IECON 2025

The 51st Annual Conference of the IEEE Industrial Electronics Society

14–17 October 2025

Tutorial SESSION

DL-based forecasting of energy related time series: tuning, evaluation, and reproducibility

Author's Name: Giuseppe La Tona

Affiliation: Institute of Marine Engineering (INM), National Research Council (CNR), Italy

Author's Name: Christoph Bergmeir

Affiliation: Department of Computer Science and Artificial Intelligence, University of Granada, Spain

Outline of the tutorial

- Introduction to energy-related time series forecasting
- ML and DL architectures for forecasting
- Tuning of DL models for forecasting
- Evaluation of forecasting results
- Reproducibility of forecasting results
- Hands-on coding examples

About me – Giuseppe La Tona

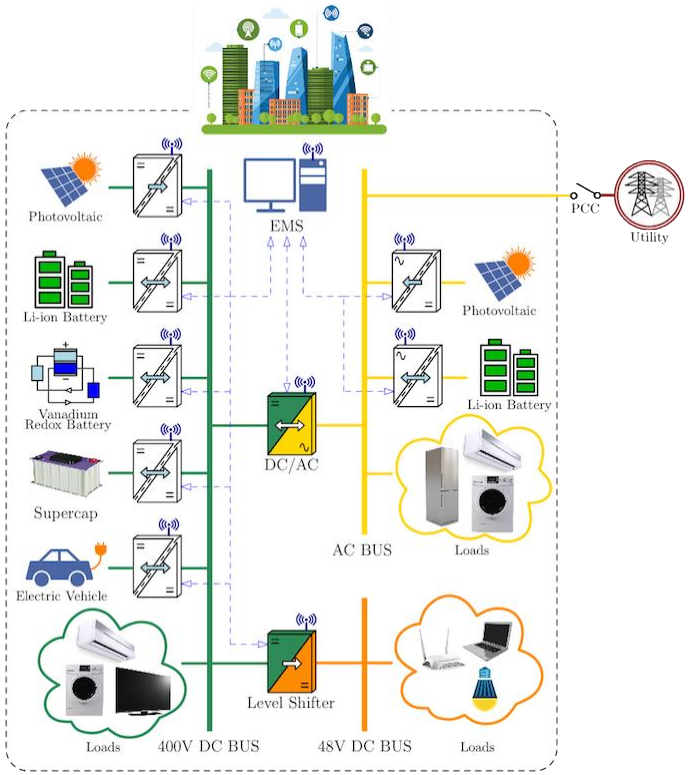
- Researcher with the **Institute for Marine Engineering (INM)** – Palermo branch, CNR
- **Ph.D. in Computer Engineering** from the *University of Palermo*
- **Senior Member, IEEE**
- Research focus: **machine learning** and **optimization** for **energy systems**
- Main topics:
  - Energy management systems (EMSs)
  - Forecasting of energy-related time series
  - Optimization-based control for microgrids and renewable integration

- Contact info:
- Email: giuseppe.latona@cnr.it
- Website: www.giuseppelatona.com

Introduction and Definitions

Forecasting: a Key Enabler of the Energy Transition

- The integration of **renewable energy sources** is accelerating the transition toward **low-carbon energy systems**.
- This shift increases **variability**, **uncertainty**, and **operational complexity**.
- **Accurate forecasting** of renewable generation and electrical load is essential to:
  - Maintain **grid stability** and **reliability**
  - Enable **smart operation** of energy communities and microgrids
  - Support **planning** and **energy trading**
- Forecasts feed **Energy Management Systems (EMSs)** that coordinate renewable sources, storage, and loads to improve efficiency and reduce emissions.

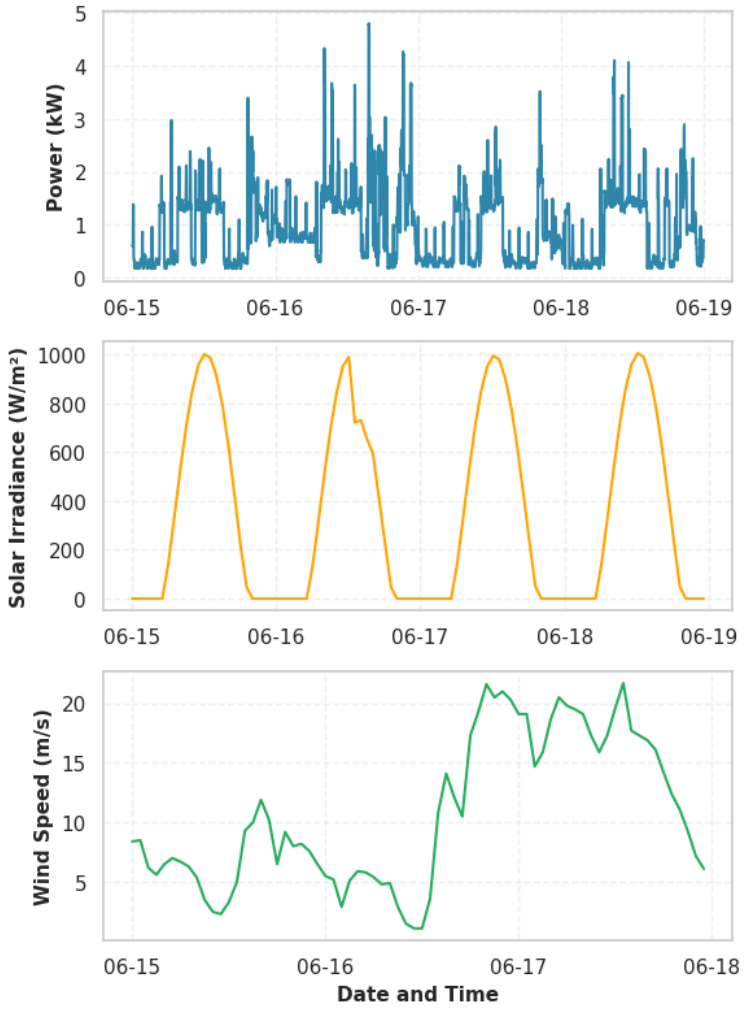


Forecasting for Different Scales and Objectives

- **Transmission and Distribution Operators:**
  - Aggregate short-/medium-term forecasts for scheduling, unit commitment, and trading.
- **Microgrids and Buildings:**
  - Short-term load and PV forecasting for day-ahead scheduling and real-time control.
- **Emerging paradigms:**
  - Energy Communities and Smart Buildings rely on EMSs integrating local generation, storage, and flexible demand.

Examples of Energy-Related Time Series

- **Electrical load demand** – residential, commercial, or aggregated
- **PV and wind generation** – renewable output affected by weather
- **Market prices** – reflecting supply–demand balance
- **Environmental variables** – temperature, irradiance, wind speed



From forecasting needs to forecasting practice

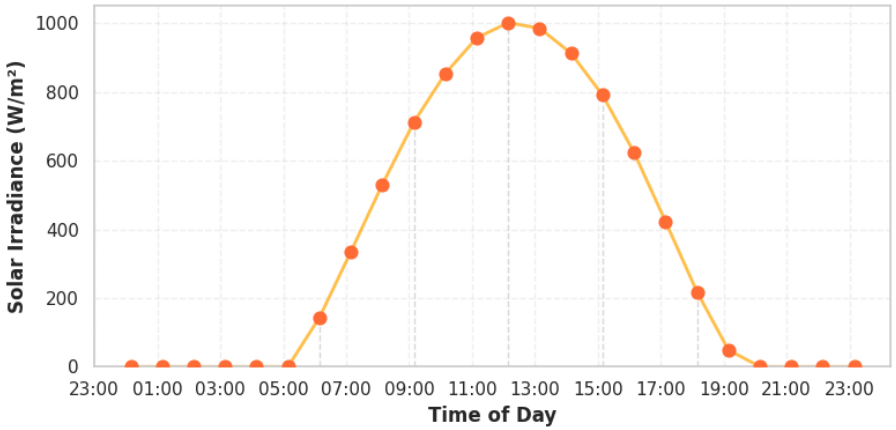
- The growing interest in machine learning and deep learning has brought remarkable advances in forecasting accuracy.
- However, challenges remain:
  - Inconsistent or inappropriate evaluation metrics
  - Data leakage and methodological errors in validation
  - Limited reproducibility of published results due to unavailable code or data
- These issues hinder both scientific progress and industrial adoption.
- This tutorial aims to share good practices, evaluation frameworks, and reproducible workflows for energy-related forecasting.

What is a time series?

- A **time series** is a set of data points indexed by time, typically sampled at **regular intervals**.
- It can be modeled as a **stochastic process**, i.e., a sequence of random variables

$$y_1, y_2, \ldots, y_t$$

- Each observation depends on previous ones — capturing temporal dependence



What is forecasting?

- **Forecasting** aims to predict the value of the variable y_t given its **past observations**:

$$\hat{y}_{t+1} = f(y_t, \dots, y_1)$$

- Often, we want to predict several future steps — a **forecasting horizon** of length *h*

$$(\hat{y}_{t+1}, \dots, \hat{y}_{t+h}) = f(y_t, \dots, y_1)$$

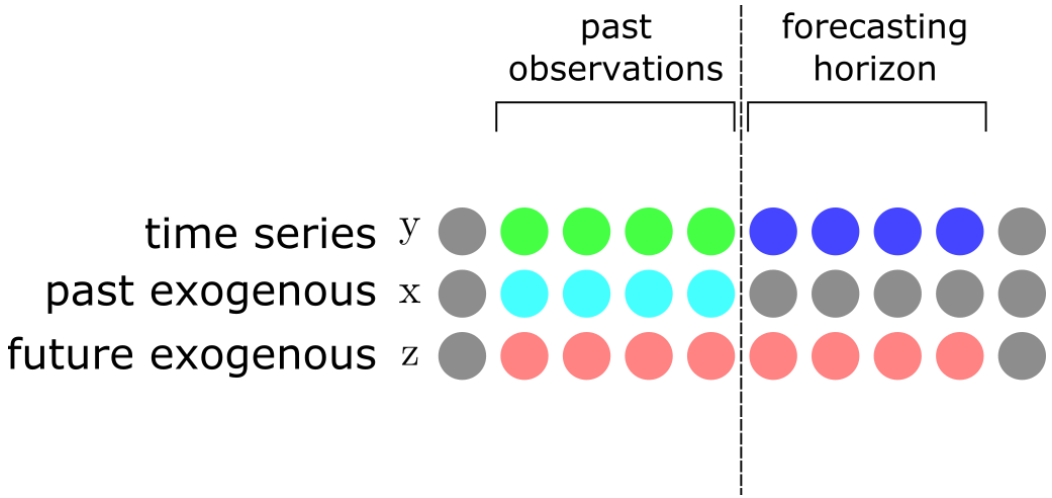
- The function *f* is unknown and it must be learned from data

Univariate and Multivariate Time Series

- **Univariate**: each time step is a scalar y_t
  - Example: total power consumption
- **Multivariate**: each time step is a vector $\boldsymbol{y_t}$
  - Example: consumption, PV generation, and temperature jointly
- Multivariate models can capture **inter-variable dependencies**.

Role of exogenous variables

- **Exogenous variables** (also called **explanatory variables**, or **covariates**) provide additional information influencing the target time series.
- Types of exogenous variables:
  - **Past exogenous variables**: known up to time t (e.g., past temperature, prices)
  - **Future exogenous variables**: known over the forecast horizon $t+1, \ldots, t+h$ (e.g., calendar features, scheduled tariffs, weather forecasts)
  - **Static exogenous variables:** variables that are static throughout the entire forecast horizon, e.g., building type, geographic location, or installed capacity
- Including exogenous inputs often improves model accuracy and robustness
- Modeling static covariates has become common with deep learning forecasting architectures



General forecasting model

In general, a forecasting model approximates the relationship between future values of the target series and all relevant inputs:

$$(\hat{y}_{t+1}, \dots, \hat{y}_{t+h}) = f(\boldsymbol{z_{t+h}}, \dots, \boldsymbol{z_{t+1}}, y_t, \dots, y_1, \boldsymbol{x_t}, \dots, \boldsymbol{x_1}, \mathbf{s})$$

- y target time series
- x past exogenous variables (time-varying)
- z future exogenous variables (time-varying, known over the forecast horizon)
- s static covariates (covariates that are static throughout the entire forecast horizon)
- f the unknown function learned from data

Modern **deep learning architectures** learn flexible approximations of f directly from raw or minimally processed data.

Single-step and multi-step forecasting

- **Single-step** forecasting:
  Predict only the next value of the time series
- **Multi-step** forecasting:
  Predict several future values
- Multi-step forecasts can be obtained by:
  - **Recursive** strategy: feed each prediction back as input
  - **Direct** strategy: train separate models for each horizon
  - **Multi-output** strategy: predict all future steps jointly

Forecasting horizons

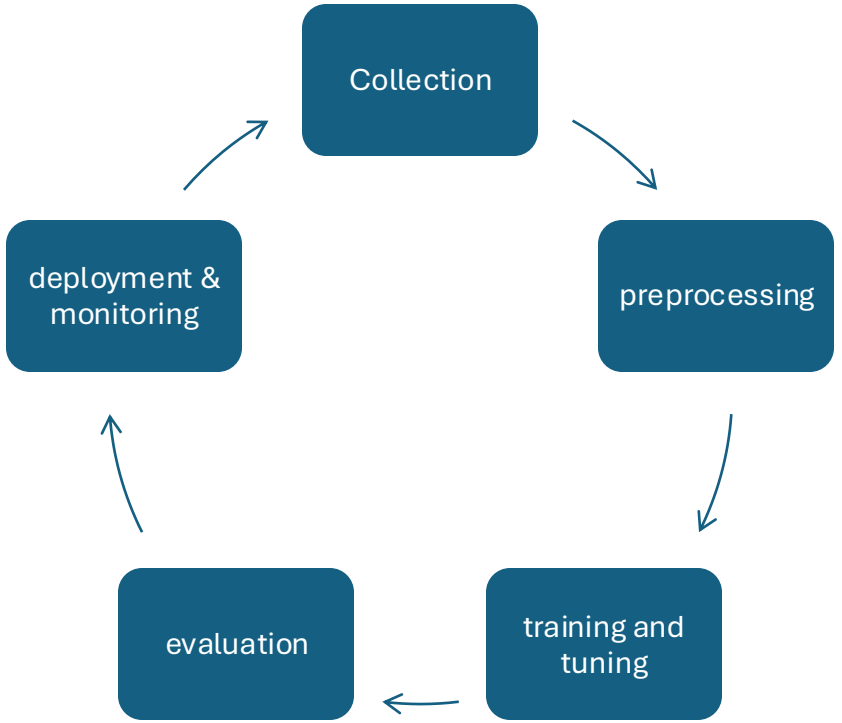
- Horizon selection depends on:
  - Temporal resolution (e.g., hourly, daily)
  - **System dynamics** and **decision timescale**

Horizon	Typical Range	Applications
Short-term	Minutes → days	Operation, dispatch, demand response, control
Medium-term	Weeks → months	Maintenance, resource planning, tariff design
Long-term	Months → years	Investment planning, capacity expansion, policy studies

Day-ahead forecasting

- A special case of short-term multi-step forecasting
- Forecasts the **next 24 hours**, typically with hourly or sub-hourly resolution
- Widely used in energy applications for:
  - Scheduling and dispatch of generation and storage
  - Energy trading in day-ahead markets
  - Demand response planning
- Often includes exogenous inputs such as weather forecasts and calendar effects

General forecasting pipeline



Data preprocessing

Typical preprocessing steps for energy-related time series:

- Data cleaning: handle missing values, outliers, and sensor errors
- Resampling: align data to a consistent temporal resolution
- Normalization / scaling: improve model convergence
- Detrending / deseasonalization: optional, depends on model type
- Feature engineering: create time-based features (hour, weekday), lags, rolling stats
- Documentation: record preprocessing decisions for reproducibility

⚙ Good preprocessing ensures data quality, model stability, and reproducible results.

Summary and outlook

- We discussed the **motivation**: the energy transition increases variability and uncertainty, making forecasting essential for stable and efficient operation
- Defined **time series**, **forecasting**, and **exogenous variables**
- Introduced forecasting **strategies** (single- vs. multi-step) and **horizons**
- Outlined the forecasting workflow and data preprocessing steps
- Next: *How to model forecasting functions f using ML and DL architectures*

ML and DL architectures for forecasting

Christoph Bergmeir

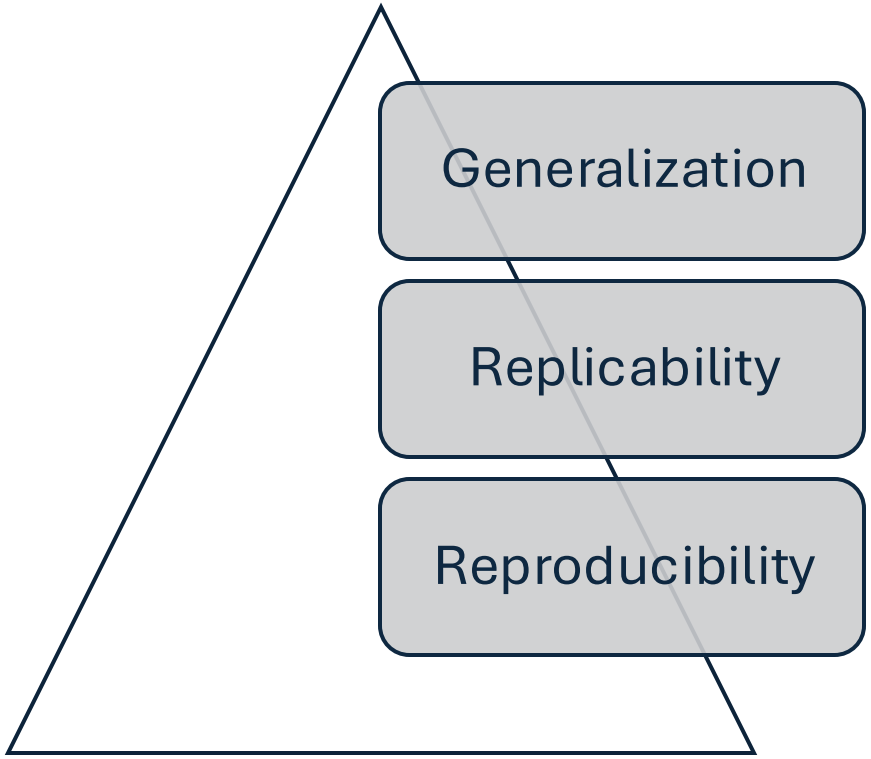
Reproducibility

Why talk about reproducibility?

- Reproducibility is the foundation of **scientific credibility**
- Increases **trust**, **transparency**, and **reliability** of research
- Enables **verification**, **comparison**, and **re-use** of methods
- Critical for bridging academic results and industrial adoption
- In ML and forecasting, reproducibility ensures models can be **replicated**, **validated**, and **improved**

Terminology

- Replicability:
  - Independent researchers obtain consistent results using new data and the same methods
- Reproducibility:
  - Independent researchers obtain the same results using the same data and methods
- Reproducibility is a necessary condition for replicability
- Without reproducibility, replication — and thus scientific progress — is impossible



The broader reproducibility crisis

- Many scientific disciplines report low replication success rates
- Common issues:
  - Incomplete or ambiguous method descriptions
  - Data and code not shared
  - Hidden assumptions or software bugs
  - Randomness and uncontrolled computation environments
- Consequences:
  - Erosion of trust in published results
  - Difficulty building on prior work

Reproducibility in forecasting research

- Forecasting studies often lack complete methodological transparency
  - Insufficient detail in data preprocessing and parameter settings
  - Proprietary software or undocumented algorithms
  - Results that depend on specific datasets or random initializations
- Even when data are public, methods are rarely reproducible without direct author communication
- Reproducibility is essential for:
  - Fair model comparison
  - Reliable performance benchmarking
  - Cumulative progress in forecasting research

Reproducibility in Machine Learning

- ML experiments are computational, yet reproducibility remains a major challenge
- Frequent causes:
  - Under-specified model and training details
  - Uncontrolled randomness (initialization, sampling, parallelism)
  - Inconsistent metric definitions
  - Selective reporting of best results
  - Software version differences
- Reproducibility is not only about sharing code, but also sharing context and documentation

Reproducibility in ML-based forecasting

- Forecasting introduces additional temporal and methodological complexities:
  - Risk of data leakage (future data influencing training)
  - Dependence on preprocessing pipelines
  - Multiple forecast horizons and error metrics
  - Use of stochastic deep learning models
- Reliable forecasting research requires:
  - Transparent workflows
  - Controlled randomness
  - Reproducible data splits and evaluation procedures

Sources of reproducibility failures

Reproducibility failures can occur at every stage of the forecasting workflow:

1. Data
2. Preprocessing
3. Model design & training
4. Evaluation
5. Reporting & dissemination

Data-related failures

Common problems:

- Unavailable or restricted data (privacy, licensing, proprietary datasets)
- Ambiguous data descriptions: unclear time ranges, units, or missing variables
- Data leakage: future information inadvertently used during training
- Uncontrolled data versioning: datasets updated without traceability
- Poor data quality: missing values, outliers, or inconsistent sampling

Preprocessing and feature engineering failures

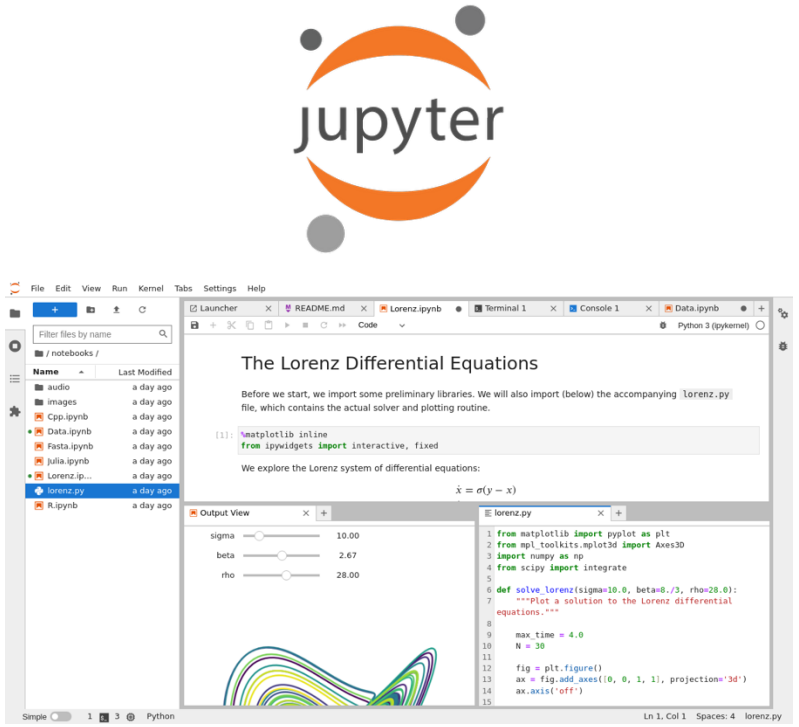
- Incomplete documentation of preprocessing pipelines
- Implicit assumptions: about scaling, normalization, or data alignment
- Non-deterministic feature generation (random splits, augmentations)
- Temporal leakage via lookahead features
- Environment-dependent scripts (different library defaults or order of operations)

Model design and training failures

- Under-specified architectures: missing layer sizes, activation functions, etc.
- Incomplete hyperparameter details (learning rate, optimizer, batch size, etc.)
- Random initialization and nondeterministic training not controlled by seeds
- Library and hardware differences (CPU vs GPU, parallelism)
- Selective reporting: only best runs reported, others omitted

Hidden Pitfalls: Notebooks and Interactive Workflows

- **Notebooks** (e.g., Jupyter, Colab) are powerful but risky for reproducibility:
  - **Out-of-order execution** can produce inconsistent results
  - **Hidden state** (variables from previous runs) affects outcomes
  - **Missing dependencies** or local paths break portability
  - **Manual intervention** prevents automation and version tracking
- Recommendations:
  - Use notebooks for exploration, not final experiments
  - Convert to scripts or pipeline stages for reproducible execution



Evaluation and validation failures

- Inconsistent data splits: unclear separation of training, validation, and test sets
- Temporal leakage in cross-validation
- Non-standardized error metrics or misreported formulas
- Selective metric reporting (best horizons, specific subsets)
- Different evaluation horizons or aggregation rules
- Overfitting to benchmark datasets

Reporting and dissemination failures

- Incomplete documentation: missing code, data, or environment details
- Unavailable trained models or scripts
- Unspecified software versions or dependencies
- Non-deterministic results reported without confidence intervals
- Insufficient methodological transparency in publications

Ensuring Reproducibility Across the Forecasting Workflow

Reproducibility requires attention at every stage:

1. Data management
2. Preprocessing and feature engineering
3. Model design and training
4. Evaluation and validation
5. Reporting and dissemination

Key principles:

- **Traceability**: record every step and change
- **Determinism**: control randomness and environments
- **Transparency**: share code, data, and decisions

Data management solutions

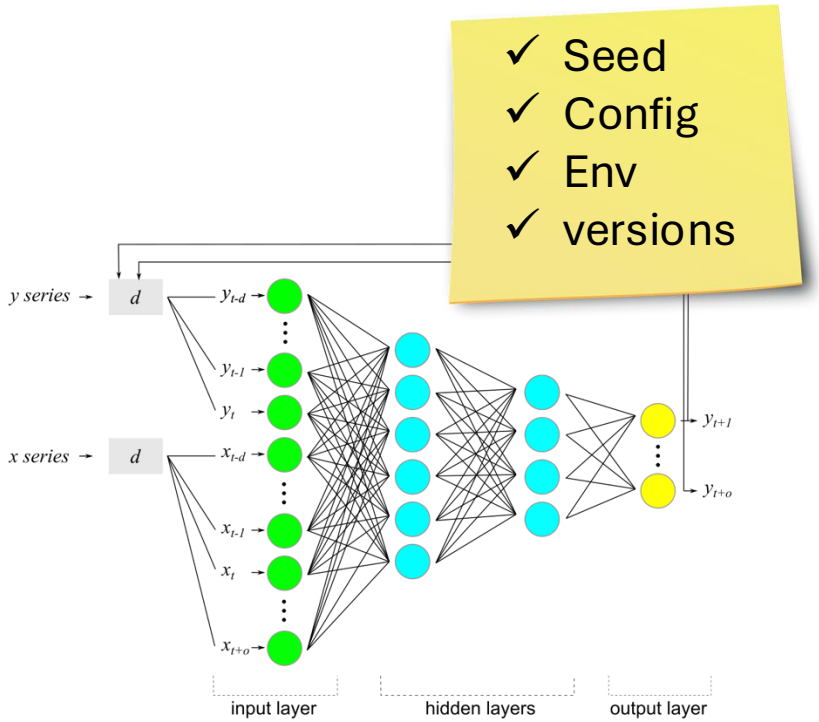
- Use **open**, **versioned datasets** whenever possible
- Track dataset versions and metadata (e.g., timestamps, features, sampling frequency)
- Apply **data version control** tools (e.g., DVC, Git-LFS)
- Store raw and processed data separately
- Document data sources, preprocessing rules, and exclusions
- Ensure **consistent random splits** across experiments

Preprocessing and Feature Engineering Solutions

- Build deterministic preprocessing pipelines
  - Fix random seeds for imputations or augmentations
  - Avoid time-dependent leakage (respect causality)
- Use pipeline orchestration frameworks (e.g., DVC stages, scikit-learn Pipeline)
- Record all preprocessing parameters in configuration files
- Keep feature engineering scripts versioned and modular
- Automate and log preprocessing steps for full reproducibility

Model Design and Training Solutions

- Fix random seeds for initialization and sampling
- Ensure **deterministic training** (when possible)
  - Control GPU operations and parallelism -> at the cost of slower training
- Record model configuration and hyperparameters (YAML/JSON files)
- Use experiment tracking tools (e.g., MLflow, Weights & Biases, DVC experiments)
- Version model code and weights
- Keep requirements.txt / environment.yml with library versions



Evaluation and Validation Solutions

- Define clear and reproducible data splits (train, validation, test)
- Use time-aware backtesting or rolling windows for temporal data
- Document evaluation horizons and metric formulas
- Report all relevant metrics — not only the best-performing ones
- Use consistent benchmarks and protocols for fair comparison
- Log evaluation results with associated configurations

Reporting and dissemination solutions

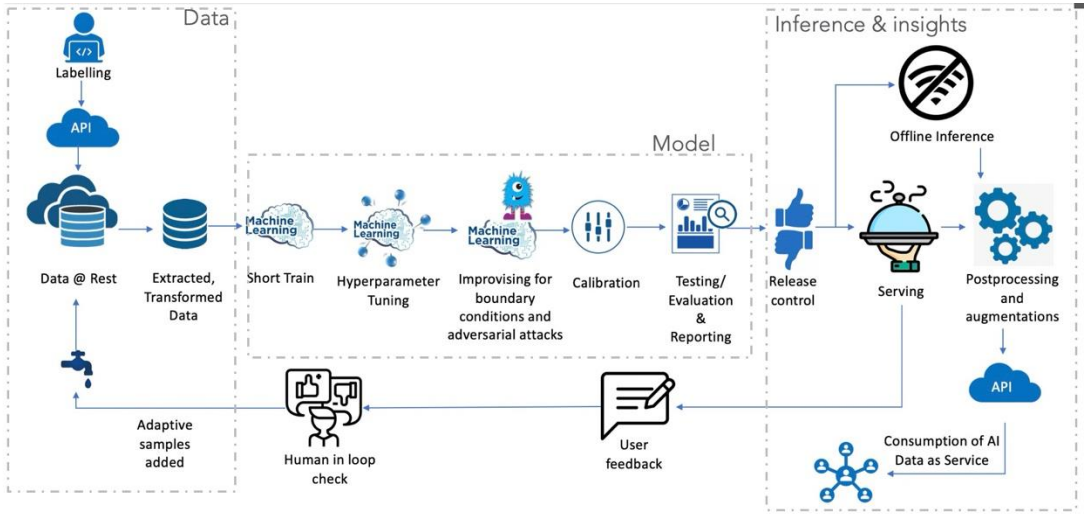
- Release code and data (or synthetic equivalents if sensitive)
- Use open repositories (GitHub, Zenodo, Hugging Face, OSF)
- Include:
  - Code, configuration, and data access instructions
  - Details of software versions and dependencies
  - Random seed and hardware info
- Consider **reproducibility checklists** and **badges**
- Write clear documentation for others to rerun experiments

MLOps and Reproducibility

- **MLOps** (Machine Learning Operations) applies DevOps principles to ML systems
- Goals: automation, traceability, reproducibility, and scalability
- Key MLOps components for reproducibility:
  - Version control of data, code, and models
  - Automated pipelines for training and deployment
  - Experiment tracking and artifact storage
  - Environment management (Docker, Conda, CI/CD)
- MLOps bridges research prototypes and production-grade forecasting systems

What is a pipeline?

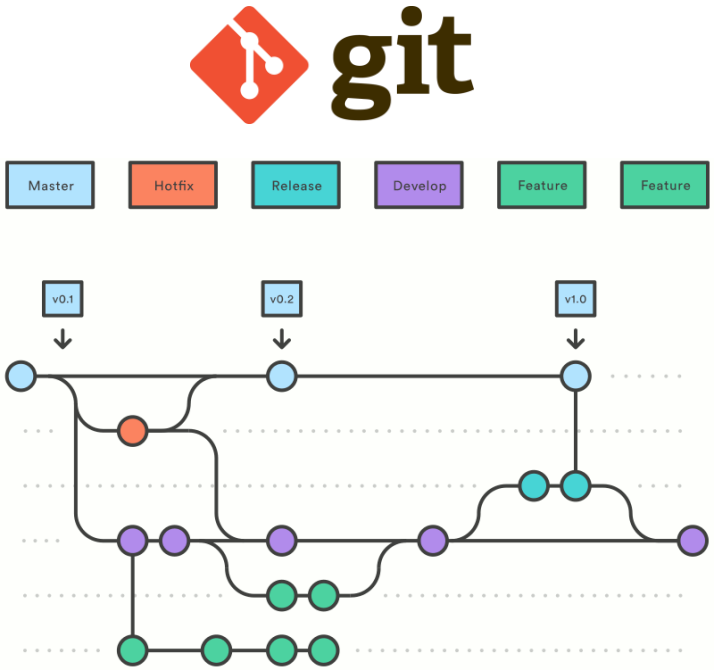
- A **pipeline** is an organized sequence of steps that transforms raw data into final results
  e.g., data collection → preprocessing → model training → evaluation → reporting
- Each stage has inputs, outputs, and dependencies
- Pipelines enable:
  - **Automation** of repetitive tasks
  - **Traceability** of intermediate results
  - **Reproducibility** of the full workflow



Source: https://suneeta-mall.github.io/blog/category/reproducible-ml/

Version control with GIT

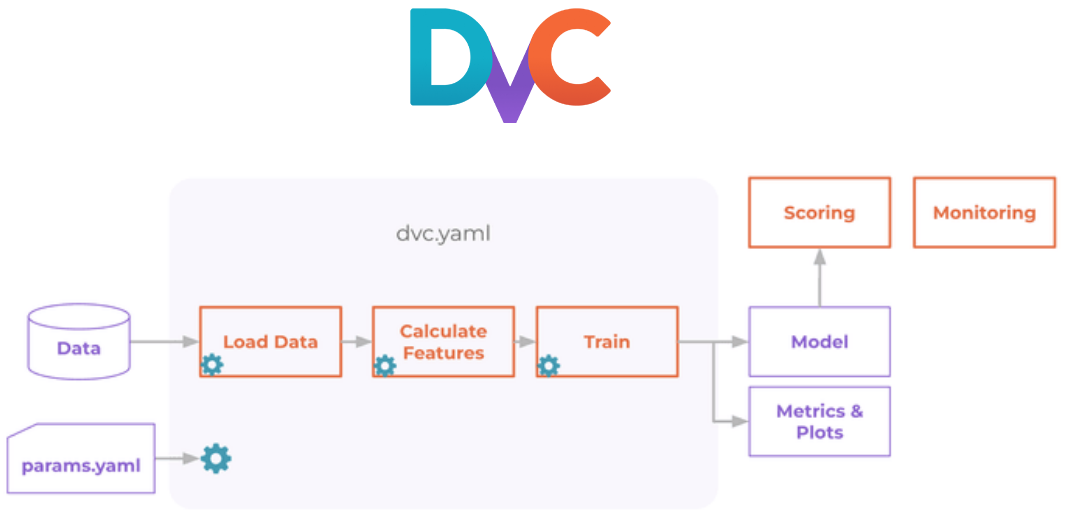
- **Git** tracks changes in code and text files over time
- Enables **collaboration**, **rollback**, and **branching**
- Essential for reproducible research because it:
  - Records *what changed, when, and by whom*
  - Links code versions to specific experiment results
  - Integrates with tools like GitHub, GitLab, Bitbucket
- Best practices:
  - Commit often, use clear messages
  - Tag releases corresponding to paper versions



Source: https://www.monolitonimbus.com.br/bitbucket-configuracao-e-branches/git_branches/
CC BY-SA

Data and model versioning with DVC

- DVC (Data Version Control) extends Git to handle large files, datasets, and ML pipelines
- Features:
  - Version control for data, models, and experiments
  - Pipeline management: define dependencies between stages
  - Experiment tracking with reproducible parameters
  - Remote storage integration (e.g., S3, Google Drive, SSH)
- Enables end-to-end reproducibility: every output can be traced to the exact input data and code



Tools for Python environment reproducibility

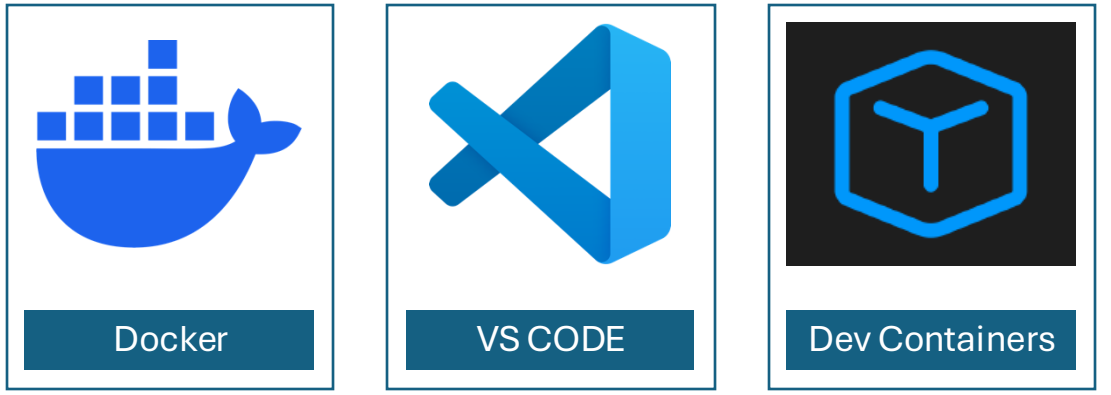
- Managing dependencies is essential for consistent results.
- Always freeze dependencies (environment.yml, Pipfile.lock, poetry.lock)
- Record environment hashes or export with conda env export or pip freeze

Common tools

Tool	Key Feature	Typical Use Case
Conda	Cross-language env manager; can fix Python & system libs	Reproducible scientific stacks
Pipenv	Combines pip & virtualenv, lockfile ensures consistency	App-level dependency control
Poetry	Modern dependency & packaging manager with lockfile	Reproducible ML projects, libraries

Containerization: Docker and Dev Containers

- **Docker** packages code, dependencies, and OS configuration into a portable container
- Ensures consistent execution across machines and platforms
- **Dev Containers** (VS Code feature) simplify reproducible development setups
- Benefits:
  - Exact replication of training environment
  - Simplified deployment and sharing
  - Integration with CI/CD and MLOps pipelines



Releasing code and data for a paper

- Publish your research artifacts with persistent identifiers
- Recommended steps:
  - Push final code to a public Git repository (tagged version)
  - Archive it with **Zenodo** or similar service
  - Obtain a **DOI (Digital Object Identifier)** for citation
  - Include data or metadata, environment files, and documentation
  - Provide a full reproduction script to regenerate all results, figures, and tables
  - Reference DOI in the paper and README

GitHub

Zenodo

doi.org

Building a reproducible ecosystem

A reproducible ML-forecasting project integrates:

- **Git + DVC** → version control for code, data, and models
- **Conda / Poetry / Pipenv** → reproducible environments
- **Docker / Dev Containers** → portable execution
- **Zenodo** → archival and citation
- **Automation pipelines** (CI/CD, MLOps) → reliability and scalability

Further reading

Forecasting Research

- Boylan et al., *Reproducibility in Forecasting Research*, *IJF*, 2015.
- Makridakis et al., *Objectivity, Reproducibility and Replicability in Forecasting Research*, *IJF*, 2018.

Machine Learning & Standards

- Pineau et al., *Improving Reproducibility in ML Research*, *NeurIPS Report*, 2020.
- SEI Blog: *The Myth of ML Non-Reproducibility and Randomness*, CMU SEI, 2021.
- **NeurIPS Reproducibility Checklist** (neurips.cc/Conferences/2024/CallForPapers)
- **AAAI Reproducibility Checklist** (aaai.org/Conferences/AAAI-24/reproducibility)

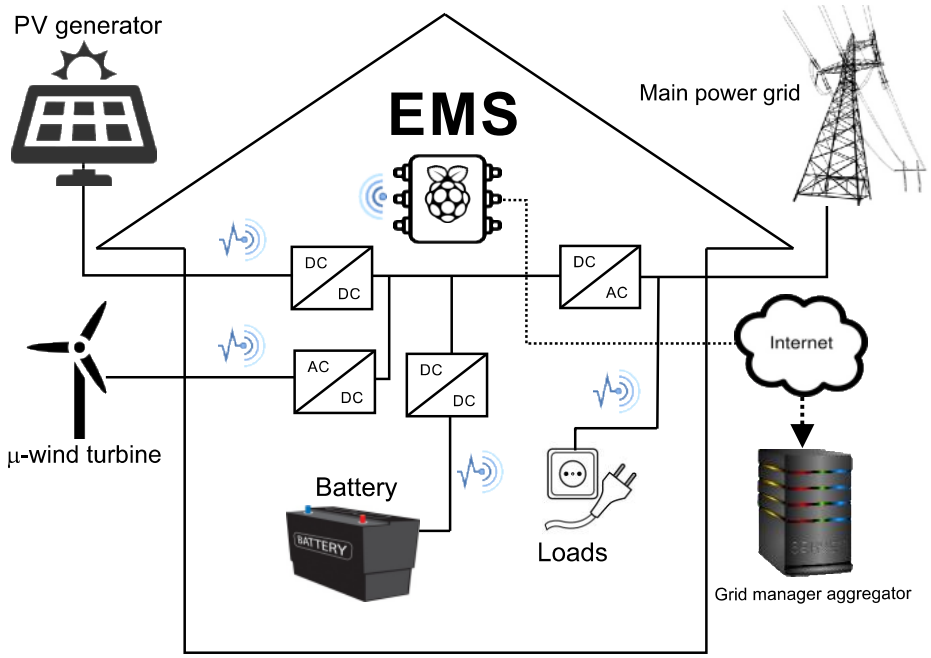
Practical Tools & Guidelines

- *Papers with Code – Releasing Research Code* (github.com/paperswithcode)
- *DVC Documentation* (dvc.org/doc)
- *Zenodo* – Archive & DOI for research artifacts (zenodo.org)

Hands-on Coding Examples

- The code of the examples is published at https://github.com/giulatona/iecon2025_tutorial
- Follow along by clicking on the badge Open in Colab

Scan to open the repository


PV generator
EMS
Main power grid
DC
DC
DC
AC
AC
DC
DC
DC
Internet
μ-wind turbine
Battery
BATTERY
Loads
Grid manager aggregator