climáticas del entorno de observación. Por ello, la validación no se limita a una comparación directa entre series, sino que requiere una secuencia ordenada de adquisición, depuración, armonización temporal, evaluación estadística y corrección posterior.

El flujo metodológico implementado comprende cuatro grandes etapas:

1) **Adquisición y organización de datos medidos, control de calidad e integración horaria**. En esta etapa se describen las características de los archivos que contienen los datos medidos, la aplicación de los filtros de control de calidad, así como la completitud y la integración de los datos minutales a la frecuencia horaria.

2) **Adquisición y organización de los datos pronosticados.** Aquí se detalla cómo se descargan los datos generales, cómo se determina la celda de extracción de datos, cómo se realiza la extracción de datos localizados, cómo se determina el marco temporal conjunto, cómo se seleccionan las variables de interés, cómo se calcula la GHI pronosticada y cómo se ensambla el dataframe de las GHI medidas (ghiqc) con las pronosticadas (GHI_Wm2) por corrida y leadtime.

3) **Validación de los pronósticos del WRF-SMN frente a las observaciones**. En esta etapa se presentan las métricas que permitirán cuantificar el desempeño del modelo. Dichas métricas son MBE, RMSE, rMBE y rRMSE. Estas métricas se

realizan de manera completa, es decir, solo considerando ghiqc vs GHI_Wm2; también se calculan métricas que consideran ghiqc vs GHI_Wm2 por corridas; por último, se consideran métricas de ghiqc vs GHI_Wm2 que consideran las corridas y las leadtimes.

4) **Entrenamiento de MLP para la adaptación al sitio; determinación de la "mejor" MLP**. La herramienta utilizada para corregir los pronósticos del WRF-SMN (disminuir el error) es una MLP. Para entrenar esta red neuronal, primero se debe dividir el conjunto de datos en TRAIN+VALIDATION y TEST. Se entrenan las MLP con los hiperparámetros seleccionados y las variables utilizadas como regresores, obteniéndose una serie de MLP. Esos modelos se prueban con el conjunto de datos VALIDATION de cada MLP y, a partir de las métricas aplicadas a los conjuntos TEST, se elige como el mejor el que minimiza la rMBE y la rRMSE.

Cada etapa se explicará detalladamente y se acompañará de sus respectivos scripts en Python. Al final de la explicación de cada etapa, se adjuntará un esquema gráfico de los pasos metodológicos correspondientes.

2.2. Procedimiento paso a paso

Ahora se desarrollarán los pasos metodológicos mencionados anteriormente. Se debe entender que los pasos 1, 2, 3 y 4 se aplican a los datos de un sitio en particular, mientras que el paso 5 se aplica a todos los sitios estudiados.

Paso 1 – Adquisición y organización de datos medidos, control de calidad e integración horaria

El procedimiento comienza con la adquisición de los datos medidos, es decir, los valores de GHI. Se considerarán, como ejemplo demostrativo, datos medidos en Buenos Aires (`BA`) con frecuencia minutal registrados durante 2023-2024. Los archivos que contienen los datos medidos tienen la extensión csv (comma-separated values).

`BA` es el acrónimo de `B`uenos `A`ires; en caso de usar datos de otra estación, conviene usar acrónimos que permitan identificarla rápidamente en los scripts y en las carpetas de datos.

P1.1) Importación de librerías

Este bloque importa las librerías y módulos necesarios para leer, procesar, graficar y modelar datos geofísicos y meteorológicos en Python 3.14.2, incluyendo herramientas para redes neuronales, control de calidad, manejo espacial y manejo de fechas.

```
import numpy as np
import pandas as pd
import xarray as xr
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import ParameterGrid, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
import joblib
import glob
import cartopy.crs as ccrs
import re
from datetime import datetime, timedelta
import os
from Geo import Geo
from NollasQC import QC
from datetime import timedelta
plt.rcParams['figure.figsize'] = (11,5)
plt.rcParams['font.size'] = 11
sns.set_style("whitegrid")
```

P1.2) Carga de archivos con datos medidos

Antes de comenzar, se debe crear manualmente la carpeta `medidas`, donde se guardan los datos medidos en el sitio `BA` , con columnas '`datetime`' y `'ghi'`.Luego se declaran el código, la latitud, la longitud, la altitud y la zona horaria a la que corresponde la hora (el GMT) del sitio a analizar.

```
SITE_COD = 'BA'
SITE_LAT = -34.60
```

```
SITE_LON = -58.48
SITE_ALT = 30
SITE_GMT = 0
```

Este bloque busca todos los archivos con formato csv de la carpeta `"medidas"` cuyo nombre empieza con el código del sitio (`'BA'`), los carga y los concatena en un único dataframe, y luego muestra qué columnas tiene y un resumen estadístico de sus datos. De esta manera, todos los datos quedan en un solo archivo.

```
#busca todos los archivos csv disponibles en la carpeta 'medidas',
#cuyo nombre siga el patron SITE_COD*
# '*' es un comodin que indica cualquier combinación alfanumérica

files = sorted(glob.glob(f"medidas/{SITE_COD}*.csv"))
print(f"lista de archivos a cargar {files}")
df = pd.concat([pd.read_csv(f,) for f in files], ignore_index=True)

print(f"Columnas dispobiles {df.columns}")
print(df.describe())
```

SALIDA

```
lista de archivos a cargar ['medidas/BA.csv']
Columnas disponibles Index(['datetime', 'ghi'], dtype='object')
                ghi
count  1.053972e+06
mean   1.901120e+02
std    3.008920e+02
min   -8.730964e+00
25%   -1.928934e+00
50%    1.827411e+00
75%    3.051777e+02
max    1.526904e+03
```

Esta SALIDA muestra que contamos con un único archivo, `BA.csv`, que contiene todos los datos disponibles (más de un millón de registros).

P1.3) Análisis de la completitud de los datos medidos

Se realiza un análisis de completitud para determinar la proporción de datos esperados, presentes, faltantes, duplicados y válidos en los datos medidos. Este análisis permite determinar si la serie es apta para la integración horaria y para su posterior comparación con el modelo. Recordemos que, en este caso, los datos medidos tienen una frecuencia minutal.

El siguiente bloque convierte la columna `datetime` al formato de fecha y hora de Python, identifica qué años están presentes en la base de datos y luego grafica la evolución temporal

de la variable `ghi` para observar, mediante una inspección visual, la cobertura y la completitud de los datos medidos.

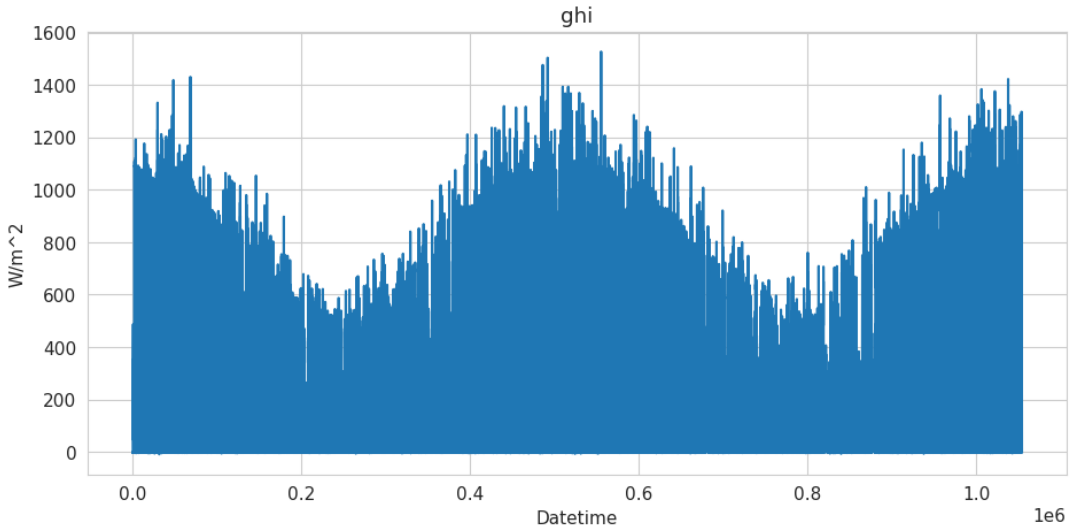
```
# Análisis de completitud  a partir de la etiqueta horaria
# Es necesario recordar que esta notebook presume que
# existe una columna 'datetime' en el set de datos
# cuyo formato respeta el estandar 'ISO8601'

df['datetime'] = pd.to_datetime(df.datetime)
years = df.datetime.dt.year.unique()

print(f"Años disponibles en el set de datos medidos \n {years}")
```

Salida

```
Años disponibles en el set de datos medidos
 [2023 2024]
```

```
plt.figure()
plt.plot(df.ghi.values)
plt.xlabel('Datetime')
plt.ylabel('W/m^2')
plt.title('ghi')
plt.show()
```

Salida



El siguiente código crea una base cronológica de referencia completa, útil para asignar posteriormente correspondencias con los valores medidos.

```
# Malla cronológica completa para los años detectados
tstart = pd.Timestamp(f"{years[0]}-01-01 00:00:00")
tend   = pd.Timestamp(f"{years[-1]}-12-31 23:59:00")
full_time = pd.date_range(start=tstart, end=tend, freq="1min")
df_full = pd.DataFrame({"control_time": full_time})
```

Ahora se realiza un control de calidad temporal, que detecta huecos y registros duplicados, con una resolución de 1 minuto.

```
# Minuto de comparación
df["control_time_min"] = df['datetime'].dt.floor("min")

# Faltantes y duplicados
faltantes =
df_full[~df_full["control_time"].isin(df["control_time_min"])].copy()
counts_per_min = df["control_time_min"].value_counts()
minutos_con_dup = counts_per_min[counts_per_min > 1].index
duplicados = df[df["control_time_min"].isin(minutos_con_dup)].copy()
duplicados["ocurrencias_en_ese_minuto"] =
duplicados["control_time_min"].map(counts_per_min)
```

El siguiente script crea la carpeta `'Procesados'` si no existe. Luego, para cada año en `years`, arma un `DataFrame` de resumen con indicadores de completitud temporal: sitio (`sitio`), año (`anio`), cantidad de minutos esperados (`minutos_esperados`), cantidad de filas observadas (`filas_observadas_crudo`), minutos únicos presentes (`minutos_unicos_presentes`), minutos faltantes (`minutos_faltantes`), filas duplicadas extra (`filas_duplicadas_extra`) y porcentaje de completitud (`porcentaje_completitud_minutos`). Ese resumen se guarda como archivo csv. También guarda, por separado, los minutos faltantes (`faltantes`) y los registros duplicados (`duplicados`).

```
output_dir = 'Procesados'
os.makedirs(output_dir, exist_ok=True)
for year in years:
    # Generación del resumen de completitud
    resumen = pd.DataFrame([{
        "sitio": SITE_COD,
        "anio": year,
        "minutos_esperados": len(df_full),
        "filas_observadas_crudo": len(df),
        "minutos_unicos_presentes": df["control_time_min"].nunique(),
        "minutos_faltantes": len(faltantes),
        "filas_duplicadas_extra": int((counts_per_min -
1).clip(lower=0).sum()),
        "porcentaje_completitud_minutos": 100.0 *
df["control_time_min"].nunique() / len(df_full)
    }])
    base_name = f"{SITE_COD}_{year}"
    resumen_path = os.path.join(output_dir, f"resumen_{base_name}.csv")
    faltantes_path = os.path.join(output_dir,
f"faltantes_{base_name}.csv")
    duplicados_path = os.path.join(output_dir,
f"duplicados_{base_name}.csv")
    clean_path = os.path.join(output_dir,
f"{SITE_COD}_GLOB_{year}_filtrado_1min.csv")
    # Guardado permanente del resumen
```

```
    resumen.to_csv(resumen_path, index=False)
    faltantes.to_csv(faltantes_path, index=False)
    duplicados.to_csv(duplicados_path, index=False)
```

Después, toma una copia de `df`, separa las columnas numéricas y no numéricas, y define una agregación por minuto: para las variables numéricas, usa el promedio; para las no numéricas, conserva el primer valor. Con eso agrupa por `control_time_min`, de modo que, si había más de un registro en el mismo minuto, los consolida en uno solo. Además, agrega una columna `dup_count` con el número de ocurrencias originales por minuto.

```
    # Generación del dataset filtrado en función de la etiqueta
temporal
    df_group = df.copy()
    num_cols =
df_group.select_dtypes(include=[np.number]).columns.difference(["contro
l_time_min"])
    non_num_cols = [c for c in df_group.columns if c not in num_cols
and c != "control_time_min"]
    agg_dict = {c: "mean" for c in num_cols}
    agg_dict.update({c: "first" for c in non_num_cols})
    grouped = df_group.groupby("control_time_min").agg(agg_dict)
    # Asegurar alineación de duplicados
    grouped["dup_count"] =
counts_per_min.reindex(grouped.index).fillna(0).astype(int)
```

Finalmente, reindexa esa tabla agrupada sobre la malla temporal completa `full_time`, de manera que el resultado quede alineado con una serie continua minuto a minuto, incluyendo huecos donde falten datos. Ese dataset final se exporta como `...filtrado_1min.csv`, y luego se imprimen en pantalla las rutas de los archivos generados junto con el resumen.

```
    # ALINEAR A SERIE COMPLETA
    aligned = grouped.reindex(full_time)
    aligned.index.name = "control_time"
    aligned = aligned.reset_index()
    # GUARDAR CSV FINAL
    aligned.to_csv(clean_path, index=False)
    # LOG
    # ==========================
    print(f"[OK] ({SITE_COD}) Resumen → {resumen_path}")
    print(f"[OK] ({SITE_COD}) Faltantes → {faltantes_path}")
    print(f"[OK] ({SITE_COD}) Duplicados → {duplicados_path}")
    print(f"[OK] ({SITE_COD}) Dataset filtrado → {clean_path}")
    print(resumen.to_string(index=False))
```

Salida

```
[OK] (BA) Resumen → Procesados/resumen_BA_2023.csv
[OK] (BA) Faltantes → Procesados/faltantes_BA_2023.csv
```

```
[OK] (BA) Duplicados → Procesados/duplicados_BA_2023.csv
[OK] (BA) Dataset filtrado → Procesados/BA_GLOB_2023_filtrado_1min.csv
sitio  anio  minutos_esperados  filas_observadas_crudo  minutos_unicos_presente
s  minutos_faltantes  filas_duplicadas_extra  porcentaje_completitud_minutos
   BA  2023           1052640                 1053972                   104816
0              4480                    5812                       99.574403
[OK] (BA) Resumen → Procesados/resumen_BA_2024.csv
[OK] (BA) Faltantes → Procesados/faltantes_BA_2024.csv
[OK] (BA) Duplicados → Procesados/duplicados_BA_2024.csv
[OK] (BA) Dataset filtrado → Procesados/BA_GLOB_2024_filtrado_1min.csv
sitio  anio  minutos_esperados  filas_observadas_crudo  minutos_unicos_presente
s  minutos_faltantes  filas_duplicadas_extra  porcentaje_completitud_minutos
   BA  2024           1052640                 1053972                   104816
0              4480                    5812                       99.574403
```

P1.4) Proceso de control de calidad de los datos medidos

Los filtros de Nollas (NollasQC, *Nollas et al., 2023*[3]) sirven para realizar un control general de los valores medidos de GHI. Estos filtros son:

- 👍 $GHI < 1.5\ S\ Esc\ (\cos SZA)^{1.2}+100\ W/m^2$ (1)
- 👍 $GHI > (6.5331 - 0.065502\ SZA + 1.831x10\text{-}4\ SZA^2)/(1+0.001113\ SZA)$ (2)
- 👍 $k_t < 1.4$ (3)

donde:

- S es el factor de corrección de la distancia Tierra-Sol,
- Esc es el valor de la constante solar (1361.1 W/m²),
- SZA es el ángulo solar cenital, definido como el ángulo entre el cenit y la posición aparente del Sol,
- k_t es el índice de claridad, definido como GHI medida / GHI extraterrestre.

Los GHI medidos (denotados como `ghi`) que cumplan estas tres condiciones se consideran aptos; de lo contrario, no se consideran para procesos posteriores. Para aplicar los filtros, deben calcularse variables de geometría solar (que varían según la posición geográfica de cada sitio) para cada valor de GHI medido, asociado a cada time_stamp.

El script toma el DataFrame de mediciones brutas de radiación solar y lo procesa usando la función `Geo` como pieza central: a partir de las fechas y la ubicación geográfica del sitio (latitud, longitud, altitud y zona horaria), `Geo` calcula, para cada instante, el ángulo cenital solar (`SZA`) y la irradiancia teórica de referencia (`GHIargp2`), construyendo la base geométrica sobre la que se apoya todo el análisis posterior. Con esa información, se eliminan automáticamente los datos nocturnos y se asignan valores NaN a todos los

[3] Nollas F, Salazar G, Gueymard C (2023) Quality control procedure for 1-minute pyranometric measurements of global and shadowband-based diffuse solar irradiance, Renewable Energy 202, 40-55, ISSN 0960-1481

registros en los que el Sol está por debajo del horizonte (`SZA > SZA_NIGHT_THRESHOLD`).

Sobre los datos diurnos resultantes se aplica un control de calidad (`QC`) que evalúa si cada medición es físicamente plausible, comparándola con los límites esperados en función del ángulo solar. Solo las mediciones que superan ese filtro se conservan en la columna `ghi` final; todo lo que resulte anómalo se descarta como `NaN`. El resultado es una tabla limpia con las columnas `datetime`, `SZA`, `GHIargp2` y `ghi`, lista para promediarse en intervalos horarios, lo que exige al menos 12 datos válidos por hora para garantizar la representatividad.

```
# Frecuencias a exportar (minutos)
RESAMPLES_MIN = [60]
# Umbral de completitud para promedios horarios
MIN_VALID_60MIN = 12

# Umbral SZA para enmascarar noche
 SZA_NIGHT_THRESHOLD = 90

# ============================
# GEOMETRÍA + PRE-PROCESADO
# ============================

# Asegurar datetime
df['datetime'] = pd.to_datetime(df['datetime'], errors='coerce')

# Geometría solar
g = Geo(
    df['datetime'],    # <-- corregido (antes d)
    lat= SITE_LAT,
    long= SITE_LON ,   # mantiene compatibilidad con Geo
    gmt= SITE_GMT,
    alt= SITE_ALT,
    beta= 0
).df

# Alineo y convierto irradiancia
g = g.copy()
g['ghimeas'] = pd.to_numeric(df['ghi'].to_numpy(), errors='coerce')

# Enmascaro noche: SZA > SZA_NIGHT_THRESHOLD => NaN
g['ghimsk'] = np.where(g['SZA'] > SZA_NIGHT_THRESHOLD, np.nan,
g['ghimeas'])

# ====================
# QUALITY CONTROL QC
# ====================
# Variables que usa NollasQC
```

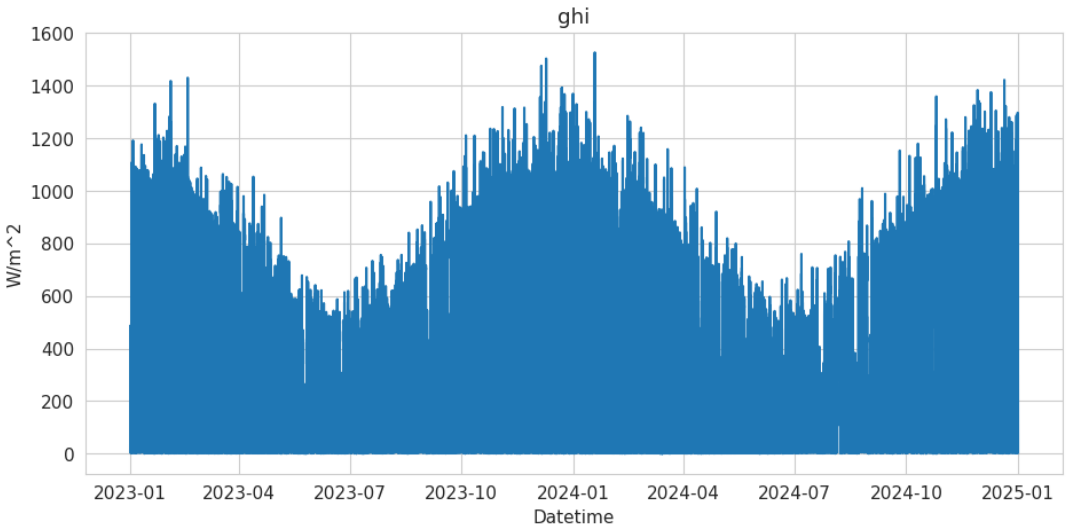
```
# (si preferís que QC corra sobre 'ghimsk', reemplaza ghimeas por
ghimsk)
g['ghi'] = g['ghimeas']
g['TZ']  = g['SZA']  # en grados
g['CTZ'] = np.cos(np.deg2rad(g['SZA']))

# Ejecutar QC (debe crear 'Acepted' o 'Accepted')
QC(g)

accepted_col = 'Acepted' if 'Acepted' in g.columns else ('Accepted' if
'Accepted' in g.columns else None)
if accepted_col is None:
    raise KeyError("La función NollasQC.QC(g) no generó 'Acepted' ni
'Accepted'.")

# Construir ghiqc: aceptado AND SZA<=SZA_NIGHT_THRESHOLD
g['ghiqc'] = np.where(g[accepted_col], g['ghimsk'], np.nan)
g = g[['date','SZA','GHIargp2','ghiqc']]
g.columns = ['datetime','SZA','GHIargp2','ghi']
```

Hasta aquí se han colocado los valores de irradiancia medidos en orden cronológico (con huecos y sin duplicados), ya filtrados (los `ghiqc`) agregando las variables `SZA` y `GHIargp2` con una frecuencia de 1 minuto. En el dataframe final, `date` se renombra a `datetime` y `ghiqc` a `ghi`.

El siguiente bloque grafica la variable de irradiancia solar global medida minutal filtrada `ghi`.

```
plt.figure()
plt.plot(g.datetime, g.ghi)
plt.xlabel('Datetime')
plt.ylabel('W/m^2')
plt.title('ghi')
plt.show()
```

SALIDA



P1.5) Cambio en la frecuencia temporal de los datos medidos

Los datos medidos, originalmente en formato minutal, se integran posteriormente a escala horaria para que sean comparables con los pronósticos del WRF-SMN. La convención adoptada asigna a HH:00 el promedio de los valores comprendidos entre HH:00 y HH:59. Para asegurar la representatividad estadística, una hora solo se considera válida si en cada intervalo de 15 minutos hay al menos tres observaciones válidas, lo que implica un mínimo de 12 datos minutales válidos por hora. Cuando ese criterio no se cumple, la hora se descarta.

El siguiente código remuestrea los datos `ghiqc` a 60 minutos y calcula promedios horarios únicamente cuando hay al menos 12 datos válidos en esa hora; de lo contrario, deja el promedio horario como NaN. Los datos minutales filtrados se guardan inicialmente en el archivo `BA_Quality_Control.csv` y, al finalizar el proceso de integración horaria, en `BA_QC_60min.csv`.

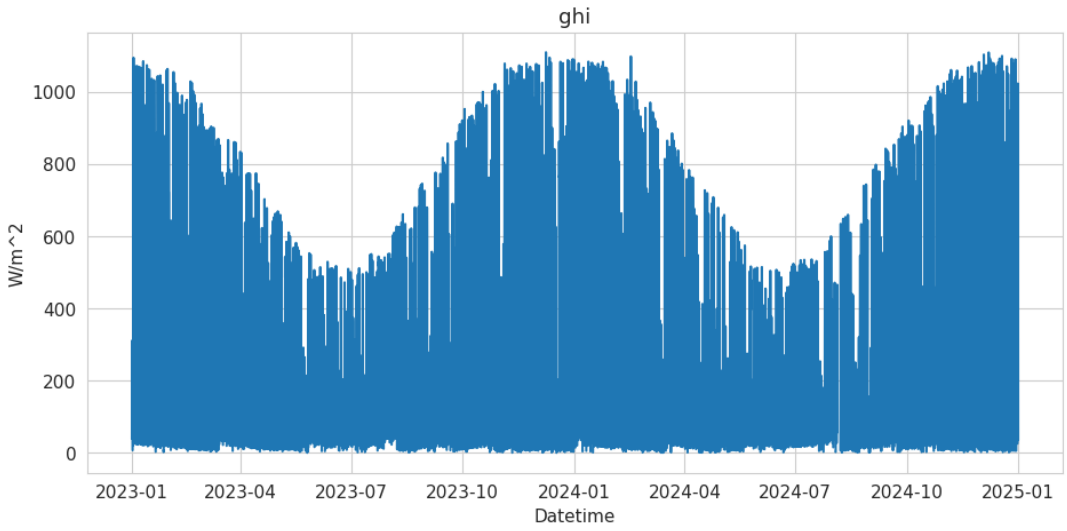
```
g.to_csv(f'{output_dir}/{SITE_COD}_Quality_Control.csv', index=False)
g['datetime'] = pd.to_datetime(g.datetime)
for res in RESAMPLES_MIN:
    # Asegurar que datetime sea índice
    g_res = g.set_index('datetime')

    # Conteo de valores válidos (no NaN)
    valid_count = g_res['ghi'].resample(f'{res}min').count()

    # Promedio
    mean_ghi = g_res['ghi'].resample(f'{res}min').mean()

    # Armar DataFrame
    mean_df = mean_ghi.to_frame(name='ghi')
```

```
    mean_df['n_valid_ghiqc'] = valid_count.astype(int)

    # Umbral dinámico (opcional pero mejor)
    min_valid = int((60 / res) * MIN_VALID_60MIN / 60) if res != 60
else MIN_VALID_60MIN
    # Aplicar filtro de completitud sobre los datos filtrados
    mean_df.loc[mean_df['n_valid_ghiqc'] < min_valid, 'ghi'] = np.nan

    # Guardar
    output_path = f'{output_dir}/{SITE_COD}_QC_{res}min.csv'
    mean_df.reset_index().to_csv(output_path, index=False)
```

Esto grafica los ghi medidos, ahora en frecuencia horaria y ya filtrados por QC.

```
plt.figure()
plt.plot(mean_df.ghi)
plt.xlabel('Datetime')
plt.ylabel('W/m^2')
plt.title('ghi')
plt.show()
```

SALIDA



Este bloque guarda los datos medidos integrados en formato horario `ghi`, con el nombre BA_QC_60min, en la carpeta `Procesados.`

```
archivosMedidas = glob.glob(f'Procesados/{SITE_COD}_QC_60min.csv')
archivosMedidas
```

SALIDA

```
['Procesados/BA_QC_60min.csv']
```

Paso 2 – Adquisición y organización de los datos pronosticados

La primera etapa del procedimiento consiste en la adquisición, organización y verificación de los datos pronosticados por el modelo WRF-SMN, correspondientes al producto *SMN Hi-Res Weather Forecast over Argentina*.

P2.1) Descarga y verificación de los pronósticos

Ese comando sincroniza desde Amazon S3 a tu computadora la carpeta `WRF_4KM/2023/` del bucket `smn-ar-wrf-dataset`, copiando en `./WRF_2023/` todos los archivos nuevos o modificados que aún no tengas localmente.

```
aws s3 sync s3://smn-ar-wrf-dataset/WRF_4KM/2023/ ./WRF_2023/
```

Los archivos tienen la etiqueta *WRFDETAR_01H_AAAAMMdd_CO_HDC.nc*, donde:

- AAAA indica el año de inicio del pronóstico, y se determina usando cuatro números, por ejemplo, 2023.
- MM indica el mes de inicio del pronóstico y se expresa como un número, donde 01 corresponde a enero, 02 a febrero, y así hasta 12 para diciembre.
- dd es el día de inicio del pronóstico, que puede ser un número del 01 al 31.
- CO indica la hora de la corrida y puede ser 00, 06, 12 o 18.
- HDC indica la leadtime L, que es un número que en el archivo va de 0 a 72, pero en realidad solo de 1 a 72, porque es la hora desde el inicio de la corrida.

Cada archivo contiene la información asociada a una ejecución específica del modelo para un horizonte temporal determinado, sobre una retícula que cubre la Argentina (excepto la Antártida), con una resolución espacial de 4 km. El sistema genera cuatro corridas diarias, inicializadas a las 00, 06, 12 y 18 UTC, con frecuencia temporal horaria y un horizonte de predicción de hasta 72 horas. Esta estructura debe considerarse explícitamente durante la descarga, el almacenamiento y el preprocesamiento de la información.

Ese bloque abre un archivo netCDF con la librería xarray (`xr`) y luego muestra en pantalla un resumen de su contenido, incluyendo dimensiones, coordenadas, variables disponibles y metadatos del dataset.

```
# Apertura de un archivo NetCDF de ejemplo
ds = xr.open_dataset("images/WRFDETAR_01H_20230101_00_011.nc")
#listado de variables disponibles
print(ds)
```

Salida

```
<xarray.Dataset> Size: 65MB
Dimensions:            (time: 1, y: 1249, x: 999)
Coordinates:
  * time               (time) datetime64[ns] 8B 2023-01-01T11:00:00
```

```
  * y                   (y) float32 5kB -2.496e+06 -2.492e+06 ... 2.496e+06
  * x                   (x) float32 4kB -1.996e+06 -1.992e+06 ... 1.996e+06
    lon                 (y, x) float32 5MB ...
    lat                 (y, x) float32 5MB ...
Data variables:
    PP                  (time, y, x) float32 5MB ...
    T2                  (time, y, x) float32 5MB ...
    PSFC                (time, y, x) float32 5MB ...
    TSLB                (time, y, x) float32 5MB ...
    SMOIS               (time, y, x) float32 5MB ...
    ACLWDNB             (time, y, x) float32 5MB ...
    ACLWUPB             (time, y, x) float32 5MB ...
    ACSWDNB             (time, y, x) float32 5MB ...
    HR2                 (time, y, x) float32 5MB ...
    dirViento10         (time, y, x) float32 5MB ...
    magViento10         (time, y, x) float32 5MB ...
    Lambert_Conformal   <U1 4B ...
Attributes: (12/17)
    title:          Python PostProcessing for SMN WRF-ARW
    institution:    Servicio Meteorologico Nacional
    source:          OUTPUT FROM WRF V4.0 MODEL
    start_lat:      -56.853172
    start_lon:      -94.33081
    end_lat:        -11.645958
    ...             ...
    TRUELAT1:       -35.0
    TRUELAT2:       -35.0
    DX:             4000.0
    DY:             4000.0
    START_DATE:     2023-01-01 00:00:00
    Conventions:    CF-1.8
```

Para poder manejar múltiples archivos, el código utiliza xarray para abrir y combinar automáticamente en un solo objeto todos los archivos NetCDF de la salida del modelo WRF con resolución temporal de una hora (01H) cuyos nombres coinciden con el patrón `WRFDETAR_01H_202301*.nc`, alineándolos cronológicamente por sus coordenadas mediante `combine='by_coords'` para ofrecer un único dataset consolidado que permite acceder a la serie temporal completa y a las variables atmosféricas proyectadas para esa región a través del atributo `time.values`.

Procesamiento de multiples archivos

```
ds = xr.open_mfdataset("images/WRFDETAR_01H_202301*.nc",
combine='by_coords')
```

```
ds.time.values
```

SALIDA

```
array(['2023-01-01T00:00:00.000000000', '2023-01-01T01:00:00.000000000'
,       '2023-01-01T02:00:00.000000000', '2023-01-01T03:00:00.000000000
',       '2023-01-01T04:00:00.000000000', '2023-01-01T05:00:00.00000000
0',       '2023-01-01T06:00:00.000000000', '2023-01-01T07:00:00.0000000
00',       '2023-01-01T08:00:00.000000000', '2023-01-01T09:00:00.000000
000',       '2023-01-01T10:00:00.000000000', '2023-01-01T11:00:00.00000
0000',       '2023-01-01T12:00:00.000000000', '2023-01-01T13:00:00.0000
00000',       '2023-01-01T14:00:00.000000000', '2023-01-01T15:00:00.000
000000',       '2023-01-01T16:00:00.000000000', '2023-01-01T17:00:00.00
0000000',       '2023-01-01T18:00:00.000000000'], dtype='datetime64[ns]
')
```

P2.2) Extracción por posición geográfica y elección de variables

Este proceso extrae, para un punto geográfico dado (latitud y longitud), los valores de ciertas variables de un archivo WRF en netCDF, convirtiendo primero la ubicación real a la proyección del modelo y agregando, además, la información temporal inferida del nombre del archivo, como la fecha base, la corrida, el leadtime y la hora válida (valid_time). Más en detalle:

- lee del nombre del archivo la fecha, la corrida y el plazo de pronóstico;
- calcula la hora válida sumando el leadtime a la fecha de inicio;
- utiliza las coordenadas geográficas del punto para calcular su posición proyectada en el plano `Lambert Conformal` del modelo
- busca cuál es la celda de grilla más cercana a ese punto;
- extrae de esa celda las variables pedidas;
- devuelve todo en un diccionario llamado `record`.

O sea: toma un archivo WRF y lo convierte en un registro puntual y temporalmente identificado para el sitio de interés.

```python
def extract_point_record(ds, filename, lat_p, lon_p, variables):
    """
    Extrae un punto del dataset teniendo en cuenta la proyección
    y agrega metadata temporal proveniente del nombre del archivo.
    """

    # ---- Extraer info temporal desde el nombre del archivo ----
    fname = os.path.basename(filename)

    match = re.match(r"WRFDETAR_01H_(\d{8})_(\d{2})_(\d{3})\.nc",
fname)
    if not match:
        raise ValueError(f"Nombre de archivo no reconocido: {fname}")

    date_str, corrida, lead_str = match.groups()
```

```
    base_date = datetime.strptime(date_str + corrida, "%Y%m%d%H")
    lead_hours = int(lead_str)
    valid_time = base_date + timedelta(hours=lead_hours)

    # ---- Transformación espacial ----
    data_crs = ccrs.LambertConformal(

central_longitude=ds["Lambert_Conformal"].attrs["longitude_of_central_m
eridian"],

central_latitude=ds["Lambert_Conformal"].attrs["latitude_of_projection_
origin"],

standard_parallels=ds["Lambert_Conformal"].attrs["standard_parallel"],
    )

    x_proj, y_proj = data_crs.transform_point(
        lon_p, lat_p, src_crs=ccrs.PlateCarree()
    )

    xi = abs(ds["x"].values - x_proj).argmin()
    yi = abs(ds["y"].values - y_proj).argmin()

    # ---- Construir registro de salida ----
    record = {
        "date": date_str,
        "corrida": corrida,
        "leadtime": lead_hours,
        "valid_time": valid_time,
    }

    for var in variables:
        if var in ds.variables:
            record[var] = ds[var].isel(x=xi, y=yi).values.item()
        else:
            record[var] = None

    return record
```

Las variables meteorológicas presentes en los archivos son las siguientes:

i) precipitación acumulada en 10 min o 1 hora (PP),
ii) humedad relativa a 2 metros (HR2),
iii) temperatura ambiente a 2 metros (T2),
iv) dirección de viento a 10 metros (dirViento10),
v) magnitud de viento a 10 metros (magViento10),
vi) presión en superficie (PSFC),
vii) radiación de onda larga entrante (ACLWDNB),
viii) radiación de onda larga saliente (ACLWUPB),

ix) radiación de onda corta entrante (ACSWDNB),
x) temperatura de suelo en la capa 0-10 cm (TSLB),
xi) humedad del suelo en la capa 0-10 cm (SMOIS).

Para la presente aplicación, la variable de interés principal es la radiación de onda corta entrante acumulada (ACSWDNB), a partir de la cual se obtiene GHI pronosticada. En el estudio de caso analizado, los datos WRF-SMN presentaron cobertura temporal completa en 2023 y 2024, sin faltantes en las variables seleccionadas y con una secuencia temporal consistente, por lo que constituyeron una base adecuada para las etapas posteriores del procedimiento.

Ese bloque recorre todos los archivos WRF de enero de 2023, extrae, para la ubicación definida al inicio (lat =-24.7°, lon=-64.5°), los valores de las variables seleccionadas de cada archivo, guarda cada extracción como un registro y, finalmente, arma con todos ellos un DataFrame tabular.

```
registros = []

# Buscar archivos reales con glob
files = sorted(glob.glob("images/WRFDETAR_01H_202301*.nc"))

variables = ["PP","HR2","T2","dirViento10",
             "magViento10","PSFC","ACLWDNB",
             "ACLWUPB","ACSWDNB","TSLB","SMOIS",
             "Freezing_level","Tmax","Tmin"]

for f in files:
    with xr.open_dataset(f, engine="h5netcdf", decode_cf=False) as ds:
        rec = extract_point_record(
            ds,
            filename=f,
            lat_p=SITE_LAT,
            lon_p=SITE_LON,
            variables=variables
        )
        registros.append(rec)

df = pd.DataFrame(registros)
df.to_csv(f'modelados/raw/test_{SITE_COD}_202301.csv')
```

Este bloque busca todos los archivos con formato csv cuyo nombre empiece con BA_ en el directorio 'modelados/raw/, los lee uno por uno y los concatena en un único DataFrame llamado df.

```
files = glob.glob(f'modelados/raw/{SITE_COD}_*.csv')
df = pd.concat([pd.read_csv(f) for f in files], ignore_index=True)
```

Este procedimiento calcula la irradiancia global horizontal horaria del modelo (`GHI_Wm2`) a partir de la variable acumulada de radiación `ACSWDNB`, expresada en J/m². Para ello, se obtiene primero el incremento temporal de ACSWDNB dentro de cada corrida y fecha de inicialización, mediante una diferencia entre registros consecutivos agrupados por `date` y `corrida`. Luego, dicho incremento energético se divide entre la duración del intervalo temporal (3600 s para archivos con resolución 01H), con el fin de convertir la energía acumulada en potencia media horaria por unidad de área, expresada en W/m². Dado que en el primer `leadtime` de cada corrida no existe un valor previo con el que calcular la diferencia, el valor de `GHI_Wm2` correspondiente se asigna como NaN. Finalmente, el DataFrame se reorganiza, conservando únicamente las variables meteorológicas de interés, junto con la nueva variable derivada `GHI_Wm2`, que representa la estimación horaria de la irradiancia modelada.

```
var = 'ACSWDNB'
df["delta_Jm2"] = df.groupby(["date", "corrida"])[var].diff()
df["dt_seconds"] = 3600  # porque usamos 01H
df["GHI_Wm2"] = df["delta_Jm2"] / df["dt_seconds"]

# Manejar NaN en el primer lead de cada corrida
df.loc[df["leadtime"] == 0, "GHI_Wm2"] = np.nan

print(df.head())

df.columns
df = df[['date', 'corrida', 'leadtime', 'valid_time', 'PP', 'HR2',
'T2',
    'dirViento10', 'magViento10', 'PSFC', 'ACLWDNB', 'ACLWUPB',
'ACSWDNB',
    'TSLB', 'SMOIS',  'GHI_Wm2']]
```

SALIDA

```
 date  corrida  leadtime           valid_time        PP        HR2  \
0  20240901        0         0  2024-09-01 00:00:00  0.000000  96.12728
9
1  20240901        0         1  2024-09-01 01:00:00  0.000035  92.74086
0
2  20240901        0         2  2024-09-01 02:00:00  0.001601  91.19950
1
3  20240901        0         3  2024-09-01 03:00:00  0.023941  90.89167
0
4  20240901        0         4  2024-09-01 04:00:00  0.086187  89.61514
3

          T2  dirViento10  magViento10         PSFC  ...    ACLWUPB  AC
SWDNB  \
0  10.562103   193.054535     3.885563  1012.641785  ...        0.0
  0.0
```

```
1  10.557678   197.190826     4.578183  1013.689514  ...  1293872.0
  0.0
2  10.162109   200.315445     5.358886  1014.447327  ...  2594926.5
  0.0
3  10.084930   200.440948     6.141666  1014.607971  ...  3891515.5
  0.0
4   9.745514   198.437744     4.636292  1014.446167  ...  5184057.5
  0.0

       TSLB     SMOIS  Freezing_level  Tmax  Tmin  delta_Jm2  dt_second
s  \
0  9.975403  0.407000     2176.957764   NaN   NaN        NaN         360
0
1  9.855621  0.406548     2126.133545   NaN   NaN        0.0         360
0
2  9.773193  0.406096     2101.175293   NaN   NaN        0.0         360
0
3  9.618683  0.405848     1990.484009   NaN   NaN        0.0         360
0
4  9.452728  0.406196     2002.267944   NaN   NaN        0.0         360
0

   GHI_Wm2
0      NaN
1      0.0
2      0.0
3      0.0
4      0.0

[5 rows x 21 columns]
```

Solo para L = 0, las variables de radiación solar presentan valores de 0. Para ese momento, `GHI_Wm2` se establece en NaN, ya que el primer valor pronosticado se observa en L = 1. De esa manera se evita asignar un valor de 0 a la `GHI_Wm2` de una date y corrida cuyo valid_time, para L = 0, puede estar en el periodo diurno.

Nota: Durante el periodo diurno, cuando el Sol está por encima del horizonte, el valor de GHI nunca puede ser 0 W/m². Esto se debe a que la GHI es la suma de la irradiancia solar difusa (DHI) y la irradiancia solar directa normal (DNI) y, si bien la DNI puede ser 0 cuando el cielo está nublado, la DHI siempre es positiva, incluso antes del amanecer.

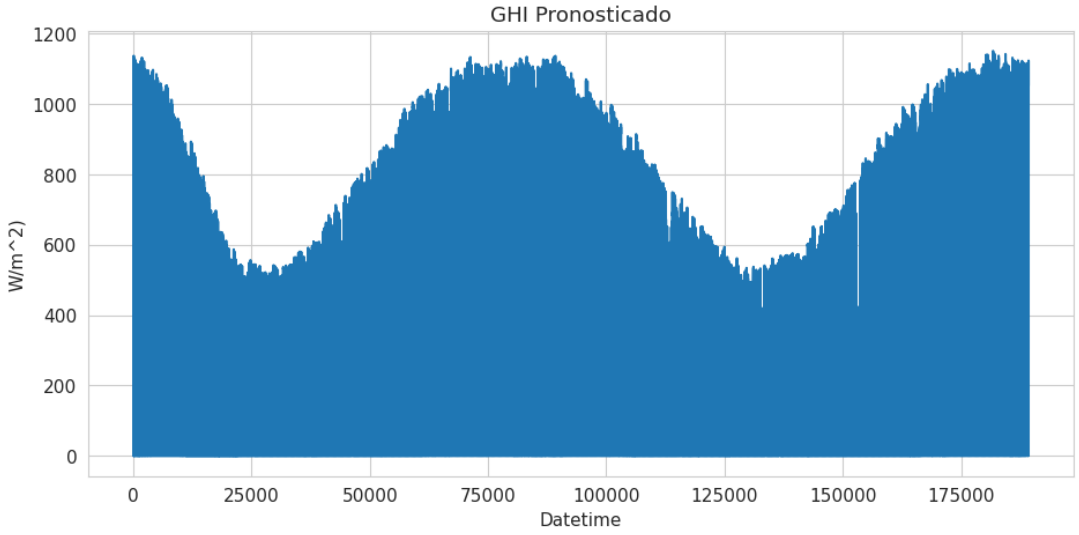
Para ordenar el DataFrame según tres claves jerárquicas: `valid_time`, por `corrida` y, dentro de cada `valid_time`, por `leadtime`, usamos el siguiente código.
Desde el punto de vista técnico, esto reorganiza los registros en orden cronológico de los pronósticos válidos y, además, mantiene un orden coherente entre las distintas corridas y

horizontes de predicción. Luego, `reset_index(drop=True)` reconstruye el índice del DataFrame de forma secuencial a partir de 0, eliminando el índice previo que quedó desordenado tras el reordenamiento.

```
df =
df.sort_values(by=["valid_time","corrida","leadtime"]).reset_index(drop
=True)
```

```
plt.figure()
plt.plot(df['GHI_Wm2'].values)
plt.xlabel('Datetime')
plt.ylabel('W/m^2)')
plt.title('GHI Pronosticado')
plt.show()
```

SALIDA



Los scripts siguientes realizan la carga de todos los archivos Excel de mediciones del sitio (`SITE_COD`) desde la carpeta medidas/, los concatena en un único DataFrame, y les ajusta la fecha sumando 1 hora (corrección de zona horaria). Luego, hace un merge entre esos datos medidos y un DataFrame de pronósticos (df) cruzando por fecha y hora de `valid_time`, de modo que solo quedan los instantes en los que ambos registros coinciden. Finalmente, toma la corrida número 0 y grafica un rango de ~5 días (horas 72 a 432 `(72*6)`) comparando la radiación solar horizontal global modelada (`GHI_Wm2`) contra la medida con control de calidad (`ghiqc`), permitiendo evaluar visualmente la precisión del pronóstico

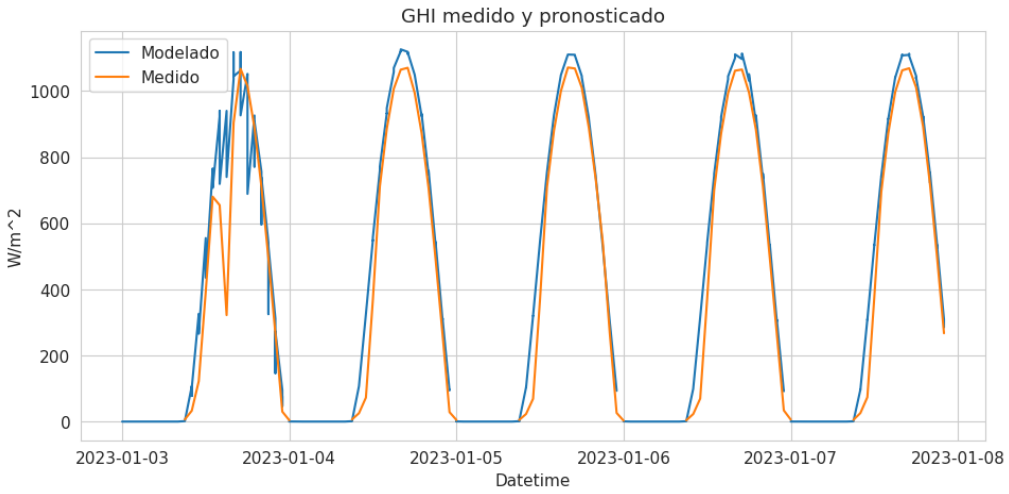
```
filesMedidos = glob.glob(f'medidas/{SITE_COD}_*.xlsx')
dfMedidos = pd.concat([pd.read_excel(f) for f in filesMedidos],
ignore_index=True)
```

```
dfMedidos['date'] = pd.to_datetime(dfMedidos['date']) +
timedelta(hours= +1)
```

```
df['valid_time'] = pd.to_datetime(df['valid_time'])
df = pd.merge(df, dfMedidos, how='inner', left_on='valid_time',
right_on='date', suffixes=('_modelado', '_medido'))
```

```
corrida = 0
msk = df['corrida'] == corrida
plt.figure()
plt.plot(df[msk]['valid_time'][72:72*6], df[msk]['GHI_Wm2'][72:72*6],
label='Modelado')
plt.plot(df[msk]['valid_time'][72:72*6], df[msk]['ghiqc'][72:72*6],
label='Medido')
plt.xlabel('Datetime')
plt.ylabel('W/m^2')
plt.title('GHI medido y pronosticado')
plt.legend()
plt.show()
```

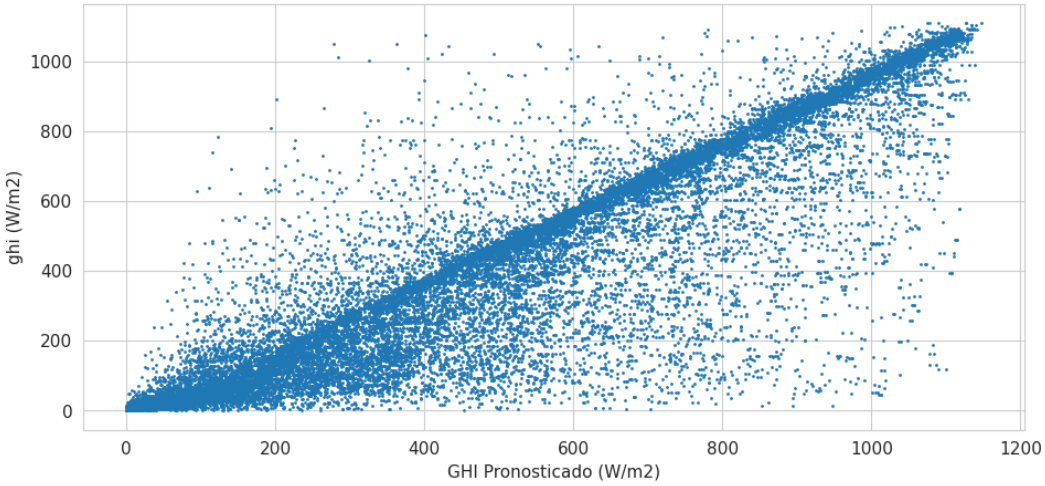
SALIDA



Este script permite graficar la variable GHI pronosticada (`GHI_Wm2`) frente a la medida (`ghiqc`).

```
plt.figure()
plt.scatter(df[msk]['GHI_Wm2'],df[msk]['ghiqc'], s=1)
plt.xlabel('GHI Pronosticado (W/m2)')
plt.ylabel('ghi (W/m2)')
plt.show()
```

SALIDA


ghi (W/m2)
GHI Pronosticado (W/m2)
0
200
400
600
800
1000
1200

Paso 3 – Validación de los pronósticos del WRF-SMN frente a las observaciones

El desempeño del modelo se cuantifica mediante las métricas MBE y RMSE, así como sus versiones relativas rMBE y rRMSE. Estas métricas se pueden calcular por corrida y por leadtime, lo que permite evaluar no solo el error global, sino también su evolución a medida que aumenta el horizonte de pronóstico.

El rMBE mide el sesgo sistemático del pronóstico e indica sobreestimación cuando es positivo y subestimación cuando es negativo.

$$MBE = \frac{1}{N}\sum_{i=1}^{N}\left(y_{i,modelado} - y_{i,medido}\right) \quad (5)$$

$$rMBE = \frac{MBE}{y_{medido}}.100 \quad (6)$$

El rRMSE mide el tamaño típico del error y penaliza más los errores grandes.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y_{i,modelado} - y_{i,medido}\right)^2} \quad (7)$$

$$rRMSE = \frac{RMSE}{y_{medido}}.100 \quad (8)$$

Las comparaciones se hicieron de tres maneras:

i) Usando todos los datos de GHI medidos y pronosticados. Ver P3.1.

ii) Usando todos los datos de GHI medidos y pronosticados, discriminados por corrida (0, 6, 12 y 18). Ver P3.2.

iii) Usando todos los datos medidos y pronosticados, discriminados por corrida (0, 6, 12 y 18) y por leadtime (1, 2, 3, …, 70, 71 y 72). Ver P3.3.

P3.1) Discriminación del error por GHI medido y modelado

El bloque siguiente define las dos métricas de error relativo expresadas en porcentaje: rMBE, que cuantifica el sesgo medio relativo del modelo, y rRMSE, que evalúa la raíz del error cuadrático medio relativo, otorgando mayor peso a los errores grandes. Luego, se eliminan los registros con valores faltantes mediante `df.dropna()` y se calculan dichas métricas comparando la irradiancia medida (`ghiqc`) con la irradiancia modelada (`GHI_Wm2`). Finalmente, los resultados se imprimen en pantalla como una evaluación global del desempeño del modelo WRF en GHI antes de aplicar cualquier corrección.

```
def rmbe(true, pred):
    mbe_loss = np.mean(pred  - true)
    return mbe_loss/ true.mean() * 100

def rrmse(true, pred):
```

```
    return np.sqrt(sum((pred - true) ** 2) / true.size)  / true.mean()
* 100
```

```
df = df.dropna()
```

```
print("Desempeño general del modelo WRF para GHI antes de la
corrección:")
print(f"rMBE: {rmbe(df['ghiqc'], df['GHI_Wm2'])}%")
print(f"rRMSE: {rrmse(df['ghiqc'], df['GHI_Wm2'])}%")
```

SALIDA

```
Desempeño general del modelo WRF para GHI antes de la corrección:
rMBE: 24.428973968300244%
rRMSE: 45.67742887337274%
```

P3.2) Discriminación del error por corrida

El siguiente bloque calcula el desempeño del modelo WRF para GHI discriminado por `corrida`. Para ello, recorre cada valor único de la variable corrida, selecciona los registros correspondientes mediante una máscara lógica y, en cada caso, calcula las métricas rMBE, rMAE y rRMSE, comparando la irradiancia modelada (`GHI_Wm2`) con la medida (`ghiqc`). De este modo, se obtiene una evaluación independiente del error para cada corrida del modelo, lo que permite analizar si el desempeño varía según la hora de inicio del pronóstico.

```
print("Desempeño del modelo WRF para GHI antes de la corrección por
corrida:")
for corrida in df['corrida'].unique():
    msk = df['corrida'] == corrida
    print(f"Corrida {corrida}:")
    print(f"  rMBE: {rmbe(df[msk]['ghiqc'], df[msk]['GHI_Wm2'])}%")
    print(f"  rRMSE: {rrmse(df[msk]['ghiqc'], df[msk]['GHI_Wm2'])}%")
```

SALIDA

```
Desempeño del modelo WRF para GHI antes de la corrección por corrida:
Corrida 0:
  rMBE: 23.802896142738696%
  rRMSE: 45.4972779437327%
Corrida 12:
  rMBE: 24.60973554098931%
  rRMSE: 44.856436575188425%
Corrida 6:
  rMBE: 24.780335372994514%
  rRMSE: 46.361016464883946%
Corrida 18:
  rMBE: 24.651016658811304%
  rRMSE: 46.300704410428715%
```

P3.3) Discriminación del error por corrida y leadtime

En el siguiente bloque se evalúa el desempeño del modelo WRF para GHI de manera desagregada por `corrida` y por `leadtime`. Para cada combinación de hora de inicialización del pronóstico y horizonte de predicción, se seleccionan los registros correspondientes y se calculan las métricas rMBE y rRMSE comparando la irradiancia modelada (`GHI_Wm2`) con la irradiancia medida (`ghiqc`). El uso de **`try`/`except`** permite evitar errores cuando no hay suficientes datos para realizar el cálculo y también informar de la situación en pantalla. Este análisis permite identificar cómo varía el error del modelo no solo entre corridas, sino también en función de la anticipación del pronóstico.

```
print("Desempeño del modelo WRF para GHI antes de la corrección por
corrida y leadtime:")
for corrida in df['corrida'].unique():
    for leadtime in sorted(df['leadtime'].unique()):
        msk = (df['corrida'] == corrida) & (df['leadtime'] == leadtime)
        try:
            print(f"Corrida {corrida}, Leadtime {leadtime}h:")
            print(f"  rMBE: {rmbe(df[msk]['ghiqc'],
df[msk]['GHI_Wm2'])}%")
            print(f"  rRMSE: {rrmse(df[msk]['ghiqc'],
df[msk]['GHI_Wm2'])}%")
        except:
            print(f"Corrida {corrida}, Leadtime {leadtime}h: No hay
datos suficientes.")
```

A diferencia del bloque anterior (que solo imprimía resultados parciales en la consola), este bloque analiza cómo varía el error del modelo con el leadtime y lo representa gráficamente por corrida, estructurando formalmente los datos que antes solo se previsualizaron en la consola. En una primera etapa, el código recorre todos los horizontes de pronóstico disponibles, selecciona los pares válidos `ghiqc - GHI_Wm2` y calcula las métricas rMBE, rMAE y rRMSE únicamente cuando se cuenta con al menos 10 datos para asegurar la representatividad estadística; estos resultados se almacenan en una lista que luego se convierte en el DataFrame `df_metrics`. En una segunda etapa, se realiza un análisis desagregado por ciclo de inicialización para calcular el rRMSE para cada combinación de corrida y leadtime, lo que permite finalmente graficar la evolución del error. Esta visualización es clave para evaluar la degradación del desempeño del modelo a medida que aumenta la anticipación del pronóstico.

```
print("Desempeño del modelo WRF para GHI antes de la corrección por
corrida y leadtime:")
for corrida in df['corrida'].unique():
    for leadtime in sorted(df['leadtime'].unique()):
        msk = (df['corrida'] == corrida) & (df['leadtime'] == leadtime)
        try:
            print(f"Corrida {corrida}, Leadtime {leadtime}h:")
            print(f"  rMBE: {rmbe(df[msk]['ghiqc'],
df[msk]['GHI_Wm2'])}%")
```

```
            print(f"  rRMSE: {rrmse(df[msk]['ghiqc'],
df[msk]['GHI_Wm2'])}%")
        except:
            print(f"Corrida {corrida}, Leadtime {leadtime}h: No hay
datos suficientes.")
```

SALIDA

```
Desempeño del modelo WRF para GHI antes de la corrección por corrida y
leadtime:
Corrida 0, Leadtime 1h:
  rMBE: nan%
Corrida 0, Leadtime 1h: No hay datos suficientes.
Corrida 0, Leadtime 2h:
  rMBE: nan%
Corrida 0, Leadtime 2h: No hay datos suficientes.
Corrida 0, Leadtime 3h:
  rMBE: nan%
Corrida 0, Leadtime 3h: No hay datos suficientes.
Corrida 0, Leadtime 4h:
  rMBE: nan%
Corrida 0, Leadtime 4h: No hay datos suficientes.
Corrida 0, Leadtime 5h:
  rMBE: nan%
Corrida 0, Leadtime 5h: No hay datos suficientes.
Corrida 0, Leadtime 6h:
  rMBE: nan%
Corrida 0, Leadtime 6h: No hay datos suficientes.
Corrida 0, Leadtime 7h:
  rMBE: nan%
Corrida 0, Leadtime 7h: No hay datos suficientes.
Corrida 0, Leadtime 8h:
  rMBE: nan%
Corrida 0, Leadtime 8h: No hay datos suficientes.
Corrida 0, Leadtime 9h:
  rMBE: -38.16161591682842%
  rRMSE: 54.656401586528546%
Corrida 0, Leadtime 10h:
  rMBE: 173.1205770333872%
  rRMSE: 240.93959689611162%
Corrida 0, Leadtime 11h:
(la lista continua )
```

Aquí solo se calcula el error por `leadtime`

```
resumen = []

for leadtime in sorted(df['leadtime'].unique()):

    subset = df[df['leadtime'] == leadtime][['ghiqc',
'GHI_Wm2']].dropna()

    if subset.shape[0] < 10:
        continue
```

```
    resumen.append({
        'leadtime': leadtime,
        'rMBE': rmbe(subset['ghiqc'], subset['GHI_Wm2']),
        'rRMSE': rrmse(subset['ghiqc'], subset['GHI_Wm2'])
    })

df_metrics = pd.DataFrame(resumen)
```

```
df.columns
```

SALIDA

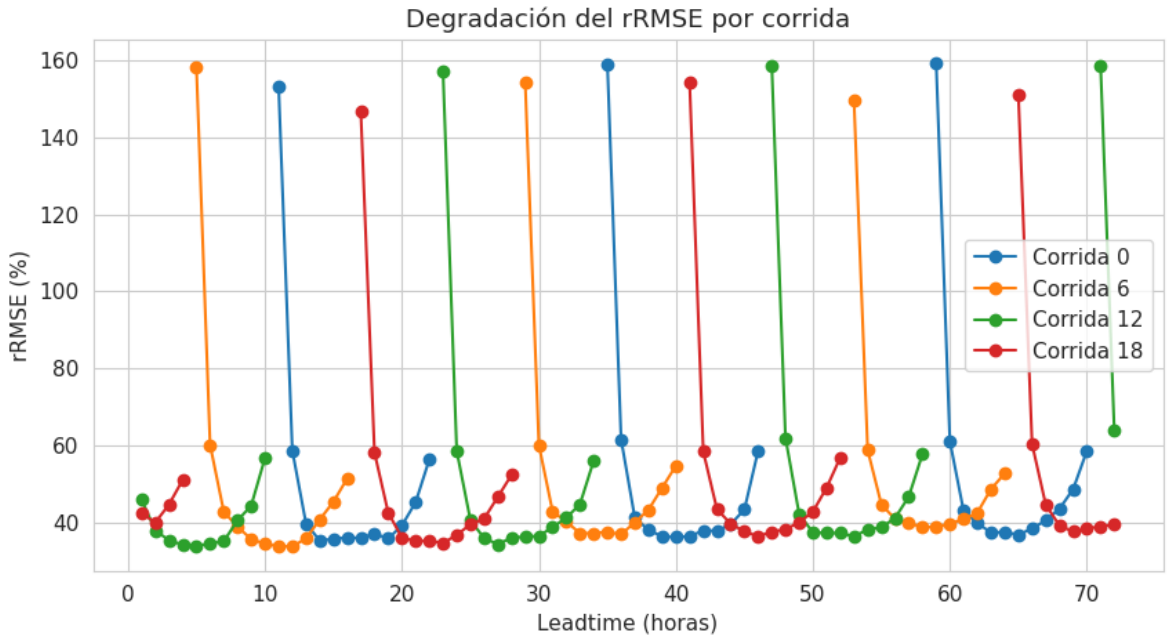
```
Index(['date_modelado', 'corrida', 'leadtime', 'valid_time', 'PP', 'HR2
', 'T2',
       'dirViento10', 'magViento10', 'PSFC', 'ACLWDNB', 'ACLWUPB', 'ACS
WDNB',
       'TSLB', 'SMOIS', 'GHI_Wm2', 'date_medido', 'ghiqc', 'SZA', 'GHIa
rgp2'],
      dtype='object')
```

Aquí se grafica el error por corrida y por leadtime.

```
plt.figure(figsize=(10, 5))

for corrida in sorted(df['corrida'].unique()):
    sub = df[(df['corrida'] == corrida) & (df['SZA'] < 80)]

    valores = []
    for leadtime in sorted(sub['leadtime'].unique()):
        ss = sub[sub['leadtime'] == leadtime][['ghiqc',
'GHI_Wm2']].dropna()

        if ss.shape[0] < 10:
            continue

        valores.append((leadtime, rrmse(ss['ghiqc'], ss['GHI_Wm2'])))

    if valores:
        lt, err = zip(*valores)

        lt = np.array(lt)
        err = np.array(err)

        # cortar líneas donde hay saltos
        x_plot = [lt[0]]
        y_plot = [err[0]]

        for i in range(1, len(lt)):
            if lt[i] - lt[i-1] > 1:
                x_plot.append(np.nan)
                y_plot.append(np.nan)

            x_plot.append(lt[i])
            y_plot.append(err[i])
```

```
        plt.plot(x_plot, y_plot, marker='o', label=f'Corrida
{corrida}')

plt.title("Degradación del rRMSE por corrida")
plt.xlabel("Leadtime (horas)")
plt.ylabel("rRMSE (%)")
plt.legend()
plt.grid(True)

plt.show()
```

SALIDA

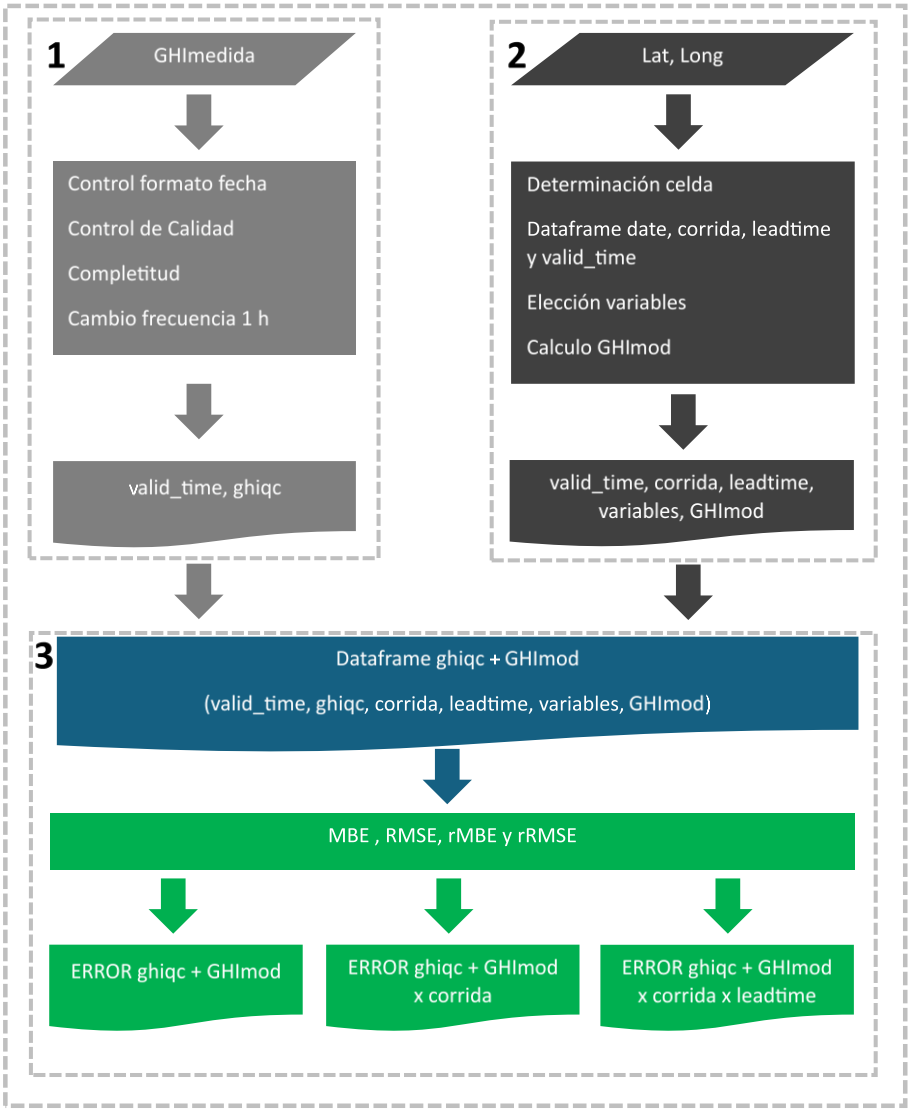


ESQUEMA CONCEPTUAL

PASO 1

PASO 2

PASO 3



Paso 4 – Entrenamiento de MLP para la adaptación al sitio; determinación de la mejor MLP

Una vez cuantificado el error del WRF-SMN, se construye una estrategia de adaptación al sitio basada en un perceptrón multicapa (MLP). El objetivo del MLP es aprender las características de los datos y generar una función de corrección que acerque los valores pronosticados de la GHI a los valores medidos. Para lograr esto, se usan los datos integrados, tanto medidos como pronosticados, emparejados por valid_time.

P4.1) Definición de los hiperparámetros de la MLP

Dado que una MLP (Multilayer Perceptron) es una red neuronal artificial, su funcionamiento depende de la configuración previa de una serie de hiperparámetros. En la librería scikit-learn de Python[4], los principales hiperparámetros a configurar son los siguientes:

i) *Hidden layers:* permiten definir la arquitectura de la red neuronal, es decir, la cantidad de capas ocultas y el número de neuronas en cada una, lo que determina la complejidad de la red y su capacidad para representar las relaciones entre las variables de entrada y de salida. Ejemplo: hidden_layer_sizes = (N, 2N+1, 1) define una red con dos capas ocultas, la primera con N neuronas, la segunda con 2N+1, y la última es 1 como salida.

ii) *Activation*: permite seleccionar la función de activación aplicada a las neuronas de las capas ocultas, introduciendo no linealidad en el modelo. Ejemplo: activation = 'relu' indica que se utiliza la función rectified linear unit, ampliamente empleada en problemas de regresión y clasificación.

iii) *Learning rate*: define la tasa de aprendizaje utilizada durante el entrenamiento, que controla el tamaño de los cambios en los pesos de la red en cada iteración. Valores altos pueden acelerar el proceso de ajuste, mientras que valores bajos favorecen un aprendizaje más gradual. Ejemplo: learning_rate_init = 0.001 establece una tasa de aprendizaje inicial de 0.001, un valor comúnmente utilizado como punto de partida.

iv) *Solver*: permite seleccionar el algoritmo de optimización para ajustar los pesos y sesgos de la red durante el entrenamiento. Su elección influye en la velocidad de convergencia y la calidad final del modelo obtenido. Ejemplo: solver = 'adam' utiliza un algoritmo de optimización eficiente y muy usado en redes neuronales.

v) *Alpha*: define el nivel de regularización aplicado al modelo. Su función es penalizar pesos excesivamente altos para reducir el riesgo de sobreajuste y mejorar la capacidad de generalización de la red. Ejemplo: Alpha = 0.0001 implica una regularización pequeña, suficiente en muchos casos para estabilizar el entrenamiento sin afectar en exceso la capacidad de ajuste.

[4] https://scikit-learn.org/stable/

Nota: El cálculo del número total de modelos MLP mencionados anteriormente se refiere a la estrategia de Grid Search (búsqueda en grilla) implementada en este estudio. Dado que cada hiperparámetro representa una dimensión distinta en el espacio de búsqueda, el número total de configuraciones que el algoritmo debe entrenar y evaluar se obtiene del producto de las opciones definidas para cada sección. Por ejemplo, al combinar 3 arquitecturas de capas ocultas, 1 función de activación, 2 tasas de aprendizaje y 2 alphas, se generan 12 modelos independientes (3 x 1 x 2 x 2 = 12). Esta aclaración es fundamental para justificar la carga computacional del proceso de optimización orientado a encontrar la arquitectura más eficiente en nuestro caso particular.

P4.2) Análisis de correlación entre las variables

Este bloque calcula y visualiza la matriz de correlación lineal de Pearson entre las variables numéricas del DataFrame. Primero, selecciona únicamente las columnas de tipo numérico con `select_dtypes(include=[np.number])`. Luego, excluye la variable `ghiqc` si está presente, para que no participe del análisis. A continuación, calcula la matriz de correlación con `df_num.corr(method="pearson")`, obteniendo para cada par de variables, un coeficiente de correlación entre −1 y 1 que cuantifica la intensidad y el signo de su relación lineal.

Esto se hace para que el usuario establezca las correlaciones entre todas las variables del modelo entre sí.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Suponiendo que ya tenés tu DataFrame cargado como df

# Seleccionar solo columnas numéricas
df_num = df.select_dtypes(include=[np.number])

# Excluir ghiqc
if "ghiqc" in df_num.columns:
    df_num = df_num.drop(columns=["ghiqc"])

# Calcular matriz de correlación lineal (Pearson)
corr_matrix = df_num.corr(method="pearson")

print(corr_matrix)

# ---- Gráfico tipo heatmap ----
plt.figure(figsize=(12, 10))

sns.heatmap(
```

```
    corr_matrix,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    square=True,
    linewidths=0.5
)

plt.title("Correlación lineal (Pearson) entre variables")
plt.tight_layout()
plt.show()
```

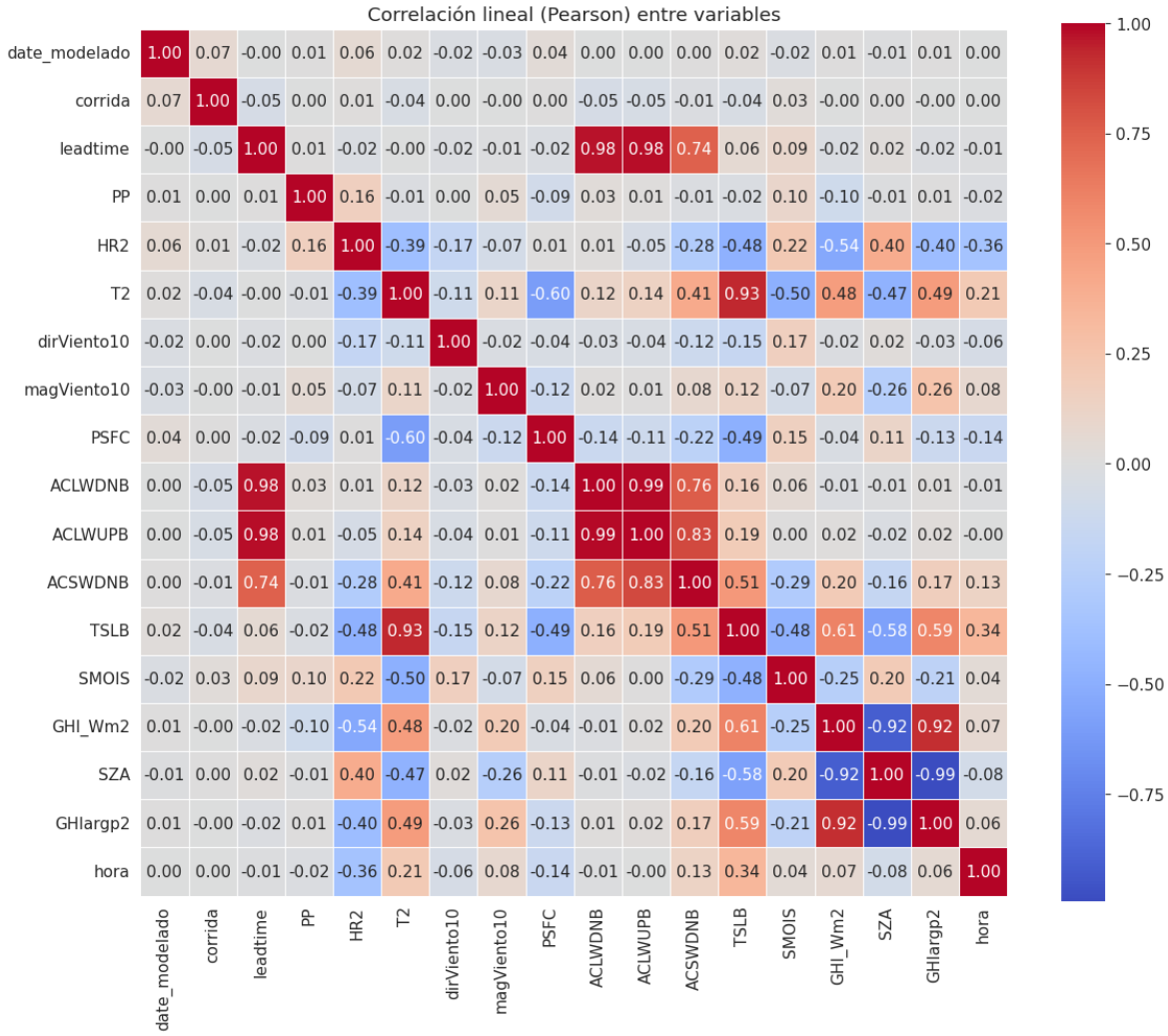
SALIDA

```
              date_modelado   corrida  leadtime        PP        HR2  \
date_modelado       1.000000  0.065964 -0.003284  0.006343  0.062041
corrida             0.065964  1.000000 -0.053988  0.001464  0.012515
leadtime           -0.003284 -0.053988  1.000000  0.005916 -0.024582
PP                  0.006343  0.001464  0.005916  1.000000  0.158780
HR2                 0.062041  0.012515 -0.024582  0.158780  1.000000
T2                  0.017338 -0.043011 -0.001353 -0.013085 -0.386837
dirViento10        -0.021964  0.001865 -0.015251  0.000303 -0.170784
magViento10        -0.032124 -0.000248 -0.009300  0.050699 -0.067473
PSFC                0.040569  0.004508 -0.023193 -0.086238  0.009828
ACLWDNB             0.002586 -0.054596  0.975249  0.027545  0.013228
ACLWUPB             0.000538 -0.051633  0.982353  0.012315 -0.054113
ACSWDNB             0.003497 -0.014633  0.743744 -0.013096 -0.282285
TSLB                0.019668 -0.037826  0.058997 -0.022687 -0.482861
SMOIS              -0.021774  0.029864  0.094981  0.096686  0.220074
GHI_Wm2             0.007663 -0.004696 -0.023283 -0.099124 -0.537469
SZA                -0.008515  0.003589  0.019561 -0.009221  0.402118
GHIargp2            0.009707 -0.004430 -0.020096  0.010562 -0.402606
hora                0.000090  0.000123 -0.007955 -0.019834 -0.358685

                        T2  dirViento10  magViento10      PSFC   ACLWDNB
\
date_modelado  0.017338    -0.021964    -0.032124  0.040569  0.002586

corrida       -0.043011     0.001865    -0.000248  0.004508 -0.054596

leadtime      -0.001353    -0.015251    -0.009300 -0.023193  0.975249

PP            -0.013085     0.000303     0.050699 -0.086238  0.027545

HR2           -0.386837    -0.170784    -0.067473  0.009828  0.013228

T2             1.000000    -0.105804     0.105749 -0.599385  0.117168

dirViento10   -0.105804     1.000000    -0.021980 -0.041704 -0.031297

magViento10    0.105749    -0.021980     1.000000 -0.124238  0.016281

PSFC          -0.599385    -0.041704    -0.124238  1.000000 -0.136862
```

ACLWDNB	0.117168	-0.031297	0.016281	-0.136862	1.000000
ACLWUPB	0.135611	-0.042063	0.008872	-0.108994	0.988589
ACSWDNB	0.411857	-0.120568	0.077184	-0.219425	0.763952
TSLB	0.933736	-0.154145	0.123997	-0.492827	0.161789
SMOIS	-0.495636	0.165641	-0.074368	0.152630	0.057041
GHI_Wm2	0.481416	-0.018223	0.199112	-0.035725	-0.012969
SZA	-0.472386	0.024134	-0.256802	0.114493	-0.012368
GHIargp2	0.487514	-0.027317	0.258910	-0.125511	0.014539
hora	0.212718	-0.058506	0.078581	-0.138222	-0.009263

	ACLWUPB	ACSWDNB	TSLB	SMOIS	GHI_Wm2	SZA
date_modelado	0.000538	0.003497	0.019668	-0.021774	0.007663	-0.008515
corrida	-0.051633	-0.014633	-0.037826	0.029864	-0.004696	0.003589
leadtime	0.982353	0.743744	0.058997	0.094981	-0.023283	0.019561
PP	0.012315	-0.013096	-0.022687	0.096686	-0.099124	-0.009221
HR2	-0.054113	-0.282285	-0.482861	0.220074	-0.537469	0.402118
T2	0.135611	0.411857	0.933736	-0.495636	0.481416	-0.472386
dirViento10	-0.042063	-0.120568	-0.154145	0.165641	-0.018223	0.024134
magViento10	0.008872	0.077184	0.123997	-0.074368	0.199112	-0.256802
PSFC	-0.108994	-0.219425	-0.492827	0.152630	-0.035725	0.114493
ACLWDNB	0.988589	0.763952	0.161789	0.057041	-0.012969	-0.012368
ACLWUPB	1.000000	0.828779	0.191300	0.003663	0.016315	-0.021015
ACSWDNB	0.828779	1.000000	0.508837	-0.289668	0.200686	-0.163470
TSLB	0.191300	0.508837	1.000000	-0.484313	0.605210	-0.580004
SMOIS	0.003663	-0.289668	-0.484313	1.000000	-0.253977	0.199652
GHI_Wm2	0.016315	0.200686	0.605210	-0.253977	1.000000	-0.915910
SZA	-0.021015	-0.163470	-0.580004	0.199652	-0.915910	1.000000

```
GHIargp2       0.023596  0.172200  0.593642 -0.213541  0.918590 -0.9942
84
hora          -0.001084  0.128232  0.344512  0.043508  0.070835 -0.0766
95

                GHIargp2      hora
date_modelado   0.009707  0.000090
corrida        -0.004430  0.000123
leadtime       -0.020096 -0.007955
PP              0.010562 -0.019834
HR2            -0.402606 -0.358685
T2              0.487514  0.212718
dirViento10    -0.027317 -0.058506
magViento10     0.258910  0.078581
PSFC           -0.125511 -0.138222
ACLWDNB         0.014539 -0.009263
ACLWUPB         0.023596 -0.001084
ACSWDNB         0.172200  0.128232
TSLB            0.593642  0.344512
SMOIS          -0.213541  0.043508
GHI_Wm2         0.918590  0.070835
SZA            -0.994284 -0.076695
GHIargp2        1.000000  0.062102
hora            0.062102  1.000000
```



El siguiente bloque cuantifica la relación entre la variable objetivo `ghiqc` y un conjunto predefinido de variables explicativas, utilizando dos medidas de asociación: Pearson, para detectar dependencia lineal, y Spearman, para detectar relaciones monótonas no necesariamente lineales.

Primero se define `ghiqc` como variable objetivo y se establece una lista de variables candidatas. Luego, se seleccionan del DataFrame únicamente las columnas numéricas y, entre ellas, solo las incluidas en posibles_vars.

A continuación, se excluye el propio target de las variables explicativas y se calculan, para cada predictor, los coeficientes de correlación de Pearson y de Spearman con `ghiqc`.

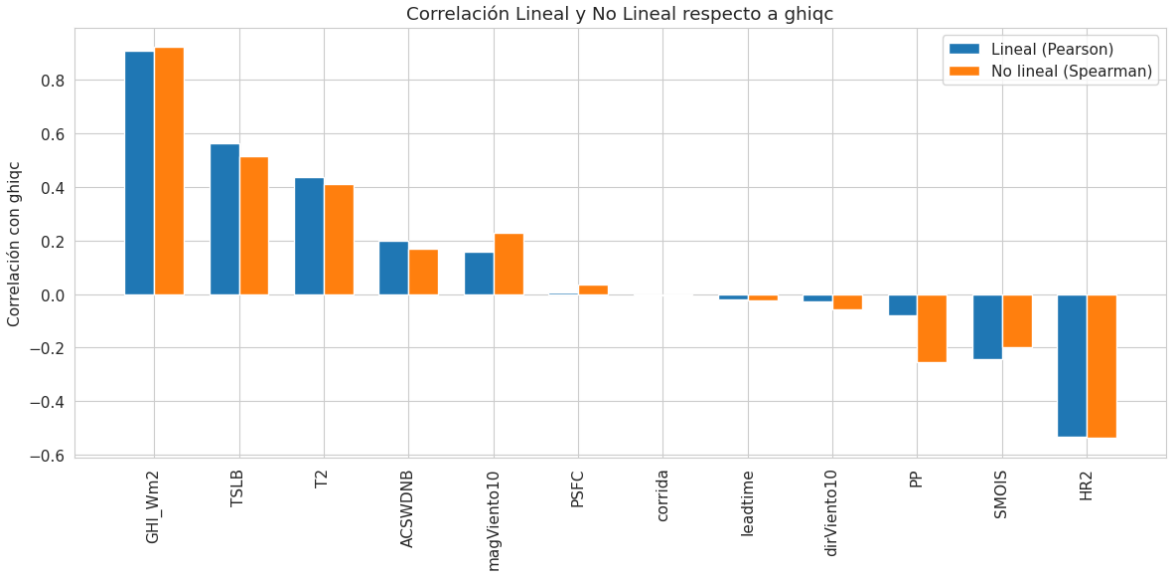
Los resultados se almacenan en un DataFrame de resumen (`corr_df`), que contiene ambas métricas y se ordena en orden descendente según la correlación de Pearson.

Finalmente, se genera un gráfico de barras comparativo en el que, para cada variable, se muestran, lado a lado, los coeficientes de Pearson y Spearman. Esta representación permite identificar qué variables presentan la mayor asociación con `ghiqc`, así como distinguir si dicha relación es predominantemente lineal o si mantiene una estructura monótona más general.

```
target = "ghiqc"

posibles_vars = [
    'corrida', 'leadtime', 'valid_time', 'PP', 'HR2', 'T2',
    'dirViento10', 'magViento10', 'PSFC', 'ACSWDNB',
    'TSLB', 'SMOIS', 'GHI_Wm2', 'ghiqc',
]

# Seleccionar solo columnas numéricas
df_num = df.select_dtypes(include=[np.number])

# Quedarse únicamente con las variables definidas en posibles_vars
df_num = df_num[[col for col in posibles_vars if col in
df_num.columns]]

# Quitar el propio target de las variables explicativas
variables = [col for col in df_num.columns if col != target]

# Calcular correlaciones lineales (Pearson)
pearson_corr = df_num[variables].corrwith(df_num[target],
method="pearson")

# Calcular correlaciones no lineales (Spearman)
spearman_corr = df_num[variables].corrwith(df_num[target],
method="spearman")

# Crear un dataframe resumen
corr_df = pd.DataFrame({
    "Pearson": pearson_corr,
    "Spearman": spearman_corr
}).sort_values(by="Pearson", ascending=False)

print(corr_df)

# ---- Gráfica ----
x = np.arange(len(corr_df))
width = 0.35

plt.figure(figsize=(12, 6))

plt.bar(x - width/2, corr_df["Pearson"], width, label="Lineal
(Pearson)")
plt.bar(x + width/2, corr_df["Spearman"], width, label="No lineal
(Spearman)")

plt.xticks(x, corr_df.index, rotation=90)
plt.ylabel("Correlación con ghiqc")
plt.title("Correlación Lineal y No Lineal respecto a ghiqc")
plt.legend()
plt.tight_layout()

plt.show()
```

SALIDA

	Pearson	Spearman
GHI_Wm2	0.907686	0.920818
TSLB	0.562967	0.514843
T2	0.437254	0.409556
ACSWDNB	0.200097	0.170037
magViento10	0.156864	0.228589
PSFC	0.005703	0.037053
corrida	-0.006506	-0.006048
leadtime	-0.018649	-0.023689
dirViento10	-0.027657	-0.056218
PP	-0.080280	-0.253180
SMOIS	-0.241338	-0.199789
HR2	-0.531965	-0.536913



P4.3) Determinación del target y de las variables regresoras

Este bloque prepara los datos de entrada (matriz de características) y de salida (vector objetivo) para el entrenamiento del modelo predictivo. La selección de las variables independientes se basó en los resultados del análisis de correlación detallado en la sección P4.1, en el que se identificaron las variables con una influencia significativa sobre la irradiancia medida.

Específicamente, se seleccionaron las variables que presentaron una correlación con `ghiqc` superior a 0.4 (40%): la irradiancia modelada por el WRF (`GHI_Wm2`), la temperatura del suelo (`TSLB`) y la temperatura a 2 metros (`T2`). Esta decisión se debe a que estas variables capturan tanto la forzante radiativa principal como el estado térmico de la atmósfera y del suelo, factores que condicionan la variabilidad local de la radiación solar.

```
features = ['T2','TSLB','GHI_Wm2']
target = 'ghiqc'
X = df[features].copy()
y = df[target].copy().values.reshape(-1,1)
```

Nota: si se necesita utilizar otras variables, solo deben incluirse en las `features`.

P4.4) División en TRAIN, VALIDATION y TEST

Este bloque divide el conjunto de datos en tres subconjuntos: entrenamiento (`train`), validación (`val`) y prueba (`test`), con una distribución de 40%, 10% y 50%, respectivamente.

```
features = ['T2','TSLB','GHI_Wm2']
target = 'ghiqc'
X = df[features].copy()
y = df[target].copy().values.reshape(-1,1)
```

```
#divide entre entrenamiento y prueba 50% a cada uno
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.5, random_state=42, shuffle=True
)
```

```
#divide entre entrenamiento y validacion 80% y 20%,
# esto es sobre el conjunto de entrenamiento definido en la celda
anterior
X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42, shuffle=True
)
```

```
print(f"Train: {len(X_train)/len(df):.1%}  ({len(X_train)} muestras)")
print(f"Validation: {len(X_val)/len(df):.1%}  ({len(X_val)} muestras)")
print(f"Test: {len(X_test)/len(df):.1%}  ({len(X_test)} muestras)")
```

SALIDA

```
Train: 40.0%  (39820 muestras)
Validation: 10.0%  (9956 muestras)
Test: 50.0%  (49776 muestras)
```

P4.5) Escalado

Este bloque realiza la estandarización de las variables de entrada (`X`) y de la variable objetivo (`y`) mediante `StandardScaler`. Este proceso transforma los datos para que tengan una media de 0 y una desviación estándar de 1, lo cual es crítico porque las variables, como la temperatura (`T2`) y la radiación (`GHI_Wm2`), tienen rangos numéricos muy diferentes, y la estandarización asegura que todas las variables contribuyan por igual

al ajuste de los pesos de la red, evitando que aquellas con valores absolutos más grandes dominen el proceso de aprendizaje. Además, los algoritmos de optimización convergen mucho más rápido cuando los datos están en una escala comparable, lo que facilita el descenso por el gradiente.

En la implementación, se utiliza el método `.fit_transform()` únicamente con los datos de entrenamiento para calcular los parámetros estadísticos (media y desviación). Luego, esos mismos parámetros se aplican mediante `.transform()` a los conjuntos de validación (`val`) y de prueba (`test`). Este procedimiento es vital para evitar la fuga de datos (data leakage), asegurando que el modelo se evalúe con datos completamente desconocidos para el escalador. Finalmente, se aplica `.ravel()` a la variable objetivo para convertirla en un vector unidimensional, cumpliendo así con los requisitos de formato de la librería scikit-learn.

```
scaler_X = StandardScaler()
scaler_y = StandardScaler()

X_train_s = scaler_X.fit_transform(X_train)
X_val_s = scaler_X.transform(X_val)
X_test_s = scaler_X.transform(X_test)

y_train_s = scaler_y.fit_transform(y_train).ravel()   # a 1D para
sklearn
y_val_s = scaler_y.transform(y_val).ravel()
y_test_s = scaler_y.transform(y_test).ravel()
```

P4.6) Configuración de MLP

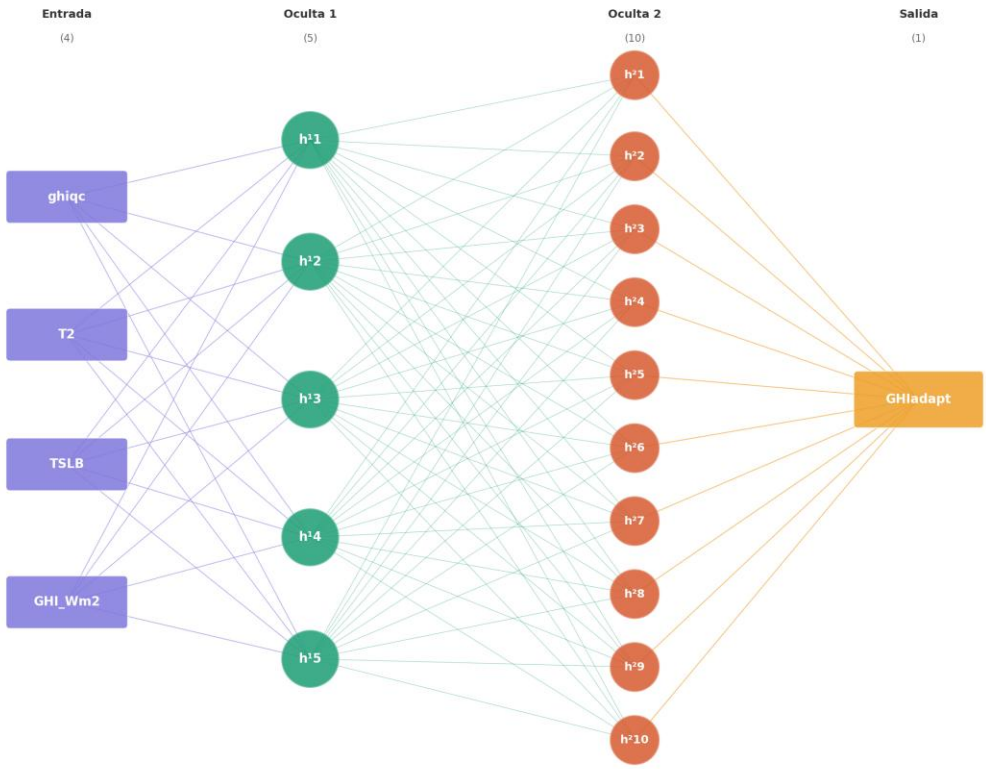
Este bloque define la configuración del MLP

```
param_grid = {
    'hidden_layer_sizes': [(5,10), (10,20),(20,40)],
    'activation': ['relu'],
    'learning_rate_init': [ 0.001, 0.01],
    'solver': ['adam'],  # adam funciona bien; podrías probar 'sgd' si
haces mini-batches
    'alpha': [1e-4, 1e-3]  # regularización L2
}
```

La elección de estas opciones (como las arquitecturas de capas ocultas (5,10), (10,20) y (20,40)) busca equilibrar la capacidad de aprendizaje con el riesgo de sobreajuste. Sin embargo, es fundamental destacar que **la configuración "óptima" no es universal,** ya que el desempeño de una MLP es altamente sensible a las condiciones locales. Por ejemplo, en regiones con alta frecuencia de cielos despejados, arquitecturas más simples suelen ser suficientes y robustas, mientras que en zonas con mayor variabilidad nubosa, las MLP pueden requerir una mayor profundidad de neuronas para capturar las relaciones no lineales entre la temperatura y la radiación. Por lo tanto, la "mejor" configuración obtenida en este

proceso es específica de los datos locales de entrenamiento y no necesariamente se replicará en otros sitios del país.

A continuación, a modo de ejemplo, se grafica una red MLP cuya primera capa oculta tiene 5 neuronas y la segunda, 10. En el código esta se denomina (5,10). También se ven las cuatro entradas y la única salida.



P4.7) Entrenamiento, métricas y elección del mejor MLP

Este bloque inicializa la estructura de control del proceso de optimización y define las variables que almacenarán el modelo de mejor desempeño y sus métricas asociadas. Un parámetro clave aquí es `MAX_EPOCHS = 200`, que establece el límite máximo de iteraciones que el optimizador podrá realizar durante el entrenamiento de cada configuración.

Se seleccionó este valor de 200 épocas con base en las curvas de aprendizaje típicas de este tipo de datos de irradiancia en Argentina. Este límite es suficiente para permitir que el optimizador (`solver`) alcance la convergencia (es decir, que el error deje de disminuir de manera significativa) sin incurrir en un gasto computacional excesivo.

```
resultados = []
MAX_EPOCHS = 200
mejor_modelo = None
mejor_params = None
mejor_rrmsd = np.inf
mejor_rmbe = np.inf
```

El siguiente bloque realiza un barrido de hiperparámetros para entrenar varias configuraciones de un `MLPRegressor` y seleccionar la de mejor desempeño en el conjunto de validación.

Para cada combinación de parámetros generada por `ParameterGrid(param_grid)`, se crea una red neuronal MLP con esos hiperparámetros y se entrena de forma incremental usando `partial_fit`, una época a la vez. En cada época, el modelo ajusta sus pesos con `X_train_s` e `y_train_s`, luego predice sobre los conjuntos de entrenamiento y validación escalados, y calcula el error cuadrático medio en ambos casos. Esos errores se almacenan en las listas `train_losses` y `val_losses`, lo que permite seguir la evolución del aprendizaje a lo largo de las épocas.

Además, se implementa una forma de early stopping: a partir de la época 100, el entrenamiento puede detenerse anticipadamente si el mejor valor de la pérdida de validación queda lo suficientemente atrás, lo que sugiere que el modelo ya no está mejorando. Una vez finalizado el entrenamiento de esa configuración, las predicciones del conjunto de validación se llevan nuevamente a la escala original mediante `scaler_y.inverse_transform`, y se calculan las métricas rRMSE y rMBE sobre los valores reales de `ghiqc`. Luego, se guarda en resultados toda la información relevante de esa corrida: parámetros, métricas, curvas de pérdida, número de épocas y el propio modelo entrenado.

Finalmente, el bloque compara el desempeño de la configuración actual con el mejor modelo encontrado hasta el momento. La selección se hace dando prioridad al menor `rrmsd` y, en caso de empate, al menor valor absoluto de `rmbe`. Si la configuración actual resulta mejor, se actualizan las variables `mejor_modelo`, `mejor_params`, `mejor_rrmsd` y mejor_rmbe, así como las curvas de entrenamiento asociadas. En síntesis, este bloque sirve para entrenar múltiples MLP, monitorear su aprendizaje y conservar la mejor red según su desempeño en validación.

```
# === Entrenamiento por grid: usamos partial_fit para tener control por
época ===
for params in ParameterGrid(param_grid):
    print(f"\nEntrenando con: {params}")

    # Configuro el MLP para entrenamiento incremental con partial_fit
    mlp = MLPRegressor(random_state=42,
                       warm_start=True,  # partial_fit no necesita
warm_start pero lo dejamos por si acaso
                       max_iter=1,        # no se usa directamente:
usaremos partial_fit
```

```
                        **params)


    train_losses = []
    val_losses = []

    for epoch in range(MAX_EPOCHS):
        mlp.partial_fit(X_train_s, y_train_s)

        y_tr_pred_s = mlp.predict(X_train_s)
        y_val_pred_s = mlp.predict(X_val_s)

        mse_tr = np.mean((y_train_s - y_tr_pred_s)**2)
        mse_val = np.mean((y_val_s - y_val_pred_s)**2)

        train_losses.append(mse_tr)
        val_losses.append(mse_val)

        if epoch > 100:
            if np.argmin(val_losses[:-100]) == len(val_losses) - 31:
                break

    # Una vez entrenado en escala, predecimos en escala original
desescalando
    y_val_pred_original =
scaler_y.inverse_transform(y_val_pred_s.reshape(-1,1)).ravel()
    y_val_original = y_val.ravel()

    rrmsd_val = rrmse(y_val_original, y_val_pred_original)
    rmbe_val = rmbe(y_val_original, y_val_pred_original)

    resultados.append({
        'params': params,
        'rrmsd': rrmsd_val,
        'train_losses': train_losses,
        'val_losses': val_losses,
        'epochs': len(train_losses),
        'modelo': mlp
    })

    print(f" --> rrmsd(val) = {rrmsd_val:.4f}, rmbe(val) =
{rmbe_val:.4f}, epochs={len(train_losses)}")

    # Selección del mejor: prioridad rrmsd y abs rmbe
    if rrmsd_val < mejor_rrmsd or (np.isclose(rrmsd_val, mejor_rrmsd)
and abs(rmbe_val) < abs(mejor_rmbe)):
        mejor_rrmsd = rrmsd_val
        mejor_modelo = mlp
        mejor_params = params
        mejor_train_losses = train_losses
        mejor_val_losses = val_losses
```

SALIDA

```
Entrenando con: {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_s
izes': (5, 10), 'learning_rate_init': 0.001, 'solver': 'adam'}
 --> rrmsd(val) = 35.4965, rmbe(val) = -0.4636, epochs=200

Entrenando con: {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_s
izes': (5, 10), 'learning_rate_init': 0.01, 'solver': 'adam'}
 --> rrmsd(val) = 35.4852, rmbe(val) = -0.1379, epochs=200

Entrenando con: {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_s
izes': (10, 20), 'learning_rate_init': 0.001, 'solver': 'adam'}
 --> rrmsd(val) = 35.3559, rmbe(val) = -0.5018, epochs=200

Entrenando con: {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_s
izes': (10, 20), 'learning_rate_init': 0.01, 'solver': 'adam'}
 --> rrmsd(val) = 35.6264, rmbe(val) = 0.4093, epochs=200

Entrenando con: {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_s
izes': (20, 40), 'learning_rate_init': 0.001, 'solver': 'adam'}
 --> rrmsd(val) = 35.3391, rmbe(val) = -1.8831, epochs=200

Entrenando con: {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_s
izes': (20, 40), 'learning_rate_init': 0.01, 'solver': 'adam'}
 --> rrmsd(val) = 35.4848, rmbe(val) = -0.6843, epochs=200

Entrenando con: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_si
zes': (5, 10), 'learning_rate_init': 0.001, 'solver': 'adam'}
 --> rrmsd(val) = 35.4914, rmbe(val) = -0.4746, epochs=200

Entrenando con: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_si
zes': (5, 10), 'learning_rate_init': 0.01, 'solver': 'adam'}
 --> rrmsd(val) = 35.4393, rmbe(val) = -1.1734, epochs=200

Entrenando con: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_si
zes': (10, 20), 'learning_rate_init': 0.001, 'solver': 'adam'}
 --> rrmsd(val) = 35.3438, rmbe(val) = -0.8513, epochs=200

Entrenando con: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_si
zes': (10, 20), 'learning_rate_init': 0.01, 'solver': 'adam'}
 --> rrmsd(val) = 35.4384, rmbe(val) = 0.1933, epochs=200

Entrenando con: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_si
zes': (20, 40), 'learning_rate_init': 0.001, 'solver': 'adam'}
 --> rrmsd(val) = 35.3389, rmbe(val) = -1.7798, epochs=200

Entrenando con: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_si
zes': (20, 40), 'learning_rate_init': 0.01, 'solver': 'adam'}
 --> rrmsd(val) = 35.5604, rmbe(val) = 0.1142, epochs=200
```

```
print(len(resultados)) ## Como se ha indicado en las instrucciones, 12
modelos entrenados.
```

SALIDA

El siguiente script define los nombres de archivos para guardar los resultados del entrenamiento: los scalers de entrada y salida (`scaler_X` y `scaler_y`) y el mejor modelo MLP encontrado, todos identificados por el código de sitio (SITE_COD). Luego, imprime un resumen con los mejores hiperparámetros hallados y sus métricas de validación (`rrmse` y `rmbe`), y finalmente serializa los tres objetos en disco usando joblib, dejando listos los archivos .pkl para reutilizar el modelo entrenado y los scalers en inferencias futuras, sin necesidad de reentrenarlos.

```
MODEL_PREFIX = f"MLP_{SITE_COD}_best"
SCALER_X_NAME = f"scaler_X_{SITE_COD}.pkl"
SCALER_y_NAME = f"scaler_y_{SITE_COD}.pkl"

# === Resultados ===
print("\n=== Mejor modelo según rrmse en validación ===")
print(mejor_params)
print(f"rrmse={mejor_rrmsd:.4f}, rmbe={mejor_rmbe:.4f}")

# Guardar scalers y modelo
joblib.dump(scaler_X, SCALER_X_NAME)
joblib.dump(scaler_y, SCALER_y_NAME)
joblib.dump(mejor_modelo, MODEL_PREFIX + ".pkl")
print(f"Guardado: {SCALER_X_NAME}, {SCALER_y_NAME},
{MODEL_PREFIX}.pkl")
```

SALIDA

```
=== Mejor modelo según rrmse en validación ===
{'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (20, 40),
 'learning_rate_init': 0.01, 'solver': 'adam'}
rrmse=37.2291, rmbe=0.5636
Guardado: scaler_X_BA.pkl, scaler_y_BA.pkl, MLP_BA_best.pkl
```

El siguiente script evalúa el modelo MLP entrenado sobre el conjunto de test: genera predicciones con los datos escalados, las devuelve a la escala original con el `scaler_y`, y calcula las métricas de error (`rrmse` y `rmbe`). Luego hace lo mismo para el modelo base WRF (el pronóstico numérico sin corrección, columna `GHI_Wm2`) sobre los mismos instantes de test, y compara ambos resultados impresos en pantalla, permitiendo ver cuánto mejora el MLP respecto al pronóstico original de WRF en términos de error relativo y sesgo.

```
# === Evaluación final en Test ===
# recuperar muestras originales (ya teníamos X_test, y_test)
y_test_pred_s = mejor_modelo.predict(X_test_s)
y_test_pred = scaler_y.inverse_transform(y_test_pred_s.reshape(-
1,1)).ravel()

y_test_orig = y_test.ravel()
rmse_test = rrmse(y_test_orig, y_test_pred)
rmbe_test = rmbe(y_test_orig, y_test_pred)

# comparar con modelo original (WRF)
```

```
test_df = df.loc[X_test.index].copy()
rmse_test_orig = rrmse(y_test_orig, test_df['GHI_Wm2'].values)
rmbe_test_orig = rmbe(y_test_orig, test_df['GHI_Wm2'].values)
print(f"WRF original Test - rrmsd={rmse_test_orig:.4f},
rmbe={rmbe_test_orig:.4f}")
print(f"WRF adaptado Test - rrmsd={rmse_test:.4f},
rmbe={rmbe_test:.4f}")
```

SALIDA

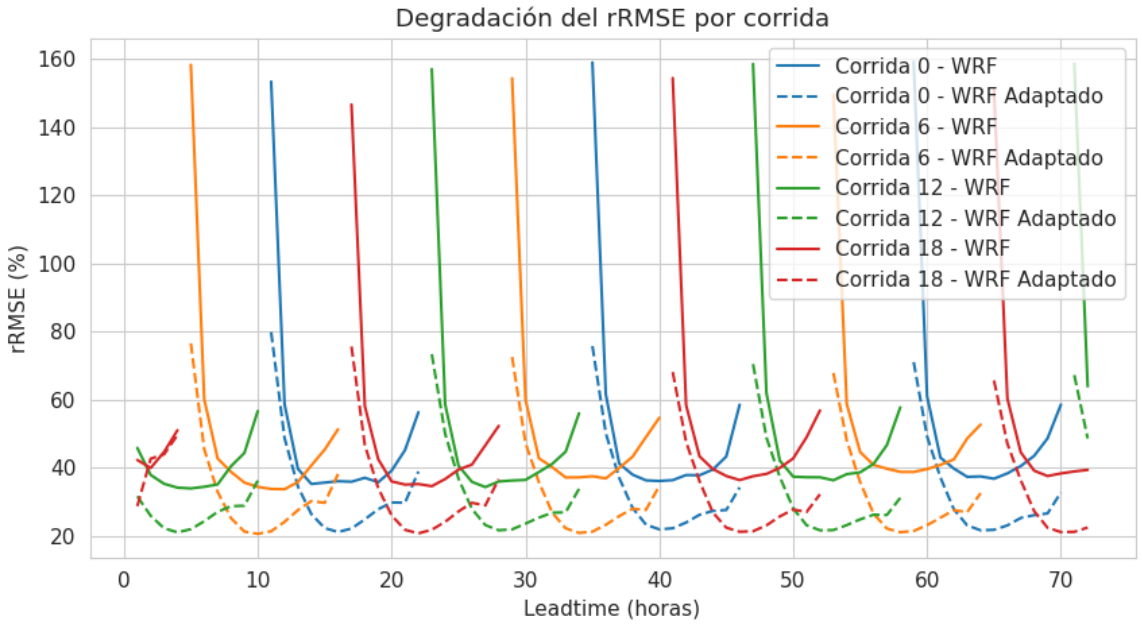
```
WRF original Test - rrmsd=46.4507, rmbe=24.3041
WRF adaptado Test - rrmsd=36.8807, rmbe=0.1086
```

El siguiente script grafica los valores del rRMSE por corrida por leadtime de los pronósticos originales de GHI del modelo WRF, vs los valores medidos para Buenos Aires, contra los mismos pronósticos pero ya adaptados usando la mejor MLP encontrada en el procedimiento.

```
df['GHI_Adap'] = scaler_y.inverse_transform(

mejor_modelo.predict(scaler_X.transform(df[features])).reshape(-1,1))
plt.figure(figsize=(10, 5))
for corrida in sorted(df['corrida'].unique()):
    sub = df[(df['corrida'] == corrida) & (df['SZA'] < 80)]
    valores_qc = []
    valores_adap = []
    for leadtime in sorted(sub['leadtime'].unique()):
        ss = sub[sub['leadtime'] == leadtime]

        # =========================
        # Curva original: ghiqc vs GHI_Wm2
        # =========================
        ss_qc = ss[['ghiqc', 'GHI_Wm2']].dropna()
        if ss_qc.shape[0] >= 10:
            valores_qc.append(
                (leadtime, rrmse(ss_qc['ghiqc'], ss_qc['GHI_Wm2']))
            )

        # =========================
        # Nueva curva: GHI_Adap vs GHI_Wm2
        # =========================
        ss_adap = ss[['GHI_Adap', 'GHI_Wm2']].dropna()

        if ss_adap.shape[0] >= 10:
            valores_adap.append(
                (leadtime, rrmse(ss_adap['GHI_Adap'],
ss_adap['GHI_Wm2']))
            )

    # ======================================================
    # Plot curva continua (ghiqc)
    # ======================================================
    if valores_qc:
        lt, err = zip(*valores_qc)
        lt = np.array(lt)
```

```
        err = np.array(err)
        x_plot = [lt[0]]
        y_plot = [err[0]]

        for i in range(1, len(lt)):
            if lt[i] - lt[i-1] > 1:
                x_plot.append(np.nan)
                y_plot.append(np.nan)
            x_plot.append(lt[i])
            y_plot.append(err[i])
        line = plt.plot(
            x_plot,
            y_plot,
            marker='o',
            ms=0.01,
            label=f'Corrida {corrida} - WRF'
        )
        color = line[0].get_color()

    # ======================================================
    # Plot curva punteada (Adap)
    # ======================================================
    if valores_adap:
        lt, err = zip(*valores_adap)
        lt = np.array(lt)
        err = np.array(err)
        x_plot = [lt[0]]
        y_plot = [err[0]]
        for i in range(1, len(lt)):
            if lt[i] - lt[i-1] > 1:
                x_plot.append(np.nan)
                y_plot.append(np.nan)
            x_plot.append(lt[i])
            y_plot.append(err[i])
        plt.plot(
            x_plot,
            y_plot,
            marker='s',
            ms=0.01,
            linestyle='--',
            color=color,
            label=f'Corrida {corrida} - WRF Adaptado'
        )

plt.title("Degradación del rRMSE por corrida")
plt.xlabel("Leadtime (horas)")
plt.ylabel("rRMSE (%)")
plt.legend()
plt.grid(True)
plt.show()
```

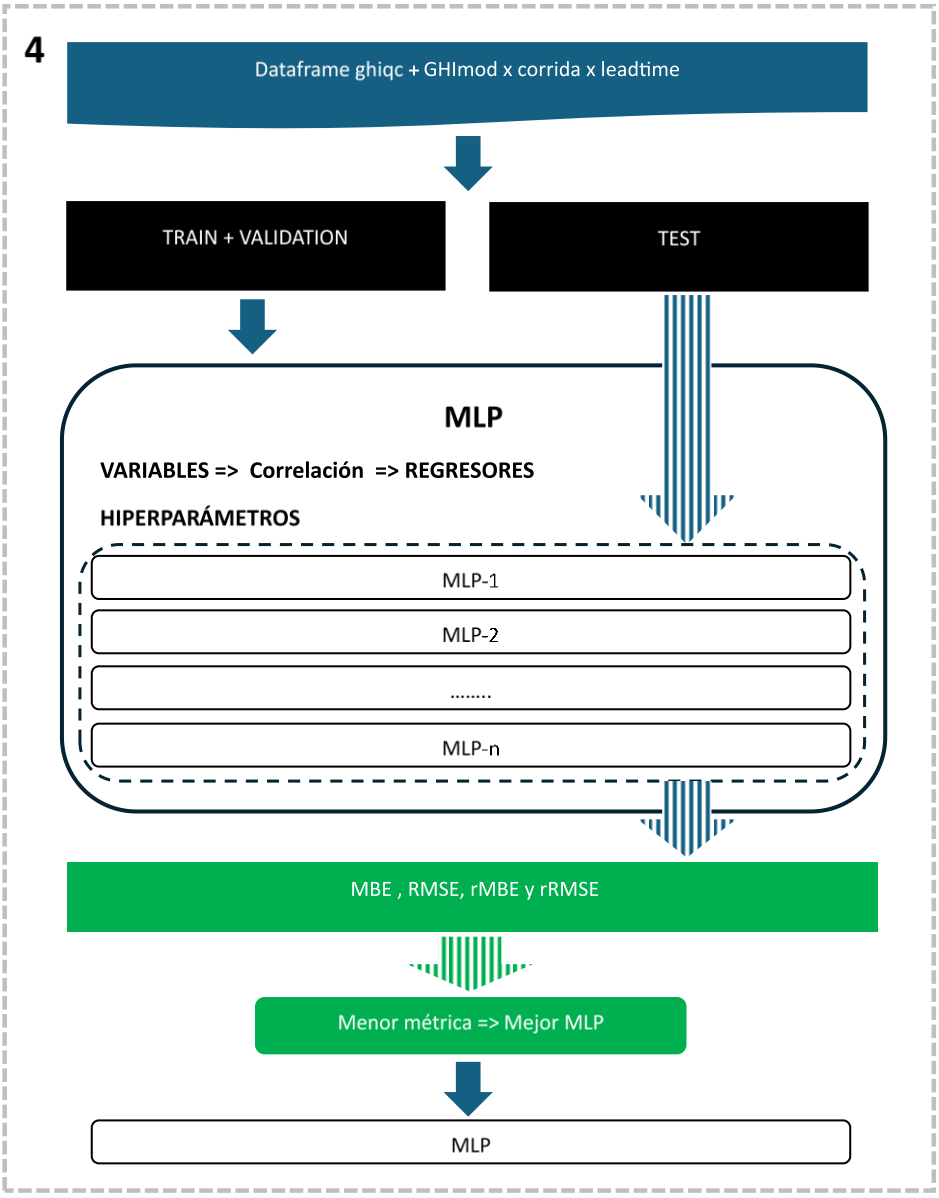
SALIDA



Esto concluye el proceso de corrección de los pronósticos del modelo WRF-SMN.

ESQUEMA CONCEPTUAL

PASO 4



3. Recomendaciones del consultor.

3.1. Limitaciones e incertidumbres del modelo.

3.1.1. Sesgo sistemático de sobreestimación.

El modelo numérico WRF-SMN exhibe un sesgo positivo sistemático en el pronóstico de GHI, con valores de rMBE entre 20% y 24% en los tres sitios analizados, lo que evidencia una sobreestimación estructural que no puede atribuirse a la variabilidad aleatoria. Si bien la adaptación al sitio mediante MLP logra reducir dicho sesgo a valores de rMBE inferiores al 1% en el conjunto de prueba, este resultado no implica la corrección de la causa física subyacente, sino una compensación estadística de sus efectos sobre la variable objetivo.

Desde el punto de vista del diagnóstico del modelo, la persistencia de un sesgo positivo de esta magnitud sugiere deficiencias en alguna o varias de las parametrizaciones físicas del WRF-SMN, en particular en aquellas que rigen el transporte radiativo de onda corta a través de la atmósfera y su interacción con los hidrometeoros y los aerosoles. La identificación precisa de los esquemas físicos responsables requeriría un análisis de sensibilidad específico de la configuración del modelo, lo que constituye una línea de investigación independiente y complementaria de los resultados aquí presentados.

3.1.2. Robustez del MLP con el tiempo.

La red neuronal fue entrenada y evaluada con datos del período 2023-2024, sin que se haya analizado la consistencia de la corrección aprendida frente a la variabilidad climática interanual. Este fenómeno se refiere a los cambios en el clima que ocurren de un año a otro; por ejemplo, la región pampeana presenta variaciones importantes en la nubosidad asociadas a forzantes como El Niño (ENSO) y el Modo Anular del Sur (SAM). Estas oscilaciones pueden alterar el comportamiento de las variables meteorológicas y, con ello, comprometer la validez de la función adaptativa obtenida.

En ausencia de una evaluación explícita de la variación temporal de dicha función, ya sea mediante la validación en años con climas contrastantes o mediante el análisis de la estacionalidad de los residuos, esta incertidumbre limita la extrapolación de los resultados. Esto condiciona la confiabilidad del sistema en un contexto operativo a largo plazo, al no saber con certeza si la precisión se mantendrá cuando las condiciones climáticas cambien en el futuro.

3.1.3. Desempeño crítico en condiciones de cielo cubierto.

El análisis revela que el modelo pronostica con alta eficacia el estado de cielo despejado (90% de acierto); sin embargo, presenta graves dificultades para predecir condiciones de

cielo cubierto (< 30%) o parcialmente cubierto (< 50%). Esta asimetría en el desempeño constituye la limitación operativa más crítica del sistema.

Si bien el MLP logra mitigar el error absoluto, no corrige la incapacidad estructural del WRF-SMN para representar adecuadamente situaciones de nubosidad intensa o de alta variabilidad espacial y temporal, precisamente las que presentan el peor desempeño del modelo físico. Cabe destacar que, para profundizar en la precisión de este diagnóstico, sería necesario contar con datos medidos de cobertura nubosa, cuya ausencia limita la validación de las transiciones nubosas..

3.2. Desarrollo futuro y líneas de investigación.

3.2.1. Incorporación de variables de nubosidad satelital.

La principal limitación del sistema de adaptación radica en que los regresores empleados por la MLP provienen exclusivamente de las salidas del modelo WRF. En consecuencia, cuando el modelo falla en representar adecuadamente el estado de la nubosidad, condición en la que presenta su peor desempeño, la red intenta corregir un desvío cuya causa física no está capturada por sus predictores de entrada. Esto genera una restricción estructural, ya que el esquema de corrección carece de información independiente sobre la variable que más condiciona el error en la estimación de la irradiancia.

Para superar esta deficiencia, se propone incorporar variables observacionales derivadas del satélite GOES-16, como la fracción de cobertura nubosa y la temperatura del tope de nube, que constituyen observaciones directas del estado atmosférico e independientes del modelo. En términos operativos, estos datos se integrarían durante la fase de preprocesamiento, sincronizando los píxeles satelitales con las salidas del modelo y con las mediciones de superficie antes de iniciar el entrenamiento. De este modo, la MLP dispondría de una caracterización más precisa del estado real de la atmósfera en el momento del pronóstico.

Esta integración permitiría que la MLP actuara como un integrador de fuentes, aprendiendo a ponderar la información del modelo frente a la evidencia satelital. Por ejemplo, si el satélite detecta nubosidad que la simulación física omite, la red contaría con el predictor necesario para ajustar dinámicamente la irradiancia a la baja. Este enfoque tiene el potencial de reducir significativamente el sesgo del sistema en condiciones de cielo cubierto o parcialmente nublado, resolviendo situaciones críticas en las que el modelo no capta la dinámica local de las nubes.

3.2.2. Evaluación ante la variabilidad climática interanual.

Incorporar el análisis de años con distintos forzantes climáticos (ENSO, SAM) para evaluar la robustez temporal del MLP y determinar si el modelo requiere reentrenamiento periódico o si la corrección aprendida es lo suficientemente estable. Para ello, también debería concretarse lo planteado en el punto 3, en cuanto a incrementar el número de estaciones de medición controladas.

3.2.3. Análisis de la importancia de las variables (feature importance)

En este informe no se aborda este tema en profundidad. Una línea de trabajo inmediata consistiría en aplicar técnicas de importancia por permutación (*permutation importance*) (Altmann et al., 2010[5]) o técnicas de análisis de sensibilidad para cuantificar qué variables del modelo aportan más información al proceso de corrección. Este procedimiento consiste en alterar aleatoriamente los valores de un predictor específico y observar cuánto aumenta el error de la MLP; un incremento significativo del error indica que la red depende fuertemente de esa variable para generar un pronóstico preciso.

La implementación de este análisis permitiría identificar la relevancia relativa de predictores como la temperatura, la humedad o la propia irradiancia del modelo. Al entender la jerarquía de estas variables, es posible detectar regresores redundantes o incluso perjudiciales para la capacidad de generalización de la red. Por ejemplo, si una variable del modelo presenta ruido excesivo que confunde a la MLP sin aportar valor predictivo, su eliminación podría simplificar la arquitectura de la red y mejorar su robustez al aplicarla en diferentes ubicaciones geográficas.

En términos prácticos, esta evaluación transformaría la MLP de un esquema opaco en un sistema más transparente y justificable desde el punto de vista físico. Identificar qué componentes del modelo son los principales motores de la mejora estadística facilitaría el diseño de configuraciones más eficientes, orientando la selección de predictores hacia aquellos que realmente capturan la dinámica de la radiación solar y descartando los que no contribuyen a la precisión final.

3.2.4. Arquitecturas alternativas de ML

El MLP es un punto de partida sólido, pero existen arquitecturas que podrían superar su desempeño en series temporales meteorológicas: redes LSTM o GRU (capturan dependencias temporales), modelos de gradient boosting (XGBoost, LightGBM, generalmente más robustos con datos tabulares pequeños) y modelos híbridos físico-ML que incorporen restricciones físicas (ej. $GHI \geq 0$, $GHI \leq$ GHI extraterrestre). Se recomienda un benchmarking sistemático.

---

[5] Altmann, A., Toloşi, L., Sander, O., & Lengauer, T. (2010). Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10), 1340-1347

3.2.5. Integración operativa con el SMN

El paso final de la cadena de valor es la implementación operativa: un pipeline automatizado que descargue las corridas del WRF-SMN, aplique el MLP entrenado y genere pronósticos corregidos de GHI en tiempo real para la red de sitios de interés. Esto requiere definir criterios de reentrenamiento, umbrales de alerta ante la degradación del modelo y una interfaz de visualización para los usuarios finales.

4. ANEXOS

- **Anexo 1. Script de los códigos Jupyter Notebook**

https://github.com/rdledesma/SMN-WRF-Site-Adaption/blob/main/Presentacion_WRF_SMN_MLP.ipynb

- **Anexo 2**

Cross-Validation. Medición de evolución de error rRMSE de los valores pronosticados adaptados.

```
# -*- coding: utf-8 -*-

import pandas as pd
import numpy as np
import joblib
import glob
from datetime import timedelta
import warnings

warnings.filterwarnings("ignore")

# ============================
SITES = ['AR', 'PI', 'BA']
FEATURES = ['T2', 'TSLB', 'GHI_Wm2']

MODEL_PREFIX = "MLP_{}_best"
SCALER_X = "scaler_X_{}.pkl"
SCALER_y = "scaler_y_{}.pkl"

LEAD_MAX = 73


def get_time_shift(site):
    return timedelta(minutes=60 if site == 'BA' else 240)


# ============================
# FUNCIÓN PRINCIPAL
# ============================
def generar_full_timeline(site):
    print(f"\nProcesando sitio {site}")

    TIME_SHIFT = get_time_shift(site)

    # ============================
    # DATOS MODELADOS (BASE ABSOLUTA)
    # ============================
    archivos = glob.glob(f"modelados/{site}*.csv")
    df_model = pd.concat(
        [pd.read_csv(a) for a in archivos],
```

```
        ignore_index=True
    )

    # Ajuste horario
    df_model['valid_time'] = (
        pd.to_datetime(df_model['valid_time']) - TIME_SHIFT
    )

    # Filtrado SOLO por leadtime (no se toca el orden)
    df_model = df_model[df_model.leadtime <
LEAD_MAX].reset_index(drop=True)

    # ============================
    # DATOS OBSERVADOS
    # ============================
    archivos_med = glob.glob(f"medidas/{site}*.xlsx")

    if archivos_med:
        df_obs = pd.concat(
            [pd.read_excel(a) for a in archivos_med],
            ignore_index=True
        )
        df_obs.columns = ['datetime', 'ghi', 'sza', 'GHIcc']
        df_obs['datetime'] = pd.to_datetime(df_obs['datetime'])
    else:
        df_obs = pd.DataFrame(columns=['datetime', 'ghi'])

    # Merge observaciones (NO altera estructura)
    df = df_model.merge(
        df_obs[['datetime', 'ghi']],
        left_on='valid_time',
        right_on='datetime',
        how='left'
    ).drop(columns=['datetime'])

    # ============================
    # FEATURES (MISMO ORDEN)
    # ============================
    X = df[FEATURES].interpolate(limit_direction='both')

    # ============================
    # PREDICCIONES (MISMA FILA = MISMO LEAD)
    # ============================
    for model_site in SITES:
        print(f"  → aplicando modelo {model_site}")

        model = joblib.load(MODEL_PREFIX.format(model_site) + ".pkl")
        scaler_X = joblib.load(SCALER_X.format(model_site))
        scaler_y = joblib.load(SCALER_y.format(model_site))

        X_scaled = scaler_X.transform(X)
        y_scaled = model.predict(X_scaled).reshape(-1, 1)
        y_pred = scaler_y.inverse_transform(y_scaled).ravel()
```

```
        df[f'GHI_MLP_{model_site}'] = y_pred

    # ============================
    # GUARDAR (FORMATO ORIGINAL + NUEVAS COLUMNAS)
    # ============================
    output = f"full_timeline_{site}_multi_modelo.csv"

    df =
df[['date','corrida','leadtime','valid_time','GHI_Wm2','ghi','GHI_MLP_A
R','GHI_MLP_PI','GHI_MLP_BA']]
    df['valid_time'] = df.valid_time +  TIME_SHIFT

    df = (
    df.sort_values(['date', 'corrida', 'leadtime'])
      .reset_index(drop=True))


    df.to_csv(output, index=False)

    print(f"✔ Generado {output}")


# ============================
# EJECUCIÓN
# ============================
for site in SITES:
    generar_full_timeline(site)
```

SALIDA

```
Procesando sitio AR
  → aplicando modelo AR
  → aplicando modelo PI
  → aplicando modelo BA
✔ Generado full_timeline_AR_multi_modelo.csv

Procesando sitio PI
  → aplicando modelo AR
  → aplicando modelo PI
  → aplicando modelo BA
✔ Generado full_timeline_PI_multi_modelo.csv

Procesando sitio BA
  → aplicando modelo AR
  → aplicando modelo PI
  → aplicando modelo BA
✔ Generado full_timeline_BA_multi_modelo.csv
```