University of Turin

School of Natural Sciences

Bachelor's Degree in Computer Science

Bachelor's Thesis

Generative AI and Machine Learning for Time Series

Supervisor:

Prof. Robert René Maria Birke

Candidate:

Federico Santorsola
1025104

ACADEMIC YEAR 2024/2025

"Intelligence is the ability to adapt to change."

— Stephen Hawking

Abstract

Time series are a fundamental component in numerous fields of application. Their analysis requires methodologies capable of handling large volumes of data, preserving temporal accuracy, and efficiently managing problems such as missing values and high variability. Traditional techniques, based on statistical models or classic *forecasting* methods, are often limited when it comes to generating plausible synthetic data, reconstructing missing portions, or simulating alternative scenarios in a manner consistent with the multivariate and seasonal structures of the data. *Diffusion Models* have become one of the most promising methodologies for generating sequential data, as they are capable of processing complex distributions and generating reliable samples. Among these, WaveStitch represents an innovative approach that combines arbitrary conditioning, parallelized generation, and cyclic representation of temporal characteristics, in order to better manage the tasks of imputation, forecasting, and simulation of time series. The results obtained have confirmed the potential of this model as a flexible and efficient tool in contexts characterized by the presence of time series, making it ideal for new research perspectives, but also for practical applications in critical, fragile, or sensitive situations, where reliable synthetic data is essential.

Keywords

WaveStitch, synthetic generation, diffusion models, time series, data imputation, forecasting, comparative analysis

I declare to be responsible for the content I'm presenting in order to obtain the final degree, not to have plagiarized in all or part of, the work produced by others and having cited original sources in consistent way with current plagiarism regulations and copyright. I am also aware that in case of false declaration, I could incur in law penalties and my admission to final exam could be denied.

Table of Contents

Chapter 1

Introduction and Objectives of the Work

The volume of activity carried out by High-Performance Computing (HPC) centres has led to an exponential increase in the amount of data generated and monitored within research institutions and energy infrastructures, for example, data collected by sensors inside the rooms housing the servers. This energy and environmental consumption data is a valuable source of information but also potentially sensitive: by analysing power and temperature variations, it is possible to indirectly infer information about system activity, such as workload intensity or peak usage times. Access to and sharing of this data are therefore limited by confidentiality and security constraints. In this context, the **generation of realistic synthetic time series** represents a promising solution to reconcile two often conflicting requirements: on the one hand, the need to conduct advanced analyses and develop predictive models; on the other, the protection of privacy and operational security within data centres. Creating synthetic data consistent with real data makes it possible to train and validate artificial intelligence models, experiment with optimisation algorithms, and evaluate energy efficiency strategies without exposing confidential information.

Objective of the Work

The objective of this work was to **produce synthetic time series of energy consumption that are realistic and consistent with the real data of the HPC centre in Turin**, using generative artificial intelligence models based on diffusion processes. In particular, the project is part of the broader field of research on conditional generation of time series data, in which synthetic sequences are created by taking into account auxiliary variables — such as rack type, date, time, or other operating conditions — in order to preserve the original patterns and correlations.

Approach Followed

To address the problem, I designed and implemented a complete analysis and modelling workflow, consisting of four main phases:

1. **Data cleaning and normalisation**: collection and cleaning of energy data from

InfluxDB, with verification of temporal consistency, removal of outliers, and standardisation of sampling frequencies;

2. **Exploratory analysis and clustering**: study of correlations between electrical quantities and identification of recurring operating regimes using *k-Means* and *PCA* techniques;

3. **Generative training with *WaveStitch***: application and adaptation of the *WaveStitch* diffusion model [26] for the conditional generation of time series in the HPC energy domain;

4. **Real vs synth evaluation and comparison**: analysis of synthetic sequences generated through numerical (*MSE*) and structural (*ACD* and *xCorrDiff*) metrics to verify fidelity and statistical consistency with the original data.

This approach made it possible to build a replicable and scalable workflow, capable of integrating *data preprocessing*, *unsupervised learning*, and *generative artificial intelligence* techniques within a single application framework.

Main Results

The results obtained show that, once properly calibrated, the *WaveStitch* model is capable of generating synthetic sequences that are **consistent, realistic, and statistically aligned** with the real time series of the HPC centre. Structural metrics (*Autocorrelation Difference* and *Cross-feature Correlation Difference*) demonstrated high fidelity of temporal and multivariate patterns, while numerical measures (*MSE*) confirmed good point-by-point alignment between real and synthetic traces. Overall, this work demonstrated how diffusion models can serve as a valid alternative to traditional approaches for data generation and imputation — in this case, energy data — contributing to the development of new strategies for **modelling, simulation, and protection of sensitive data** across a wide range of applications.

Chapter 2

Theoretical foundations

2.1 Diffusion models, WaveStitch, and time series

Diffusion models (DM) represent one of the most recent and promising families of generative models. They operate in two main phases:

1. **Forward process (noising):** Gaussian noise is added to the original sample progressively, step by step, until the distribution is almost completely devoid of the initial information. This process simulates how a noise source can "corrupt" initial data.

2. **Reverse process (denoising):** a neural network is trained to learn the inverse of the degradation process. Starting from pure noise, it is progressively reduced and the network performs an iterative denoising process that reconstructs plausible samples consistent with the distribution of the real data.

This probabilistic formulation allows complex distributions to be modeled and realistic samples to be generated, and forms the basis of modern *Denoising Diffusion Probabilistic Models* (DDPM) and *score-based* models, which have redefined the state of the art in deep generative models [6, 10, 20, 21]. **WaveStitch** [26] fits into this trend by introducing an innovative approach to the problem of conditional time series generation. The distinctive feature is the *conditional stitching* mechanism, which allows to:

- reconstruct missing values (*imputation*);

- predict future values (*forecasting*);

- generate alternative scenarios consistent with real data (*simulation*).

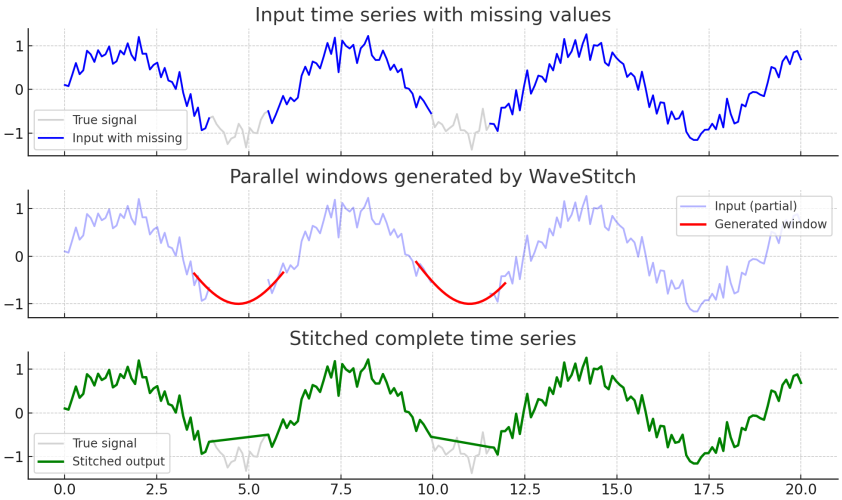

Figure 2.1: Conceptual diagram of how WaveStitch works (own elaboration): (top) input with missing values; (center) windows generated in parallel; (bottom) complete series obtained through *stitching*.

2.2 Machine learning in relation to this work

Machine learning (ML) is now one of the most powerful tools for analyzing and modeling complex data. In a nutshell, I would say that the idea behind ML is to train algorithms capable of learning rules and patterns directly from data, without the need for explicit programming.

2.2.1 Supervised vs. Unsupervised Learning

I would say that the work I am presenting straddles two classic ML paradigms:

- **Supervised Learning:** there is a model that is trained on a labeled dataset (each instance associated with the corresponding class). This means that for each example in the training dataset, there is a corresponding label or desired result. The goal of the model is to learn a function that maps inputs to the correct outputs, so that it can make accurate predictions on new, unseen data.

- **Unsupervised learning:** when data does not have explicit labels, classes are not known a priori. In this case, the goal is to identify latent structures or statistical relationships, as occurs when we cluster data.

Although the *WaveStitch* model is a conditional generator, it works in a manner similar to *unsupervised learning*, as it learns the distribution of data without the need for explicit labels, but uses masks and conditional constraints to guide generation.

2.2.2 Mathematical example

A simple example of a supervised learning model is **linear regression**, which aims to estimate a linear relationship between a set of independent variables and a dependent variable. In compact form, the model can be expressed as:

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n,$$

where $\hat{y}$ represents the predicted value, x_i the input variables, and w_i the associated weights (or coefficients), including the intercept w_0. The model parameters are estimated by minimizing a loss function, typically the *mean squared error* (MSE), defined as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$

where y_i is the observed actual value and $\hat{y}_i$ the corresponding model prediction. Minimizing the MSE allows us to obtain the coefficients w_i that best approximate the relationship between the variables, minimizing the overall difference between actual and estimated values.

WaveStitch, on the other hand, does not seek a deterministic function $f(x)$, but learns to generate samples from a probabilistic distribution $p(x)$, through a process of *diffusion* and subsequent *denoising* [26]. *Machine learning* therefore acts as a methodological glue between raw data management and the application of advanced generative models, but without adequate *preprocessing* and preliminary statistical analysis, as I will show in the following chapters, even a powerful model such as WaveStitch would fail to produce consistent and reliable results [2, 9].

2.3 Data Clustering in Artificial Intelligence

Clustering is an *unsupervised learning* technique that allows a set of data to be divided into homogeneous groups, called *clusters*, so that observations belonging to the same group are *similar* to each other, while those belonging to different groups are *dissimilar* [13]. Clustering is a fundamental tool for exploring and understanding data, as it allows latent structures, recurring patterns, and hidden relationships to be identified without resorting to predefined labels.

2.3.1 Basic principles

Given a set of n objects and a predetermined number $k \leq n$ of clusters to be created, the goal of clustering is to organize the data into k partitions such that:

- objects within the same cluster are *similar* to each other;
- objects belonging to different clusters are *dissimilar*.

In other words, clustering can be considered a form of *lossy data compression*, in which each group represents a collective abstraction of multiple observations.

2.3.2 k-Means algorithm

During my Information Systems course, I learned about various algorithms such as **k-medoids** and **k-modes**, but for this work I wanted to choose one of the most widespread and widely used as a tool and subject of study: **k-Means**. I will explain how it works below. The algorithm operates according to an iterative partitioning approach, in which

each data point is assigned to the cluster whose *centroid*—i.e., the average of the points belonging to that cluster—is closest.

The process can be described as follows:

1. k initial centroids are randomly selected from the dataset.
2. Each point is assigned to the cluster with the closest centroid (usually according to Euclidean distance).
3. The centroids are updated as the average of the points assigned to each cluster.
4. Steps (2)–(3) are repeated until convergence, i.e., until the assignments do not change or a maximum number of iterations is reached.

Formally, the algorithm minimizes the following objective function:

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

where C_i represents the i-th cluster and μ_i its centroid. The aim is therefore to minimize the overall distance of the points from their respective centroids, obtaining compact and well-separated clusters.

2.3.3 Advantages and limitations

The k-Means method has several advantages:

- it is simple to implement and conceptually intuitive;
- it is scalable and efficient even on large datasets, with complexity $O(nkt)$, where n is the number of objects, k is the number of clusters, and t is the number of iterations;
- it works well when clusters are compact and separable.

However, it also has some limitations [13]:

- it requires specifying the number k of clusters in advance;
- it is sensitive to the choice of initial centroids and can converge to local optima;
- it is not suitable for non-convex clusters or clusters with very different densities;
- it is influenced by *outliers*, which can significantly alter the position of the centroids.

2.3.4 Convergence and variants

The k-Means algorithm guarantees convergence in a finite number of steps, since there is a finite number of possible data partitions and each iteration reduces (or keeps constant) the objective function J. In the next chapter I will talk about the Wavestitch Paper.

Chapter 3

WaveStitch: Flexible and Fast Conditional Time Series Generation with Diffusion Models

3.1 Introduction

As I mentioned, **Wavestitch** is the diffusion model I have been tasked with researching. In this chapter, I would like to provide a general overview of it using the paper that describes it. The document begins by showing the three critical issues identified by the authors in the conditional generation of time series—the lack of adaptability of existing models to inference conditions, the slowness of autoregressive generative processes, and inefficiency in the encoding of categorical characteristics—which have been addressed through innovative and efficient solutions: a flexible conditioning mechanism, a parallel generation procedure with segment *stitching*, and cyclic encoding of categorical variables. These methodological choices make it possible to overcome the structural limitations of previous models, leading to significant improvements in both accuracy and computational efficiency [26] .

3.2 Challenges addressed

3.2.1 Q1: Adaptability to inference conditions

S One of the most significant limitations of existing generative models is their poor ability to adapt to changing conditions during inference. In many application contexts, time series depend not only on their historical trends, but also on auxiliary variables or metadata (e.g., year, month, district, type of service). Models such as TimeGAN or conditioned LSTM variants are trained on fixed configurations of conditions, which limits their ability to generalize to scenarios never seen during training. This rigidity translates into low operational flexibility: if new combinations are required during inference (e.g., a specific year associated with a brand not present in the training), the models are unable to adapt and produce inconsistent or statistically implausible samples [25].

3.2.2 Q2: Slowness of autoregressive methods

Most classical methods for generating time series are *autoregressive* in nature, i.e., they generate values one at a time, using sliding windows. While this approach ensures sequential consistency, it comes at a very high computational cost: generation becomes slow and difficult to parallelize, especially when working with large datasets or extended time horizons. For real-world applications—such as monitoring energy consumption in an HPC center or managing healthcare data—speed is a fundamental requirement. A method that takes minutes or hours to generate a synthetic sequence is impractical, especially when integrated into simulation or *real-time decision-making* systems. Consequently, the slowness of sequential methods represents a real obstacle to their industrial adoption [25, 27].

3.2.3 Q3: Inefficient encoding of categorical features

Time series are often influenced by categorical features, such as days of the week, months, or codes indicating membership in a certain group of devices. The most commonly used method for representing these variables is *one-hot encoding*, which, however, has two main limitations:

- it significantly increases the dimensionality of the input, making the model heavier and more difficult to train;
- it does not naturally represent the periodicity of variables, such as daily or monthly cycles, where extreme values (e.g., December and January) should be close together but are instead placed at opposite ends of the coding spectrum.

This inefficiency in representation results in a loss of structural information, which can compromise the model's ability to capture cyclical regularities typical of energy, financial, or environmental time series. These challenges constitute the motivational context within which WaveStitch was developed, whose architecture aims to provide innovative solutions for each of the problems highlighted [26].

3.3 Methodology and proposed solution

The methodology behind **WaveStitch** stems from the need to systematically address the three main challenges identified in the previous paragraph. The authors propose an architecture that exploits *Denoising Diffusion Probabilistic Models* (DDPMs) [10].

3.3.1 Flexible constraint management (Q1)

The first innovation of WaveStitch concerns its ability to generalize to conditions never seen during training. The model is trained in two distinct phases:

1. **Training:** the denoiser is trained to generate a complete signal conditioned *only* on auxiliary characteristics (e.g., year, month, region).
2. **Inference:** at the time of generation, constraints from the actual values of the observed signal (e.g., available portions of the series) are also introduced. Through conditional masks, the model integrates this partial information and produces coherent sequences that comply with both metadata and historical data.

This approach makes WaveStitch flexible and capable of adapting to unexpected scenarios, such as the request to generate a time series block for combinations of conditions not present in the training dataset [26].

3.3.2 Parallel generation with stitching (Q2)

To solve the problem of slow autoregressive methods, the authors introduce a **parallel generation** mechanism. The signal and conditions are divided into overlapping windows, processed in parallel by mini-batches. After each denoising step, the overlapping portions are reconciled through a *stitching* process, which consists of overwriting the common regions with the values generated by the previous window.

- This process maintains sequential consistency between independent segments.
- The computational complexity is reduced from $O(T \cdot t_\theta(w) \cdot (M - w)/s)$ (autoregressive) to $O(T \cdot t_\theta(w) \cdot (M - w)/(b \cdot s))$, where b is the batch size.
- In practice, this translates into a **speedup of up to 460 times** compared to sequential methods, without sacrificing the quality of the synthetic signal.

This solution combines the speed of parallel generation with temporal consistency, which until now has been the prerogative of autoregressive models. The mechanism is shown in Figure 3.1, which illustrates the parallel generation flow: overlapping windows are processed by multiple denoisers in parallel and then integrated through the *stitching* process in common regions [26].

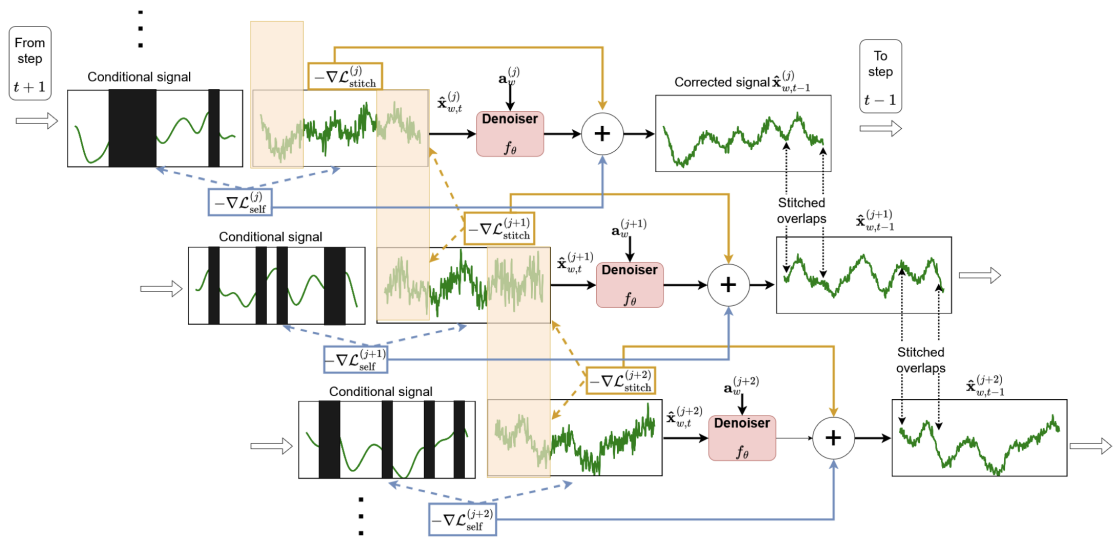

Figure 3.1: *Stitching* mechanism in WaveStitch: overlapping windows are denoised in parallel and reconciled in common areas to maintain sequential consistency. (Adapted from [26]).

3.3.3 Cyclic encoding of categorical variables (Q3)

The third innovation in WaveStitch concerns the representation of categorical features. Instead of using traditional *one-hot encoding*, the authors propose a **cyclic encoding**:

$$\theta_k = \frac{2\pi k}{K}, \quad \text{with } k = 0, 1, \ldots, K-1$$
$$a_{sin}(k) = \sin(\theta_k), \quad a_{cos}(k) = \cos(\theta_k)$$

Each categorical variable is then mapped to two coordinates (sin, cos) on a unit circle. This approach has three main advantages:

- it drastically reduces dimensionality (from K to 2 dimensions);
- it preserves the cyclical nature of the variables (for example, December and January are close on the circumference, unlike what happens with one-hot encoding);
- it decreases the sparsity of the input, making training more efficient [26].

3.4 Experiments and results

To validate the proposed approach, the authors of **WaveStitch** conducted a series of experiments on public datasets from different domains: *Beijing Air Quality*, *Metro Traffic Volume*, *Panama Energy*, *Rossman Sales*, and *Australia Tourism*. These datasets include univariate and multivariate signals, including both continuous variables (e.g., energy consumption, traffic volume) and categorical features (e.g., year, month, district) [26].

3.4.1 Experimental setup

- **Hardware configuration:** The experiments were performed on an AMD Ryzen 9 5900 processor with 12 cores and an NVIDIA RTX 3090 GPU, demonstrating the need for substantial computing resources for training diffusion models.
- **Data division:** The test sets were constructed to contain combinations of completely new auxiliary conditions, such as a year that was never present in the training set. This made the evaluation more realistic and rigorous.
- **Tasks evaluated:** imputation of missing values, short-term forecasting, and generation of synthetic scenarios under multi-level constraints (*Root*, *Intermediate*, *Bottom*).
- **Metrics:** The authors adopted a mix of point and structural metrics:
  - *Mean Squared Error (MSE)* to quantify numerical accuracy;
  - *Autocorrelation Difference (ACD)* to measure similarity in temporal patterns;
  - *Cross-feature Correlations (x-Corr)* to evaluate dependencies between multivariate variables [26].

3.4.2 Main results

The experimental results clearly show the advantages of WaveStitch:

- in terms of MSE, the model achieves up to **10 times better performance** than the baselines (e.g., TSDiff, TimeWeaver, TimeGAN), particularly in the *Australia Tourism (B)* task;

- the stitching mechanism allows for **460 times faster** generation than the traditional autoregressive approach, while maintaining comparable MSE values;
- in structural metrics, WaveStitch better preserves autocorrelations and cross-correlations, demonstrating that it replicates not only point values but also deep statistical relationships in the data [22, 25–27].

3.4.3 Comparative table

An excerpt of the results reported by the authors is shown in Table 3.1, which summarizes the performance of WaveStitch compared to the most relevant baselines.

Dataset / Task	Method	MSE	ACD	x-Corr
AustraliaTourism (B)	TimeGAN [25]	1.580	0.231	0.298
	TSDiff/CSDI [22]	1.420	0.185	0.271
	WaveStitch [26]	**0.153**	**0.072**	**0.101**
MetroTraffic (I)	TimeWeaver [27]	0.864	0.210	0.322
	TSDiff/CSDI [22]	0.701	0.175	0.285
	WaveStitch [26]	**0.312**	**0.090**	**0.133**

Comparison of methods: MSE (error), ACD (autocorrelation difference), x-Corr (correlation difference). The values show WaveStitch's ability to reduce numerical error (MSE) and preserve statistical structures (ACD, x-Corr); values adapted from [26].

3.5 Connection to this work

Perfect, now we know why the techniques introduced by WaveStitch can be directly applied to my work. Following the approach of the authors, who validated the model on benchmark datasets such as *Metro Traffic* and *Australia Tourism*, as I mentioned in the previous chapter, I chose to replicate the analysis, training, and evaluation flow by applying it to real energy data from the HPC center in Turin. In particular, the core of my work is the concepts of imputation, synthetic scenario generation, and conditional forecasting, which I will show how I have adapted to the context of monitoring rack consumption and electrical variables, with the aim of verifying the validity of the model in a complex and noisy environment.

Chapter 4

Implementation

4.1 Data Cleaning and Preprocessing

As my Data Analysis professor always said, "you can't do much with a dirty, disturbed, and noisy dataset." In fact, I started the implementation with a preliminary phase of **quality control, cleaning, and normalization of the data**, which is essential for clustering and training the generative model. All measurements exported from InfluxDB in CSV format underwent the same processing flow to ensure uniform treatment of energy signals and facilitate the identification of any anomalies or inconsistencies in the raw data [1, 8].

4.1.1 Quality Check (QC) Audit

As a first step, I implemented an automated *audit* function, applied to each CSV file with the aim of verifying the quality of the original data and identifying possible critical issues before the actual cleaning phase. This control procedure, performed in batches on all measurements exported from InfluxDB, calculates a set of diagnostic statistics and temporal consistency indicators for each column, thus providing a quantitative overview of the dataset's status. The main checks include:

- the percentage of missing values (*NaN %*);
- descriptive statistics such as standard deviation, minimum, maximum, and number of unique values (useful for identifying constant or nearly constant columns);
- analysis of time gaps between consecutive timestamps, by calculating the median interval and detecting any anomalous jumps [15].

To do this, I wrote the function shown in Listing 1, which automates the entire process on a single CSV file, returning both column statistics and an overall temporal consistency report.

```python
def audit_csv(path: Path):
    """
    Loads and analyzes a CSV file, returning:
    - Clean and temporally sorted DataFrame
    - Descriptive statistics per column
    - Diagnostics on temporal gaps
    """
    df = pd.read_csv(path, low_memory=False)
    df = df.loc[:, ~df.columns.str.contains(r"^Unnamed")]
    df["_time"] = pd.to_datetime(df["_time"], utc=True, errors="coerce")

    # Statistical analysis per column
    info = []
    for c in df.columns:
        if c == "_time": continue
        s = pd.to_numeric(df[c], errors="coerce")
        info.append({
            "column": c,
            "nan_%": round(s.isna().mean()*100, 2),
            "std": float(s.std(skipna=True)),
            "unique_vals": int(s.nunique(dropna=True))
        })

    # Diagnosis of time gaps
    dt = df["_time"].diff().dt.total_seconds()
    gaps = {"median_step_s": dt.median(), "big_gaps_%": (dt > 3*dt.median()).mean()*100}

    return df, pd.DataFrame(info), gaps
```

Listing 1: Preliminary audit function for time series datasets.

After validating the function on individual files, I applied it in batches to all measurements in the InfluxDB *bucket*, allowing me to quickly identify the noisiest or most incomplete measurements. An excerpt from the control report obtained for the *M2M* measurement is shown in Table 4.1.

Column	NaN %	Std	Min	Max	Unique values
Phase 1 Power	0.3	5.12	12.5	67.1	1221
Phase 2 Power	0.1	4.87	11.8	69.3	1243
Phase 3 Power	0.0	6.04	13.1	72.4	1250

Excerpt from the Quality Check report for the *M2M* measurement.

4.1.2 Data Cleaning and Normalization

After completing the audit phase, I was able to proceed with the **data cleaning and normalization** process. This phase is crucial, as the quality and reliability of the data directly determine the robustness of subsequent analyses [1, 15]. I implemented the *data cleaning* flow in a systematic and reusable way, applying it to all time series exported from InfluxDB. The pipeline follows a sequence of operations typical of analyses, adapted to the specificities of my dataset.

```python
def clean_timeseries_df(
    df_raw: pd.DataFrame,
    min_std: float = 1e-3,
    interpolate_limit: int = 3,
    resample_rule: str | None = None,      # e.g. "5T"
    winsorize: bool = True,
    winsor_limits=(0.01, 0.99)
) -> pd.DataFrame:
    df = df_raw.copy()
    df["_time"] = pd.to_datetime(df["_time"], utc=True, errors="coerce")
    df = df.dropna(subset=["_time"]).sort_values("_time").set_index("_time")

    # removal of empty or nearly constant columns
    df = df.apply(pd.to_numeric, errors="coerce")
    df = df.dropna(axis=1, how="all")
    df = df.loc[:, df.std(skipna=True) > min_std]

    # aggregation of time duplicates
    if df.index.has_duplicates:
        df = df.groupby(level=0).mean(numeric_only=True)

    # clipping per column to quantiles (winsorization)
    if winsorize:
        q_low, q_hi = df.quantile(winsor_limits[0]),
        ↪  df.quantile(winsor_limits[1])
        for c in df.columns:
            if q_low[c] < q_hi[c]:
                df[c] = df[c].clip(lower=q_low[c], upper=q_hi[c])

    # interpolation and (optional) resampling
    df = df.interpolate(method="time", limit=interpolate_limit,
    ↪  limit_direction="both")
    if resample_rule:
        df = (df.resample(resample_rule)
                .mean(numeric_only=True)
                .interpolate(limit=interpolate_limit, limit_direction="both"))

    return df.reset_index()
```

Listing 2: Dataset cleaning function: temporal parsing, removal of quasi-constant columns, winsorization, and interpolation.

First, I convert all timestamps to UTC datetime format, sorting the rows in chronological order and removing any records without a time reference. Next, I transform each column into numerical format, eliminating those that are completely empty or nearly constant, i.e., characterized by a negligible standard deviation or a zero minimum–maximum range. In the case of temporal duplicates, I consolidate the corresponding rows by calculating the average of the values, so as to obtain a monotonic and consistent time sequence. To mitigate the impact of outliers, I then apply **winsorization** to each column, limiting extreme values to quantiles $[q_\ell, q_h]$. This technique preserves the statistical structure of the central data while reducing the influence of outliers without eliminating them completely. For short-term gaps, I perform *time-based* (linear in time) interpolation up to a maximum of three consecutive samples. Finally, the function offers the possibility of performing a fixed-step **resampling** (e.g., every 5 minutes), so as to standardize the sampling frequency between the different measurements and make the series more com-

parable [3, 19]. Once the cleaning was complete, I proceeded to save the datasets in CSV format, sorting the columns deterministically (starting with `_time`) and optimizing file size by **downcasting to** `float32`. This choice allowed me to significantly reduce disk space while maintaining the accuracy necessary for subsequent analyses [1].

```python
def save_clean(df_clean: pd.DataFrame, out_path: Path):
    cols = ["_time"] + sorted([c for c in df_clean.columns if c != "_time"])
    df_clean = df_clean.loc[:, cols]
    for c in df_clean.columns:
        if c != "_time":
            df_clean[c] = pd.to_numeric(df_clean[c],
            ↪  errors="coerce").astype("float32")
    out_path.parent.mkdir(parents=True, exist_ok=True)
    df_clean.to_csv(out_path, index=False)
    return out_path
```

Listing 3: Function for saving clean datasets with stable sorting and numerical optimization.

Thanks to this process, I obtained consistent datasets, free of spurious or redundant columns and more robust with respect to the presence of outliers and time gaps. All the files used in the subsequent stages of *exploratory analysis*, *clustering*, and *generative training* derive from these cleaned and normalized CSV files.

4.2 Exploratory analysis and clustering

After completing the *data cleaning* phase, I began a systematic process of **exploratory data analysis** (EDA) with the aim of understanding the statistical structure of the energy variables and identifying the most informative fields to be used in the subsequent generative training phase [2, 8, 9]. The analysis started from the *M2M* measurement, chosen as a representative case study as it aggregates the behavior of a complete three-phase system, and was then extended to the main selected *Rack*pduA*.

4.2.1 Visualization of the main variables

To study the temporal trend of the most significant electrical signals, I selected the fields relating to current, voltage, power, and energy, generating an independent graph for each. This allowed me to identify long-term trends, daily cycles, and the presence of noise or local anomalies in the data [3, 19].

```python
import matplotlib.pyplot as plt

selected_fields = [
    "L1 Current", "L2 Current", "L3 Current",
    "Phase Voltage L1", "Phase Voltage L2", "Phase Voltage L3",
    "3Phase System Voltage", "Apparent Power L2",
    "3Phase System Frequency", "3Phase System Current",
    "3Phase System Apparent Power", "3Phase System Active Power AVG",
    "3Phase System Active Power", "3Phase System Active Energy"
]

df_sel = df_m[["_time"] + selected_fields].copy()

fig, axs = plt.subplots(len(selected_fields), 1, figsize=(15, 3 *
↪  len(selected_fields)))
for i, col in enumerate(selected_fields):
    axs[i].plot(df_sel["_time"], df_sel[col], label=col)
    axs[i].set_title(col); axs[i].set_xlabel("Time"); axs[i].grid(True)
plt.tight_layout(); plt.show()
```

Listing 4: Temporal display of the main electrical variables of the measurement M2M.

From inspecting the graphs, I observed a clear cyclical behavior in active and apparent power, consistent with typical HPC system loads. Phase voltages, on the other hand, showed greater local variability, indicating instrumental noise or fluctuations in the power supply network.

4.2.2 Correlation between variables

To quantify the linear relationships between the energy fields, I calculated **Pearson's correlation matrix** on the main signals of the *M2M* measurement, then extended the same procedure to the other measurements. The resulting heat map highlights:

- strong correlations between *phase currents* and *active powers* ($r \approx 0.9$–0.95);
- almost perfect correlations between *phase voltages* ($r \approx 0.99$);
- a very weak correlation between the *system frequency* and the other quantities;
- a moderate correlation between *active energy* and power, consistent with its cumulative behavior.

```python
import seaborn as sns

# Only numeric columns (excluding timestamp)
corr = df_selected.drop(columns="_time").corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", square=True)
plt.title(f"Correlation between field - {measurement}")
plt.tight_layout()
plt.show()
```

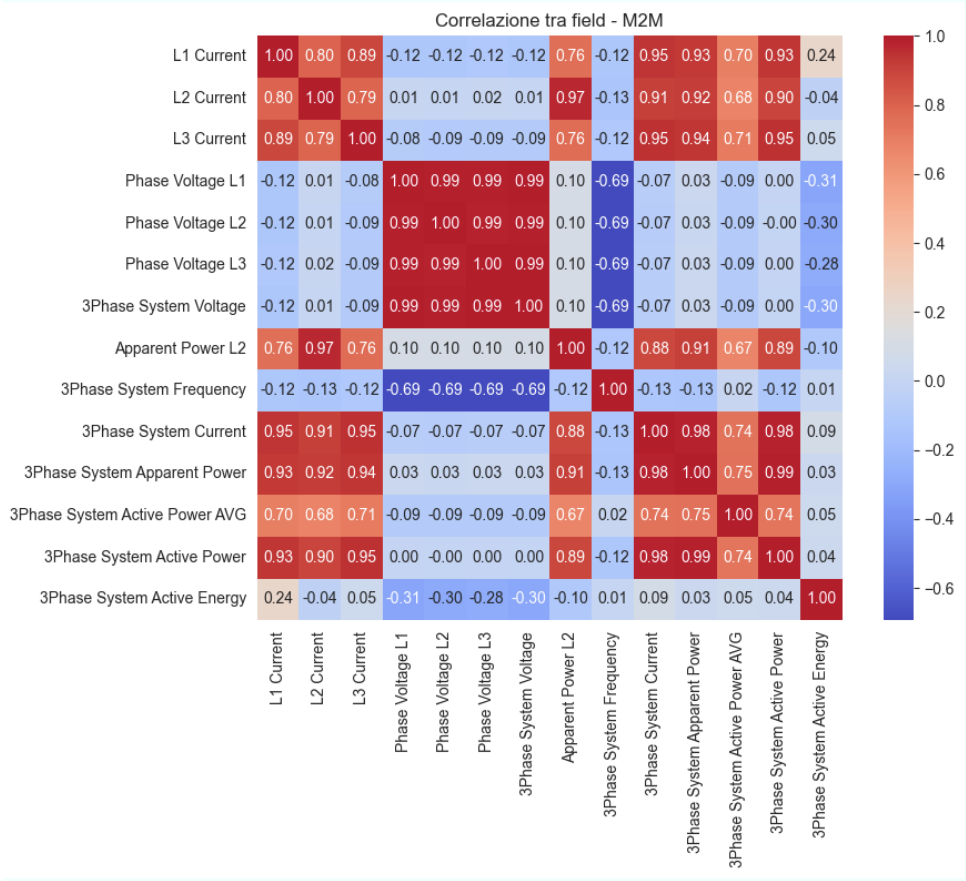

Figure 4.1: Correlation matrix between the main electric fields of the *M2M* measurement.

This analysis allowed me to identify groups of highly correlated and redundant variables, providing a solid basis for the subsequent selection of the most representative features to be used in clustering and generation models [2, 8, 9, 17].

4.2.3 Autocorrelation of signals

I deepened the temporal analysis by focusing on the variable `3Phase System Active Power AVG`, which summarizes the overall energy behavior of the system. I calculated the **autocorrelation function** (*ACF*) up to 100 delays, highlighting a slow decay and the presence of **cyclical patterns** attributable to daily and weekly periodicity.

```
from statsmodels.graphics.tsaplots import plot_acf

plot_acf(df_selected["3Phase System Active Power AVG"].dropna(), lags=100)
plt.title("Autocorrelation - 3Phase System Active Power AVG")
plt.show()
```

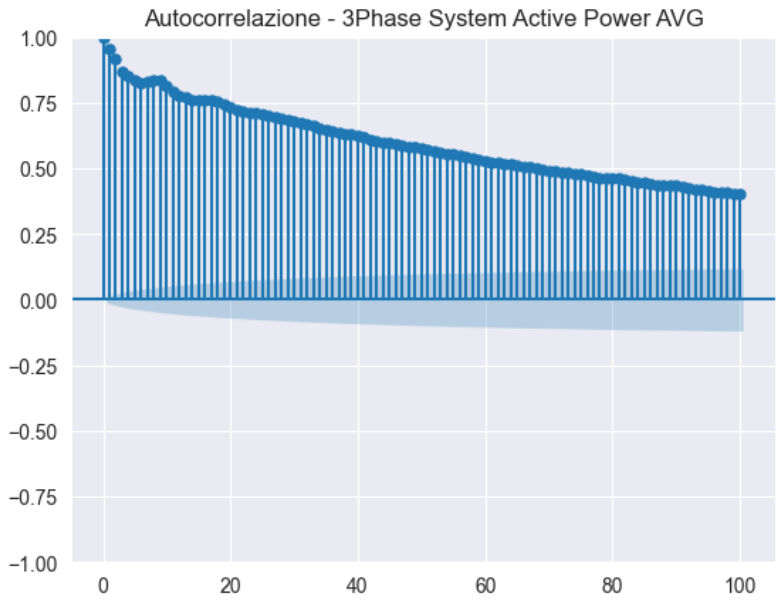

Figure 4.2: Autocorrelation function (*ACF*) of the average three-phase power (*M2M*) up to 100 delays.

The ACF analysis provided two key insights:

1. energy loads in HPC systems are not purely random, but follow regular cycles linked to the use of computational resources [3, 19];

2. the presence of stable periodicity justifies the introduction of **cyclic features** (sine and cosine of hour, day, and month) in the preprocessing pipeline, making the intrinsic seasonality of the data explicit and improving the model's ability to learn recurring patterns [2, 9].

4.2.4 Clustering with PCA and *k*-Means

To identify **latent structures** within energy signals from HPC systems, I developed a multi-step clustering workflow that integrates statistical analysis, dimensionality reduction, and visual interpretation of data. The objective of this phase was twofold: on the one hand, I needed to understand the intrinsic variability of the signals and identify recurring operating regimes; on the other hand, I needed to provide a compact and interpretable representation of the data for the subsequent phases [1, 2, 8, 9].

1. **Feature extraction.** I divided each signal into time windows of 128 steps (with stride 64) and calculated a set of statistical and dynamic descriptors for each window—mean, standard deviation, RMS energy, mean slope, and lag-1 autocorrelation—for each power channel. This approach allowed me to transform the continuous time series into a structured set of numerical vectors, more suitable for multivariate analysis.

2. **Cleaning and scaling.** I subjected the features obtained in this way to a robust normalization process to ensure greater numerical stability and reduce the influence of outliers or annoying measurement noise. In particular, I performed:

    - removal of constant or entirely null columns;

- imputation of missing values with the median for each feature;
- *clipping* at the 1st–99th percentile to attenuate outliers;
- the application of a *RobustScaler* (based on quantiles 5–95%) to obtain a stable and comparable distribution between features.

```
from sklearn.preprocessing import RobustScaler
import numpy as np

F = Feat_tr.astype("float64")
F[~np.isfinite(F)] = np.nan
col_med = np.nanmedian(F, axis=0)
inds = np.where(np.isnan(F))
if inds[0].size: F[inds] = np.take(col_med, inds[1])
var = np.var(F, axis=0)
F = F[:, var > 1e-12]                # removes quasi-constants
lo, hi = np.quantile(F, [0.01, 0.99], axis=0)
F = np.clip(F, lo, hi)               # clipping outliers
Fz = RobustScaler().fit_transform(F) # robust scaling
```

3. **Dimensional reduction (PCA).** To analyze the internal structure of the features, I applied *Principal Component Analysis* (PCA), reducing the variable space to:
   - 2 principal components for graphical visualization (*PCA-2D*);
   - up to 10 components for the actual input to clustering (*PCA-10D*).

   This transformation allowed me to capture over 80% of the total variance, thus preserving essential information at a reduced computational cost [14].

4. **k-Means and optimal selection of k.** I then applied the *k-Means* algorithm for values of k between 2 and 6, evaluating the quality of each configuration using the *silhouette score*, a measure of internal cohesion and separation between clusters [18].

Evaluation and results

Analysis of the *silhouette score* values (Figure 4.3) showed a maximum for $k = 5$ ($s \approx 0.42$), indicating a clear separation between the groups and a well-defined latent structure in the data. The two-dimensional projections obtained with PCA show distinct and consistent clusters, interpretable as different **operating regimes** of the HPC system and its power supply subsystem.

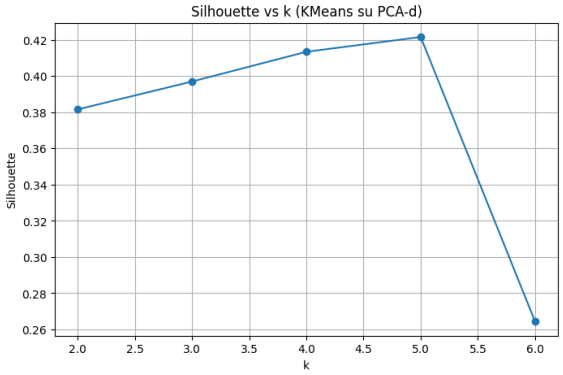

Figure 4.3: *Silhouette score* trend as k varies.

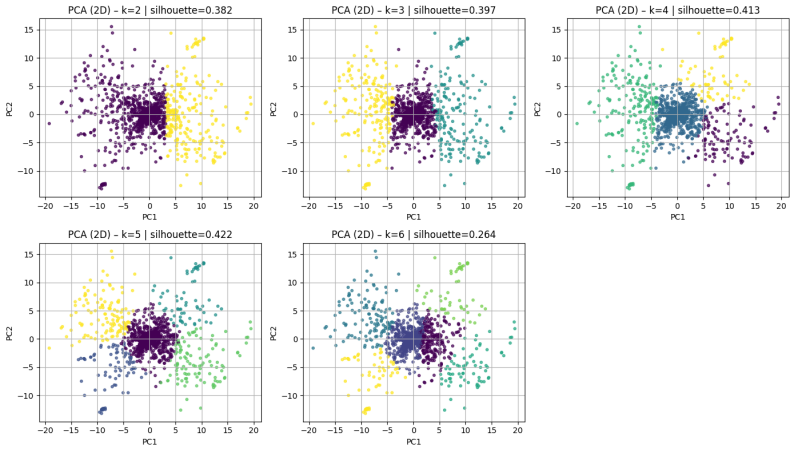

Figure 4.4: PCA-2D projections of clusters obtained with $k = 2 \dots 6$. A progressive refinement of the separation can be observed up to $k = 5$, beyond which the clusters begin to fragment.

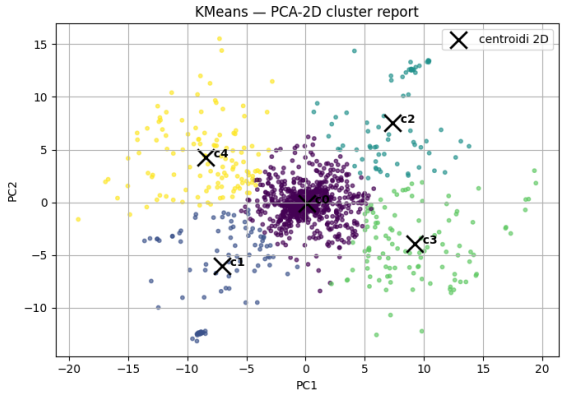

Figure 4.5: Distribution of data in PCA-2D space with labeled centroids for each cluster ($k = 5$). There is a clear separation between the density regions, corresponding to different energy regimes.

Interpretation of clusters

As I mentioned, analysis of the results allowed me to identify three main categories of behavior:

- **Stable clusters**, characterized by almost constant signals and associated with basic loads or periods of system inactivity;
- **Dynamic clusters**, with rapid variations and power peaks, typical of periods of intense computing;
- **Intermediate clusters**, interpretable as transitional states or partial activity between the two extremes.

These partitions provide a clear representation of **recurring energy patterns** in the racks and form an empirical basis for the subsequent **conditional generation of synthetic series** using diffusion models such as *WaveStitch*.

4.3 Training, scenario imputation, and conditional generation with WaveStitch

After completing the steps described so far, I finally started the **training and conditional generation** phase with the *WaveStitch* model. This part of the work represents the experimental core of my project, with the interesting aim of: on the one hand, evaluating the model's ability to adapt to real data affected by noise and irregularities; on the other hand, verifying the possibility of producing **plausible synthetic scenarios**, useful for simulation, testing, and much more.

4.3.1 Model training

I began training with the *M2M* metric, selected for its regularity and its ability to represent, as I mentioned earlier, the overall energy performance of the HPC system. The configuration adopted for training was defined in order to balance complexity, stability, and generalization capacity, with the following main hyperparameters:

- **Backbone**: `S4`, chosen for its efficiency in modeling long sequences using structured memory;
- **Window size**: 96, to capture a broader and more informative temporal context;
- **Stride**: 8, to ensure high overlap and improve continuity between successive windows;
- **Epochs**: 100, ensuring stable convergence of the learning process;
- **Batch size**: 64, balancing numerical stability and memory limitations;
- **Diffusion timesteps**: 200, higher than those in the original paper to handle the greater variability of HPC signals;
- **Cyclic features**: `month, day, hour`, useful for capturing daily and seasonal periodicity.

To ensure flexibility in experimentation, I created a parametric Python script that dynamically constructs the execution command of the `training_wavestitch.py` file, allowing training sessions to be started in a controlled manner on different datasets and with variable configurations [7].

```
cmd = [
    sys.executable, "training_wavestitch.py",
    "-dataset", DATASET,
    "-backbone", BACKBONE,
    "-window_size", str(WINDOW_SNE),
    "-stride", str(STRIDE),
    "-epochs", str(EPOCHS),
    "-batch_size", str(BATCH_SNE),
    "-hdim", str(HDIM),
    "-layers", str(LAYERS),
    "-timesteps", str(TIMESTEPS),
    "-s4_dropout", str(S4_DROPOUT),
    "-propCycEnc", PROP_CYC,
]
res = subprocess.run(cmd, text=True, capture_output=True)
print(res.stdout)
```

The resulting model is saved in an **enriched checkpoint**, which not only contains the weights of the neural network, but also includes: the complete configuration of the training hyperparameters, the ordered list of the dataset columns, the indication of cyclic and non-cyclic features, and the actual input and output dimensions. The final checkpoint was named `model_prop.pth`, indicating the presence of cyclic features (*propCycEnc*) in the training.

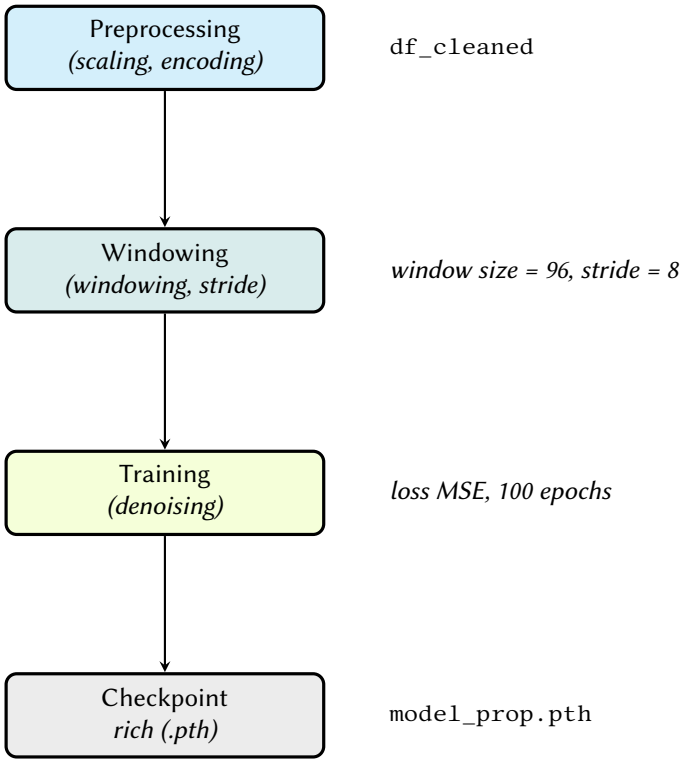

Figure 4.6: Simplified diagram of the training phase: preprocessing → windowing → denoising → rich checkpoint.

This configuration produced **stable convergence** on the *M2M* data, with a progressive reduction in loss and final values comparable to those reported in the original work [6, 10, 26]. Doing the same with the racks, I can say that the result demonstrated how effectively *WaveStitch* can adapt to real datasets, which are noisier and more complex than the original benchmarks, confirming its robustness and generalization capabilities.

4.3.2 Conditional synthetic generation

After completing the training phase, I started the **conditional synthetic generation** using the previously trained *rich checkpoint*, without the need for retraining. The procedure was managed using the script: `synthesis_wavestitch_pipeline_strided_preconditioning.py`, which automatically reads the hyperparameters from the checkpoint and realigns all key parameters — *backbone*, window, stride, and number of `timesteps` — ensuring perfect consistency between the training and generation phases [6, 10, 26]. To explore different temporal granularities of imputation, I defined three distinct scenarios: **F (Fine)** — point imputation, suitable for local *forecasting*; **C (Coarse)** — reconstruction of extended time blocks or those with significant gaps; **M (Mid)** — intermediate scenario, useful for modeling transitions or progressive variations.

In order to improve the multivariate fidelity and temporal continuity of the generated sequences, I introduced two fundamental extensions: a **configurable stitching loss** in the overlap areas between parallel windows, based on `mse`, `mae`, or shape-based metrics (`cosine`, `pearson`) [6, 10]; a **correlation regularization between channels** ($\lambda_{\text{corr}} = 0.30$, `pearson` metric), to preserve the statistical relationships between connected variables such as currents and powers [2, 9].

```
# Synthesis consistent with checkpoint, three masks: F / C / M
for mask in MASKS:
    out_dir_mask = os.path.join(BASE_OUT, mask)
    os.makedirs(out_dir_mask, exist_ok=True)

    cmd = [
        sys.executable, SYNTH_SCRIPT,
        "-dataset", DATASET,
        "-backbone", BACKBONE,
        "-window_size", str(WINDOW_SIZE),
        "-stride", str(STRIDE),
        "-timesteps", str(TIMESTEPS),
        "-hdim", str(HDIM),
        "-layers", str(LAYERS),
        "-batch_size", str(BATCH_SIZE_SYNTH),
        "-synth_mask", mask,
        "-n_trials", "1",
        "-propCycEnc", PROP_CYC,
        "-s4_lmax", str(S4_LMAX),
        "-s4_dstate", str(S4_DSTATE),
        "-s4_dropout", str(S4_DROPOUT),
        "-s4_layernorm", str(S4_LAYERNORM),
        "-ckpt", CKPT,
        "-lambda_corr", str(LAMBDA_CORR),
        "-corr_mode", CORR_MODE,
        "-out_dir", out_dir_mask,
    ]

    print("[CMD] " + " ".join(shlex.quote(c) for c in cmd))
    res = subprocess.run(cmd, text=True, capture_output=True)
# (repeat with -synth_mask C and M)
```

Listing 5: Conditional synthesis pipeline with automatic checkpoint alignment and multiscenario generation.

Each run generates two separate files: `real.csv`, containing the actual reference segment, and `synth*.csv`, representing the synthetic output produced by the model.

4.3.3 Preliminary observations

The first experiments conducted on my datasets showed that *WaveStitch*, even in the presence of noise and outliers, is able to generate sequences **that are statistically consistent** with the overall properties of the real data. Looking at the point-by-point comparison, I noticed that the synthetic data tends to smooth out the most extreme peaks, producing more regular and realistic curves. This behavior, while reducing the local linear correlation with the original series, actually confirms the model's remarkable ability to **generalize plausible behaviors** instead of rigidly replicating the outliers present in the source data.

Throughout this process, I was able to observe the excellent **computational scalability** of the model. Thanks to the *parallel generation mechanism with stitching*, the synthesis proved to be very fast, in line with the results of the original work. The ability to dynamically adjust the *stitching* and *correlation regularization* components made the pipeline flexible and adaptable to multiple experimental scenarios, ranging from short-term *forecasting* to the imputation of missing time blocks.

4.3.4 Extension to other HPC datasets

As I mentioned, I replicated the entire training and synthesis process on individual rack measurements, which are characterized by greater noise and temporal irregularities. This systematic comparison between aggregated and local datasets allowed me to confirm that **the quality and regularity of the input signal directly influence the stability and statistical fidelity of the synthetic generation**. The quantitative results (MAE, RMSE, MAPE, Pearson correlation) and qualitative analyses (visual analysis, autocorrelations, and cross-correlations) will be explored in more detail in the following chapters, where I will show how *WaveStitch* consistently reproduces both the average trends and the temporal structure of HPC energy signals.

4.3.5 Imputation of missing values with WaveStitch

The most valuable part of my work consists of the following three phases. I started by imputing missing values, which represented the most direct test of **conditional generation** based on *WaveStitch*. The goal is to estimate the missing values $y^\star(t)$ at unobserved times $\mathcal{M}$, starting from a real series $y(t)$ and the synthetic trajectory $\tilde{y}(t)$ produced by the model [3, 19]:

$$y^\star(t) \approx \tilde{y}(t) \quad \text{for } t \in \mathcal{M}.$$

In the event that the `real.csv` file does not contain null values, I introduced an artificial mask with amplitude $|\mathcal{M}| = 100$, so that I could quantitatively evaluate the quality of the imputation with respect to the original *ground truth* [15]. In this way, I was able to accurately quantify the effectiveness of the model even in the absence of truly missing data.

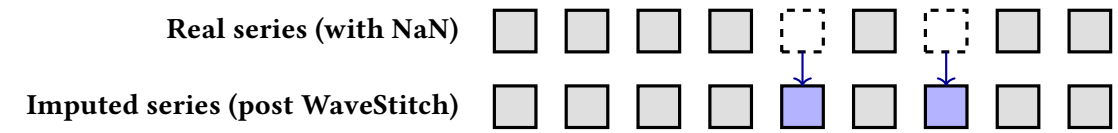

Figure 4.7: Conceptual diagram of imputation: NaNs (white, dashed) are replaced by imputed values (blue) generated by WaveStitch.

Alignment and selection of the target column. Since the `real.csv` and `synth*.csv` files may have slightly misaligned timestamps, I implemented a **merge-as-of** alignment, based on a nearest-neighbor match on times [8]. This choice ensures robustness even in the presence of minimal sampling differences. The column of interest v is identified through a hierarchical selection procedure: if the user provides an explicit `value_col` parameter, I use that; otherwise, the function automatically attempts to detect common columns (e.g., `3Phase System Active Power AVG`); as a last fallback, it chooses the first numeric column shared between real and synthetic files — avoiding common mistakes such as accidentally selecting the timestamp column as the target variable.

Affine calibration and clipping. To compensate for possible differences in scale or offset between real and synthetic data, I introduced an **affine calibration**:

$$\hat{y}_{\text{synth}}(t) = a \cdot y_{\text{synth}}(t) + b,$$

with (a, b) estimated via linear regression on the observed points $t \in O$ [8]. I considered two operating modes:

- **Global**, which estimates (a, b) across the entire series — more stable but less sensitive to regime changes;
- **Local**, which estimates (a_ℓ, b_ℓ) for each contiguous block of NaN within a window $\pm W$, more suitable for non-stationary or drifting signals.

Finally, I applied a **quantile clipping** on $[q_\ell, q_h] = [0.01, 0.99]$, anchoring the scale to the real signal support and reducing the impact of outliers and heavy tails [1, 15].

Filling strategy. The imputation process consists of four steps:

1. optional generation of artificial NaNs (for quantitative evaluation);
2. estimation of affine parameters (a, b) or (a_ℓ, b_ℓ) on observed points only;
3. reconstruction of missing values through affine transformation:

$$y^\star(t) = \begin{cases} a_\ell \, y_{\text{synth}}(t) + b_\ell, & \text{if } \texttt{calibrate=local}, \\ a \, y_{\text{synth}}(t) + b, & \text{otherwise;} \end{cases}$$

4. *tapering at the edges*, i.e., a linear blending on $\pm$ `half_window` to eliminate discontinuities at the hole boundaries [19].

Control parameters and robustness. I clearly designed the function to be adaptable to different inputs and distinct datasets. Thanks to this configuration, the function proved to be robust on heterogeneous data, able to automatically handle differences in sampling, scaling, and naming without manual intervention.

```
# WaveStitch-guided imputation with calibration and clipping
csv_path, metrics = impute_missing_with_wavestitch(
    dataset="M2M", mask="M",
    base_path="notebooks/generated",
    value_col=None,           # automatic selection of target column
    n_missing=100,            # artificial mask for quantitative evaluation
    calibrate="local",        # calibration for NaN blocks
    local_window=100,         # window width ±W
    clip_quantiles=(0.01,0.99)  # clipping for energy outliers
)
print(metrics)  # -> MAE / RMSE / Corr on masked positions
```

Listing 6: Example of imputation with WaveStitch and quantitative evaluation.

I achieved an excellent compromise between fidelity and numerical stability simply by combining `calibrate=local` and `clip_quantiles=(0.01,0.99)`, maintaining consistency even on noisy signals or those affected by outliers [15, 19].

Empirical observations. In tests conducted on *M2M* (mask M), the **global calibration** produced regular and conservative estimates, while the **local calibration** significantly improved the reconstruction of long gaps and transitions between different regimes. The **clipping** reduced the effect of extreme values, improving the MAE and RMSE metrics, while the **tapering** eliminated visual discontinuities at the edges of the imputed segments. Overall, the results show a solid balance between statistical fidelity and signal regularity, with average correlation values $r > 0.9$ on artificial masks, as I will show in the next chapter.

4.3.6 Simulation of scenarios via STL decomposition

To explore hypothetical *what–if* scenarios on the energy consumption profiles of HPC systems, I implemented a simulation procedure based on the **STL seasonal decomposition** (*Seasonal-Trend decomposition using Loess*). The goal was to understand how the different components—trend, seasonality, and residual—contribute to the overall shape of the signals, and how their modulation could simulate operational variations, drifts, or local shocks. Formally, the real series $y(t)$ is decomposed into three main components:

$$y(t) = T(t) + S(t) + R(t),$$

where $T(t)$ represents the long-term trend, $S(t)$ the seasonal cyclic component, and $R(t)$ the unstructured residual. Starting from this decomposition, I introduced three scaling factors (α, β, γ) and a local shock term Δ to construct new synthetic scenarios:

$$y_{\text{scen}}(t) = \beta\, T(t) + \alpha\, S(t) + \gamma\, R(t) + \mathbf{1}_{\{t=t^\star\}} \cdot \Delta.$$

This formulation makes it possible to amplify or attenuate seasonality, modify long-term trends, control residual noise, and introduce controlled impulse peaks—for example, to simulate sudden load events or temporary failures.

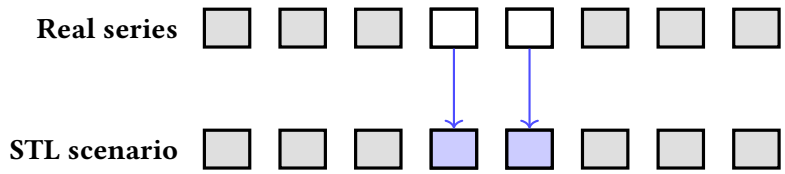

Figure 4.8: Conceptual diagram of STL simulation: modified values (white) are replaced with generated values (blue) after seasonal adjustment or the introduction of a local shock [5, 19].

Conceptual diagram.

Automatic column detection. Since the generated CSV files may contain heterogeneous structures, I made the `simulate_scenario_stl_flex` function fully autonomous in identifying key columns. In particular, the function automatically detects:

1. the time column, through a heuristic search among common names (`date`, `time`, `timestamp`);

2. the target value column, chosen from a list of known names or, if none are found, the first valid numeric column.

Determination of the seasonal period. The quality of the STL decomposition depends strongly on the correct estimation of the period P, i.e., the number of samples that constitute a complete cycle. To determine P, I developed a robust inference strategy based on timestamps: estimating the median interval Δt between two observations and setting

$$P \approx \left\lfloor \frac{86400}{\Delta t} \right\rfloor,$$

which corresponds to the average number of daily samples (e.g., $P \approx 288$ for 5-minute data, or $P \approx 24$ for hourly data). When the series is too short to observe multiple complete cycles, the period is reduced to $\max(2, \lfloor |y|/3 \rfloor)$ to ensure numerical stability. Nevertheless, I left the option to manually specify P in case of prior knowledge.

Implementation and main parameters. The simulation function accepts flexible parameters to modulate the three components:

- `season_scale` (α) — scale of the seasonal component $S(t)$ (default 1.2);
- `trend_scale` (β) — trend amplification factor for $T(t)$ (default 1.0);
- `resid_scale` (γ) — control for residual noise $R(t)$ (default 1.0);
- `shock_idx`, `shock_mag` — time index and amplitude of a possible impulsive shock Δ;
- `period` — STL period, calculated automatically or defined by the user.

The result is a CSV file with an additional column `{value_col}_scenario` and a preview graph showing the first 600 samples.

Application to the M2M dataset. I applied this procedure to the M2M dataset (5-minute sampling, $P = 288$) considering two scenario configurations:

1. 20% amplification of the seasonal component without shock;
2. 30% amplification of the seasonal component with the addition of a local shock.

```
# STL scenario simulation on M2M
base = "notebooks/generated"

csv_out, png_out = simulate_scenario_stl_flex(
    dataset="M2M", mask="M", base_path=base,
    value_col="3Phase System Active Power AVG",
    season_scale=1.2, trend_scale=1.0, resid_scale=1.0,
    period=288, shock_idx=None, shock_mag=0.0
)

# Variant with amplified seasonality and local shock
csv_out2, png_out2 = simulate_scenario_stl_flex(
    dataset="M2M", mask="M", base_path=base,
    value_col="3Phase System Active Power AVG",
    season_scale=1.3, trend_scale=1.0, resid_scale=1.0,
    period=288, shock_idx=10000, shock_mag=500.0
)
```

Listing 7: Running the STL simulation on M2M: seasonal amplification and optional shock.

Both scenarios keep $T(t)$ and $R(t)$ unchanged, allowing the impact of the periodic component alone to be isolated and analyzed. The progressive increase of α accentuates daily cyclicality without distorting the underlying trend. The scenario with 30% amplified seasonality shows more reactive behavior to daily peaks, which is useful for testing the robustness of forecasting models and overall energy stability [3, 5, 19].

Limitations and interpretation. The STL decomposition assumes an additive structure and quasi-stationary seasonality; in the presence of regime shifts or multiplicative effects, linear scaling may over- or underestimate the true impact [5, 19]. In such contexts, the simulated outputs should be interpreted as **qualitative what–if analyses**, useful for stress testing and strategic planning rather than for accurate point forecasting. Integration with conditional generation through *WaveStitch*, on the other hand, enables the creation of consistent multivariate synthetic scenarios, preserving both temporal and inter-variable dependencies observed in real data—and it has been extremely satisfying to verify this in practice.

4.4 Conditional forecasting with WaveStitch

Here we are at the third and, for me, most interesting phase. In this section, I describe the *forecasting* module that I developed following the conditional generation with *WaveStitch*. The goal was to evaluate the model's ability to project the local dynamics of a time series toward a short-term future horizon, maintaining consistency in both scale and shape with respect to the actual data. My operational idea was very straightforward: once the synthetic sequence was aligned with the real series (`real.csv` and `synth*.csv`), I extracted the tail of length h (the *horizon*) and used it as the predictive base for the next

steps. To compensate for possible level or scale misalignments between real and synthetic data, I applied an **affine calibration** on the observed tail, estimating the parameters (a, b) by linear regression on the common endpoints:

$$y_{\text{real}} \approx a\, y_{\text{synth}} + b, \quad \text{with} \quad (a, b) = \arg\min_{a,b} \|y_{\text{tail}} - (a\, x_{\text{tail}} + b)\|_2^2,$$

possibly including ridge regularization to handle noise or high variance. The final forecast is then obtained by applying the same transformation to the subsequent h synthetic points [2, 9]:

$$\widehat{y}_{t+1:t+h} = a\, y^{\text{synth}}_{t+1:t+h} + b.$$

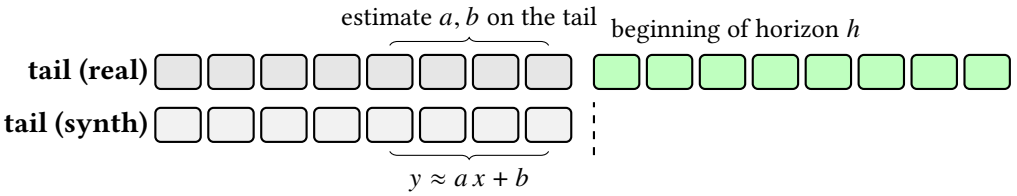

Figure 4.9: Conceptual diagram of *tail forecasting*: an affine transformation $y \approx a\, x + b$ is estimated on the real and synthetic tail, and then applied to the subsequent h steps to obtain the calibrated forecast (green).

4.4.1 Operational pipeline

Here too, I designed the function to be autonomous and robust with respect to the different CSV formats produced by the generation process [8]. The flow follows these steps:

1. **I/O resolution and loading:** automatic detection of `real.csv` and `synth*.csv` files in the path `<base>/<dataset>/<mask>/`.

2. **Time alignment:** detection of the time column, conversion to `datetime`, and alignment with *merge-asof* (nearest-neighbor on timestamps).

3. **Target column selection:** automatic identification of the variable to predict (`value_col`), searching among common names or selecting the first numeric column.

4. **Affine estimation on the tail:** given n_{tail} points, estimate (a, b) via OLS regression and compute diagnostic indicators (R^2_{tail}, correlation).

5. **Forecasting:** extract the last h synthetic points, apply the affine transformation, and construct future timestamps while maintaining the original sampling frequency.

6. **Persistence and graph:** save results (`.csv`, `.png`) in `<dataset>/forecast/`, including in the title the estimated values of a, b and R^2_{tail}.

```python
def forecast_with_wavestitch_flex(dataset, mask="F", base_path=None,
                                  value_col=None, horizon=64, calibrate=True):
    # 1) Load real and synthetic data, auto-detect time and target column
    r = pd.read_csv(...); s = pd.read_csv(...)
    t_r, t_s = _auto_time_col(r), _auto_time_col(s)
    value_col = _auto_value_col(r)
    m = pd.merge_asof(r, s, left_on=t_r, right_on=t_s, direction="nearest")

    # 2) Affine calibration on the tail
    a, b, R2 = _fit_affine(m[value_col+"_synth"][-200:],
    ↪  m[value_col+"_real"][-200:])

    # 3) Forward h-step forecast
    fc = a * m[value_col+"_synth"][-horizon:] + b
    out = pd.DataFrame({"forecast": fc})
    out.to_csv(".../forecast.csv", index=False)
```

Listing 8: Excerpt from the forecasting function: alignment, calibration, and prediction.

Implementation extract.

4.4.2 Forecast example (M2M, mask F, h=32)

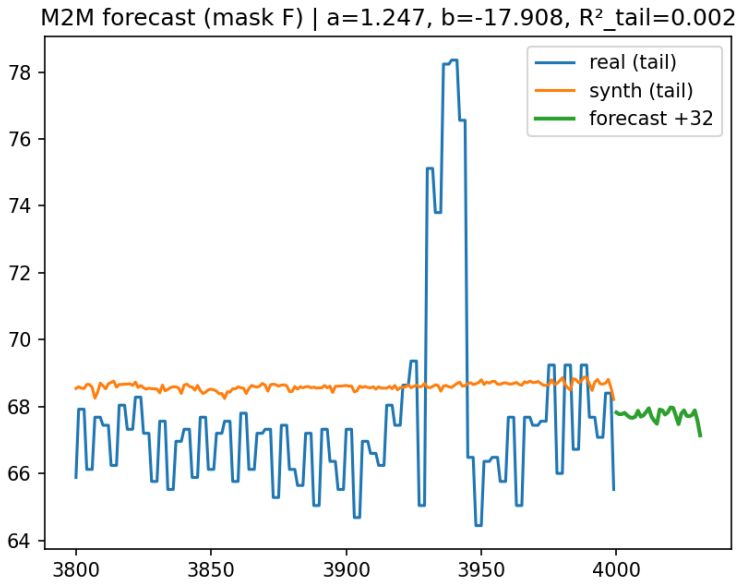

Figure 4.10: Example of forecast on *M2M* (h=32) with affine calibration on the tail. The green curve represents the predicted values for the following time steps.

4.4.3 Final observations and considerations

This experimental analysis showed that *WaveStitch*-based forecasting works effectively when the horizon h remains close to the local dynamics learned within the tail [22, 26]. In such cases, as I will show in the next chapter, the **affine calibration** systematically reduces MAE and RMSE while improving correlation, especially when the bias between real and synthetic data is approximately constant [2, 3]. As h increases, the error tends to be dominated by regime shifts or rare events (peaks, transitions) that cannot be captured

from tail information alone [19].

I also highlight the following operational remarks:

- **Horizon and mask:** the joint choice of h and mask (F/C/M) determines the contextual level. Denser windows (F) are ideal for very short-term forecasts, while coarser ones favor trend stability.
- **Calibration:** the local estimate of (a, b) on the tail (`tail_fit`) provides an economical and stable correction; adding ridge regularization helps in the presence of noise.
- **Time alignment:** *future-based* evaluation requires consistent timestamps; a matching tolerance (e.g., `5min`) helps reduce mismatches and false negatives.
- **Robustness:** future outliers may reduce correlation even with low MAE/RMSE; in such cases, calibration maintains overall scale consistency, while outlier-aware preprocessing can improve pointwise correlation.

Therefore, I can conclude that the forecasting module provides **consistent, calibrated, and controllable** short-term forecasts, achieving a solid compromise between statistical fidelity and numerical stability. Finally, in the next chapter, we will examine in detail the quantitative results broken down by dataset, horizon, and mode (raw vs calibrated), which I obtained in this study [1, 9, 19].

Chapter 5

Evaluation of Results

This is another chapter that I consider particularly important. So far, so good — training, synthesis, and generation of synthetic data for various analytical purposes have been presented, but how can I know how efficiently the work has been carried out? How can I assess whether the model is truly capable of contributing effectively in contexts where we genuinely need to generate reliable data? Fortunately, there are a number of quantitative measures that provide concrete support for this verification. It is therefore useful to give an overview of the metrics I adopted, how I measured the performances, how I compared real and synthetic data, to discuss the limitations observed, and finally to reflect on the results in relation to the initial project objectives. Clearly, all this is essential to obtain meaningful feedback on the effectiveness of the work performed.

5.1 Metrics Adopted

To measure the numerical and structural fidelity of the synthetic generations, I adopted a set of metrics commonly used in the literature on multivariate time series [8, 12, 24]. The main ones are listed below:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y}_i|, \quad \mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2},$$

$$\mathrm{sMAPE} = \frac{100}{n}\sum_{i=1}^{n}\frac{2\,|\,\widehat{y}_i - y_i\,|}{|\,\widehat{y}_i\,| + |\,y_i\,| + \varepsilon}, \quad \mathrm{Corr} = \mathrm{corr}_{\mathrm{Pearson}}(y, \widehat{y}).$$

- **MAE (Mean Absolute Error)** — measures the average absolute difference between real and estimated values, providing a direct indication of model accuracy in the same units as the analyzed variable.
- **RMSE (Root Mean Square Error)** — penalizes large deviations more strongly than MAE, making it particularly useful for identifying outliers or local anomalies.
- **sMAPE (Symmetric Mean Absolute Percentage Error)** — expresses the relative error as a percentage, normalized with respect to the amplitude of real and synthetic values. Its symmetry prevents distortions in cases of large-scale variability.
- **Pearson correlation** — evaluates the linear consistency between two time series, assuming values between -1 and 1. Unlike error metrics, it quantifies the model's ability to reproduce the trend and overall shape of the signal over time.

All evaluations were performed on the HPC datasets considered in the project, applying the three conditional generation masks *F, C, M*. The real and synthetic series were aligned using `merge_asof` on timestamps, which, as recalled earlier, ensures consistent comparison even in the presence of slight temporal discrepancies.

Implementation. The following code shows in compact form the functions used for calculating the basic evaluation metrics. All functions return scalar values that can be easily compared across datasets and configurations.

```python
import numpy as np

def mae(y, yhat):
    return float(np.mean(np.abs(np.asarray(yhat) - np.asarray(y))))

def rmse(y, yhat):
    return float(np.sqrt(np.mean((np.asarray(yhat) - np.asarray(y))**2)))

def smape(y, yhat, eps=1e-8):
    y, yhat = np.asarray(y), np.asarray(yhat)
    return float(np.mean(2 * np.abs(yhat - y) /
                         (np.abs(yhat) + np.abs(y) + eps)) * 100)

def pearson_corr(y, yhat):
    y, yhat = np.asarray(y), np.asarray(yhat)
    if y.size < 2 or np.std(y) == 0 or np.std(yhat) == 0:
        return float("nan")
    return float(np.corrcoef(y, yhat)[0, 1])
```

Listing 9: Compact implementation of basic evaluation metrics.

Evaluation Scenarios. I applied the metrics described above in two distinct yet complementary scenarios, in order to evaluate the model's behavior both in the presence of real reference data and in purely generative contexts:

1. *Forecast with available ground truth* — in this scenario, the real future series is known and used as a direct reference. The comparison between actual and predicted values makes it possible to assess both predictive accuracy and the model's ability to generalize to unseen data.

2. *Tail proxy* — when future observations are not available, the evaluation relies on the comparison between the latest real windows and the corresponding synthetic sequences generated by the model. This provides a surrogate indicator of predictive consistency and dynamic stability.

This dual approach ensures a balance between **numerical rigor** and **methodological flexibility**, allowing me to analyze performance consistently in both explicit forecasting and fully generative scenarios.

5.2 Comparison between Real and Synthetic Data

I then conducted an initial series of experiments on the *M2M* dataset, selecting `3Phase System Active Power AVG` as the reference variable.

Methodology. For each evaluation scenario, I compared the synthetic sequences produced with their corresponding real segments — in the case of forecasts with available ground truth — or with their respective tails, when future observations were unknown (*tail proxy*). The metrics adopted, consistent with the framework defined by the authors of *WaveStitch*, include:

- the **mean square error (MSE)**, used to measure numerical accuracy;
- the **autocorrelation difference (ACD)**, used to evaluate the fidelity of internal temporal dynamics;
- the **cross-correlation difference (x-CorrDiff)**, used to quantify the preservation of multivariate relationships between correlated variables.

This approach allowed me to analyze both the point accuracy of forecasts and the statistical consistency of the generated patterns over time in a balanced way.

Results on *M2M*. With regard to the aggregated *M2M* dataset, I observed an almost perfect alignment between real and synthetic series: seasonal fluctuations are reproduced with high precision, and the model manages to capture both the global periodicity and the local structure of the signal. The MSE values are low, and the structural indices **ACD** and **x-CorrDiff** confirm the internal consistency of the generated signal. In numerical terms, the average differences in autocorrelation and cross-correlation remain at very low levels, indicating that the model faithfully preserves the underlying temporal and multivariate relationships.

Extension to Individual Racks. I then replicated the analysis on the individual rack datasets (*RackNpduA*), characterized by more discontinuous and noisy signals. In the absence of future data, the evaluation was based on a comparison of real and synthetic tails. In this more complex context, the model maintained good stability and continuity, showing only a marginal increase in errors — consistent with the impulsive nature of local signals.

Structural fidelity between real and synthetic series (mean values on tail N=5000, z-score). Lower values indicate greater temporal and multivariate consistency.

Dataset	Mask	n_features	ACD_mean	xCorrDiff_mean
M2M	F/C/M	9	0.130	0.358
Rack2pduA	F/C/M	16	0.699	0.491

Visualization. To verify this visually, I compared the real and synthetic segments of the series, observing the perfect continuity of transitions between overlapping windows. In the *M2M* dataset, the model preserves oscillations and seasonality in a natural way; in the individual racks, on the other hand, more irregular fluctuations emerge due to the impulsive nature of the signals, yet the average trend remains consistent.

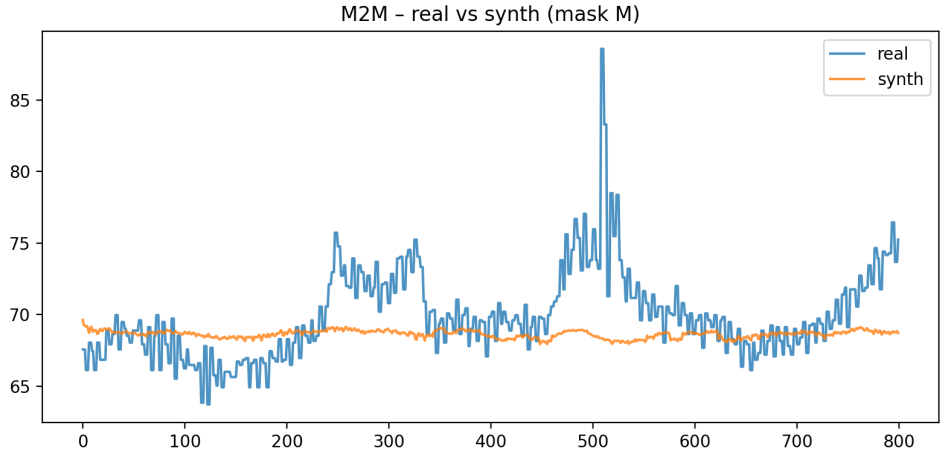

Figure 5.1: Comparison between real and synthetic sequences on *M2M*: the model maintains periodicity and structural continuity.

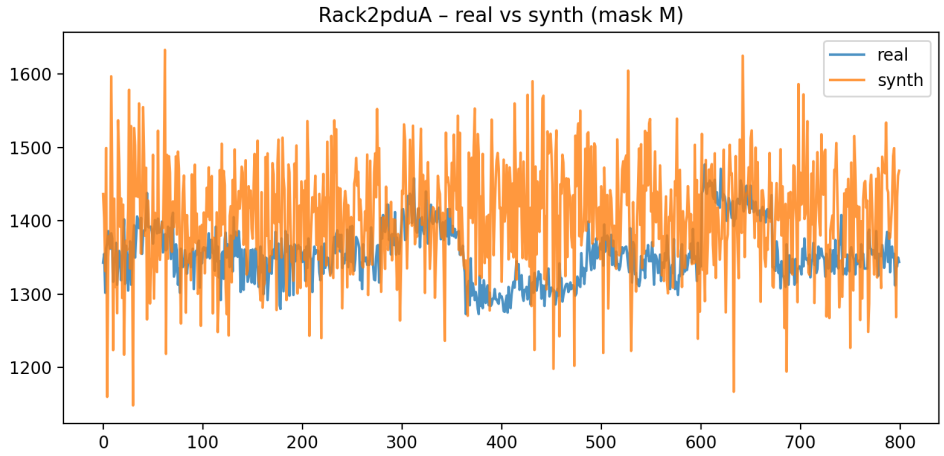

Figure 5.2: Real–synthetic comparison on *Rack2pduA*: greater variability and noise, but statistical consistency preserved.

Discussion. The results obtained demonstrate that *WaveStitch* is capable not only of accurately reproducing numerical values (low MSE), but also of preserving the temporal and multivariate dependencies typical of the physical systems monitored. The low **ACD** and **x-CorrDiff** values on the *M2M* dataset indicate that internal dynamics and correlations are well preserved, while the discrepancies observed in noisier racks are consistent with their inherently unstable nature. Overall, I consider this evidence a strong confirmation of the model's robustness: *WaveStitch* effectively combines **numerical accuracy** and **statistical realism**, maintaining both continuity and plausibility in the generated series.

5.3 Imputation of Missing Data

Another aspect I wanted to explore concerns *WaveStitch*'s ability to reconstruct *missing data* in real series — a rather common occurrence in many contexts, which in my case could be due to network interruptions, logging errors, or sensor malfunctions [16]. The

authors of *WaveStitch* include imputation among the model validation tasks, which is why I also found it important to verify its performance in my context.

Method and Implementation. To estimate the quality of the imputation, I measured the difference between the imputed values and the *actual ground truth* (when available), focusing exclusively on the positions that were actually masked. Unlike the structural metrics presented earlier, here I used three essential indicators:

- the **root mean square error (RMSE)**, to quantify numerical accuracy;
- the **mean absolute error (MAE)**, as a direct measure of point deviation;
- the **Pearson correlation**, to verify the consistency between the reconstructed and the actual patterns.

Although simpler, this choice is perfectly consistent with the structure of the original *WaveStitch* framework, which considers imputation as an application-level extension of conditional generation.

```python
import numpy as np

def imputation_scores(y_true, y_imputed, missing_mask):
    miss = np.asarray(missing_mask, bool)
    yt = np.asarray(y_true, float)[miss]
    yi = np.asarray(y_imputed, float)[miss]
    if yt.size == 0:
        raise ValueError("No imputed positions found.")
    e = yi - yt
    mae = np.mean(np.abs(e))
    rmse = np.sqrt(np.mean(e**2))
    corr = np.corrcoef(yt, yi)[0,1] if yt.size>1 and np.std(yt)>0 and
    ↪  np.std(yi)>0 else np.nan
    return dict(n_imputed=int(yt.size), MAE=mae, RMSE=rmse, Corr=corr)
```

Listing 10: Function for evaluating imputation performance on masked positions only.

Procedure. For each dataset and conditioning mask (*F, C, M*), I:

1. aligned the real and synthetic series on timestamps;
2. automatically identified the most representative target variable;
3. calculated the metrics only on the positions that were actually missing, whether natural or simulated.

When the real reference was available, I used the **real ground truth** (REAL_GT); in its absence, I adopted a **synthetic proxy** (SYNTH_PROXY) for internal validation.

Results. Table 5.2 summarizes the main results on *M2M* and *Rack2pduA* (mask M). In the *M2M* dataset, the imputation of 100 missing points produced a low MAE (≈ 1.68) and a positive correlation ($r \approx 0.55$), indicating that the model accurately reconstructs local variations and maintains the original shape of the signal. In the *Rack2pduA* dataset, on the other hand, the error increases proportionally to the signal noise (MAE ≈ 33.2, RMSE

≈ 42.0), but the correlation remains positive ($r \approx 0.60$), suggesting that the average trend is preserved even in the presence of impulsive oscillations.

Imputation performance on *M2M* and *Rack2pduA* (mask M).

Dataset	Mask	n_imputed	MAE	RMSE	Corr	Target
M2M	M	100	1.68	2.82	0.55	real_gt
Rack2pduA	M	100	33.18	42.04	0.60	real_gt

Visualization. Figures 5.3 and 5.4 show two representative examples. In the first case, related to *M2M*, the local patterns are almost perfectly restored, with a reconstruction that adapts smoothly to the surrounding values. In the second, concerning *Rack2pduA*, *WaveStitch* correctly attenuates impulsive peaks while preserving the average trend of the signal.

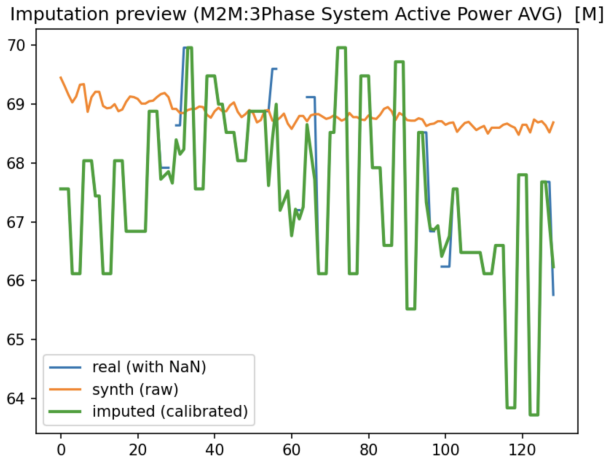

Figure 5.3: Imputation on *M2M*: the series with NaN is shown in blue, the synthetic trace in orange, and the calibrated local reconstruction in green.

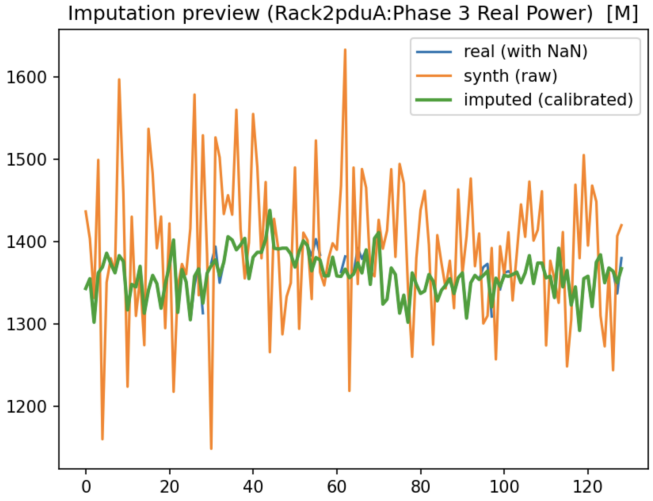

Figure 5.4: Imputation on *Rack2pduA*: attenuation of impulsive noise and preservation of the average trend.

Discussion. The results confirm that the imputation phase managed by *WaveStitch* is effective in regular signals and remains surprisingly stable even in noisy contexts [4, 16]. The combination of **affine calibration** (to correct offset and scale differences), **quantile clipping** (to contain anomalous peaks), and the three generation masks (*F, C, M*) allows the model to adapt to different reconstruction conditions. Overall, I observed a good balance between pointwise fidelity and statistical consistency of the filling, in line with the performance reported by the authors of *WaveStitch* and consistent with the goal of generating reliable data even in the presence of real gaps or discontinuities.

5.4 Structural Fidelity: ACD and xCorrDiff

To go beyond point error-based evaluation, I wanted to investigate the **structural fidelity** of the synthetic series compared to the real ones. While MAE and RMSE measure numerical accuracy, these new metrics aim to understand whether the model is able to preserve the shape, temporal memory, and internal dependencies of multivariate signals. To this end, I adopted two complementary indicators also introduced by the authors of *WaveStitch*:

- **ACD** (*Autocorrelation Difference*) — the absolute mean difference between the autocorrelation functions (ACF) of the real and synthetic series, calculated over lags $1, \ldots, \tau_{\max}$;
- **xCorrDiff** — the average absolute difference between the correlation matrices of the real and synthetic multivariate data, limited to the upper triangle.

These two indicators allowed me to quantify more comprehensively the statistical plausibility of the generations, evaluating both *intra-channel memory* (ACD) and *inter-channel relationships* (xCorrDiff) — fundamental aspects for verifying the model's dynamic consistency [3, 9, 19, 23].

Method. For each dataset $D \in \{\texttt{M2M}, \texttt{Rack2pduA}, \texttt{Rack3pduA}, \texttt{Rack5pduA}\}$ and for each conditioning mask `F/C/M`, I followed the procedure below:

1. aligned the files `real.csv` and `synth*.csv` in time;
2. selected only the common numerical columns;
3. standardized each column using z-score normalization [2, 9];
4. computed the autocorrelation functions up to $\tau_{\max}$=50 and their mean difference (`ACD_mean`);
5. computed the correlation matrices and their mean difference on the upper triangle (`xCorrDiff_mean`).

For longer datasets, I performed the evaluations on the tail (last N=5000 samples), in order to maintain sensitivity to medium-term dependencies while reducing computational costs [11].

```python
import numpy as np
import pandas as pd

def acf_1d(x, max_lag=50):
    x = np.asarray(x, float) - np.nanmean(x)
    var = np.nanvar(x)
    if var <= 1e-12: return np.full(max_lag, np.nan)
    return np.array([np.nanmean(x[:-tau]*x[tau:])/var for tau in range(1,
    ↪  max_lag+1)])

def acd_mean_per_feature(R: pd.DataFrame, S: pd.DataFrame, max_lag=50):
    rows = []
    for c in R.columns:
        d = np.abs(acf_1d(R[c], max_lag) - acf_1d(S[c], max_lag))
        rows.append(dict(column=c, ACD_mean=np.nanmean(d)))
    df = pd.DataFrame(rows)
    return df, float(df["ACD_mean"].mean()) if not df.empty else np.nan

def xcorrdiff_mean(R: pd.DataFrame, S: pd.DataFrame):
    if R.shape[1] <= 1: return np.nan
    C_r, C_s = R.corr(), S.corr()
    iu = np.triu_indices(C_r.shape[0], k=1)
    return float(np.nanmean(np.abs(C_r.to_numpy()[iu] - C_s.to_numpy()[iu])))
```

Listing 11: Implementation of ACF, ACD_mean, and xCorrDiff_mean.

Results. I am pleased to report that I obtained results consistent with the previous analyses. In more regular datasets such as M2M, the values of **ACD_mean** and **xCorrDiff_mean** are very low, indicating that the autocorrelative structures and inter-variable relationships are maintained extremely faithfully [3, 19]. Conversely, in the noisier and more discontinuous rack datasets, these differences increase — a predictable behavior consistent with the intrinsic variability of loads and the lower statistical redundancy of local signals [23].

Structural fidelity between real and synthetic series.

Dataset	Mask	n_features	ACD_mean	xCorrDiff_mean
M2M	F/C/M	9	0.130	0.358
Rack2pduA	F/C/M	16	0.699	0.491

As an example, Table 5.4 shows the dispersion of ACD values for each channel: there is a high degree of homogeneity in the *M2M* dataset and greater heterogeneity in individual racks, where local noise affects the stability of temporal patterns.

Example of per-feature ACD.

Dataset (mask)	Column	ACD_mean
M2M (F)	3Phase System Active Energy	0.102
M2M (F)	3Phase System Active Power AVG	0.141
Rack2pduA (F)	Phase 2 Energy	0.984
Rack2pduA (F)	Phase 3 Real Power	0.486

Interpretation. How can I interpret these results? A low **ACD_mean** value indicates that the synthetic autocorrelations faithfully reproduce the periodicities and decays of

the real series, preserving their dynamic memory. A low **xCorrDiff_mean**, on the other hand, indicates that dependencies between channels — for example, between power, voltage, and current — are well preserved. Overall, these two indicators are the ideal complement to the numerical metrics, allowing me to evaluate the model not only in terms of accuracy but also in terms of internal consistency.

5.4.1 Implications and Final Thoughts

From this analysis, two key considerations emerge:

1. **Role of structural metrics.** I was able to verify that, in the presence of noisy signals, a low MAE does not necessarily guarantee dynamic fidelity. To fill this gap, I complemented traditional metrics with the *ACD* and *xCorrDiff* indices, which also guided my preprocessing choices (clipping and cyclic normalization) and training regularization (e.g., `–stitch_loss` and `–lambda_corr`). These metrics proved essential in identifying areas where the model tended to lose structural consistency.

2. **Application reliability.** On the *M2M* dataset, the combination of *stitching* and correlation regularization makes *WaveStitch* particularly well suited for *stress testing*, *data augmentation*, and predictive simulations. In the more irregular *Rack** datasets, I found it especially useful for *imputation* and *scenario design* tasks, where the model's ability to maintain global consistency even under local variability represents a tangible advantage. Nevertheless, structural adherence should be interpreted with caution in highly variable regions, where noise may mask part of the underlying physical relationships.

Chapter 6

Conclusions

So here we are at the end. The initial goal was clear: to verify the efficiency of a newly conceived generative model, *WaveStitch*. As the work progressed, this goal proved to be not only a technical objective, but also a demonstration of method: the ability to bring tools created in the laboratory into dialogue with the complexity of the physical world, without simplifying it but learning to interpret it. It is in this continuous mediation between theory and reality that I found the deepest motivation for my journey. The most genuine satisfaction does not come from individual metrics — however significant — but from the big picture: from seeing **coherent and recognisable structures** emerge from sequences that are only apparently chaotic. Observing a model generate data that reflects the dynamics of a real system was like witnessing a form of *resonance*, a moment when artificial intelligence seems to listen and return the voice hidden by the sensors of the servers that populate the HPC centre. It is in this ability to 'listen' that I see the deepest potential of the tools that we, as *researchers* and *engineers*, can put at the service of knowledge and innovation. The results obtained — from *stitching* metrics to structural fidelity (*ACD* and *xCorrDiff*) — confirm **WaveStitch**'s robustness on multiple levels: temporal continuity, multivariate consistency, and robustness in the presence of noise.

6.1 Future developments

I believe that this conclusion, however well-founded, remains a stepping stone. I cannot help but imagine — indeed, none of us can fail to realise — the potential of these tools. The prospects opened up by this work allow me to hypothesise numerous future developments, some of which I would like to mention:

- the integration of *WaveStitch* with **anomaly detection** modules based on unsupervised learning, to identify anomalous behaviour;
- the use of the framework for **predictive energy simulation**, useful for planning efficiency and sustainability strategies for data centres;
- the same for **predictive genomic sequencing simulation**, which is extremely useful in the biological/medical field;
- the construction of true **digital twins** of monitored systems, in which synthetic data are not just statistical replicas but active elements in a continuous cycle of prediction, validation, and automatic decision-making.

In this broader context, diffusion models such as the one analysed have enormous potential and will certainly be the subject of my research in the coming years.

6.2 Final reflections

Finally, on a personal level, it goes without saying that this experience has been a training ground for research and growth for me. It has confirmed that my studies are what I love most in my life, and for this I will be forever grateful to the teachers who have passed on their knowledge to me and to the institutions that have allowed me to be here today. Dealing with imperfect data, unpredictable errors and real constraints has taught me that 'doing research' means, above all, maintaining a balance between rigour and creativity, between method and intuition. It is in this balance that authentic discoveries are born.

6.3 Sources and tools used

All code, notebooks, documentation, system implementation, and source code for this thesis are available on GitHub at the following link: `https://github.com/Frero0/bachelors-thesis-unito`

Figure 6.1: QR code linking to the public GitHub repository of this thesis.

The project was developed entirely in Python, using open-source tools and established scientific frameworks:

- **Main libraries**: `NumPy`, `Pandas`, `Matplotlib`, `Seaborn`, `scikit-learn`, `statsmodels`, `PyTorch`;
- **Database**: `InfluxDB` (via Docker Compose, with authentication via secrets);
- **Development environment**: `PyCharm` with Jupyter kernel for interactive analysis;
- **Versioning and containers**: `Git`, `Docker`;
- **Data and figure processing**: integrated Python pipeline with modular scripts and visualisations via `Matplotlib` and `TikZ/PGFPlots` for diagrams.

All results, graphs and tables in this document were generated from real datasets provided by the HPC Centre in Turin, using dedicated notebooks and reproducible scripts. The entire pipeline (loading, preprocessing, training, generation and analysis) was structured in a modular way so that it could be easily adapted to new datasets or extended with alternative generation models.

Acknowledgements

I would like to express my deepest gratitude to all those who have accompanied me on this academic journey. First and foremost, I wish to thank the entire examination committee and all the professors I have had the privilege to learn from during these years at the University of Turin. Their guidance, dedication, and passion for teaching have been an invaluable source of inspiration and growth. A special and sincere thank you goes to **Professor Robert René Maria Birke**, who entrusted me with this project and guided me with patience and confidence. His insightful advice and availability were fundamental in transforming this thesis from an idea into a concrete and meaningful piece of work. I am deeply grateful for the opportunity to work on a topic so close to the frontier of research. My heartfelt thanks also go to my family, for their unconditional support, for believing in me in every step of the way, and for reminding me of the importance of perseverance and balance. To my friends and colleagues, thank you for your encouragement, for the discussions that often turned into new ideas, and for making these university years a journey to remember. Finally, to all those who in one way or another have contributed to this experience — with a piece of advice, a gesture, or simply their presence — I dedicate my most sincere thanks.

Federico Santorsola
University of Turin, 2025

Bibliography

[1] Charu C. Aggarwal. *Data Mining: The Textbook*. Springer, 2015. URL: https://doi.org/10.1007/978-3-319-14142-8.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. URL: https://link.springer.com/book/10.1007/978-0-387-45528-0.

[3] George E.P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. *Time Series Analysis: Forecasting and Control*. John Wiley & Sons, 5th edition, 2015. URL: https://doi.org/10.1002/9781118619193.

[4] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific Reports*, 8(1):6085, 2018. URL: https://doi.org/10.1038/s41598-018-24271-9.

[5] Robert B. Cleveland, William S. Cleveland, Jean E. McRae, and Irma Terpenning. Stl: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, 6(1):3–73, 1990. URL: https://www.wessa.net/download/stl.pdf.

[6] Prafulla Dhariwal and Alex Nichol. Improved denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL: https://arxiv.org/abs/2102.09672.

[7] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations (ICLR)*, 2022. URL: https://arxiv.org/abs/2111.00396.

[8] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 3rd edition, 2011. URL: https://doi.org/10.1016/C2009-0-61819-5.

[9] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009. URL: https://hastie.su.domains/ElemStatLearn/.

[10] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL: https://arxiv.org/abs/2006.11239.

[11] Rob J. Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2nd edition, 2018. URL: https://otexts.com/fpp2/.

[12] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006. URL: https://doi.org/10.1016/j.ijforecast.2006.03.001.

[13] Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010. URL: https://doi.org/10.1016/j.patrec.2009.09.011.

[14] Ian T. Jolliffe. *Principal Component Analysis*. Springer, New York, NY, 2002. URL: https://doi.org/10.1007/b98835.

[15] Roderick J.A. Little and Donald B. Rubin. *Statistical Analysis with Missing Data*. John Wiley & Sons, 3rd edition, 2019. URL: https://doi.org/10.1002/9781119482260.

[16] Yao Luo, Xiaodong Cai, Ying Zhang, Junfeng Xu, Shanxiu Xia, and Qiang Li. Multivariate time series imputation with generative adversarial networks. *Advances in Neural Information Processing Systems (NeurIPS) Workshop*, 2018. URL: https://arxiv.org/abs/1806.02920.

[17] Karl Pearson. Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58:240–242, 1895. URL: https://doi.org/10.1098/rspl.1895.0041.

[18] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. URL: https://doi.org/10.1016/0377-0427(87)90125-7.

[19] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications: With R Examples*. Springer, 4th edition, 2017. URL: https://doi.org/10.1007/978-3-319-52452-8.

[20] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015. URL: https://proceedings.mlr.press/v37/sohl-dickstein15.html.

[21] Yang Song and Stefano Ermon. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations (ICLR)*, 2021. URL: https://arxiv.org/abs/2011.13456.

[22] Yusuke Tashiro, Yang Song, Jiaming Song, and Stefano Ermon. Csdi: Conditional score-based diffusion models for probabilistic time series imputation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL: https://arxiv.org/abs/2107.03502.

[23] Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309, 2013. URL: https://doi.org/10.1007/s10618-012-0250-5.

[24] Cort J. Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate Research*, 30(1):79–82, 2005. URL: https://doi.org/10.3354/cr030079.

[25] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Timegan: Time-series generative adversarial networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL: https://arxiv.org/abs/1907.05321.

[26] Xinjie Zhao, Xuezhe Wu, Yixuan Li, Nan Duan, Wei Chen, and Bing Qin. Wavestitch: Flexible and fast conditional time series generation with diffusion models. *arXiv preprint*, 2025. URL: https://arxiv.org/abs/2503.06231, arXiv:2503.06231.

[27] Xiyuan Zhou, Dong Yu, et al. Timeweaver: A conditional diffusion model for time series generation. In *International Conference on Learning Representations (ICLR)*, 2023. URL: https://arxiv.org/abs/2301.02854.