MASTER IN COMPUTER SCIENCE

Automated Model Selection for Time Series Data Generation

Master Thesis

Matej Kutirov

Dr. Mourad Khayati

Prof. Dr. Philippe Cudré-Mauroux

University of Bern

October 2025

UNIVERSITÄT BERN

UNIVERSITÉ DE NEUCHÂTEL

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Acknowledgments

I would like to express my deepest gratitude to my distinguished supervisor of the thesis, Dr. Mourad Khayati, for his invaluable guidance, constructive feedback, and continuous encouragement throughout the course of this work. His expertise and insightful advice have been essential to the successful completion of this work. I am equally grateful to the distinguished Prof. Dr. Philippe Cudré-Mauroux, for providing a stimulating academic environment and for supporting my research endeavors within the group.

Furthermore, I owe my deepest appreciation to my family and my partner. Their unwavering love, patience, and encouragement have been a constant source of strength throughout this journey. To my family, thank you for instilling in me the values of perseverance and curiosity that made this work possible. To my partner, your support, understanding, and belief in me have carried me through the most challenging moments and made the accomplishments along the way all the more meaningful.

Abstract

Data that captures temporal dynamics is indispensable for understanding and forecasting real-world phenomena. Time series emerge across diverse domains, ranging from financial markets and meteorology to healthcare monitoring and energy management. However, obtaining sufficient volumes of high-quality data remains a formidable challenge, and rare or intricate patterns are seldom represented in existing datasets. Synthetic time series generation addresses this limitation by producing realistic artificial data that preserves the underlying patterns and dependencies of real sequences. Such data facilitates downstream tasks, enables the study of rare events, and supports more informed decision-making when authentic data is scarce.

This thesis introduces a novel methodology that integrates time series generation with automated machine learning (AutoML) to identify the most suitable generative models for a given dataset. The approach is grounded in two complementary aspects. First, it benchmarks a wide variety of existing data generation models, including statistical models, variational autoencoders (VAE), generative adversarial networks (GAN), and transformers. Second, it applies AutoML-based model selection to determine the best-performing model for each dataset. The selected datasets span four distinct domains, Financial, Medical, Air, and Energy, each exhibiting unique intrinsic characteristics such as seasonality, trend, stationarity, transitions, and structural shifts.

The evaluation was carried out using twelve metrics to measure predictive accuracy, statistical similarity, temporal consistency, and computational efficiency. The findings demonstrate that deep generative models, particularly TimeGAN, achieve the lowest predictive error on highly dynamic datasets such as Medical, whereas TimeVAE and TimeTransformer are more effective at preserving distributional properties and latent structures. Classical statistical approaches remain competitive on smoother datasets, such as Financial and Air, due to their robustness and computational efficiency.

An AutoML framework was further applied to automate model selection across varying time budgets. The AutoML procedure proved to be an order of magnitude (10x) more efficient than manual selection, although with a 45.5% accuracy decrease compared to manual selection.

The principal contributions of this work include the development of a reproducible AutoML-driven pipeline for time series generation, a comprehensive experimental evaluation across multiple datasets and metrics, and open resources including code, results, and guidelines to support reproducibility and practical application. The code is publicly available at: `https://github.com/matejkutirov/AutoTSG`.

Contents

1 Introduction

Time has fascinated researchers for millennia, yet even today, deciphering its underlying dynamics remains one of science's most enduring challenges. Many of these dynamics are captured as sequences of observations collected over time, commonly known as time series. Each observation in a time series tells a part of a story, capturing the influence of past events while offering hints about what the future may hold. As a matter of fact, they form patterns that can sometimes be obvious or hard to notice, with insights hidden within the flow of time.

To fully appreciate their complexity, it is essential to highlight the defining properties of time series data. Unlike non-temporal datasets, time series display dependencies across time, with patterns such as trends, seasonality, transitions, and shocks shaping their behavior. Moreover, many series are nonstationary, with changing means or variances, and may include irregular measurements or missing values. These characteristics complicate analysis but also provide the foundation for generating synthetic sequences that capture actual dynamics.

Time series can take diverse forms, ranging from daily stock prices and hourly weather readings to yearly population statistics or minute-by-minute heart rate data. Their behaviors often exhibit trends, seasonal fluctuations, or sudden irregular shocks, revealing both predictable cycles and unexpected surprises. Researchers and practitioners analyze these patterns to forecast future outcomes, detect anomalies, and guide decision-making in areas as diverse as finance, healthcare, energy, and climate science. Regardless of the wealth of potential insights, the availability of sufficient, high-quality time series data often limits how accurately we can uncover and leverage these temporal stories.

In many fields, there is simply not enough data available in order to extract reliable insights. In healthcare, patient records are often restricted by privacy regulations, making it difficult to study detailed disease progression or rare conditions. Similarly, in finance, sensitive transaction data cannot always be shared, and extreme market events, so-called "black swan" events [44], occur so infrequently that there are few historical examples to learn from. Even in scientific research, infrequent phenomena, such as anomalous weather events or uncommon genetic disorders, offer limited observations, leaving gaps in our understanding.

In addition to scarcity, strict regulations such as HIPAA [36] and GDPR [38] further restrict access to sensitive data, particularly in healthcare. In like manner, synthetic time series generation (TSG) offers not only a way to overcome scarcity but also a means to preserve privacy and enable ethically sound

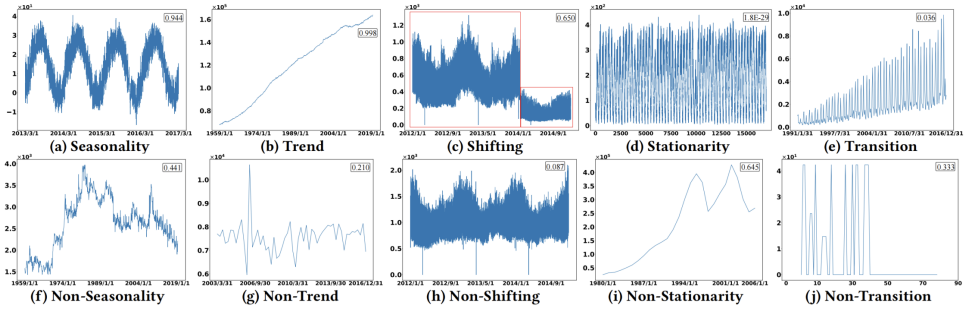

Figure 1.1: Visualization of data with different characteristics. [40]

experimentation. For example, synthetic patient trajectories can simulate disease progression, treatment strategies, or rare conditions that would be impractical to study directly. Consequently, these constraints pose significant challenges for traditional analytical methods, which rely on large volumes of data to identify meaningful patterns. As a result, researchers have turned to innovative strategies that can expand or simulate datasets, creating opportunities to overcome scarcity and explore new insights.

This leads us to data augmentation, a set of techniques designed to expand the size and diversity of datasets without the need for additional data collection. It is important to distinguish time series forecasting from generation: whereas forecasting extrapolates future values of an existing sequence, generation produces entirely new sequences that mimic the statistical and temporal structure of real data. This distinction underpins the role of data augmentation, where synthetic sequences are used to enrich existing datasets and address the challenges of scarcity. Additionally, adding controlled noise, scaling or shifting data, or using advanced algorithms such as generative models to simulate realistic sequences. By increasing the variety of training examples, data augmentation helps models generalize better, improves robustness, and enables the study of rare events that are otherwise underrepresented. As the field has advanced, researchers have developed numerous approaches and established benchmarks to systematically evaluate their effectiveness.

Such benchmarks and techniques are now central to both generating and evaluating synthetic time series. Methods range from simple statistical models and simulation-based approaches to state-of-the-art machine learning models like RNNs, GANs, and transformers [43, 46, 52]. Some approaches even integrate domain expertise with machine learning models to capture complex patterns while retaining interpretability. Equally important is the evaluation of synthetic sequences, which relies on metrics that assess realism, diversity, and utility. Frameworks such as TSGBench [2] provide a structured basis for this assessment. By coupling generation with systematic evaluation, it is possible to produce high-quality synthetic datasets that effectively compensate for the limitations of the real data.

However, a critical challenge lies in determining how we can automatically find the best model for a given time series dataset. Automated Machine Learning (AutoML) [18] offers a solution by systematically exploring model architectures, hyperparameters, and training strategies to identify high-performing solutions with minimal human intervention. When applied to time series, it can quickly discover which techniques work for different types of data, adapting to patterns and complexities that might be hard to spot manually. This not only accelerates the modeling process but also reduces reliance on manual trial-and-error, making model selection more consistent and scalable. The diversity of time series behaviors across domains further complicates this task: financial series often contain abrupt shocks, medical signals combine cycles with noise, and industrial sensor data may involve multivariate dependencies. Consequently, no single generative approach is universally optimal. This motivates the use of automated strategies to adapt model choice to the characteristics of each dataset.

Inspired by research on time series generation and AutoML, this thesis proposes a novel approach that unifies both paradigms. It conducts experiments and provides results, code, and analysis to support reproducibility. The overarching objective is to automate the identification of the most suitable generative models, thereby accelerating discovery while ensuring consistency across diverse datasets. The principal contributions of this work are:

- Framework development: Designing and implementing an AutoML-driven pipeline for time series generation that systematically explores and evaluates multiple generative models.
- Experimental evaluation: Conducting comprehensive experiments to assess model performance, generation quality, and practical utility across heterogeneous datasets.
- Open resources and guidelines: Releasing code, results, and methodological recommendations to promote reproducibility and enable practitioners to apply automated generation techniques efficiently.

The structure of this thesis is organized as follows:

- Chapter 2 reviews related work, outlining key benchmarks in time series generation. It identifies existing challenges and positions the present study within the broader research landscape.
- Chapter 3 examines methods for time series generation, covering transformation-based techniques, classical statistical models such as ARIMA and Exponential Smoothing, and modern deep learning approaches including GANs, VAEs, Fourier Flows, Diffusions, and Transformers. The discussion emphasizes their applicability, strengths, and limitations in generating synthetic time series.
- Chapter 4 introduces automated model selection, explaining the principles of AutoML and the workings of FLAML. It describes how these tools can be leveraged to automate and optimize the selection of generative models for time series data.
- Chapter 5 presents the experimental design, detailing the implemented pipeline, datasets, evaluation metrics, and results. It analyzes model performance and discusses observed trends, practical implications, and directions for future research.
- Chapter 6 concludes the thesis by summarizing the main findings and contributions, reflecting on their significance, and suggesting potential extensions of the proposed approach.

2
Related Work

Several benchmarks have been developed to systematically evaluate time series generation models. They aim to address challenges such as dataset heterogeneity, inconsistent evaluation protocols, and limited reproducibility. Among the most influential efforts in recent years are TSGBench [2], CTBench [3], Human Motion Generation Benchmark [22] and TS Generation Benchmark [16].

TSGBench [2] constitutes the first dedicated benchmark for synthetic time series generation, explicitly addressing the lack of unified and comprehensive evaluation standards in this domain. It introduces three tightly integrated modules: (i) a curated collection of publicly accessible, real-world datasets together with a standardized preprocessing pipeline, (ii) a broad suite of evaluation measures encompassing traditional statistical metrics, novel distance-based assessments, and visualization tools, and (iii) a pioneering generalization test grounded in domain adaptation, applicable to all model families. TSGBench evaluates a wide spectrum of state-of-the-art generative models across ten datasets and twelve metrics, thereby providing nuanced insights into the trade-offs between statistical fidelity, predictive utility, and cross-domain generalization. Crucially, it contributes statistical analyses of performance rankings that highlight variability in model behavior across datasets. Despite its depth, TSGBench presents a significant limitation: while the framework implements the evaluation routines, it does not open-source the code of the evaluated models, requiring users to integrate and adapt their implementations manually.

CTBench [3] introduces a domain-specific benchmark for cryptocurrency time series generation, a field uniquely characterized by extreme volatility, non-stationarity, and continuous trading activity. CTBench compiles data from 452 tokens and organizes evaluations across thirteen metrics grouped into five dimensions: statistical fidelity, predictive utility, rank preservation, trading performance, and computational efficiency. A major innovation lies in its dual-task evaluation framework. The Predictive Utility task assesses whether synthetic data can effectively preserve patterns for forecasting, while the Statistical Arbitrage task evaluates whether generated series maintain signals relevant to trading strategies, such as mean reversion. By benchmarking eight representative models across multiple methodological families and four distinct market regimes, CTBench provides actionable insights for practitioners in quantitative finance. Nevertheless, its limitations are notable: the benchmark is restricted to cryptocurrency markets, lacks publicly available code for seamless adoption, and its highly specialized nature reduces generalizability to broader domains of time series research outside finance.

Establishing a Unified Evaluation Framework for Human Motion Generation: A Comparative Analysis

of Metrics [22] introduces a comprehensive benchmark aimed at standardizing the evaluation of generative models for temporal data, with a specific focus on human motion generation. The framework reviews and unifies eight quantitative metrics to assess both fidelity and diversity of generated sequences, and proposes a novel measure—Warping Path Diversity (WPD)—to capture differences in temporal distortions such as frequency shifts and time warping. By providing a consistent evaluation setup, publicly available code, and clear methodological guidelines, this benchmark advances reproducibility and comparability across studies. However, its experimental scope is limited to models from the Variational Autoencoder (VAE) family, omitting other influential approaches such as Generative Adversarial Networks (GANs), Diffusion Models, Transformer-based architectures, and Fourier Flow models. Consequently, while it establishes a solid foundation for unified evaluation, its findings remain constrained to a narrow subset of generative methods. The authors acknowledge these limitations and suggest extending the framework to encompass additional model families and diverse datasets, thereby enhancing its generality and applicability to broader time series generation tasks.

Comparison of Synthetic Time Series Data Generation Techniques [16] provides a complementary study that empirically evaluates six augmentation methods: Anomalies Injection, Dynamic Time Warping Barycentric Averaging (DBA), autoregressive models, GAN, InfoGAN, and TimeGAN. These methods are tested on datasets with varying characteristics, such as pronounced cyclical patterns and the presence of anomalies. The work also introduces a tool for automating evaluation, parameterizing techniques, and visualizing generated series. Results indicate that machine learning–based approaches, particularly GAN variants, are often the most effective; however, simpler methods such as DBA, anomaly injection, and autoregressive models can still be competitive under certain conditions. Despite these contributions, the study has notable limitations: it is primarily centered on GAN-based approaches, leaving out other prominent model families such as Variational Autoencoders (VAEs) and Transformer-based architectures. In addition, the evaluation lacks widely used qualitative techniques such as t-SNE visualizations, which are valuable for intuitively assessing the similarity between real and synthetic data distributions.

3
Time Series Generation

This chapter examines various methodologies for generating synthetic time series, categorized into three principal classes: Transformation-Based, Classical Statistical, and Deep Learning techniques. To ensure a balanced comparison, at least one representative method was selected from the latter two categories. Statistical models provide interpretable and theoretically grounded baselines, while deep learning methods, including Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and Transformers, represent the forefront of generative modeling. The empirical evaluation applies datasets from finance, medicine, air quality, and energy, deliberately chosen for the heterogeneous characteristics - seasonality, trend, non-stationarity, abrupt transitions, and structural shifts.

3.1 Transformation-based Techniques

Transformation-based augmentations are methods that modify existing time series data to create useful variations. While they do not generate entirely new sequences, they are extremely valuable because they expand the training set with additional examples that capture more diversity. This is especially important for rare or underrepresented patterns, which models might otherwise overlook. By introducing these controlled variations, augmentations improve the robustness of learning, act as a form of regularization, and ultimately strengthen the model's ability to generalize to unseen data. These augmentation strategies encompass techniques situated in two principal analytical domains, namely the time domain and the frequency domain.

3.1.1 Time-Domain

- **Jittering (Noise Injection)** is a method that adds small random noise, usually Gaussian or uniform, to the original data. It is commonly used to make models more robust to measurement errors, sensor inaccuracies, or natural fluctuations that occur in real-world data. For instance, it can be useful in physiological signals where small variations are common. The limitation, however, is that too much noise can distort the true signal and obscure important patterns, reducing the quality of the training data.

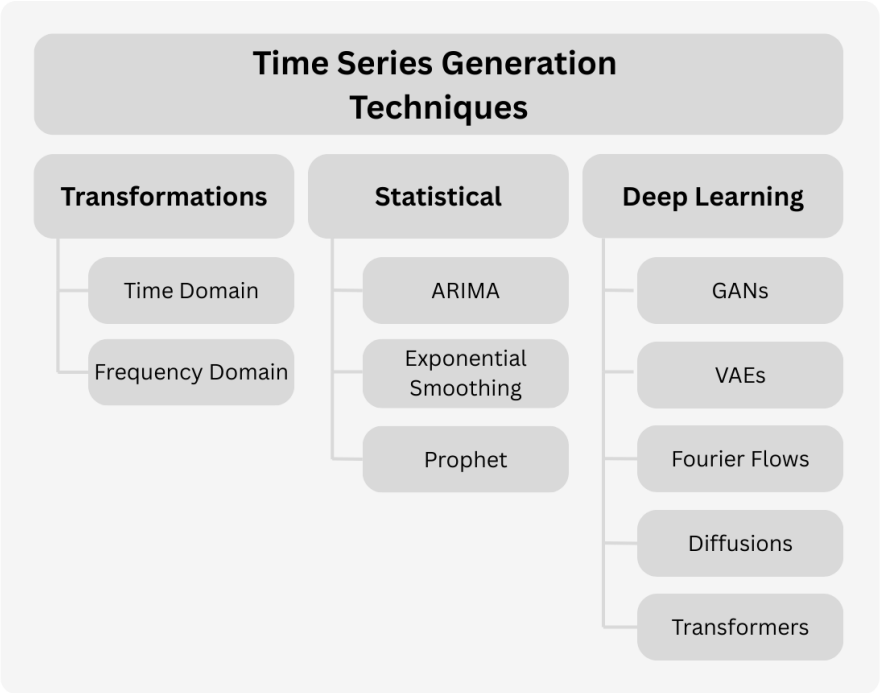

Figure 3.1: Taxonomy of Time Series Generation Techniques

- **Scaling** multiplies the whole time series by a random factor, changing the amplitude but keeping the overall structure intact. This is particularly helpful in domains like energy consumption or financial data, where values can fluctuate in magnitude depending on external conditions. Its drawback is that it may introduce unrealistic ranges if the scaling factors are not chosen carefully, which can mislead the learning model.
- **Permutation** divides the time series into subsequences, which are then shuffled. It encourages the model to learn local patterns without being tied strictly to the global order. This can be beneficial in activities like sensor-based human motion analysis, where the same motions might appear in different orders. The limitation is clear: shuffling can disrupt temporal dependencies, making the data less realistic for tasks where order is critical.
- **Time warping** stretches or compresses certain sections of the series along the time axis, simulating changes in speed. For example, in speech or gait analysis, the same sequence might occur faster or slower, and time warping prepares the model for such variations. Its limitation lies in the risk of creating unrealistic distortions if applied too aggressively, which can confuse rather than help the model.
- **Window Slicing** crops a random subsequence and rescales it back to the original length. This is especially useful when datasets are long and contain redundant segments, as it teaches the model to generalize from smaller parts. However, slicing may remove essential global patterns, such as long-term seasonality or trend, and thus may not be suitable for all applications.
- **Window Warping** selects a window and stretches or compresses it while leaving the rest unchanged. It allows local timing variations while preserving the overall order of events. It works well in sensor

data or gesture recognition, but the main drawback is that overuse can create unrealistic distortions in timing that do not naturally occur in the original system.

- **Time Shifting** moves the entire series forward or backward, simulating delays or phase differences. This is useful in domains like industrial monitoring or finance, where events may occur with a delay. The limitation is that it does not create fundamentally new patterns, only displaced versions of the existing data.
- **Trend Injection** adds synthetic trends to the data, such as gradual increases or decreases over time. This helps models learn to recognize and adapt to long-term drifts, which are common in domains like climate change or economic markets. However, if the injected trends are unrealistic or too strong, they may bias the model and lead to false generalizations.

Again, this set of techniques does not contribute to generating brand new synthetic data but to data augmentation as a precaution against overfitting the model when the datasets are smaller.

3.1.2 Frequency-Domain

- **Fourier Perturbation** modifies the coefficients of the Fast Fourier Transform (FFT) and then converts the series back to the time domain. It is useful for introducing subtle variations in the underlying frequencies of the signal, helping models learn to recognize patterns that may shift slightly over time. The limitation is that large perturbations can create unrealistic signals that no longer reflect the original behavior.
- **Filtering** applies low-pass, high-pass, or band-pass filters to emphasize or suppress certain frequency components. This is particularly effective in sensor data or biomedical signals, where noise removal or focus on specific frequency bands improves model performance. The main drawback is that filtering may remove meaningful information if not carefully tuned, potentially altering key patterns in the series.
- **Spectrogram Augmentations** [37] hides or masks parts of the signal in either the time or frequency domain, similar to creating small gaps in the data. It is often used in audio processing to help models learn from incomplete or partially hidden signals, which makes them more robust. The main limitation is that if too much of the signal is masked, important patterns can be lost, making the augmented data less realistic and less useful for training.

3.2 Classical Statistical Techniques

Classical statistical models rely on mathematical formulas to capture patterns in time series data. Fortunately, these models are easily explainable, computationally efficient, and well-suited for datasets with stable and predictable patterns. Although they may struggle with highly complex or nonlinear dynamics, they provide a strong foundation for time series generation and serve as a baseline for comparing more advanced methods. Within this class of approaches, two of the most prominent representatives are ARIMA and Exponential Smoothing, which exemplify the versatility and enduring relevance of classical statistical models in time series analysis.

3.2.1 ARIMA

ARIMA [5, 50], which stands for AutoRegressive Integrated Moving Average, is a widely used statistical model for analyzing and forecasting time series data. It combines three components:

- the autoregressive part - predicts values based on past observations;
- the integrated part - removes trends to make the series stationary;
- the moving average part - accounts for past forecast errors.

ARIMA is especially effective for datasets with consistent patterns and linear relationships, such as economic indicators, sales figures, or sensor readings that follow stable trends. Again, limitations exist everywhere, and ARIMA is not excluded either. It struggles with highly nonlinear or irregular data and requires careful tuning of its parameters to achieve good performance. Despite these challenges, it remains a reliable choice for generating synthetic sequences in situations where interpretability and stability are important.

3.2.2 Exponential Smoothing

Exponential smoothing [51] is another classical statistical method used for forecasting and generating time series. The idea behind it is simple: recent observations are given more weight than older ones, allowing the model to adapt to short-term changes while still capturing long-term patterns. Variants of exponential smoothing can model different types of series, from those with steady levels to those with clear trends and seasonal cycles. This makes it useful in areas such as demand forecasting, weather prediction, or monitoring energy consumption. However, the method assumes that future behavior can be explained by past patterns in a relatively smooth way, which limits its ability to handle sudden shifts, irregular fluctuations, or strongly nonlinear dynamics. Still, because it is easy to implement, computationally efficient, and interpretable, exponential smoothing continues to be a valuable tool in many practical applications.

3.3 Deep Learning Techniques

Deep learning methods bring a different perspective to time series generation by relying on layered neural networks capable of learning complex, non-linear relationships directly from the data. In the following sections, we explore different families of deep learning methods that have been developed for this purpose. These families encompass a diverse range of architectures and generative paradigms, including GANs, VAEs, Fourier Flows, Diffusion Models, and Transformers, which are covered in the following sections.

3.3.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) have become one of the most widely explored approaches for time series generation. At their core (Figure 3.2), GANs consist of two competing models: a generator that produces synthetic sequences and a discriminator that tries to distinguish real data from generated samples. Through this adversarial process, the generator learns to produce increasingly realistic outputs. While initially designed for images, GANs have been adapted to capture the sequential dependencies and temporal dynamics of time series, leading to several notable frameworks.

TimeGAN [52] combines adversarial training with supervised sequence modeling by jointly learning an autoencoding representation and a generative process in latent space. The architecture (Figure 3.3) comprises four modules: (i) an *embedding function* that maps real sequences into latent codes via recurrent networks, (ii) a *recovery function* that reconstructs input features from latent codes, (iii) a *generator* that produces synthetic latent trajectories from random noise, and (iv) a *discriminator* that distinguishes real from synthetic latent sequences. Training is governed by three complementary objectives. First, a reconstruction loss L_R ensures that the embedding–recovery pair preserves sufficient information for faithful sequence reconstruction. Second, a standard adversarial loss L_U encourages the generator to

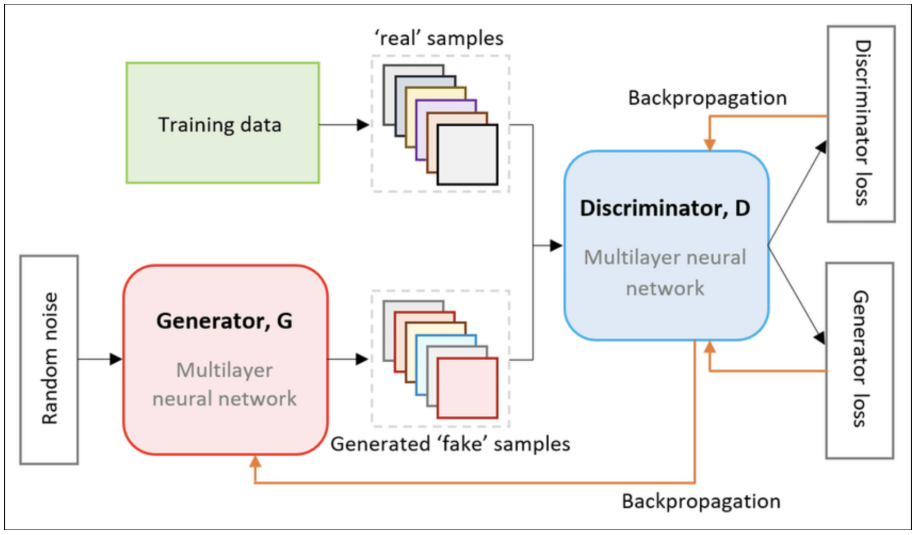

Figure 3.2: GAN Architecture. Figure taken from [30].

produce latent sequences indistinguishable from real ones. Third, a supervised stepwise loss L_S aligns the generator's conditional transitions with those of the embedding network, ensuring that synthetic sequences respect the temporal dynamics of the data. The overall optimization alternates between (i) minimizing $(L_R + \lambda L_S)$ for the embedding–recovery networks, and (ii) solving a minimax game between the generator and discriminator with an additional regularization ηL_S:

$$\min_{\theta_g} \left(\eta L_S + \max_{\theta_d} L_U \right) \tag{3.1}$$

3.3.2 Variational Autoencoders

In contrast to GANs, which rely on a competitive training process, Variational Autoencoders (VAEs) work by learning a compressed representation of time series data in a latent space, from which new sequences can be generated. Therefore, VAEs are especially powerful for capturing uncertainty and generating diverse samples, while also providing an interpretable structure that connects the latent space to real-world dynamics. Because of these qualities, they have been widely adapted to time series generation, with different extensions designed to better model temporal dependencies and complex multivariate patterns.

TimeVAE [11] extends the VAE framework to time-series generation by designing encoder–decoder architectures that both reconstruct input sequences and incorporate interpretable temporal structures (Figure 3.6). At its core, TimeVAE assumes a latent generative process where a sequence $x_{1:T}$ is produced via a latent variable z sampled from a prior distribution $p_\theta(z)$, followed by a conditional likelihood $p_\theta(x|z)$. Since the true posterior $p_\theta(z|x)$ is generally intractable, it is approximated with a variational distribution $q_\phi(z|x)$, parameterized by the encoder. The encoder maps input sequences $X \in \mathbb{R}^{N \times T \times D}$ into the parameters (mean and variance) of a multivariate Gaussian distribution in latent space, from which z is sampled using the reparameterization trick to ensure differentiability. The decoder then reconstructs sequences $\tilde{X}$ by transforming z through fully connected and transposed convolutional layers, restoring the original temporal structure. The dimensionality m of the latent space is a central hyperparameter, balancing representational capacity against overfitting. Training is based on the evidence lower bound (ELBO),

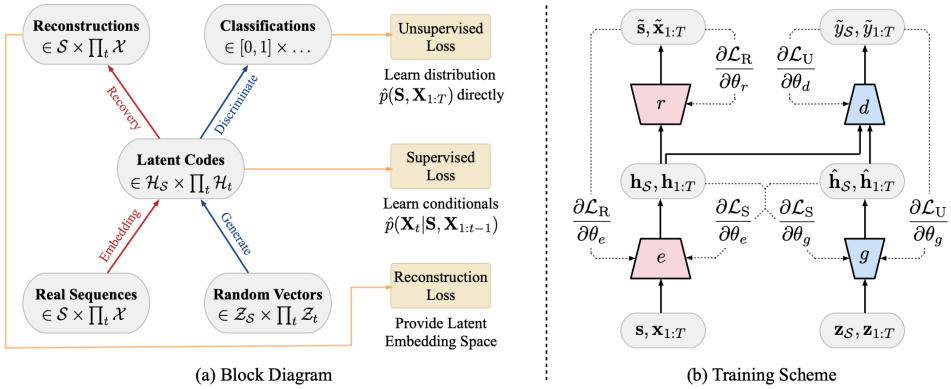

Figure 3.3: (a) TimeGAN Block diagram of component functions and objectives. (b) TimeGAN Training scheme; solid lines indicate forward propagation of data, and dashed lines indicate backpropagation of gradients. Figure taken from [52].

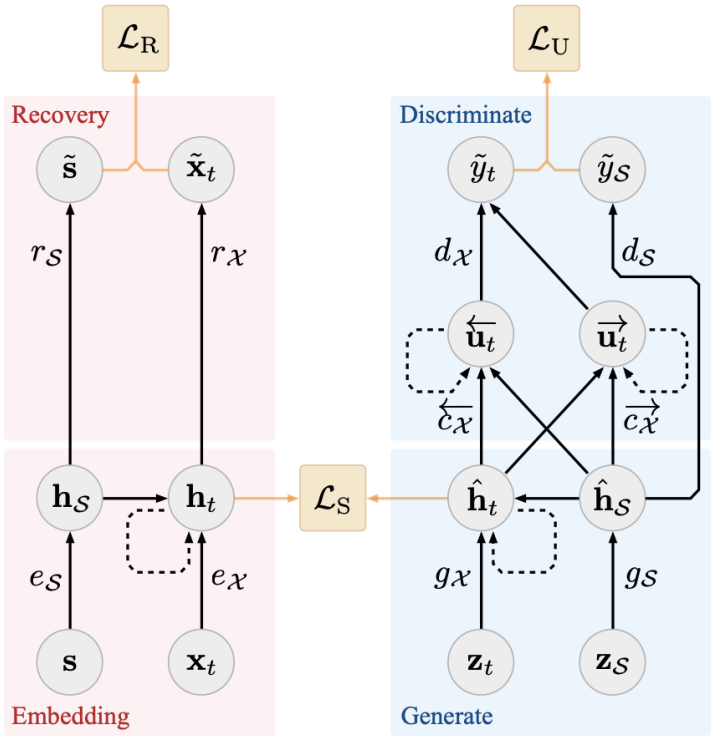

Figure 3.4: TimeGAN instantiated with RNNs. Solid lines denote function application, dashed lines denote recurrence, and orange lines indicate loss computation. Figure taken from [52].

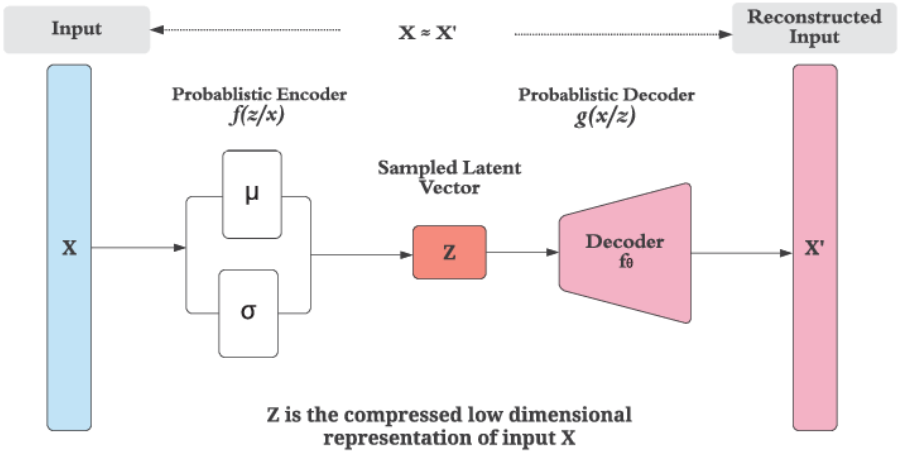

Figure 3.5: VAE Architecture. Figure taken from [25]

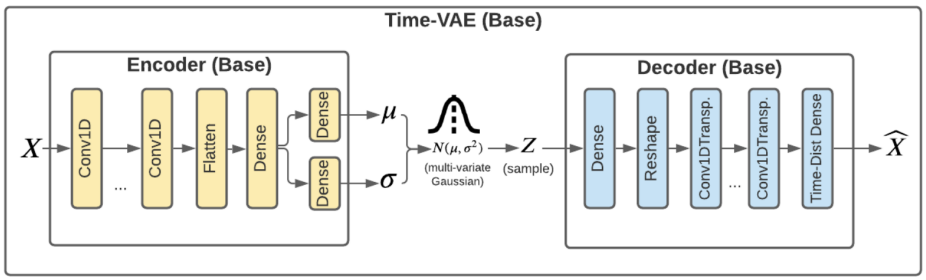

Figure 3.6: TimeVAE Base block diagram of components. Figure taken from [11].

$$\mathcal{L}_{\theta,\phi} = -\mathbb{E}_{q_\phi(z|x)}\big[\log p_\theta(x \mid z)\big] + D_{\mathrm{KL}}(q_\phi(z \mid x) \,\|\, p_\theta(z)) \tag{3.2}$$

which combines a reconstruction loss with a KL divergence term enforcing regularization toward the chosen prior, typically $\mathcal{N}(0, I)$. TimeVAE further introduces a weighting factor on the reconstruction term, allowing practitioners to emphasize fidelity of generated samples versus latent space regularization. Contrary to generic VAEs, TimeVAE's architecture allows the injection of explicit temporal components, such as polynomial trends or seasonal patterns, into the decoder, providing interpretable control over generated dynamics. This feature is particularly advantageous in domains with limited data, where domain expertise can guide the generative process. By uniting probabilistic latent modeling with temporal priors, TimeVAE produces synthetic sequences that are both realistic and structurally consistent with the dynamics of real-world time series.

3.3.3 Fourier Flows

Fourier flows generate time series by first moving from the time domain into the frequency domain. The process starts with a discrete Fourier transform (DFT), which converts sequences of any length and sampling rate into fixed-length spectral representations. Once in this form, the data can be modeled with

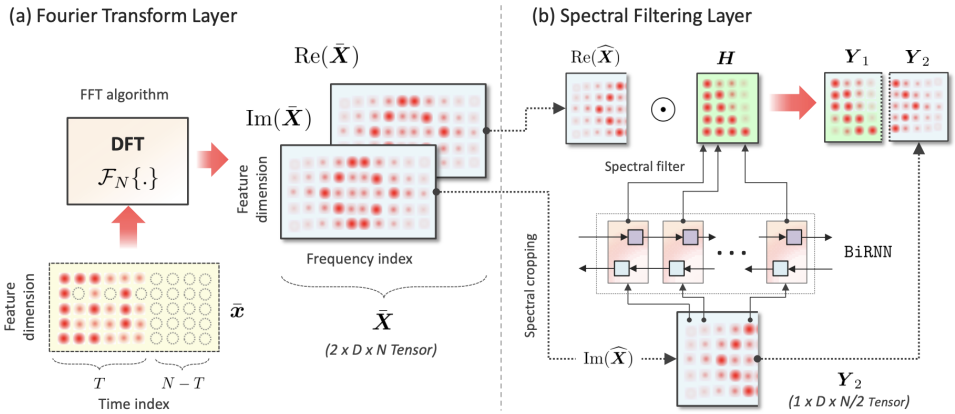

Figure 3.7: Fourier Flow pictorial depiction of the two layers. Figure taken from [1].

normalizing flows that apply flexible, data-dependent transformations in the frequency space. This design avoids the difficulties of handling variable-length time series directly in the time domain and creates a common representation for many types of signals.

Fourier Flows [1] extend normalizing flow models to time-series data by exploiting the frequency-domain representation of sequences (Figure 3.7). The flow consists of two main layers: a *Fourier Transform layer* and a *spectral filtering layer*. First, each time series $x = [x_0, \ldots, x_{T-1}]$ is zero-padded to a fixed length $N \geq T$ and transformed via a discrete Fourier transform (DFT) applied independently to each feature dimension d, producing a complex-valued tensor $X_c \in \mathbb{R}^{2\times D\times N/2}$ consisting of the real and imaginary components. This step ensures lossless, invertible conversion between time and frequency domains while enabling uniform handling of variable-length sequences and inconsistent sampling rates. Next, the spectral filtering layer applies a data-dependent affine transformation in the frequency domain. Specifically, a bi-directional recurrent neural network (BiRNN) processes the imaginary component of X_c to produce frequency-domain scale and shift parameters H and μ, which are then applied to the real component via element-wise multiplication and addition. Formally, the transformation is

$$Y_1 = H \odot \mathrm{Re}(X_c) + \mu, \quad Y_2 = \mathrm{Im}(X_c), \quad Y = \mathrm{concat}(Y_1, Y_2) \tag{3.3}$$

where $\odot$ denotes the Hadamard product. The inverse DFT of Y recovers the transformed time-domain sequence, corresponding to a circular convolution between the even component of x and the learned impulse response of the spectral filter, while the odd component is preserved. This frequency-domain formulation provides a richer, convolutional mapping compared to traditional time-domain flows, allowing the model to capture complex temporal dependencies without increasing the computational cost: the DFT runs in $O(DN \log N)$ and the spectral filtering operates in $O(N)$ per feature. Moreover, the DFT Jacobian has a unit determinant, preserving exact likelihood evaluation. By combining invertible frequency transformations with data-dependent spectral filtering, Fourier Flows produce expressive, fully tractable generative models that maintain both temporal coherence and exact likelihood computation for sequential data.

3.3.4 Diffusions

Diffusion models generate data by gradually transforming random noise into structured sequences. In the context of time series, this process involves learning how temporal patterns emerge as noise is progressively

removed, allowing the model to capture both short-term fluctuations and long-term dependencies. By iteratively denoising, diffusion models can produce highly realistic sequences that maintain the statistical and temporal characteristics of the original data. This approach has recently shown strong results in applications such as forecasting, imputation, and synthesis of complex multivariate time series.

Diffusion-TS [54] adapts denoising diffusion probabilistic models (DDPMs) to time-series generation by incorporating an interpretable decomposition into the denoising network (Figure 3.8 and Figure 3.9). As in standard diffusion frameworks, the forward process gradually perturbs data $x_0 \sim q(x)$ into Gaussian noise $x_T \sim \mathcal{N}(0, I)$ through transitions $q(x_t|x_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}x_{t-1}, \beta_t I)$. The reverse process, parameterized by $p_\theta(x_{t-1}|x_t)$, learns to denoise iteratively using a neural network approximation of the posterior mean $\mu_\theta(x_t, t)$. Training reduces to minimizing a weighted mean squared error:

$$\mathcal{L}(x_0) = \sum_{t=1}^{T} \mathbb{E}_{q(x_t|x_0)} \left\| \mu(x_t, x_0) - \mu_\theta(x_t, t) \right\|^2 \tag{3.4}$$

which can be interpreted as optimizing a variational lower bound on the log-likelihood. The key innovation of Diffusion-TS is an encoder–decoder transformer with a decomposition-based decoder that disentangles latent dynamics into *trend*, *seasonality*, and *residual error* components. Polynomial regression layers capture smooth low-frequency trends, Fourier layers model periodic seasonal components via learned frequency bases, and residual layers absorb remaining variability. The overall reconstruction of the clean signal is then expressed as

$$\hat{x}_0(x_t, t, \theta) = V_{\text{trend}}^t + \sum_{i=1}^{D} S_{i,t} + R \tag{3.5}$$

where V^ttrend is the polynomial trend, Si, t seasonal terms from Fourier bases, and R the residual output of the final decoder block. To enforce accurate recovery of both time-domain and frequency-domain structures, training employs a Fourier-augmented loss:

$$\mathcal{L}_\theta = \mathbb{E}_{t,x_0}\Big[w_t \Big(\lambda_1 \|x_0 - \hat{x}_0\|^2 + \lambda_2 \|\mathrm{FFT}(x_0) - \mathrm{FFT}(\hat{x}_0)\|^2 \Big) \Big] \tag{3.6}$$

Diffusion-TS further extends to conditional generation (e.g., imputation and forecasting) by guiding the reverse process with gradient-based constraints, steering synthetic samples toward conditions or observed subsequences. By integrating interpretable trend–seasonality decomposition into the diffusion mechanism, Diffusion-TS yields both high-quality generative performance and transparent representations of temporal dynamics.

3.3.5 Transformers

Transformers, originally developed for natural language processing, have shown strong potential in time series generation. Their attention mechanism allows them to capture both short-term and long-term dependencies within sequences, making them well-suited for complex, multivariate data. When they are processing entire sequences in parallel, transformers can identify global trends while preserving local variations, thus offering advantages over traditional recurrent models.

Time-Transformer [35] integrates a layer-wise parallel architecture within an adversarial autoencoder (AAE) framework for time-series generation (Figure 3.10 and Figure 3.11). The overall model first reconstructs a prototype of the input sequence via a deconvolutional block, which is then refined by the Time-Transformer to capture both local and global temporal dependencies. Within each Time-Transformer

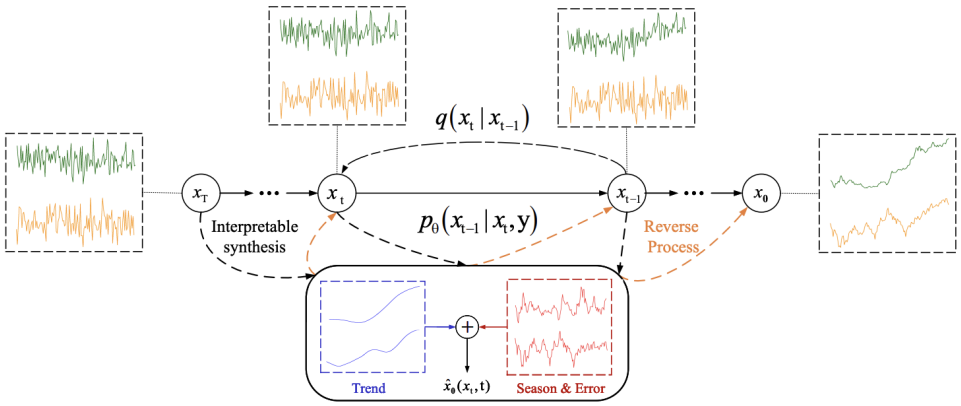

Figure 3.8: Diffusion-TS forward and reverse diffusion on time series data. The denoising network learns to predict the clean time series $\hat{x_0}$ from x_t based on an interpretable decomposition. In the reverse pass, the generator network gradually injects the expert knowledge to make the synthetic series target move toward the real ones. Figure taken from [54]

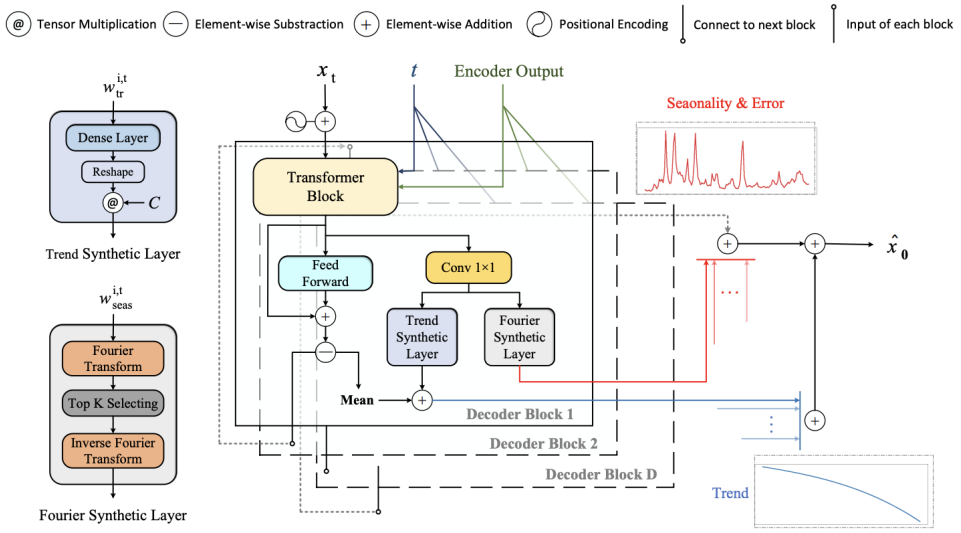

Figure 3.9: Diffusion-TS network architecture of the decoder in proposed $\hat{x_0}$ approximator. Figure taken from [54].

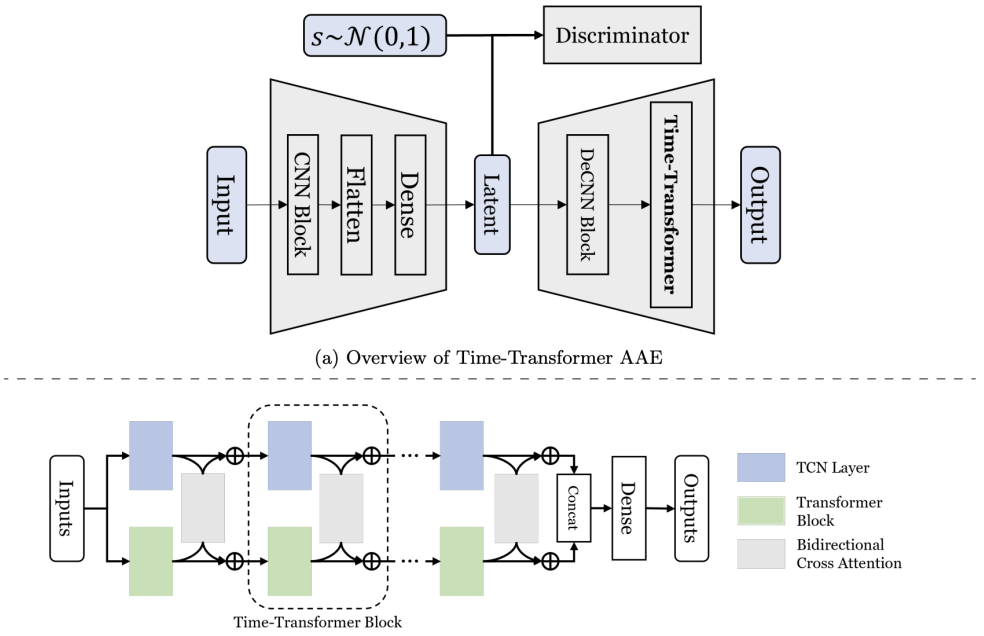

Figure 3.10: Time-Transformer Structure [35]

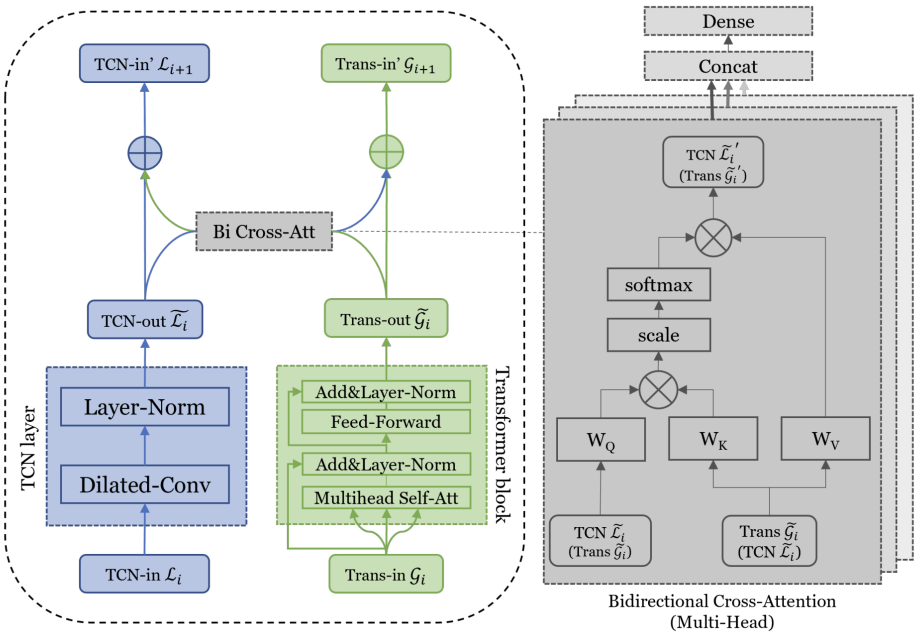

Figure 3.11: Time-Transformer detail design of the block. Figure taken from [35].

block, temporal convolutional networks (TCNs) and Transformer blocks operate in parallel. The TCN stream employs dilated convolutions to extract local, short-term features, while the Transformer stream leverages multi-head self-attention to model global dependencies. Outputs from both streams are fused via bidirectional cross-attention, allowing each branch to refine its representation based on the complementary features of the other. Formally, let $L_f^i \in \mathbb{R}^{t \times c}$ denote the local features from the TCN and $G_e^i \in \mathbb{R}^{t \times c}$ denote the global features from the Transformer at block i. The cross-attention updates are defined as:

$$L^{i+1} = L_f^i + A_{G_f^i \to L_f^i} \cdot G_e^i W_{ev}, \qquad A_{G_f^i \to L_f^i} = \text{softmax}\Big(\frac{L_f^i W_{eq} (G_e^i W_{ek})^T}{\sqrt{c}}\Big), \tag{3.7}$$

$$G^{i+1} = G_e^i + A_{L_f^i \to G_f^i} \cdot L_f^i W_{dv}, \qquad A_{L_f^i \to G_f^i} = \text{softmax}\Big(\frac{G_e^i W_{dq} (L_f^i W_{dk})^T}{\sqrt{c}}\Big), \tag{3.8}$$

where $W_{eq}, W_{ek}, W_{ev}, W_{dq}, W_{dk}, W_{dv}$ are learnable linear projections. This mechanism allows the TCN and Transformer streams to iteratively exchange information, producing features that jointly capture short-term fluctuations and long-range dependencies. The concatenated outputs are mapped to the desired dimensions through a fully connected layer and reshaped to reconstruct the time series. By embedding the Time-Transformer within an AAE, the model combines the advantages of adversarial learning with structured temporal representation, enabling the synthesis of realistic sequences while maintaining both local fidelity and global temporal coherence.

3.4 Evaluation of Generated Time Series

Evaluating synthetic time series is one of the most challenging aspects of time series generation. In forecasting, we can compare predictions directly to future values. Generative models are different; they do not aim to produce one correct answer, but instead try to mimic the behavior of the original data while creating new, realistic variations. Hence, the evaluation is more complex and requires multiple perspectives. Researchers usually look at three key dimensions:

- how faithful the sequences are to the real data,
- how diverse they are, and
- how useful they can be in practice

The latest benchmarking efforts, most notably TSGBench, have proposed a comprehensive framework for evaluating generative models. To capture complementary aspects of quality, it combines model-based, feature-based, distance-based, and visualization-driven measures.

- **The Discriminative Score (DS)** focuses on fidelity; thus, it measures how easy it is to tell apart synthetic data from real data. To calculate it, a classifier such as an LSTM or GRU is trained to distinguish between the two. Real sequences are labeled as authentic, while synthetic ones are labeled as artificial. If the classifier often makes mistakes, the DS will be low, which indicates that the synthetic data looks very close to the original.
- **The Predictive Score (PS)** highlights usefulness. In this case, a forecasting model is trained only on synthetic data and then tested on real data. If the model performs well, it suggests that the synthetic sequences capture the temporal patterns that matter for prediction. Mean absolute error is often used as the performance metric here, but other losses can also apply.

- **Contextual-FID (C-FID)** looks at fidelity in a different way. Instead of working directly with raw values, it compares real and synthetic sequences in a learned feature space. This helps capture higher-level contextual similarities that go beyond point-by-point matching. A lower C-FID means the generated data is statistically closer to the real distribution, not just in values but also in structure.
- **Marginal Distribution Difference (MDD)** shifts the focus to statistics and checks whether the overall distribution of values in synthetic data matches that of the real data. Should the synthetic sequences produce very different histograms, MDD will reveal the mismatch.
- **AutoCorrelation Difference (ACD)** targets temporal dependencies. Many time series display patterns over time, such as seasonality or lagged relationships. ACD measures how well these correlations are preserved in the synthetic data. Smaller differences indicate that the generated sequences capture the rhythms and repetitions present in the original.
- **Skewness Difference (SD)** examines asymmetry in distributions. Some datasets lean heavily to one side; for instance, stock returns often have more extreme drops than rises. SD checks whether the synthetic data reflects the same imbalance.
- **Kurtosis Difference (KD)** captures another aspect of distributional shape: the heaviness of the tails. High kurtosis indicates frequent extreme values. KD evaluates whether the synthetic series also reproduce such rare but important events.
- **Euclidean Distance (ED)** provides a straightforward comparison. It simply measures the average point-by-point distance between synthetic and real sequences. As much as it is easy to compute, it is sensitive to small shifts in time.
- **Dynamic Time Warping (DTW)** addresses exactly that limitation. Instead of comparing points at identical positions, DTW stretches or compresses time so that similar shapes can align, even if they occur at different speeds or start times. This makes it especially useful when evaluating sequences that follow the same pattern but with slight misalignments.
- **Visualization** tools add an intuitive layer of evaluation. t-SNE projects high-dimensional time series into two dimensions, letting researchers see whether real and synthetic data overlap in the reduced space. Distribution plots, on the other hand, offer a direct side-by-side view of value distributions, making differences immediately visible.
- **Training efficiency** provides a practical perspective. TSGBench records metrics like training time, reminding us that even the most accurate model may be of limited use if it is too slow or resource-intensive. Efficiency helps balance quality with feasibility in real applications.
- Finally, error-based measures such as **RMSE and MAE** provide a more direct approach. They compare values between generated and real sequences one by one. RMSE gives extra weight to large errors by squaring them, which makes it sensitive to outliers. MAE, on the other hand, measures the average size of the errors in a more balanced way. These metrics are easy to interpret, quick to compute, and useful when the generated series can be aligned with the original data. Unlike DS and PS, they do not require training an extra model, which makes them less sensitive to randomness in initialization.

Each metric has its drawbacks. For instance, DS and PS depend heavily on the Train-on-Synthetic-Test-on-Real (TSTR) setup, which can lead to inconsistent results since deep learning models are sensitive to their architecture and starting conditions. They also struggle with small datasets or very short sequences, where predictive models may fail to learn. For this reason, it is often best to pair them with stable, deterministic measures like RMSE and MAE.

Overall, no single measure can capture everything, but some of them provide more valuable insight for the matter at hand. For the purpose of this thesis, only DS, PS, RMSE, and MAE are used further. DS reveals how realistic the data looks, PS shows whether it is useful for prediction, and RMSE and MAE provide clear numerical comparisons. Using these measures together allows one to better judge not only the appearance of synthetic sequences, but also their practical value in scientific and industrial settings.

4
Automated Model Selection

This chapter examines automated machine learning (AutoML) [18], which systematically optimizes model selection, hyperparameter tuning, and feature engineering with minimal human intervention, with a particular focus on FLAML [47], a lightweight yet efficient AutoML framework designed to deliver accurate, resource-aware solutions through cost-effective search strategies.

4.1 AutoML

Automated model selection, or AutoML [18], is the process of choosing the most suitable predictive model and hyperparameters for a dataset with minimal manual intervention. Instead of relying on intuition and trial-and-error, AutoML formalizes this task as a structured optimization problem. At its core, an AutoML pipeline typically consists of four main stages. Data preparation ensures that raw inputs are cleaned, transformed, and formatted to handle issues such as missing values, irregular sampling, or noise. Feature engineering then derives informative representations, including temporal, statistical, or domain-specific features, that improve model performance. Next is model generation, which explores different algorithms and hyperparameter settings, guided by systematic search strategies or meta-learning principles. Finally, model evaluation compares candidate solutions against defined metrics, balancing predictive accuracy with constraints like computational efficiency. In the following sections, a closer look at each of these stages is taken in order to understand their roles, challenges, and importance in building effective AutoML pipelines for time series data.

4.2 Data Preparation and Feature Engineering

Data preparation and feature engineering form the foundation of any AutoML pipeline. In this stage, raw time series data is collected, cleaned to handle missing or inconsistent entries, and optionally augmented to improve generalization. Feature engineering then transforms the data into informative representations: relevant variables and lagged observations are selected, new features such as rolling statistics or Fourier components are constructed to highlight temporal patterns, and latent representations are extracted using

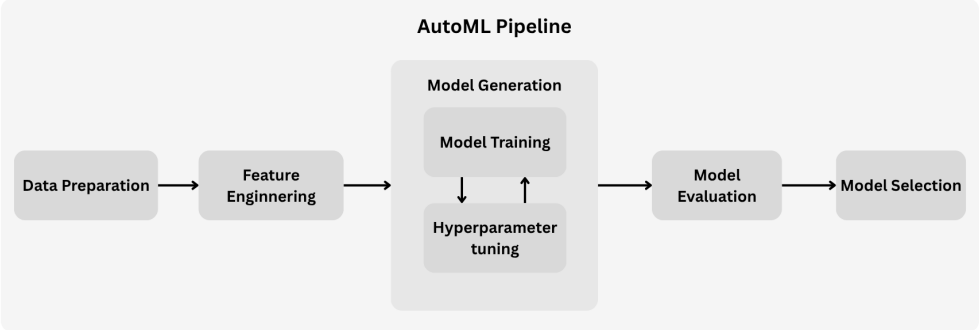

Figure 4.1: AutoML Pipeline.

methods like PCA or neural encoders to capture complex dependencies. These steps ensure the dataset is structured, informative, and ready for effective model generation.

4.3 Model Generation

Model generation involves creating machine learning architectures or models that can perform well on a given task. Namely, this process consists of two main parts: defining the search space of possible architectures and using optimization methods to find the best one. The search space determines what kind of networks can be designed, ranging from traditional models like SVM and KNN to deep neural networks (DNNs). Furthermore, optimization methods adjust either the network structure or the training hyperparameters, such as learning rate, number of layers, or filter sizes. Neural architecture search (NAS) is a modern approach that automates this design process, thus reducing the need for human trial-and-error.

4.3.1 Search Space

The search space defines the set of candidate model architectures that an AutoML pipeline can explore. In neural architecture search (NAS) [13], models are typically represented as directed acyclic graphs (DAGs), where nodes correspond to feature tensors and edges correspond to operations, such as convolutions, pooling, or skip connections. Candidate operations may also include advanced modules, e.g., depth-wise separable convolutions, dilated convolutions, or squeeze-and-excitation blocks.

Five main types of search spaces are commonly applied:

- **Entire-structured** [6, 39, 57]: The entire-structured search space defines a network by stacking layers with specified operations, optionally including skip connections [17] to improve information flow. Each layer is explicitly chosen, making this approach straightforward and easy to implement, as the full network architecture is directly specified rather than composed from smaller motifs or cells.

  However, this simplicity comes at a cost. The search space grows exponentially with depth and the number of candidate operations, making the search computationally expensive for deep networks. Moreover, networks discovered in this space tend to have limited transferability: a model optimized on a small dataset may not generalize well to a larger dataset, often requiring a new search. These challenges have motivated alternative search spaces, such as cell-based and hierarchical designs, which reduce complexity while maintaining strong performance.

- **Cell-based** [39, 56, 58]: The cell-based search space simplifies neural architecture search (NAS) by defining a network as a stack of repeated cells rather than searching each layer individually. Inspired by human-designed networks like ResNet [17], each cell acts as a modular building block, allowing the network to scale by repeating the same motif. Typically, networks use two types of cells: normal cells, which preserve spatial dimensions, and reduction cells, which downsample feature maps while increasing channels. This design improves transferability, as cells optimized on smaller datasets can generalize to larger ones.

  Each cell consists of multiple blocks, and each block contains nodes that apply operations (e.g., convolutions, pooling, skip connections) to selected inputs. The outputs of nodes within a block are combined through addition or concatenation. Each block can be represented as a tuple specifying its inputs, operations, and combination method. To handle varying input dimensions, calibration operations like 1×1 convolutions align channel numbers and adjust spatial resolution when needed. This structured design allows efficient search, as only the cell architecture is optimized while the overall network is formed by stacking these cells.

  Cell-based search significantly reduces complexity compared to layer-wise search. For example, with typical settings (5 candidate operations, 3 blocks per cell), the number of possible cell structures is orders of magnitude smaller than the number of entire networks, making NAS tractable. Standard methods follow a two-stage pipeline, where cells are first searched on shallow networks and then evaluated on deeper networks. Techniques like progressive DARTS [10] gradually increase network depth during search to bridge the gap between search and evaluation, achieving better performance without excessive computational cost.

- **Hierarchical** [32]: The hierarchical search space extends the cell-based paradigm by recursively composing cells into higher-level modules, allowing the architecture to capture both cell-level and network-level patterns. Unlike standard cell-based NAS, which often uses the same cell repeated throughout the network, hierarchical search enables flexible combinations of different cells across layers, supporting varied feature-map sizes, channel numbers, and computational budgets. This approach improves architecture diversity and better adapts models to different tasks and hardware constraints.

  At the cell level, lower-level primitives such as 1×1 or 3×3 convolutions and pooling operations are combined to form higher-level cells. These higher-level cells, in turn, can be treated as building blocks for even larger cells, forming a multi-level hierarchy. Methods like HierNAS [33] and PNAS [31] use this recursive strategy to progressively construct more complex cells while controlling search space size. Each higher-level cell can be represented by a learnable adjacency matrix that defines connections and operations between nodes, enabling flexible topologies that go beyond fixed cell repeats.

  Hierarchical search also allows layer-specific cell structures, unlike standard cell-based NAS where all layers share the same cell. For example, MnasNet [45] uses a factorized hierarchical design, where each layer may have a different cell composed of a variable number of repeated blocks, balancing accuracy and latency. This flexibility makes hierarchical search suitable for optimizing models across diverse datasets and target platforms, though it typically requires more computation, motivating techniques like proxy datasets or progressive search to improve efficiency.

- **Morphism-based** [8, 23, 49]: Morphism-based NAS focuses on transforming existing networks rather than designing architectures from scratch. By applying function-preserving operations, a trained "parent" network can be morphed into a new "child" network while retaining its learned knowledge. This approach leverages the principle behind transfer learning [53] and knowledge distillation [20] but directly modifies the network structure itself, enabling rapid exploration without restarting training from zero.

The transformations, often called network morphisms, include changes in depth, width, and kernel size. Early methods like Net2Net [9] introduced identity morphisms (IdMorph) to make networks deeper or wider while keeping their function intact. However, these methods were limited in scope and required separate operations for each type of change. More advanced network morphisms [49] can simultaneously adjust multiple aspects of the architecture and support non-linear layers, allowing a child network to inherit all knowledge from its parent. This capability accelerates training substantially, often reducing it to a fraction of the time required to train a network from scratch.

Beyond accelerating learning, morphism-based search provides a flexible framework for progressively exploring the search space. For instance, convolutional layers can be morphed into more complex modules [48], or networks can be adapted to different resolutions and computational constraints. Frameworks combining morphism with optimization strategies, such as Bayesian-guided morphism [24], have shown strong performance while minimizing search cost. Overall, this approach emphasizes efficiency and knowledge reuse, making it particularly suitable for iterative and resource-aware NAS scenarios.

- **Pipeline-based** [26]: The pipeline-based search space is designed for AutoML scenarios where the goal is to select or configure complete processing pipelines rather than individual neural network layers. In this paradigm, a candidate pipeline consists of multiple components, such as feature extraction, normalization, dimensionality reduction, and learning or imputation algorithms, each of which may have several parameter configurations. The search process explores combinations of these components to identify the pipeline that performs best for a given dataset or task. This approach is particularly useful for structured data, time series, or other domains where the challenge lies in finding the optimal combination of preprocessing and modeling steps rather than designing a deep neural network from scratch.

  Search in this space typically follows an iterative expand-and-prune strategy. Methods such as A-DARTS [26] begin with an initial seed set of pipelines and progressively generate new variations by adjusting one component or parameter at a time. Candidate pipelines are evaluated on subsets of data or through cross-validation, and low-performing pipelines are pruned early to focus computational resources on the most promising configurations. This iterative refinement allows the method to balance exploration of the search space with efficiency, avoiding the need to exhaustively evaluate all possible pipelines.

  One key advantage of pipeline-based search is the ability to leverage prior knowledge. Existing high-performing pipelines or component configurations can be reused and adapted, effectively transferring information from previously seen datasets to new tasks. Additionally, aggregation strategies such as soft voting can combine recommendations from multiple top-performing pipelines, improving stability and robustness across datasets. This approach has been shown to perform well in practical applications like time series imputation, where the search space includes diverse algorithms with varying strengths and runtime trade-offs, making pipeline-based search a powerful alternative to traditional NAS in non-deep-learning domains.

Designing an effective search space is critical. It must be flexible enough to cover a wide variety of architectures while avoiding excessive computational cost and human bias. The choice of search space directly constrains the optimization process and determines the potential performance of the generated models.

4.3.2 Architecture Optimization

Once the search space is defined, architecture optimization (AO) guides the discovery of high-performing models. AO methods aim to automate the selection of neural architectures, reducing reliance on human

expertise and trial-and-error. Common AO strategies include evolutionary algorithms, reinforcement learning, gradient-based optimization, surrogate model-based optimization, and hybrid methods.

Evolutionary Algorithms (EA) use population-based metaheuristic search inspired by biological evolution. Architectures are encoded either directly, as explicit graphs or binary strings, or indirectly, through generative rules. Key steps include selection of high-fitness networks, crossover to combine genetic information, mutation to introduce diversity, and population updates to maintain computational efficiency. EA is robust and broadly applicable, but computationally intensive.

Reinforcement Learning (RL) treats architecture search as a sequential decision-making problem. A controller, often an RNN, samples architectures from the search space and receives performance-based rewards from the environment. Policy gradients or proximal policy optimization update the controller to bias future samples toward higher-performing architectures. Parameter-sharing techniques, such as in ENAS [39], accelerate RL-based search by avoiding full training of every candidate.

Gradient-based Optimization methods, such as DARTS [34], relax the discrete search space to a continuous, differentiable space. Each candidate operation is weighted, enabling joint optimization of architecture parameters and model weights via gradient descent. Techniques such as bilevel or mixed-level optimization and Gumbel-Softmax sampling mitigate overfitting and reduce memory usage during search.

Surrogate Model-based Optimization (SMBO) builds predictive models of architecture performance, such as Gaussian processes, random forests, or neural networks, to guide search. By estimating the most promising architectures before training, SMBO methods substantially reduce computational cost, particularly for high-dimensional or variable-length architectures.

Hybrid Methods combine multiple optimization strategies to leverage their strengths. For example, EA can be augmented with RL or SMBO to improve efficiency and stability, while gradient-based methods can be coupled with surrogate models to accelerate evaluation. Hybrid approaches often achieve superior performance while maintaining practical search times.

Effective AO requires balancing exploration of diverse architectures with computational efficiency, and the choice of method often depends on the size of the search space, available resources, and target application constraints.

4.3.3 Hyperparameter Optimization

After identifying a promising neural architecture, hyperparameter optimization (HPO) is required to further improve performance. HPO aims to select the optimal configuration of learning rates, regularization coefficients, batch sizes, and other parameters to maximize model performance. Common HPO strategies include grid search, random search, Bayesian optimization, and gradient-based methods.

Grid and Random Search are the simplest HPO methods. Grid search (GS) divides the hyperparameter space into discrete intervals and evaluates all combinations, guaranteeing coverage but suffering from exponential growth in computational cost for high-dimensional spaces. Random search (RS) selects hyperparameters at random, which often explores more diverse configurations in fewer trials, especially when only a subset of hyperparameters significantly affects performance. Techniques such as coarse-to-fine GS [21], contracting GS [19], and Hyperband [28] improve efficiency by focusing resources on promising regions of the hyperparameter space and discarding poorly performing configurations early.

Bayesian Optimization (BO) treats HPO as a black-box optimization problem and builds a probabilistic surrogate model to predict the performance of hyperparameter configurations. BO iteratively balances exploration of the hyperparameter space and exploitation of promising configurations. Common surrogate models include Gaussian processes (GP) [41], random forests (RF), and tree-structured Parzen estimators (TPE) [4]. Advanced variants, such as BOHB [14] and FABOLAS [27], combine BO with resource allocation strategies or sub-dataset evaluations to further accelerate the search, often achieving orders-of-magnitude faster optimization.

Gradient-based Optimization (GO) leverages differentiable relationships between hyperparameters

and model performance. GO methods compute gradients of the validation loss with respect to hyperparameters, enabling efficient updates for thousands of continuous hyperparameters. Reverse-mode and forward-mode gradient optimization strategies allow hyperparameters to be updated during or before training, with techniques such as reversible dynamics and approximate gradients reducing computational and memory costs. GO can also optimize optimizer-specific hyperparameters, such as momentum or Adam coefficients, providing fine-grained control over training dynamics.

HPO is critical for bridging the gap between a promising neural architecture and its peak performance, and the choice of method depends on the dimensionality of the hyperparameter space, computational budget, and the degree of differentiability in the target function. Efficient HPO often complements architecture search, resulting in fully optimized models suitable for deployment.

4.4 Model Evaluation

After generating a model for time series prediction, its performance must be assessed to determine forecasting accuracy. Fully training each network on historical data and testing on a separate set is straightforward but computationally expensive, especially for large datasets or complex architectures. To reduce cost, researchers use strategies that estimate performance without full training.

Low-fidelity evaluation simplifies training by using shorter sequences, smaller data subsets, or reduced network sizes. For instance, a model may be trained on only a few months of data or on a downsampled series. Even partial training often identifies promising architectures, enabling faster comparisons.

Weight sharing reuses parameters from previously trained networks, leveraging knowledge of common temporal patterns. By transferring or adapting weights to new architectures, redundant computation is reduced, and evaluation becomes faster. Network morphism techniques similarly allow efficient exploration of new architectures using learned weights.

Surrogate models predict network performance based on previously evaluated architectures, avoiding full training. Semi-supervised approaches can incorporate unlabeled models to improve these predictions.

Early stopping halts training for networks unlikely to perform well based on partial learning curves or error trends. These techniques enable efficient architecture exploration while conserving computational resources.

4.5 FLAML

FLAML [47] (Fast and Lightweight AutoML) is an automated machine learning library designed to efficiently build high-performing models with minimal computational cost. Unlike many traditional AutoML tools that require large clusters or long runtimes, FLAML is tailored for speed and resource efficiency, making it suitable for ad-hoc datasets or applications where only limited CPU or time is available. Here, the library automatically searches for the best model, hyperparameters, and training settings by balancing trial cost and predictive performance, gradually moving from cheaper, rough trials to more expensive, accurate ones.

4.5.1 Design Overview

Figure 4.2 illustrates the design of FLAML, a fast and lightweight AutoML library structured to achieve efficient and flexible architecture search. The system consists of two layers: the ML layer, which contains candidate learners, and the AutoML layer, which orchestrates the search process. The AutoML layer includes a learner proposer, a hyperparameter and sample size proposer, a resampling strategy proposer, and a controller. In each iteration, the proposers select a learner, its hyperparameters, sample size, and resampling strategy, after which the controller executes a trial on the ML layer and observes the validation

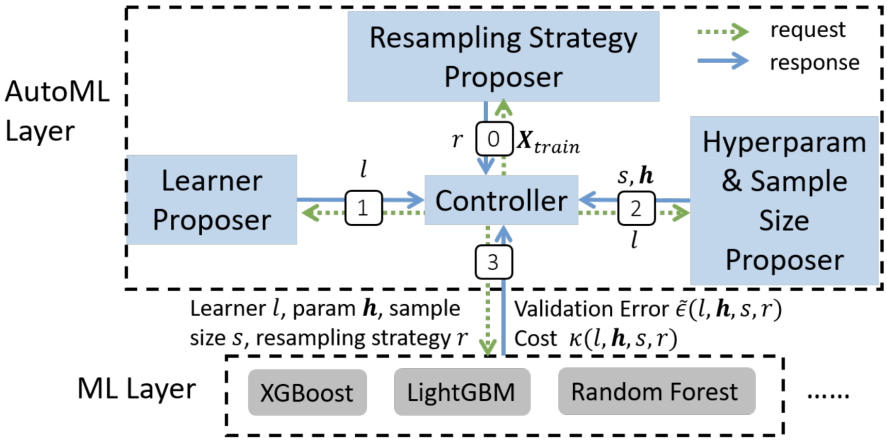

Figure 4.2: FLAML Architecture with its major components. Figure taken from [47].

error $\tilde{\epsilon}$ and computational cost κ. This process repeats until the allocated budget is exhausted, and in parallel environments, multiple iterations can be executed simultaneously to improve throughput.

FLAML decouples the search strategies of each component, allowing changes in one to occur independently of the others, which enables easy integration of new search schemes. Compared to previous AutoML frameworks, FLAML couples hyperparameter and sample size selection to respect their dependency while decoupling learner selection, focuses on low-cost evaluation for ad-hoc and large-scale datasets without relying on meta-learning or ensembling, and employs specialized search strategies to efficiently explore interactions between learners, hyperparameters, and resampling methods. This design ensures a lightweight, modular, and efficient AutoML framework suitable for practical use on diverse datasets.

4.5.2 Search Strategy

FLAML's search strategy relies on the notion of *estimated cost for improvement* (ECI) to guide efficient exploration of learners, hyperparameters, and sample sizes. For each learner l, $\text{ECI}_1(l)$ estimates the cost of finding a configuration with lower validation error than the current best for l, $\tilde{\epsilon}_l$, at the current sample size. $\text{ECI}_2(l)$ estimates the cost to evaluate the current configuration with an increased sample size, scaled by a factor c. Finally, $\text{ECI}(l)$ represents the estimated cost to improve over the current global best error $\tilde{\epsilon}^*$, combining both ECI_1 and ECI_2:

$$\text{ECI}_1 = \max(K_0 - K_1, K_1 - K_2), \quad \text{ECI}_2 = c \cdot \kappa_l, \tag{4.1}$$

$$\text{ECI} = \max\left(\frac{(\tilde{\epsilon}_l - \tilde{\epsilon}^*)(K_0 - K_2)}{\delta}, \min(\text{ECI}_1, \text{ECI}_2)\right), \tag{4.2}$$

where K_0, K_1, and K_2 denote the cumulative cost spent on l at successive improvements, δ is the observed error reduction, and κ_l is the cost of the current trial. ECI_1 captures the increasing difficulty of finding improvements later in the search, while ECI_2 reflects the expected benefit of increasing sample size. When a learner has not yet been evaluated, ECI_1 is initialized based on the minimum observed trial cost from the fastest learner.

The search proceeds in three steps. Step 0 determines the resampling strategy r, using a simple thresholding rule based on dataset size and budget. Cross-validation is applied to small datasets, while holdout is used for larger datasets. Step 1 selects a learner l with probability inversely proportional to ECI(l), giving higher priority to learners expected to improve error at low cost while maintaining randomization to ensure fair exploration. As new observations are collected, ECIs are updated dynamically, allowing self-correction if previous estimates were inaccurate, where an unsuccessful XGBoost trial increases its ECI and decreases its selection probability.

Step 2 involves the hyperparameter h and a sample size s proposer. Hyperparameters are optimized via randomized direct search, which iteratively samples a random direction u in the $(|h| - 1)$-dimensional unit sphere and moves along u or its opposite based on observed changes in validation error. Step size is adaptively adjusted and occasionally restarted to escape local optima. Sample size is also adaptively modified: if $\text{ECI}_1(l) \geq \text{ECI}_2(l)$, the current hyperparameters are retained and the sample size is increased; otherwise, a new hyperparameter configuration is explored at the current sample size. Once the full dataset is used, the sample size remains fixed until convergence, mitigating the risk of pruning promising configurations due to small sample sizes. Stratified random shuffling is applied to generate samples for classification tasks, ensuring representative subsets.

This design allows FLAML to achieve both strong anytime performance and high final accuracy. ECI-based prioritization favors inexpensive learners early but adapts to penalize slow-improving learners later. The combination of randomized direct search and adaptive sample sizing enables efficient exploration of both low-cost and high-complexity configurations, while the computational overhead of ECI calculations and hyperparameter updates remains negligible compared to the cost of model training. The result is an AutoML framework capable of navigating large, ad-hoc search spaces efficiently while dynamically balancing cost and expected improvement.

5 Experiments

This chapter presents a comprehensive experimental evaluation of time series generation models. The first set of experiments assesses models using multiple evaluation dimensions, including error-based, feature-based, model-based, and distance-based metrics, to capture predictive accuracy and structural alignment. Subsequently, t-SNE and distributional visualizations are used to examine how effectively the models replicate the latent and statistical properties of the data. The following experiments compare manual evaluation outcomes with AutoML predictions, elucidating the degree to which automated selection aligns with manual selection. Finally, the influence of training time on model selection accuracy is analyzed, highlighting the trade-offs between computational efficiency and generative performance across datasets.

5.1 Pipeline

Building on the methodological foundations outlined earlier, the experimental setup is implemented as a structured pipeline that organizes the entire process of time series generation and automated model selection. As illustrated in Figure 5.1, the standard pipeline is composed of six consecutive stages: data loading, preprocessing, model training, generation, evaluation, and visualization. Raw datasets are first imported and prepared for analysis, after which candidate models are trained and used to generate synthetic time series. These generated sequences are then evaluated using both quantitative metrics and qualitative assessments, with visualizations complementing the numerical results by highlighting patterns and discrepancies in the outputs.

Alongside this, an AutoML pipeline mirrors the general structure but introduces two additional components: a reduced model training phase, which quickly benchmarks a wide range of candidate architectures, and an automated model selection step, which identifies the most suitable model for final evaluation. As a matter of fact, the two pipelines provide a complementary framework: the standard pipeline enables fine-grained control and in-depth analysis of the generation process, while the AutoML pipeline emphasizes efficiency and scalability through automation.

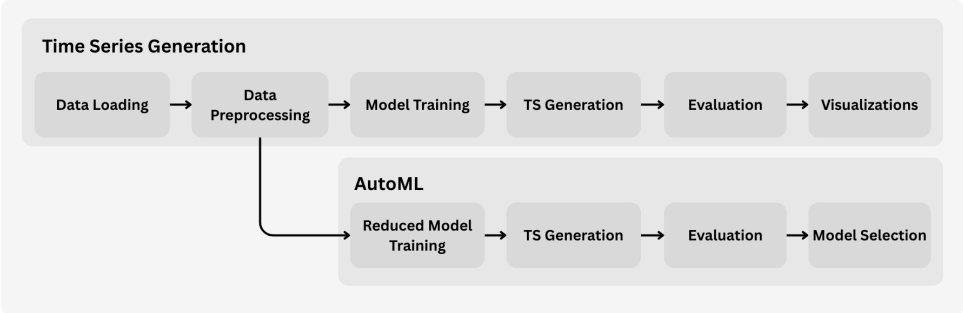

Figure 5.1: Full Pipeline

5.2 Data Handling

5.2.1 Datasets

Four datasets were selected to evaluate the models across different domains. Each dataset contains temporal information with distinct characteristics and patterns.

The first dataset consists of Google's stock price history from 2004 to 2024 [15]. It contains daily observations over a span of roughly 20 years, resulting in about 5,000 entries. Furthermore, the features include opening and closing prices, daily highs and lows, and trading volume. Stock data is particularly relevant because financial time series often exhibit recurring patterns such as volatility clustering and structural shifts. Different from Appliances or Air Quality datasets, which are dominated by regular seasonal or environmental cycles, financial data is more stochastic and heavily influenced by external shocks.

The second dataset focuses on air quality measurements collected in four major Chinese cities: Tianjin, Beijing, Guangzhou, and Shenzhen [55]. The data includes environmental variables such as temperature, humidity, and wind speed, along with pollution indicators like PM2.5, carbon monoxide, and ozone. In fact, each city contributes between 5,000 and 8,000 measurements. Tianjin is used for training with a 9:1 ratio of training and validation, while the other cities are reserved for testing, allowing models to be evaluated in a domain adaptation setting.

The third dataset records household energy consumption [7]. It combines external conditions such as temperature and humidity with contextual information like the time of day, resulting in approximately 20,000 entries. Energy usage serves as the primary target variable; it typically follows daily and seasonal cycles. This dataset is well-suited for studying recurring consumption patterns.

Finally, the last dataset represents electroencephalography (EEG) recordings of brain activity, and it provides about 15,000 measurements of electrical signals captured from different regions of the brain [42]. The signals are generally flat or slowly varying for long periods, but they contain sudden spikes of activity across the 14 sensors whenever an activation occurs, making them challenging for generative models.

The visualizations of the raw data for each feature across all datasets are presented in Figure 5.2, Figure 5.3, Figure 5.4, and Figure 5.5. These plots provide an overview of the input signals before any preprocessing or transformation steps, offering insight into their variability, scale, and temporal dynamics. Such visualizations form the foundation for understanding the characteristics of the data that will later be used in the pipeline for time series generation.

Dataset	*samples*	*seq*	*features*	Domain
Stock [15]	4,751	125	5	Financial
Air [55]	22,726	30	12	Sensor
Appliances [7]	19,735	125	26	Energy
EEG [42]	14,980	30	14	Medical

Table 5.1: Summary of datasets

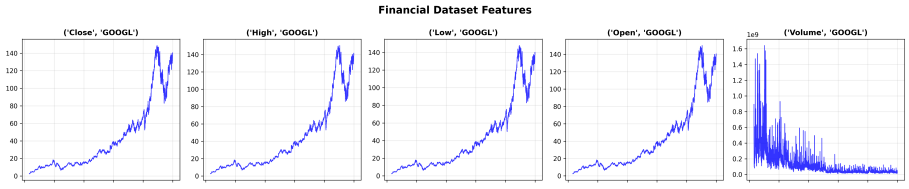

Figure 5.2: Google Stock Data

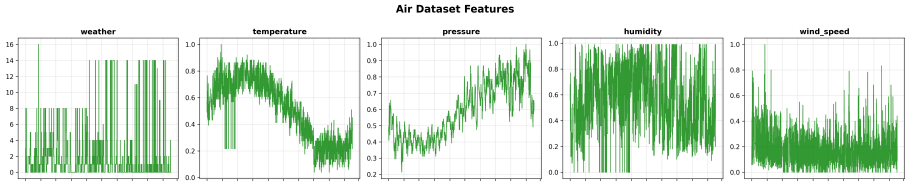

Figure 5.3: Air Data

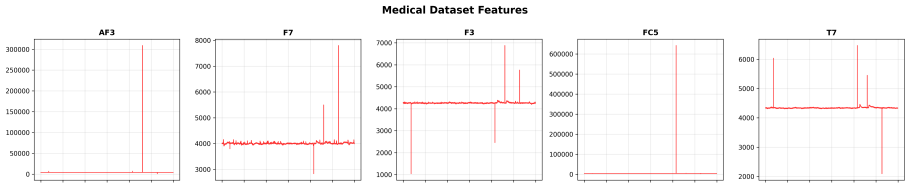

Figure 5.4: Medical EEG Data

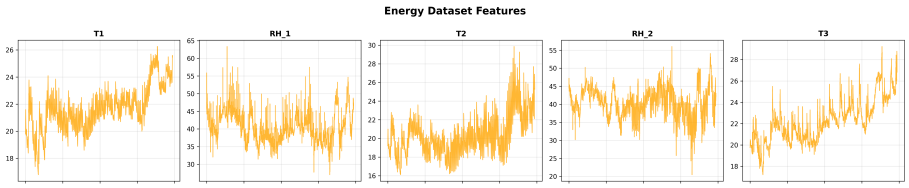

Figure 5.5: Energy Appliances Data

5.2.2 Preprocessing

All datasets undergo a consistent preprocessing pipeline before being used for model training. The first step involves data cleaning; missing values, empty cells, and irrelevant features are removed. Non-numerical columns such as dates or categorical labels are excluded. Only continuous numerical variables that contribute to learning remain.

The next step is creating time sequences. Instead of predicting from individual observations, consecutive measurements are grouped into sequences of 125 time steps. This sliding window approach enables the models to capture temporal dependencies while keeping the sequences computationally manageable. After sequences are formed, the data is divided into training and testing sets, with 90% of sequences used for training and 10% used for testing. Logically, the larger portion ensures sufficient data for learning, while the smaller portion provides a reliable basis for evaluation.

Normalization is then applied using min–max scaling. All variables are rescaled to a range between 0 and 1. This transformation ensures that features with larger ranges, such as trading volumes or pollution levels, do not dominate others, and it also accelerates the convergence of the models during training. Compared to z-score normalization, min–max scaling has the advantage of preserving the relative relationships in the original data distribution and does not suppress outliers, which can be important in domains where extreme values carry meaningful information.

The final output of preprocessing is a clean dataset consisting of overlapping sequences, normalized values, and clearly separated training and testing sets. The models receive input in a uniform structure, which makes the learning task consistent across different domains.

5.3 Time Series Generation

5.3.1 Model Training

Six models were trained and evaluated, each representing a distinct methodological family. Configurations followed the recommendations of the original papers, with adjustments only to match the datasets and the preprocessing pipeline. With this, reproducibility and fair comparison were ensured, without introducing bias from extensive manual tuning.

- **ARIMA**: Parameters p, d, and q were chosen automatically using an information-criterion-based search. The model was fitted on 125-step sequences and completed training within seconds.
- **Exponential Smoothing**: Additive trend and seasonality components were used, and smoothing parameters were estimated automatically. Like ARIMA, training was completed quickly and required minimal configuration.
- **TimeVAE**: The latent space was set to 8 dimensions as suggested in the original implementation. Input sequences were fixed at 125 time steps with multiple features. Training was performed for 200 epochs with early stopping to prevent overfitting.
- **TimeGAN**: Here, the generator, discriminator, and embedding networks were configured according to the default architecture. Training was run for 500 epochs, as longer training is required for adversarial models to stabilize. Batch size and learning rate were kept at the recommended defaults.
- **Fourier Flows**: This model was implemented with 10 flow layers as proposed in the reference paper. Each sequence was transformed to the frequency domain before training and inverted back for generation. Convergence was achieved within 200 epochs.

- **Time-Transformer**: The encoder–decoder structure with convolutional layers and transformer blocks was adopted directly from the original design. Then, the sequence length was fixed at 125 steps, with hidden dimensions and attention heads left at default values. Training was completed in 100 epochs for all datasets, except for the appliances dataset, where resource limitations made it difficult to obtain results and visualizations.

5.3.2 Evaluation

In order to provide a comprehensive and fair comparison of the six models, the evaluation stage was designed. Metrics were selected to capture different aspects of time series quality: accuracy, similarity, distributional consistency, and practical usability. Since no single metric is sufficient to fully assess generative performance, a diverse set was chosen to highlight complementary perspectives.

- **Error metrics**: RMSE and MAE were used to quantify direct reconstruction accuracy. These measures provide a straightforward baseline for how closely the generated sequences align with the real data on a point-by-point basis. Training time was also recorded, as computational efficiency is an important factor when comparing deep learning models with classical statistical approaches.
- **Distance-based metrics**: Euclidean Distance (ED) and Dynamic Time Warping (DTW) were applied to measure similarity between real and synthetic sequences. ED captures direct alignment, while DTW accounts for temporal shifts and distortions, making it more robust in settings where patterns may appear at different times. Including both ensures that results are not biased toward a single alignment assumption.
- **Statistical distribution metrics**: Marginal Distribution Difference (MDD), Autocorrelation Difference (ACD), Skewness Difference (SD), and Kurtosis Difference (KD) were chosen to evaluate whether generated data preserves key statistical properties of the original time series. MDD measures overall distribution similarity, ACD captures temporal dependencies, while SD and KD ensure that higher-order distributional characteristics such as asymmetry and tail behavior are not lost. With the inclusion of these metrics, the point was to ensure that models are judged not only on short-term accuracy but also on their ability to reproduce the underlying statistical structure of the data.
- **Model-based metrics**: Discriminative Score (DS), Predictive Score (PS), and Contextual Fréchet Inception Distance (C-FID) were used to assess generation quality from a model-driven perspective. DS quantifies how distinguishable real and synthetic data is, PS evaluates whether generated sequences retain predictive utility, and C-FID provides a modern, feature-based similarity measure. These metrics were chosen because they extend beyond simple comparisons and reflect how generated data would perform in practical downstream tasks.

All metrics were computed for every dataset–model combination. Results were stored in a structured JSON format, organized hierarchically by dataset and model. Using such a design allows seamless addition of new experiments without overwriting existing results, thus ensuring reproducibility and traceability. The evaluation process was fully automated and executed after each model completed training, which guarantees consistency across runs and prevents human error in metric calculation.

5.4 AutoML

The AutoML system was implemented using the FLAML framework to automatically select the best time series generation model within a predefined time period, here named time budget. Furthermore, the system relies on sequential training with strict enforcement of time constraints. That ensures efficient use of

computational resources while maintaining model quality. Next, the pipeline begins with a data preparation phase, where preprocessed sequences are formatted and aligned for all candidate models. During the model registration phase, each model type is wrapped as a custom FLAML estimator, inheriting from BaseEstimator. Like this, the system handles both classical and deep learning models uniformly.

Time is allocated according to a budget distribution strategy. Classical models receive a small fraction of the total budget (5%) due to their fast training times. Deep learning models receive the remaining 95%, with the TimeTransformer assigned the largest share (35%) because of its complexity and longer training requirements. As a result, all models can complete training while giving sufficient resources to those requiring more computation. The sequential training phase runs models one after another, respecting the allocated budget for each. After training, each model generates synthetic data for evaluation. The primary objective is to minimize the evaluation metric specified at runtime through a command-line argument, and the secondary objective is to maximize time efficiency.

All results are stored in a structured format, with each entry containing the model name, time used, allocated budget, and the chosen evaluation metric. Broadly speaking, a clear comparison across models can be made, and reproducibility facilitated, thanks to the aforementioned design. More specifically, the AutoML implementation ensures that the best-performing model with respect to the selected metric is chosen automatically while balancing computational efficiency, model quality, and strict adherence to the time budget.

5.5 Results

This section presents the results of three experiments conducted in the study, focusing on five models: ARIMA, Exponential Smoothing, TimeVAE, TimeGAN, and Time Transformer. Their performance was evaluated across a diverse set of metrics, including RMSE, MAE, ED, DTW, MDD, ACD, SD, KD, DS, PS, C-FID, and training time. Fourier Flow was initially included in the analysis; however, its performance proved extremely poor across all datasets, likely due to implementation issues or difficulties integrating it into the experimental pipeline. For this reason, it was excluded from the main analysis to preserve the clarity and reliability of the reported results.

5.5.1 Data Generation Performance

The performance of the models across different datasets was evaluated using a range of quantitative measures. Figures 5.6 through 5.9 summarize the results across four categories: error-based, distance-based, feature-based, and model-based measures.

When focusing on prediction accuracy and structural similarity, the deep generative models generally show clear advantages. TimeGAN consistently achieves the lowest RMSE and MAE on the EEG dataset and performs strongly on DTW and dependency measures. This indicates that the GAN-based approach is particularly effective for highly dynamic data with complex temporal dependencies. TimeTransformer demonstrates stable performance across datasets and remains competitive with TimeGAN in several cases, although it rarely surpasses it. In contrast, TimeVAE struggles with EEG data, yielding substantially higher error values and highlighting its limitations with non-stationary, high-frequency signals.

For the Google and Air datasets, which are comparatively smoother, classical models such as ARIMA and Exponential Smoothing remain competitive, achieving error levels close to those of the deep models. This suggests that for simpler time series, the added complexity of deep generative approaches does not necessarily translate into improved predictive accuracy.

When training time is taken into account, the tradeoff becomes evident. ARIMA and Exponential Smoothing achieve solid results with minimal computational effort, which makes them attractive for applications where efficiency is essential. In contrast, the deep generative models, particularly TimeGAN

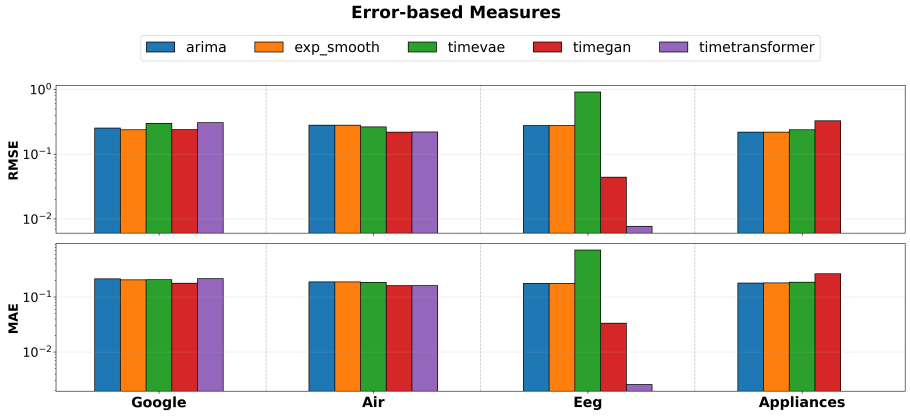

Figure 5.6: Error-based Measures results

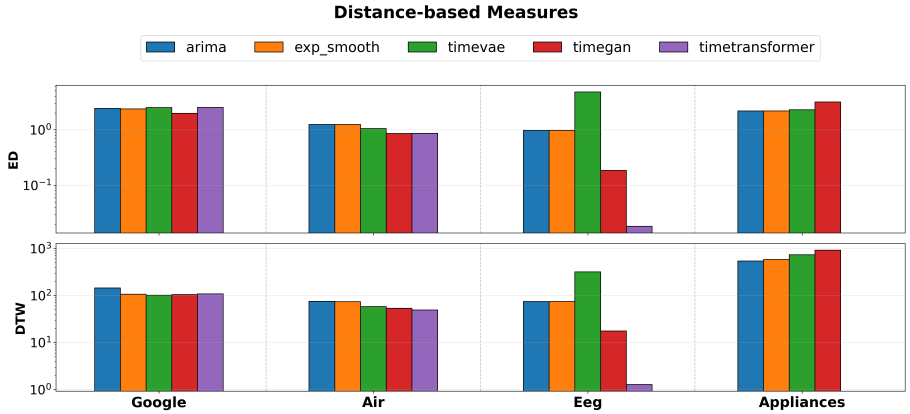

Figure 5.7: Distance-based Measures results

and TimeTransformer, require extensive training times, especially on the EEG dataset. TimeVAE also shows high training time without delivering consistently superior results. This highlights that improvements in predictive accuracy and structural fidelity through deep models often come at the cost of computational efficiency.

5.5.2 T-SNE and Distribution Visualizations

The picture changes when analyzing Figure 5.11, which compares the distributional fidelity and latent structure preservation. Here, TimeVAE and TimeTransformer show strong alignment between the synthetic and real data distributions for specific datasets. On the Google and Air datasets, the latent spaces produced by TimeVAE overlap closely with the real data, indicating that variational autoencoder approaches can effectively capture the underlying distribution in smoother series. TimeTransformer also preserves distributional characteristics well across several datasets, particularly in the Appliances dataset, where the generated series follow the real series distribution closely. By contrast, TimeGAN shows less consistent alignment in the distributional plots despite its superior predictive metrics. This suggests that while GANs excel at short-term accuracy and dependency capture, they do not always replicate the full data distribution

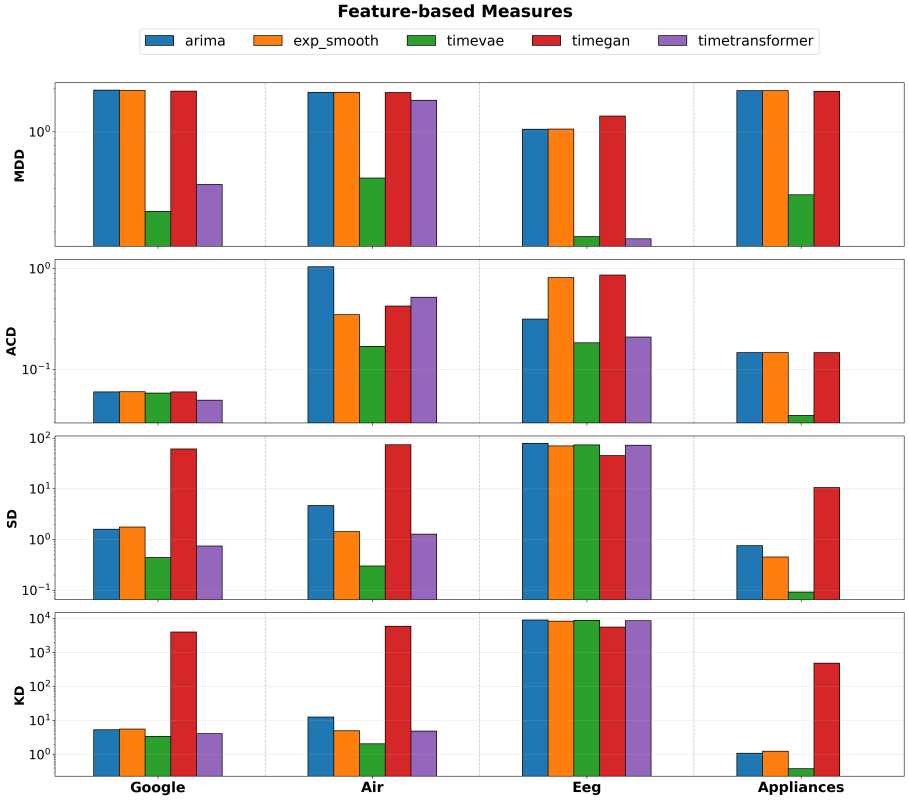

Figure 5.8: Feature-based Measures results

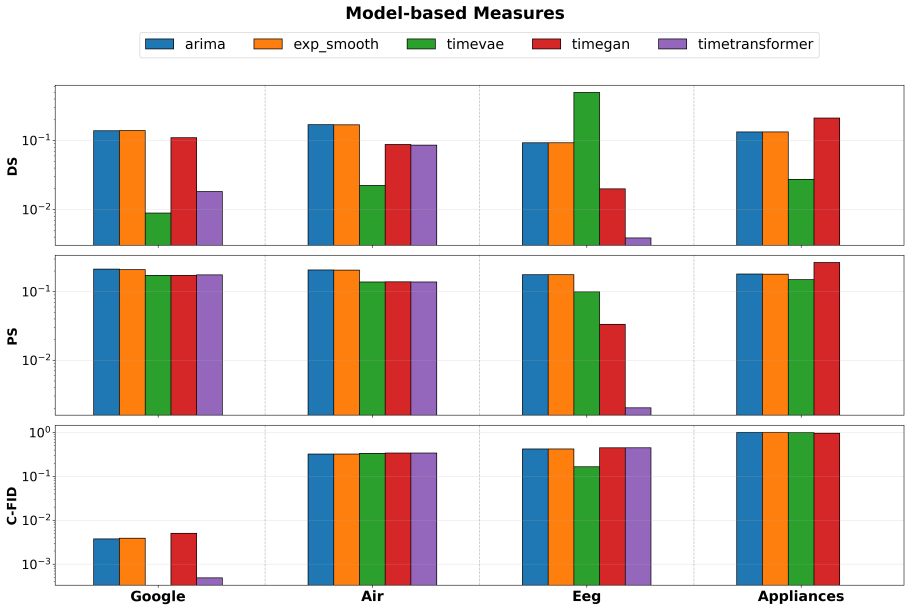

Figure 5.9: Model-based Measures results

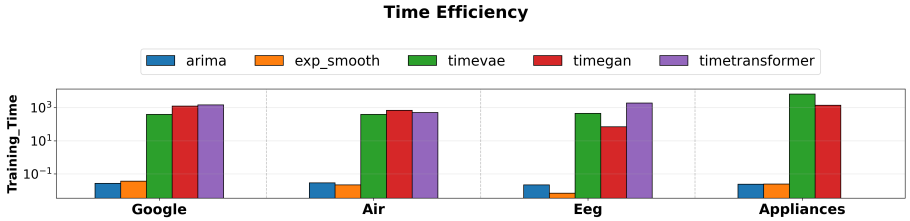

Figure 5.10: Time efficiency results

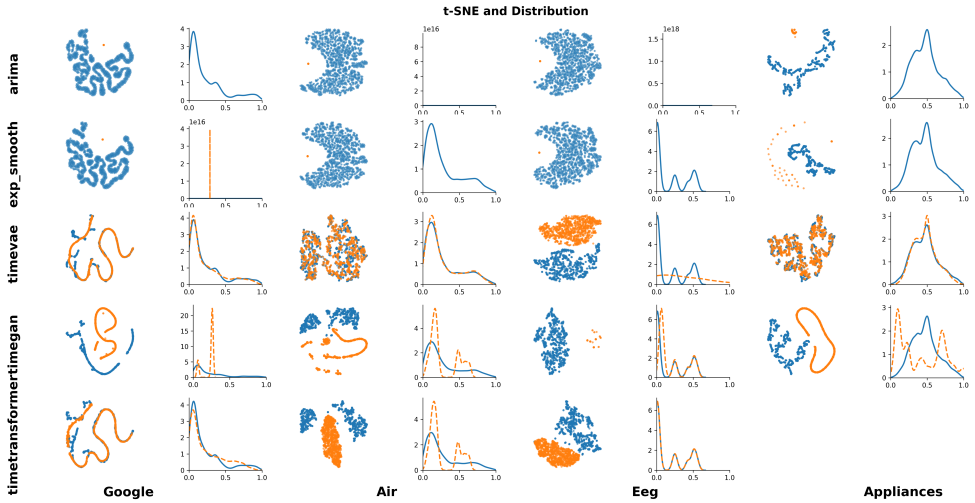

Figure 5.11: Distribution and t-SNE visualization

as reliably as VAEs or Transformers.

Taken together, the results show that the choice of model depends strongly on the dataset characteristics and the evaluation criteria. GAN-based models such as TimeGAN are well-suited for datasets with complex dynamics, offering superior accuracy and dependency preservation, but they come with high training costs and less consistent distributional fidelity. VAE-based models perform strongly on distributional similarity and preserve latent structures well, making them suitable when the statistical representation of the data is more important than pointwise accuracy. Transformer-based models achieve a balance, offering competitive results across both predictive and distributional metrics, but they also require significant computational resources. Classical models remain highly competitive for simpler time series, where their efficiency and robustness are unmatched.

5.5.3 AutoML Results

Figures 5.12 through 5.15 compare the models selected by AutoML under a 300-second time budget with the best-performing models identified through exhaustive evaluation using TSG benchmarking. During AutoML training, different percentages of time were allocated to each model:

- ARIMA - 2.5%
- Exponential Smoothing - 2.5%

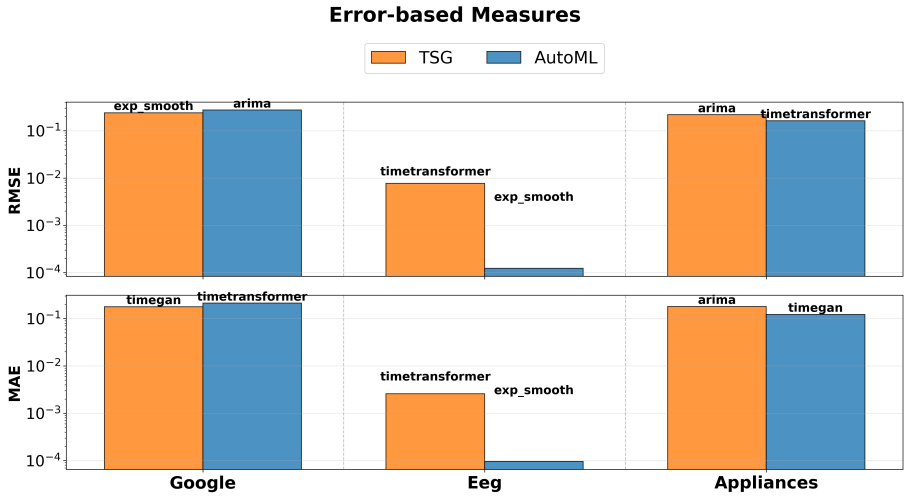

Figure 5.12: Comparison between best TSG models and the AutoML model selection using error-based measures.

- TimeVAE - 20%
- TimeGAN - 20%
- Fourier Flow - 20%
- Time Transformer - 35%

These percentages were chosen to ensure that each model receives a proportion of the total time appropriate to its complexity and learning requirements. The comparisons in Figures 5.11–5.14 highlight systematic differences between the two approaches depending on the type of evaluation metric.

For error-based measures Figure 5.11, AutoML does not perform well. It fails to consistently select the models with the lowest RMSE and MAE, highlighting its limitations in optimizing for predictive accuracy under strict time constraints.

For distance-based measures, Figure 5.12, AutoML again struggles to match the models identified by TSG benchmarking. It does not select the best-performing models for Euclidean Distance or Dynamic Time Warping in any dataset, which shows that pointwise accuracy and temporal alignment are difficult to capture with limited search time.

For feature-based measures Figure 5.13, AutoML demonstrates mixed but often favorable performance. It correctly identifies the best model in all datasets for Skewness Difference, and in two out of three datasets for both Marginal Distribution Difference and Kurtosis Difference. Additionally, it selects the best model in two datasets for Autocorrelation Difference. This indicates that AutoML can capture feature-level statistical properties of the data distribution, even with limited training time, although its performance is not always consistent across all datasets.

For model-based measures, Figure 5.14, AutoML performs relatively well. It always selects the correct model for the Discriminative Score and matches TSG benchmarking results in two out of three cases for the Predictive Score. For Contextual-FID, however, AutoML only aligns with TSG benchmarking in one dataset. These results suggest that AutoML is effective in capturing dependency structures and predictive fidelity under time constraints, although it is less reliable when evaluation depends on distributional similarity in latent space.

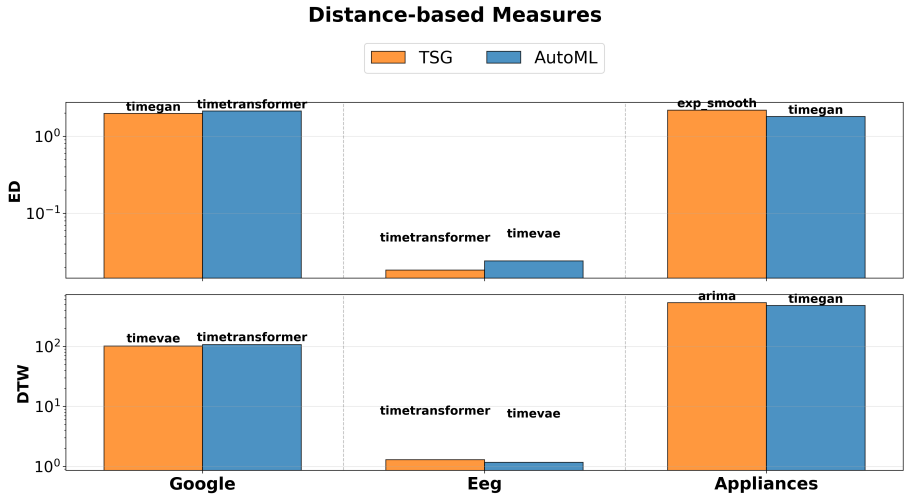

Figure 5.13: Comparison between best TSG models and the AutoML model selection using distance-based measures.

5.5.4 Effect on Training Time on Model Selection

To further investigate the performance of AutoML, it was analyzed how different time budgets affect its ability to select the best model. Figure 5.15 reports the percentage of cases where AutoML successfully selected the same model as identified by the TSG benchmark for three datasets under time budgets of 120, 300, 600, and 1200 seconds.

The results show that the time budget has a significant impact on AutoML's effectiveness. With a budget of only 120 seconds, AutoML achieves an accuracy of 18.2% across all datasets, indicating that the search space is too constrained to explore more complex models. Increasing the budget to 300 seconds improves overall accuracy to 45.5%, allowing AutoML to evaluate more candidate models and make selections that better align with exhaustive TSG benchmarks.

Extending the budget to 600 seconds yields mixed results. Accuracy increases to 63.6% for the Google dataset, but drops to 9.1% for EEG and 36.4% for Appliances, resulting in a total accuracy of 36.4%. This suggests that longer search times may encourage exploration of more complex architectures or hyperparameter configurations that do not necessarily improve performance.

With the 1200-second budget, results are again uneven. Google achieves 54.5% accuracy, EEG improves to 27.3%, and Appliances reaches 45.5%, giving an overall accuracy of 42.4%. These findings highlight that longer budgets do not guarantee better model selection. Instead, AutoML performance depends on the dataset and the balance between exploration and evaluation time. A moderate budget of 300 seconds provides the most consistent results, while very long budgets may reduce robustness for certain datasets.

5.6 Discussion and Future Work

The results of this thesis demonstrate that the effectiveness of time series generative models depends strongly on the characteristics of the dataset and the evaluation metrics used. Deep generative models such as TimeGAN perform particularly well on complex, high-frequency datasets like EEG, while TimeVAE and TimeTransformer excel in preserving distributional properties. Classical models remain competitive on smoother series due to their efficiency and stability. The AutoML experiments further highlight that

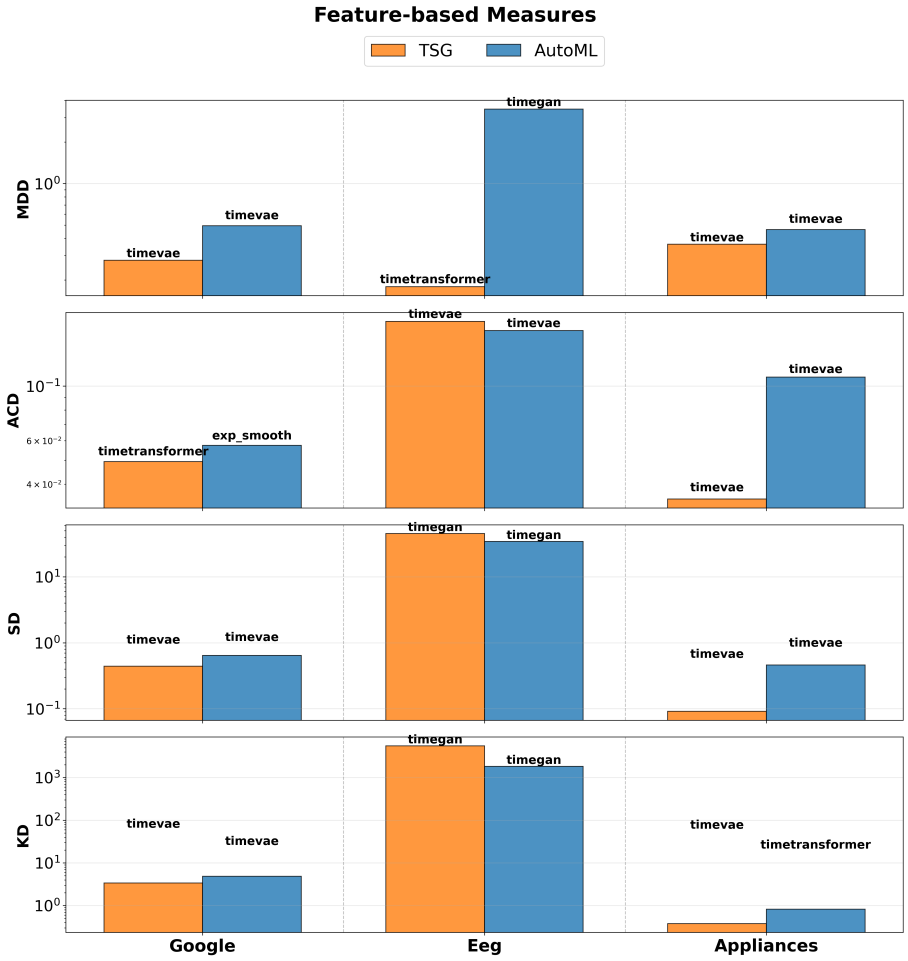

Figure 5.14: Comparison between best TSG models and the AutoML model selection using feature-based measures.

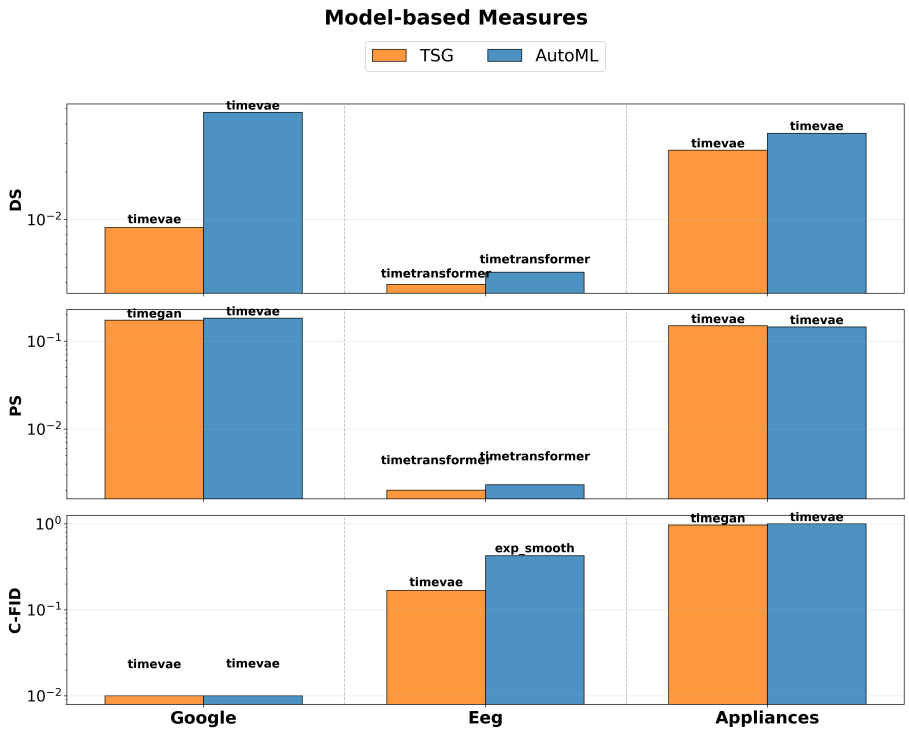

Figure 5.15: Comparison between best TSG models and the AutoML model selection using model-based measures.

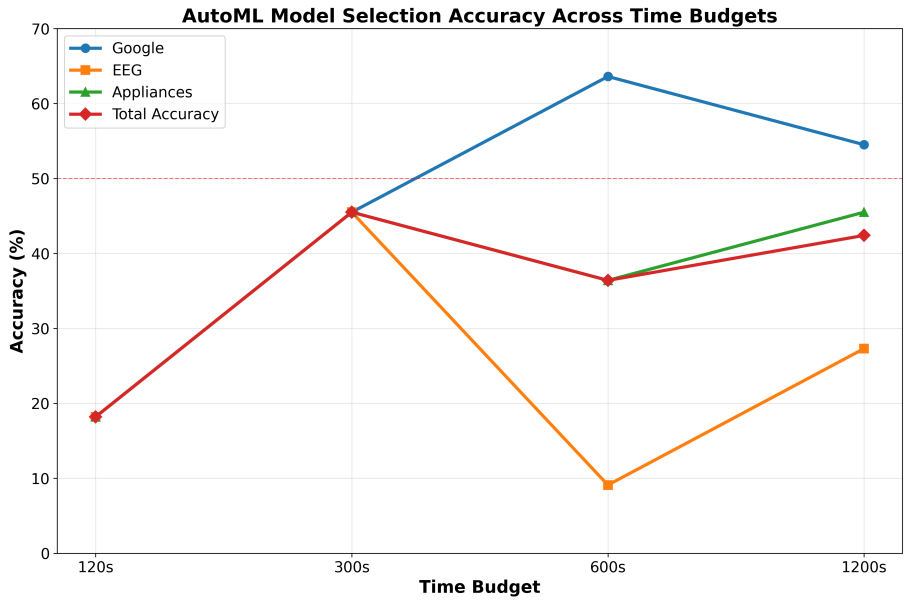

Figure 5.16: The effect of different time budgets on the model selection accuracy across different datasets.

model selection can be automated effectively, but the choice of time budget plays a critical role, and longer search times do not always guarantee better results.

While these findings provide valuable insights, a more robust analysis is needed to fully understand the limitations and generalizability of these models. Conducting additional experiments on a wider variety of datasets would strengthen the conclusions and allow for more nuanced recommendations. In particular, exploring time series from audio domains, using models such as WaveGAN [12] or DoppelGANger [29], could reveal how generative approaches perform on highly structured temporal data with different characteristics.

In healthcare, future work could focus on generating rare or hard-to-collect signals, such as biomarkers, patient vitals, or longitudinal disease progression data. Such studies would not only improve model evaluation but could also support data augmentation for clinical research and personalized medicine. Further investigations could also include domain-specific adaptations of generative models, hybrid approaches combining statistical and deep learning techniques, and systematic studies of AutoML strategies tailored to specific datasets or applications.

6 Conclusion

The thesis demonstrates that combining systematic evaluation with automated selection can efficiently identify high-performing generative models while balancing predictive accuracy, distributional fidelity, and computational cost.

The experimental results provide practical insights into the strengths and limitations of different models across diverse datasets. Deep learning models such as TimeGAN excel at capturing complex temporal dependencies, while TimeVAE and TimeTransformer preserve distributional properties, and classical models remain effective for simpler, smoother series. The AutoML framework proved capable of accelerating model selection and providing reproducible results, although performance depends on dataset characteristics and resource allocation.

Finally, this work delivers a reproducible pipeline, comprehensive evaluation metrics, and open resources for practical application. It establishes a foundation for systematic, data-driven approaches to time series generation and provides guidance for researchers and practitioners seeking to generate high-quality synthetic data efficiently.

Bibliography

[1] Ahmed Alaa, Alex James Chan, and Mihaela van der Schaar. Generative time-series modeling with fourier flows. In *International Conference on Learning Representations*, 2021.

[2] Yihao Ang, Qiang Huang, Yifan Bao, Anthony K. H. Tung, and Zhiyong Huang. Tsgbench: Time series generation benchmark. In *Proceedings of the VLDB Endowment*, 2024.

[3] Yihao Ang, Qiang Wang, Qiang Huang, Yifan Bao, Xinyu Xi, Anthony K. H. Tung, Chen Jin, and Zhiyong Huang. Ctbench: Cryptocurrency time series generation benchmark. In *Proceedings of the ACM Conference*, 2025.

[4] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24 (NIPS 2011)*, 2011.

[5] George E. P. Box and Gwilym M. Jenkins. Some statistical aspects of adaptive optimization and control. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1962.

[6] Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. Smash: One-shot model architecture search through hypernetworks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[7] Luis Candanedo. Appliances energy prediction, 2017.

[8] Tianqi Chen, Ian J. Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In *4th International Conference on Learning Representations (ICLR)*, 2016.

[9] Tianqi Chen, Ian J. Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In Yoshua Bengio and Yann LeCun, editors, *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016.

[10] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[11] Abhyuday Desai, Cynthia Freeman, Zuhui Wang, and Ian Beaver. Timevae: A variational auto-encoder for multivariate time series generation. In *International Conference on Learning Representations (ICLR)*, 2022.

[12] Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis. In *International Conference on Learning Representations (ICLR)*, 2019.

[13] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. In *Journal of Machine Learning Research*, 2018.

[14] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[15] Yahoo Finance. Google stock, 2025.

[16] Jonas Fontana. Comparison of synthetic time series data generation techniques. In *Universit of Fribourg*, 2021.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[18] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 2021.

[19] J. Y. Hesterman, L. Caucci, M. A. Kupinski, H. H. Barrett, and L. R. Furenlid. Maximum-likelihood estimation with a contracting-grid search algorithm. *IEEE Transactions on Nuclear Science*, 2010.

[20] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[21] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. *A Practical Guide to Support Vector Classification*.

[22] Ali Ismail-Fawaz, Maxime Devanne, Stefano Berretti, Jonathan Weber, and Germain Forestier. Establishing a unified evaluation framework for human motion generation: A comparative analysis of metrics. In *Computer Vision and Image Understanding*, 2025.

[23] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2019.

[24] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In Ankur Teredesai, Vipin Kumar, Ying Li, R. Rosales, E. Terzi, and George Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2019.

[25] Manoj Joshi, Dibakar Raj Pant, Rupesh Raj Karn, Jukka Heikkonen, and Rajeev Kanth. Meta-learning fast adaptation and latent representation for head pose estimation. In *31st IEEE FRUCT Conference*, 2022.

[26] Mourad Khayati, Guillaume Chacun, Zakhar Tymchenko, and Philippe Cudré-Mauroux. A-darts: Stable model selection for data repair in time series. In *Proceedings of the IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, 2025.

[27] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[28] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Ameet Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 2017.

[29] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the Internet Measurement Conference (IMC)*, 2020.

[30] Claire Little, Mark Elliot, Richard Allmendinger, and Sahel Shariati Samani. Generative adversarial networks for synthetic data generation: A comparative study. In *Proceedings of the University of Manchester*, 2021.

[31] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *European Conference on Computer Vision Workshops (ECCVW)*, 2018.

[32] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *6th International Conference on Learning Representations (ICLR)*, 2018.

[33] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[34] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *7th International Conference on Learning Representations (ICLR)*, 2019.

[35] Yuansan Liu, Sudanthi Wijewickrema, Ang Li, Christofer Bester, Stephen O'Leary, and James Bailey. Time-transformer: Integrating local and global features for better time series generation. In *SDM24*, 2024.

[36] U.S. Department of Health and Human Services. Health insurance portability and accountability act (hipaa), 1996.

[37] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le. Specaugment: A simple data augmentation method for automatic speech recognition. In *Interspeech*, 2019.

[38] European Parliament. General data protection regulation), 2016.

[39] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[40] Xiangfei Qiu, Jilin Hu, Lekui Zhou, Xingjian Wu, Junyang Du, Buang Zhang, Chenjuan Guo, Aoying Zhou, Christian S. Jensen, Zhenli Sheng, and Bin Yang. Tfb: Towards comprehensive and fair benchmarking of time series forecasting methods. In *PVLDB*, 2024.

[41] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Lecture Notes in Computer Science*, 2003.

[42] Oliver Roesler. Eeg eye state, 2013.

[43] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404, March 2020. Special Issue on Machine Learning and Dynamical Systems.

[44] Nassim Nicholas Taleb. *The Black Swan: The Impact of the Highly Improbable*. Random House, 2007.

[45] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems (NIPS)*, 2017.

[47] Chi Wang, Qingyun Wu, Markus Weimer, and Erkang Zhu. Flaml: A fast and lightweight automl library. In *Proceedings of the Fourth Conference on Machine Learning and Systems (MLSys 2021)*, 2019.

[48] Tian Wei, Chang Wang, and Chih-Wei Chen. Modularized morphing of neural networks. *arXiv preprint arXiv:1701.03281*, 2017.

[49] Tian Wei, Chang Wang, Yu Rui, and Chih-Wei Chen. Network morphism. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

[50] Peter Whittle. *Hypothesis Testing in Time Series Analysis*. Almqvist and Wiksells boktr., 1951.

[51] Peter R. Winters. Forecasting sales by exponentially weighted moving averages. *Management Science*, 1960.

[52] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[53] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems (NeurIPS 2014)*, 2014.

[54] Xinyu Yuan and Yan Qiao. Diffusion-ts: Interpretable diffusion for general time series generation. In *International Conference on Learning Representations (ICLR)*, 2024.

[55] Yu Zheng, Xiuwen Yi, Ming Li, Ruiyuan Li, Zhangqing Shan, Eric Chang, and Tianrui Li. Forecasting fine-grained air quality based on big data. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2015.

[56] Zhun Zhong, Jianhua Yan, Wei Wu, Jingjing Shao, and Chunhua Liu. Practical block-wise neural network architecture generation. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[57] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations (ICLR)*, 2017.

[58] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[1]

[1] Large language models were applied solely for proofreading and improving the clarity of the text. They were not used in the development of original ideas, analyses, or conclusions, which are entirely my own.