Série Temporais de Compressores: Análise de Modelos ODEJump Attention + TSDiffusion

Inovação em Geração de Séries Temporais Irregulares e com Dados Faltantes

CPE883 - Tópicos Especiais em Aprendizado de Máquina

Rodrigo Petrus Domingues (rodrigopd@petrobras.com.br)

14 de setembro de 2025

Motivação e Objetivo

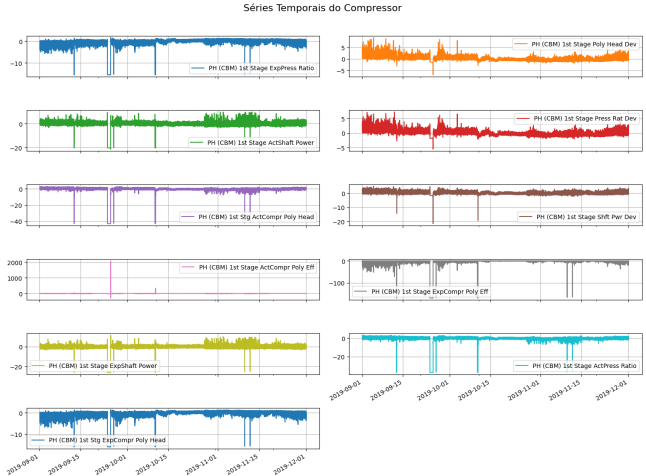
1 Introdução

- **Cenário**: Séries temporais industriais (compressores) com **amostragem irregular**, **dados faltantes** e **alta dimensionalidade**.
- **Objetivo do trabalho**: avaliar uma arquitetura *Encoder ODEJump + Attention + (reconstrução)* e seu papel em um pipeline maior com *TSDiffusion*[1].
- **Contribuição prática**: encoder contínuo-discreto (ODE + GRU) que produz estados latentes robustos para dados irregulares e faltantes.
- **Escopo deste código**: implementação do **Encoder ODEJump**, cabeças de saída (reconstrução e missingness), **pipeline de dados**, perdas e **métricas** de avaliação.

Apresentação da série temporal

1 Introdução

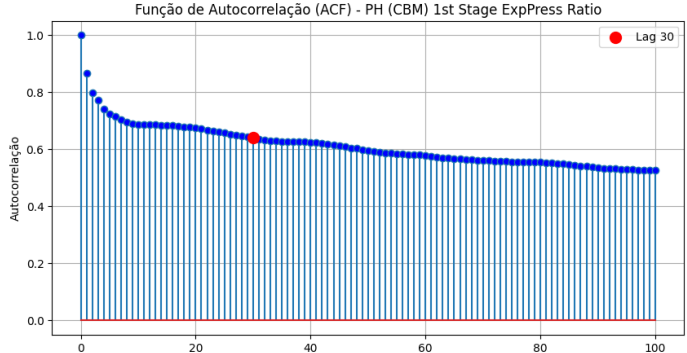
- Curvas com anomalias, mas sem base de dados sem classificação.



Autocorrelação (ACF) com janela de 100 eventos

1 Introdução

- ACF (*Autocorrelation Function*) mede a correlação da série consigo mesma em diferentes atrasos (lags).
- Aqui, consideramos uma **janela de 100 eventos**, analisando como o sinal se correlaciona até esse limite.
- Indica periodicidade, dependência temporal e "memória" da série.
- Iremos adotar janelamento de **30 eventos** por ser o momento que a série estabiliza em um plateau local.



O que foi implementado para classificar automaticamente

2 Segmentador Não Supervisionado (Multivariado)

- **Suporte multivariado**: entrada $x \in \mathbb{R}^{T \times C}$; janelamento $X \in \mathbb{R}^{N_w \times w \times C}$.
- **Features por canal** (concatenadas): média, mediana, desvio, IQR, amplitude, *skew*, *kurtosis*, *slope* OLS, % outliers (MAD), ACF (até L *lags*), *bandpowers*.
- **Features cruzadas**: covariâncias e correlações par-a-par (lag 0) entre canais.
- **Escalonamento robusto**: `RobustScaler`.
- **HMM Gaussiano (diag)**: emissões $\mathcal{N}(\mu_k, \operatorname{diag} \sigma_k^2)$; π e A estocásticas por linha.
- **Treino EM** (log-espaço), inicialização `KMeans`, critério de tolerância.
- **Decodificação Viterbi**: $z_{1:N_w}$.

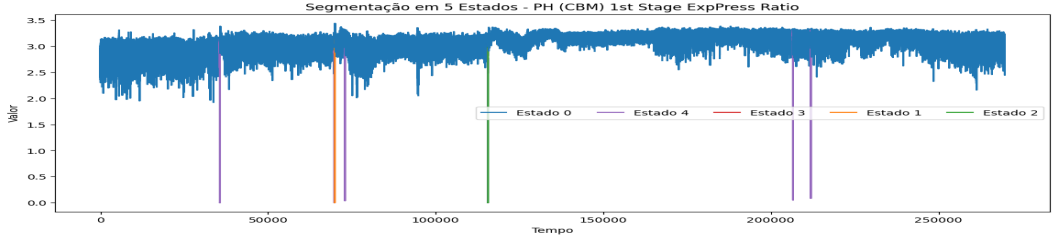
Pipeline de classificação (etapas)

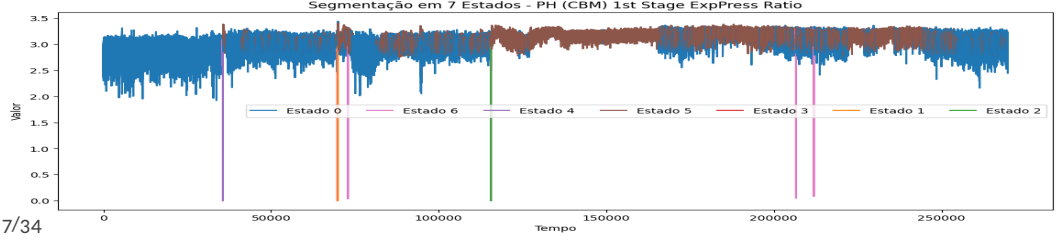
2 Segmentador Não Supervisionado (Multivariado)

1. **Janelamento**: $(T,)$ ou $(T, C) \to X \in \mathbb{R}^{N_w \times w \times C}$; guardar índices.
2. **Features**: por canal + cruzadas $\Rightarrow F \in \mathbb{R}^{N_w \times d}$.
3. **Escalonamento**: $F \mapsto F_z$ (`RobustScaler`).
4. **EM**:
   - Init: `KMeans` em F_z.
   - E-step: $\gamma_{t,k}$, $\xi_{t,i,j}$ via forward/backward.
   - M-step: atualizar π, A, μ_k, σ_k^2; monitorar $\log \mathcal{L}$.
5. **Decodificação**: Viterbi $\Rightarrow z_{1:N_w}$ e expansão temporal.

Resultados da Classificação Não Supervisionada

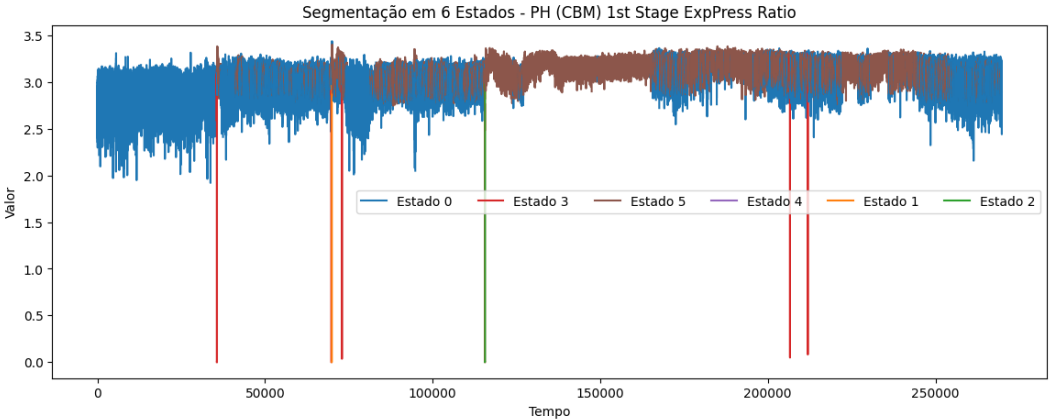
2 Segmentador Não Supervisionado (Multivariado)





Resultados da Classificação Não Supervisionada

2 Segmentador Não Supervisionado (Multivariado)



Pré-processamento dos Estados para Treinamento

2 Segmentador Não Supervisionado (Multivariado)

- **Objetivo**: reduzir desbalanceamento entre classes no treinamento do modelo.
- **Agrupamento**:
  - Estados **1, 2, 3 e 4** foram **agregados em um único estado** ($\tilde{g} = 1$).
  - Estados **0 e 5** foram mantidos como classes distintas, preservando sua interpretação original.
- **Resultado**: distribuição final de classes mais equilibrada:

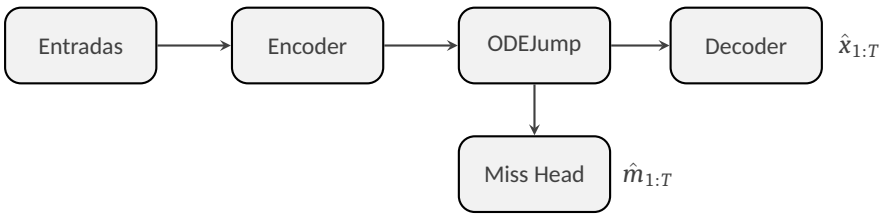
$$\text{Classes finais: } \tilde{g} \in \{0, 1, 5\}$$

- **Impacto no treinamento**:
  - Permite **amostragem estratificada** (`WeightedRandomSampler`) de forma proporcional.
  - Reduz a dominância de estados raros, estabilizando a convergência da loss.
  - Melhora a capacidade do modelo de aprender padrões robustos para classes minoritárias.

Pipeline: ODEJump (baseline)

3 Componentes da arquitetura

$x_{1:T}$
timestamps $t_{1:T}$
máscara $m_{1:T}$
estáticas c (opcional)
denoised x $z_{1:T}$
(opcional)



Encoder (Embedding)

3 Componentes da arquitetura

- **Entradas** (por timestep):

$$\left[\tilde{x}_t,\ \tilde{m}_t\right] \quad (\text{ou } \left[\tilde{x}_t,\ \tilde{z}_t^{\text{denoised}},\ \tilde{m}_t\right])$$

  Obs: a última foto é zerada nos vetores x e z e é criada uma nova máscara $\tilde{m}$, em que a última foto é substituída por 1.

- **MLP por timestep**:

$$h_t^{(0)} = \mathrm{ReLU}(W_e\,\mathrm{concat}(\cdot) + b_e)$$

- **Estáticas (opcional)**:

$$c \in \mathbb{R}^{D_c} \quad \Rightarrow \quad c_e = \mathrm{ReLU}(W_c c + b_c), \quad h_t = h_t^{(0)} + c_e$$

  (broadcast em t)

- **Saída do Encoder**:

$$H = \left[h_1, \ldots, h_T\right] \in \mathbb{R}^{T \times D}$$

ODEJump (dinâmica contínua + salto discreto)

3 Componentes da arquitetura

- **Entrada**: recebe a sequência de estados latentes

$$H^{(0)} = [\, h_1^{(0)}, \ldots, h_T^{(0)} \,] \quad \text{saída do } \textbf{Encoder}.$$

- **Evolução contínua**: entre eventos, resolve-se a ODE $\dot{h} = f_\theta(h)$ no espaço latente, integrando de t_{i-1} até t_i:

$$h(t) : \ \dot{h} = f_\theta(h) \quad \text{(MLP sobre } h\text{)}$$

- **Aproximação numérica**: usa-se **um único passo RK4** por intervalo $\Delta t_i = t_i - t_{i-1}$ para obter o estado pré-salto $\tilde{h}_i$:

$$k_1 = f_\theta(h_{i-1}),\ k_2 = f_\theta(h_{i-1} + \tfrac{\Delta t_i}{2} k_1)$$

$$k_3 = f_\theta(h_{i-1} + \tfrac{\Delta t_i}{2} k_2),\ k_4 = f_\theta(h_{i-1} + \Delta t_i k_3)$$

$$\tilde{h}_i = h_{i-1} + \frac{\Delta t_i}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

- **Por que RK4?** Balanceia custo computacional e estabilidade, dispensando um solver adaptativo completo — suficiente para passos Δt_i típicos.

ODEJump (dinâmica contínua + salto discreto)

3 Componentes da arquitetura

- **Salto discreto**: no instante t_i, aplica-se uma atualização GRU sobre o estado latente vindo do RK4:

$$h_i = \mathrm{GRUCell}\big(h_i^{(0)},\ \tilde{h}_i\big)$$

  onde $h_i^{(0)}$ é o embedding do timestep i.
- **Iteração**: o processo (integração + salto) se repete para todos os T passos.
- **Saída**: a sequência de estados refinados

$$H = [h_1, \ldots, h_T] \in \mathbb{R}^{T \times D}$$

Decoder e Miss Head

3 Componentes da arquitetura

- **Entrada**: recebe a sequência de estados latentes $H = [h_1, \ldots, h_T] \in \mathbb{R}^{T \times D}$, produzidos pelo ODEJump (ou ODEJump+Transformer).
- **Decoder (Reconstrução)**:

$$\hat{x}_t = W_d \, \mathrm{ReLU}(h_t) + b_d \quad \Rightarrow \quad \hat{X} \in \mathbb{R}^{T \times C}$$

  Restaura os sinais nos C canais observáveis, servindo como reconstrução $\hat{x}_{1:T}$.
- **Miss Head (Probabilidade de Observação)**:

$$\hat{m}_t = \sigma(W_m h_t + b_m) \quad \Rightarrow \quad \hat{M} \in [0, 1]^{T \times 1}$$

  Prediz a probabilidade de cada timestep ser observado, usada na **loss de missingness** para regularização.

Função de Custo — ODEJump

3 Componentes da arquitetura

- **Reconstrução ($\mathcal{L}_1$)**: é o **erro quadrático médio** apenas nos **valores ausentes** de x:

$$\mathcal{L}_1 = \frac{\sum (1 - m^{\text{train}}) \cdot (x - \hat{x})^2}{\max\left(\sum (1 - m^{\text{train}}),\ 1\right)}$$

  onde $m^{\text{train}} = 1$ apenas para valores *mantidos* no treino.

- **Probabilidade de Observação ($\mathcal{L}_4$)**: Binary Cross-Entropy entre máscara verdadeira e predita:

$$\mathcal{L}_4 = -\frac{1}{T} \sum_t \left[m_t^{\text{true}} \log \hat{m}_t + (1 - m_t^{\text{true}}) \log(1 - \hat{m}_t) \right]$$

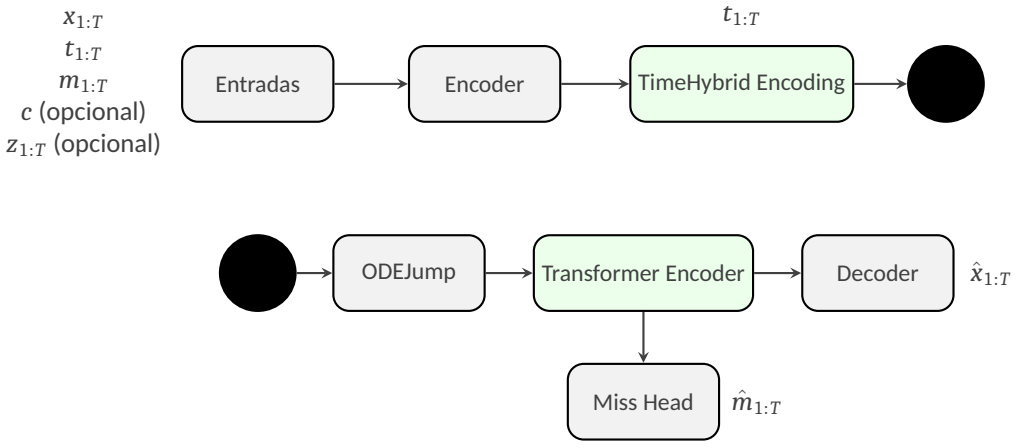
- **Loss total**: combinação ponderada

$$\mathcal{L} = \lambda_1 \mathcal{L}_1 + \lambda_4 \mathcal{L}_4, \qquad \lambda_1 + \lambda_4 = 1$$

  onde $\lambda_1 \gg \lambda_4$ (ênfase na reconstrução).

Pipeline: ODEJump + Transformer

3 Componentes da arquitetura



TimeHybrid Encoding

3 Componentes da arquitetura

- **Objetivo**: enriquecer o estado h vindo do **Encoder** com uma representação contínua e diferenciável do tempo, para uso no **ODEJump** e na **atenção temporal**.
- **Entrada**: timestamps normalizados

$$t_i^{\text{norm}} = \frac{t_i - t_0}{\text{TS\_SPAN}} \qquad \Rightarrow \qquad t_i^{\text{norm}} \in [0, 1]$$

- **Codificação híbrida**: combina rampas aprendidas (linear) e codificação senoidal (frequências log):

$$\text{Para índices ímpares:} \quad e_i^{\text{ramp}} = a \cdot t_i^{\text{norm}} + b$$

$$\text{Para índices pares:} \quad e_i^{\text{sin/cos}} = \left[\sin(\omega_k t_i^{\text{norm}}),\ \cos(\omega_k t_i^{\text{norm}}) \right]_{k=1}^{K}$$

- **Saída final**: vetor concatenado

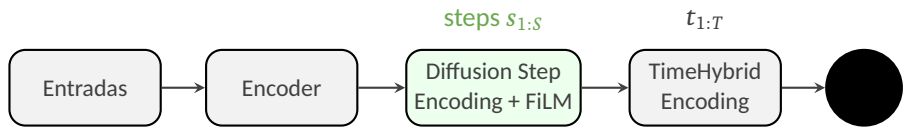
$$e_i^{\text{time}} = \text{concat}\left(e_i^{\text{ramp}},\ e_i^{\text{sin/cos}}\right) \quad \Rightarrow \quad h_i^{\text{enriquecido}} = h_i + e_i^{\text{time}}$$

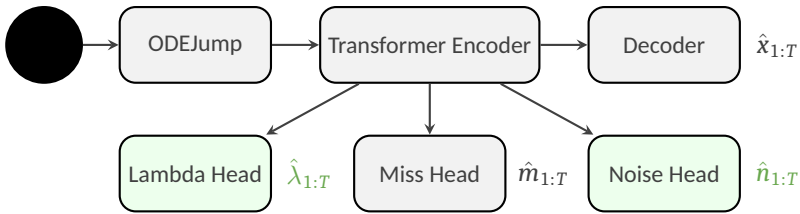
- **Uso**: $h_i^{\text{enriquecido}}$ é passado ao **ODEJump** (para integração contínua) e ao **Transformer Encoder** (atenção sensível ao tempo).

Pipeline: ODEJump + Transformer + Diffusion (TSDiffusion[1](like) como um denoiser gaussiano)

3 Componentes da arquitetura

$x_{1:T}$
$t_{1:T}$
$m_{1:T}$
number of steps S
c (opcional)





Diffusion Step (DDPM) com `cosine_beta_schedule`

3 Componentes da arquitetura

- **Agendamento coseno[2]**:

$$\bar{\alpha}_s \;=\; \cos^2\!\left(\frac{\frac{s}{S}+\delta}{1+\delta}\,\frac{\pi}{2}\right), \qquad \beta_s \;=\; 1 \;-\; \frac{\bar{\alpha}_s}{\bar{\alpha}_{s-1}}, \qquad \alpha_s \;=\; 1-\beta_s$$

$$\bar{\alpha}_s \;=\; \prod_{k=1}^{s} \alpha_k$$

- **Forward (difusão)** em um passo s:

$$\mathbf{h}_s \;=\; \sqrt{\bar{\alpha}_s}\,\mathbf{h}_0 \;+\; \sqrt{1-\bar{\alpha}_s}\,\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0,\mathbf{I})$$

- **Reverse (denoising)** — atualização ancestral ($\sigma_t^2 = \beta_t$):

$$\mathbf{h}_{s-1} \;=\; \frac{1}{\sqrt{\alpha_s}}\Big(\mathbf{h}_s - \frac{1-\alpha_s}{\sqrt{1-\bar{\alpha}_s}}\,\hat{\boldsymbol{\epsilon}}_\theta(\mathbf{h}_s, s)\Big) \;+\; \sigma_s\,\boldsymbol{\xi}, \qquad \boldsymbol{\xi} \sim \mathcal{N}(0,\mathbf{I})$$

Embedding do Passo de Difusão

3 Componentes da arquitetura

- **Objetivo**: representar o passo de difusão $s \in \{0, \ldots, S-1\}$ em um espaço vetorial contínuo de dimensão D.
- **Codificação sinusoidal (estilo Vaswani et al.[3])**:

$$\text{emb}(s) = \left[\sin\left(\frac{s}{10000^{2k/D}}\right),\ \cos\left(\frac{s}{10000^{2k/D}}\right) \right]_{k=0}^{D/2-1}$$

- **Projeção linear**:

$$\mathbf{t}_s = W_{\text{proj}}\ \text{emb}(s) + b_{\text{proj}} \qquad \Rightarrow \qquad \mathbf{t}_s \in \mathbb{R}^D$$

- **Intuição**:
  - Frequências log-espalhadas garantem **resolução alta em passos pequenos** e **suavidade** em passos maiores.
  - Projeção linear adapta o embedding ao espaço latente do modelo.
- **Uso no modelo**: $\mathbf{t}_s$ é somado a h (estado do encoder) e também usado como entrada do FiLM para modular o embedding temporal contínuo.

FiLM: Feature-wise Linear Modulation no Tempo

3 Componentes da arquitetura

- **Objetivo**: condicionar o estado latente h usando a informação de tempo contínuo, de forma **suave** e **diferenciável**.
- **Processo (no TSDiffusion[1] adaptado)**:

$$\mathbf{g}_s = \mathrm{MLP}_{\mathrm{FiLM}}(\mathrm{t}_s) \quad \Rightarrow \quad (\boldsymbol{\gamma}_s, \boldsymbol{\beta}_s) = \mathrm{split}(\mathbf{g}_s)$$
$$\tilde{\mathbf{e}}_t = (1 + \boldsymbol{\gamma}_s) \odot \mathbf{e}_t \; + \; \boldsymbol{\beta}_s$$

  onde $\mathbf{e}_t$ é o embedding de tempo contínuo e $\odot$ denota produto elemento a elemento.
- **Intuição**:
  - $\boldsymbol{\gamma}_s$ aplica **ganho** dinâmico (reescala a importância das dimensões).
  - $\boldsymbol{\beta}_s$ aplica **deslocamento** (ajusta o centro das features no tempo).
- **Justificativa de uso**:
  - Permite ao modelo **modular** a representação em função do passo de difusão s e do tempo contínuo.
  - Melhora a **consistência temporal**, tornando o aprendizado mais estável (evita sobre/under-weighting em regiões de alta ou baixa incerteza).

Noise Head

3 Componentes da arquitetura

- **Noise Head** ($\hat{\varepsilon}_t$):
  Prediz o **ruído gaussiano** adicionado no passo s da difusão.

$$\hat{\varepsilon}_s = \mathrm{MLP}_\varepsilon(h_s)$$

  Serve para estimar o ruído a ser removido na etapa de *denoising*:

$$\mu_\theta(h_s, s) = \frac{1}{\sqrt{\alpha_s}}\left(h_s - \frac{1-\alpha_s}{\sqrt{1-\bar{\alpha}_s}}\,\hat{\varepsilon}_s\right)$$

  Garante reconstrução consistente do estado latente h_{s-1}.

Lambda Head — Medida de Confiança

3 Componentes da arquitetura

- **Lambda Head** ($\hat{\lambda}_t$):
  Prediz o **grau de confiança** que o modelo tem sobre sua própria predição $\hat{x}_t$ naquele timestep — funciona como um **estimador de incerteza local**.

$$\hat{\lambda}_t = \text{softplus}\big(W_\lambda h_t + b_\lambda\big)$$

- **Interpretação**:
  - $\hat{\lambda}_t$ alto: rede está **segura** de que a predição $\hat{x}_t$ é confiável.
  - $\hat{\lambda}_t$ baixo: rede sinaliza **alta incerteza**, sugerindo cautela na interpretação de $\hat{x}_t$.
- **Uso prático**:
  - Permite **ponderar a loss** (dar mais peso a predições confiáveis).
  - Pode ser visualizado como um **mapa de confiança** sobre a série reconstruída.

Função de Perda — TSDiffusion[1] (adaptada)

3 Componentes da arquitetura

- **L1 — NLL Gaussiano ponderado por $\hat{\lambda}_t$ (em *todos* os canais):**

$$\text{sse}_t = \sum_c \left(\hat{x}_{t,c} - x_{t,c}\right)^2, \qquad \hat{\lambda}_t = \text{softplus}\left(W_\lambda h_t + b_\lambda\right)$$

$$\log p(x_t \mid h_t) = -\tfrac{1}{2}\,\hat{\lambda}_t\,\text{sse}_t + \tfrac{1}{2}\,n\,\log\hat{\lambda}_t - \tfrac{1}{2}\,n\log(2\pi)$$

$$\mathcal{L}_1 = -\sum_t \log p(x_t \mid h_t)$$

- **L2 — Denoising no latente (ruído):**

$$\mathcal{L}_2 = \sum_{t,d} \left\| \text{noise\_head}(h_{t,d}) - \epsilon_{t,d} \right\|_2^2$$

- **L4 — Missingness por timestep (Bernoulli)**
- **Função de perda combinada:**

$$\mathcal{L} = \lambda_1\mathcal{L}_1 + \lambda_2\mathcal{L}_2 + \lambda_4\mathcal{L}_4$$

Melhor configuração com cross validation até 100 épocas

Tabela: Melhor resultado (entre 8, 16 e 32 dimensões por canal).

Configuração	macro-MSE ↓	micro-MSE ↓
8 dimensões	204.289083 ± 203.480588	1.759255 ± 0.590695
16 dimensões	1.112684 ± 0.414279	0.786424 ± 0.070351
32 dimensões	0.556403 ± 0.361469	0.716240 ± 0.063003

Tabela: Melhor resultado (entre 2, 4 e 6 camadas de Transformers).

Configuração	macro-MSE ↓	micro-MSE ↓
2 camadas	1.731701 ± 0.958092	0.758033 ± 0.067674
4 camadas	1.378706 ± 0.636514	0.752427 ± 0.065560
6 camadas	0.670777 ± 0.313602	0.721729 ± 0.064226

Melhor configuração com cross validation até 100 épocas

Tabela: Melhor resultado (entre 2, 4 e 8 cabeças de Atenção).

Configuração	macro-MSE ↓	micro-MSE ↓
2 cabeças	0.838581 ± 0.286205	0.746875 ± 0.063159
8 cabeças	0.690819 ± 0.346612	0.765505 ± 0.066242
4 cabeças	0.594687 ± 0.357233	0.723179 ± 0.059632

Tabela: Melhor resultado (entre 50, 100 e 150 passos de denoining).

Configuração	macro-MSE ↓	micro-MSE ↓
150 passos	0.025409 ± 0.016501	0.026400 ± 0.010262
100 passos	0.018939 ± 0.016202	0.023044 ± 0.008627
50 passos	0.018255 ± 0.016645	0.022559 ± 0.009060

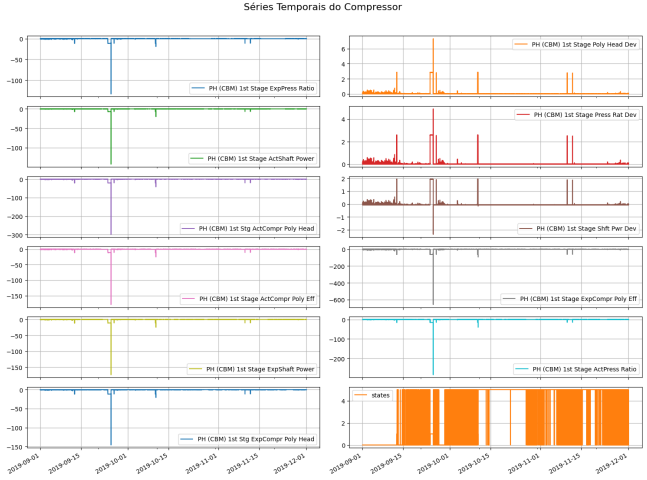
Resultados aplicando arquitetura TSDiffusion com função de perda adaptada ($\mathcal{L}_2$)

Tabela: Melhor resultado após 65 épocas de treinamento

macro-MSE	micro-MSE
0.070653 $\pm$ 0.002102	0.068604 $\pm$ 0.000826

Série temporal limpa de ruído gaussiano (ruído branco)

5 Resultados Denoising



Evolução da perda durante o treinamento (macro e micro MSE).

Comparação das arquiteturas

6 Resultados

Tabela: Resultados por modelo

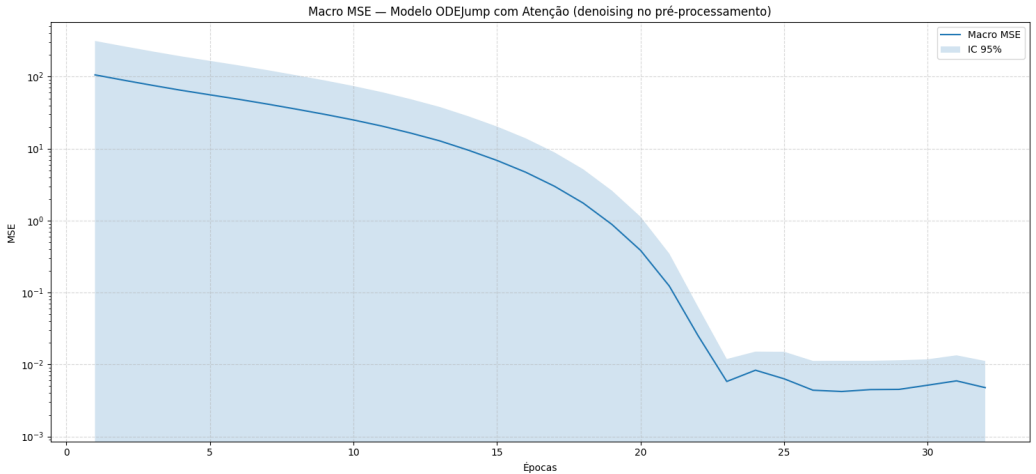
Modelo	macro-MSE	micro-MSE
ODEJump (baseline)	0.291084 ± 0.163770	0.331472 ± 0.025534
ODEJump + Transformers	0.371475 ± 0.147604	0.337681 ± 0.035874
ODEJump com denoising	0.343405 ± 0.135264	0.308247 ± 0.026885
ODEJump + Transformers com denoising	0.260986 ± 0.160533	0.323708 ± 0.025023

Tabela: Resultado com entrada somente denoised

Modelo	macro-MSE	micro-MSE
ODEJump + Transformers somente com denoising	0.004218 ± 0.003596	0.005153 ± 0.002070

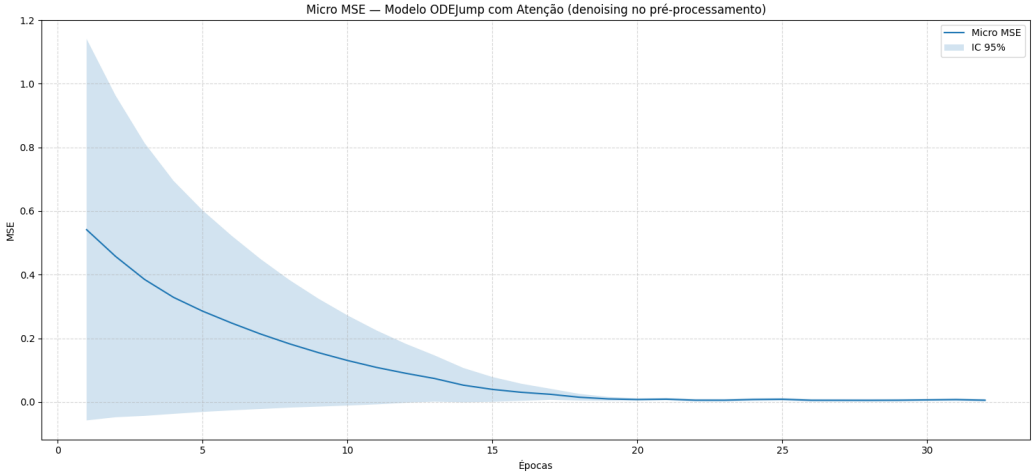
Evolução do treinamento

6 Resultados



Evolução do treinamento

6 Resultados



Conclusões

7 Conclusões

- **Arquitetura**: o pipeline **Encoder ODEJump** (dinâmica contínua + saltos GRU) $\rightarrow$ **Transformer** (atenção temporal) $\rightarrow$ **TSDiffusion** (denoiser gaussiano) mostrou-se **eficaz** para séries irregulares e com faltas.
- **Codificações de tempo**: o **TimeHybrid Encoding** + **FiLM** estabilizou o aprendizado ao modular a representação temporal de acordo com o passo de difusão e o tempo contínuo.
- **Cabeças auxiliares**:
  - **Noise Head** melhorou o *denoising* no latente (consistência com o passo reverso do DDPM).
  - **Miss Head** regularizou a *missingness*, ajudando a reconstrução apenas onde havia alvo.
  - **Lambda Head** forneceu um **mapa de confiança** por timestep, útil para ponderar erros e interpretação.
- **Balanceamento**: a agregação dos estados $\{1,2,3,4\}$ em um único grupo e a manutenção de $\{0\}$ e $\{5\}$ separadamente reduziram o desbalanceamento e facilitaram o **oversampling** estratificado no treino.
- **Resultados**: a combinação com **entrada denoised** alcançou **macro-MSE** e **micro-MSE** baixos, com melhora consistente ao aumentar a capacidade (dimensão/camadas) até um patamar de saturação.

Limitações e Trabalhos Futuros

7 Conclusões

- **Limitações**:
  - Sensibilidade a **hiperparâmetros** (nº de passos de denoising, dimensão por canal, camadas/heads).
  - **Estados raros**: mesmo com agregação, a incerteza permanece alta onde há poucos exemplos; o **SE macro** reflete variação entre grupos.
  - Modelo se limitou a desenvolver um preditor generativo, não tendo como alvo predições de **mudança de estado**.
- **Extensões**:
  - **Avaliação de probabilidade de mudança de estado no tempo**: Implementar mecanismo de previsão de mudança de estado calibrado pelo nível de certeza.
  - **Diffusion guiado** por priors físicos/operacionais do equipamento (consistência de limites e balanços).
  - **Uncertainty-aware decoding**: ponderar reconstruções por $\hat{\lambda}_t$ e calibrar incertezas (p. ex., *temperature scaling*).

Referências Bibliográficas

8 Referências Bibliográficas

[1] Y. Li, “Ts-diffusion: Generating highly complex time series with diffusion models,” *arXiv preprint arXiv:2311.03303*, 2023.

[2] A. Nichol and P. Dhariwal, “Improved denoising diffusion probabilistic models,” 2021.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, vol. 30, 2017.