A HALF-DAY SHORT COURSE

Time Series Forecasting

From VAR to Transformers and Diffusion

Attention · Patching · Foundation Models · Denoising Diffusion

Weining Wang

University of Bristol · Nova Math Week

Course Roadmap (about 4.5 hours)

0:00 – 0:50	**Module 1 — Classical Multivariate Models**	VAR/VARMA, impulse response, Granger causality, cointegration & ECM
0:50 – 1:50	**Module 2 — The Transformer Era**	From VAR to attention, attention math, Informer to PatchTST
1:50 – 2:45	**Module 3 — Foundation Models**	Zero-shot forecasting, tokenizing time, the 2026 model zoo, live demo
2:45 – 2:55	**Break**	Coffee
2:55 – 3:55	**Module 4 — Diffusion Models**	Gaussian densities to scenarios, TimeGrad, CSDI/SSSD, the frontier
3:55 – 4:45	**Module 5 — Hands-On & Practice**	Six coding exercises, evaluation, model selection, open problems

What You Will Be Able to Do

1. **Specify and estimate** classical multivariate models — VAR/VARMA — and read impulse-response functions, Granger-causality tests, and cointegration/ECM.
2. **Formulate** point and probabilistic forecasting problems and pick the right metrics (MASE, CRPS, quantile loss).
3. **Explain** self-attention and how transformers were adapted to time series — and why a linear model briefly beat them all.
4. **Use** zero-shot foundation models (Chronos-2, TimesFM 2.5, Moirai-2) and know when they are the right tool.
5. **Derive** the intuition behind denoising diffusion and map it to forecasters such as TimeGrad, CSDI, and TSDiff.
6. **Decide** which model class fits a given scenario, and backtest it without fooling yourself.

MODULE 1 0:00 – 0:50

Classical Multivariate Models

VAR & VARMA · impulse response · Granger causality · cointegration & ECM

From Univariate AR to VAR

A univariate AR(p) predicts a series from its own past:

$$y_t = \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \varepsilon_t.$$

But series rarely move alone — interest rates, demand, and prices co-evolve.

A **vector autoregression** lets every variable depend on the recent past of *all* variables, capturing feedback and co-movement.

Why multivariate?

- cross-series predictive power (does x help forecast y?),
- shock propagation through a system,
- long-run equilibria between series.

These three questions map onto the three tools in this module: Granger causality, impulse response, and cointegration.

VAR(p): Definition and Estimation

For an $n \times 1$ vector y_t (zero mean):

$$\mathrm{y}_t = A_1 \mathrm{y}_{t-1} + \cdots + A_p \mathrm{y}_{t-p} + \varepsilon_t, \qquad \varepsilon_t \sim \text{iid } \mathcal{N}(0, W).$$

Estimation

- Each equation has the *same* regressors (the stacked lags) $\Rightarrow$ estimate **equation-by-equation OLS**.
- W may be non-diagonal: shocks are contemporaneously correlated.

Stability / stationarity

- Stable if all roots of $\det(I - A_1 z - \cdots - A_p z^p) = 0$ lie *outside* the unit circle.
- Then a vector MA(∞) representation exists — the basis for impulse response.

Lab preview: in Exercise 2 you simulate a VAR(1), recover A by OLS ($\hat{A} \approx A$), and read off its impulse responses.

VARMA Models

Adding moving-average terms gives the general **VARMA(p, q)**:

$$\mathrm{y}_t = \underbrace{A_1 \mathrm{y}_{t-1} + \cdots + A_p \mathrm{y}_{t-p}}_{\text{autoregressive}} + \varepsilon_t + \underbrace{B_1 \varepsilon_{t-1} + \cdots + B_q \varepsilon_{t-q}}_{\text{moving average}}.$$

- $\varepsilon_t \sim \text{iid}\, \mathcal{N}(0, W)$, with W possibly non-diagonal (contemporaneous correlation).
- Assumes *no* serial correlation in residuals and conditional homoskedasticity (no GARCH).
- More parsimonious than a long pure VAR, but estimation is harder (the MA part is nonlinear).

In practice a well-specified VAR(p) is often preferred for its simple OLS estimation; VARMA matters when the MA structure is strong.

Impulse-Response Analysis

Idea. How do shocks propagate? Invert a stable VAR into a vector MA(∞):

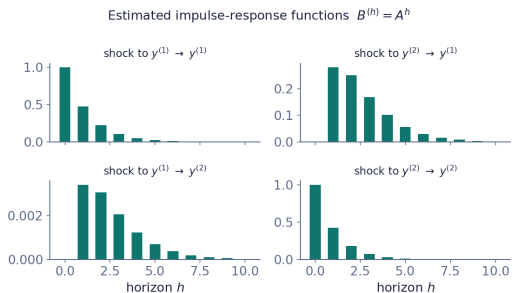
$$y_t = \varepsilon_t + B_1\varepsilon_{t-1} + B_2\varepsilon_{t-2} + \cdots$$

The matrices B_h are built recursively from the A_i.

The **impulse-response function** plots

$$B_{ij}^{(h)} = \text{effect on } y_{t+h}^{(i)} \text{ of a unit shock to } \varepsilon_t^{(j)},$$

tracing one variable's response to another's shock across the horizon h.



Estimated IRFs for a fitted VAR(1). Bottom-left is ≈ 0: a shock to $y^{(1)}$ barely moves $y^{(2)}$ — a preview of Granger non-causality.

Granger Causality

Question. Do past values of x_t help predict y_t, beyond y's own past?

$$y_t = a_1 y_{t-1} + \cdots + a_p y_{t-p} + b_1 x_{t-1} + \cdots + b_p x_{t-p} + u_t.$$

Null hypothesis (no causality $x \to y$):

$$H_0 : \; b_1 = b_2 = \cdots = b_p = 0.$$

Tested with an F- (Wald) test on the joint restriction.

Lab: on the simulated system you will find $F_{y_1 \to y_2} \approx 0$ (cannot reject H_0) but $F_{y_2 \to y_1} \gg 0$ — recovering the asymmetry built into A.

"Granger causality" is predictive, not structural — it says x helps forecast y, not that x *causes* y in a deeper sense.

Unit Roots, Cointegration, and the ECM

Unit roots. $y_t \sim I(1)$ if Δy_t is stationary. Regressing one $I(1)$ series on another risks *spurious regression*.

Cointegration. If $\mathrm{y}_t \sim I(1)$ but a linear combination $\boldsymbol{\beta}^\top \mathrm{y}_t$ is $I(0)$, the series share a long-run equilibrium; $\boldsymbol{\beta}$ is a cointegrating vector.

Error-correction model. A cointegrated system is a VAR in differences plus an equilibrium term:

$$\Delta \mathrm{y}_t = -\alpha \boldsymbol{\beta}^\top \mathrm{y}_{t-1} + \sum_j B_j \Delta \mathrm{y}_{t-j} + \varepsilon_t.$$

- $\boldsymbol{\beta}$: cointegrating vectors (long run),
- α: speed of adjustment back to equilibrium.

Key implication: if y_t is cointegrated, a VAR in differences *alone* is mis-specified — you must keep the error-correction term. We revisit this when comparing to neural models, which do not enforce it.

Bridge: From VAR to Attention

Everything that follows generalizes the VAR you just saw.

Classical — VAR(p)

$$\mathrm{y}_t = \sum_{i=1}^{p} A_i \, \mathrm{y}_{t-i} + \varepsilon_t$$

- Coefficients A_i *fixed*, estimated by OLS.
- *Linear*, finite-lag combination of the past.

Modern — a transformer

$$\hat{\mathrm{y}}_t = \sum_{i=1}^{L} \underbrace{a_i(\mathrm{y}_{t-L:t-1})}_{\text{data-dependent}} \, v(\mathrm{y}_{t-i})$$

- Weights a_i *recomputed for every window*.
- *Nonlinear* combination of learned values.

Attention weights play the role of the VAR coefficients A_i — but data-dependent instead of fixed. A causal mask keeps attention backward-looking, exactly like a VAR. **On to Module 2.**

MODULE 2 0:50 – 1:50

The Transformer Era

From VAR to attention · attention math · Informer to PatchTST

The Forecasting Problem

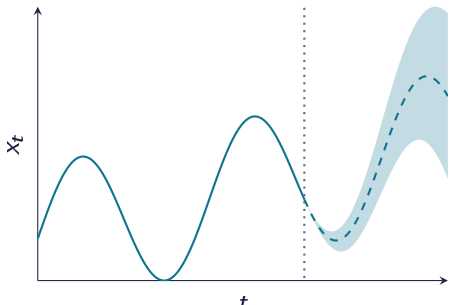
Given history $\mathrm{x}_{1:T} \in \mathbb{R}^{T \times D}$ and covariates c, predict the next H steps.

- **Point forecast:** $\hat{\mathrm{x}}_{T+1:T+H} = f(\mathrm{x}_{1:T}, \mathrm{c})$
- **Probabilistic forecast:** model the full predictive distribution

$$p(\mathrm{x}_{T+1:T+H} \mid \mathrm{x}_{1:T}, \mathrm{c})$$

- Real data brings trend, multiple seasonalities, regime changes, missing values, and cold starts.

The whole course is a story about ever-richer models of this conditional distribution.



History, median forecast, and uncertainty fan

Baselines You Must Beat First

Statistical

Seasonal naive: repeat last season, $\hat{y}_t = y_{t-m}$. The bar every model must clear.

ETS: exponential smoothing with Error/Trend/Seasonality components.

ARIMA

Theta: split into a trend line and an exaggerated-curvature line, forecast each, average; won the M3 competition.

Fast, interpretable, shockingly hard to beat on a single series.

Tree-based ML

Gradient boosting on lag and calendar features.

Won the M5 competition; still a production workhorse.

Early deep learning

DeepAR: an RNN (LSTM) that outputs a full distribution, not a point.

N-BEATS: deep stacks of MLP blocks with trend/seasonality basis expansions; interpretable, no recurrence.

N-HiTS: adds multi-rate sampling to N-BEATS for efficient long-horizon forecasts.

Global models — they shine when you have many *related* series to learn from jointly.

Lesson from the M-competitions: report a seasonal-naive and an ETS baseline next to every deep model. If you can't beat them, you don't have a model — you have overhead.

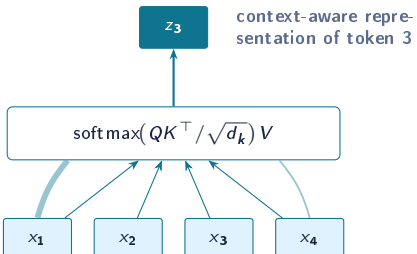
Transformer Basics I: Self-Attention

Each token emits a **query**, a **key**, and a **value**:

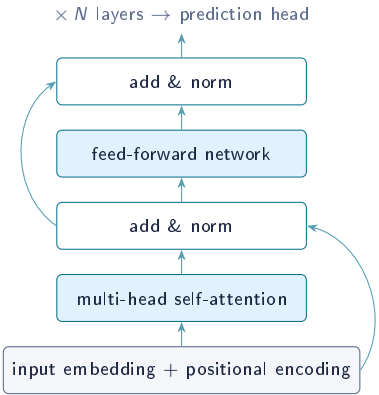
$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

$$\mathrm{Attn}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V$$

- Every output is a *data-dependent weighted average* of all values.
- No recurrence: all pairs interact in one step — long-range dependencies become easy.
- Cost is $\mathcal{O}(L^2)$ in sequence length L — this will matter for long histories.



Transformer Basics II: The Full Block



- **Multi-head:** h attention maps in parallel subspaces; different heads can track different periodicities.
- **Positional encoding:** attention is permutation-invariant, so order must be injected — sinusoidal, learned, or rotary; for time series often enriched with calendar features (hour, weekday, holiday).
- **Residuals + normalization:** make deep stacks trainable.
- **Decoder-only vs encoder–decoder:** autoregressive generation vs direct multi-horizon output — both appear in forecasting models.

Open question for time series: what is a "token"? A single timestep? A patch? A variable? Each answer defines a model family.

Transformer Math I: One Head, Step by Step

Input sequence $X \in \mathbb{R}^{L \times d}$ (L tokens, model width d). Project into three roles:

$$Q = XW_Q, \quad K = XW_K \in \mathbb{R}^{L \times d_k}, \qquad V = XW_V \in \mathbb{R}^{L \times d_v}$$

1. Scores — compatibility of every query with every key:

$$S = \frac{QK^\top}{\sqrt{d_k}} \in \mathbb{R}^{L \times L}, \qquad S_{ij} = \frac{\mathrm{q}_i^\top \mathrm{k}_j}{\sqrt{d_k}}$$

2. Weights — row-wise softmax, so each row sums to 1:

$$A_{ij} = \frac{\exp(S_{ij})}{\sum_{j'=1}^{L} \exp(S_{ij'})}$$

3. Mix — each output is a convex combination of values:

$$\mathrm{z}_i = \sum_{j=1}^{L} A_{ij}\, \mathrm{v}_j, \qquad Z = AV$$

Why divide by $\sqrt{d_k}$?

If entries of q, k are unit-variance and independent, $\mathrm{q}^\top \mathrm{k}$ has variance d_k. Without scaling, large dot products push softmax into saturated regions where gradients vanish. Dividing by $\sqrt{d_k}$ keeps scores at unit scale.

Causal masking: for autoregressive forecasting, set $S_{ij} = -\infty$ for $j > i$ so a token attends only to its past.

Transformer Math II: Multi-Head and the Full Layer

Multi-head attention runs h independent heads in parallel and concatenates:

$$\mathrm{head}_m = \mathrm{Attn}(XW_Q^m, XW_K^m, XW_V^m), \qquad \mathrm{MHA}(X) = \big[\mathrm{head}_1 \,\|\, \cdots \,\|\, \mathrm{head}_h\big]\, W_O$$

Each head uses $d_k = d/h$, so total cost matches one full-width head while letting heads specialize (e.g. different seasonal lags).

One transformer layer (pre-norm form):

$$X' = X + \mathrm{MHA}\big(\mathrm{LN}(X)\big)$$
$$Y = X' + \mathrm{FFN}\big(\mathrm{LN}(X')\big)$$

with a position-wise feed-forward network

$$\mathrm{FFN}(\mathrm{u}) = W_2\, \sigma(W_1 \mathrm{u} + \mathrm{b}_1) + \mathrm{b}_2$$

σ is GELU/ReLU; the hidden width is usually $4d$.

Sinusoidal positional encoding injects order:

$$\mathrm{PE}_{t,2i} = \sin\Big(\tfrac{t}{10000^{2i/d}}\Big), \quad \mathrm{PE}_{t,2i+1} = \cos\Big(\tfrac{t}{10000^{2i/d}}\Big)$$

added to token embeddings so attention can read relative offsets.

Cost: scores S are $L \times L$, so compute and memory are $\mathcal{O}(L^2 d)$ — the bottleneck every long-horizon forecasting variant attacks.

Why Time Series Stress-Test the Transformer

- **Quadratic cost:** hourly data with a one-year context is $L \approx 8{,}760$ — L^2 attention gets expensive fast.
- **Weak tokens:** a single scalar timestep carries almost no semantics (unlike a word); attention over point-wise tokens is noisy.
- **Channel strategy:** mix variables in one token (channel mixing) or model each variate independently (channel independence)?
- **Distribution shift:** train/test statistics drift; normalization (e.g. RevIN) becomes a first-class design choice.

2019–2023 in one sentence: the field tried efficient attention first, got humbled by a linear model, then fixed the token definition.

From VAR to Attention

For anyone coming from classical multivariate time series, attention is a familiar idea in disguise.

Classical — VAR(p)

$$\mathrm{y}_t = \sum_{i=1}^{p} A_i \, \mathrm{y}_{t-i} + \varepsilon_t$$

- Coefficient matrices A_i are *fixed* — estimated once by OLS.
- Forecast is a *linear*, finite-lag combination of the past.

Modern — a transformer

$$\hat{\mathrm{y}}_t = \sum_{i=1}^{L} \underbrace{a_i(\mathrm{y}_{t-L:t-1})}_{\text{data-dependent}} \, v(\mathrm{y}_{t-i})$$

- Weights $a_i = \mathrm{softmax}(QK^\top/\sqrt{d_k})$ are *recomputed for every window*.
- Forecast is a *nonlinear* combination of learned value vectors.

One sentence: attention weights play the role of the VAR coefficients A_i — but they vary with the input instead of being fixed, and a causal mask makes attention strictly backward-looking like a VAR.

Reading the Model: IRF and Granger $\leftrightarrow$ Attention

Impulse response

Classical: $B_{ij}^{(h)}$ = effect on $y_{t+h}^{(i)}$ of a unit shock to $\varepsilon_t^{(j)}$, from the MA(∞) inversion.

Modern analog: attention / gradient-sensitivity maps show which past inputs drive a forecast — learned, nonlinear, horizon-dependent.

Granger causality

Classical: does x's past help predict y? F-test of $H_0 : b_1 = \cdots = b_p = 0$.

Modern analog: multivariate models (iTransformer) attend *across variables*; cross-variable weights measure predictive relevance.

Caveat worth teaching: attention is *associational*, not structural — it is not an orthogonalized shock and is not a hypothesis test. For causal effects use interventions or gradient sensitivities; for inference, keep the Granger F-test or add ablation importance.

The Efficiency Race (2019–2022)

Model	Key idea	Contribution
Informer AAAI 2021	ProbSparse attention: keep only the most informative queries	$\mathcal{O}(L \log L)$ attention; generative-style decoder for one-shot long-horizon output
Autoformer NeurIPS 2021	Series decomposition blocks + auto-correlation instead of dot-product	Builds trend/seasonal inductive bias directly into the architecture
FEDformer ICML 2022	Attention in the frequency domain (Fourier/wavelet bases)	Linear complexity; global spectral view of the series

All three beat vanilla transformers on the long-horizon benchmarks (ETT, Electricity, Traffic, Weather)... which set the stage for an uncomfortable question.

2022: "Are Transformers Effective for Time Series Forecasting?"

DLinear (Zeng et al., AAAI 2023): decompose into trend + seasonal, fit *one linear layer* per component, map history directly to horizon.

- Outperformed Informer, Autoformer, and FEDformer on most long-horizon benchmarks.
- Diagnosis: point-wise tokens lose temporal semantics; iterated decoding accumulates error; benchmarks rewarded smooth extrapolation.
- Not the end of transformers — the end of *naive* transformers.

1 layer

linear model, no attention

> 3 SOTA

transformers on 8/9 datasets

Takeaway for practitioners: complexity must earn its keep — always run the linear baseline.

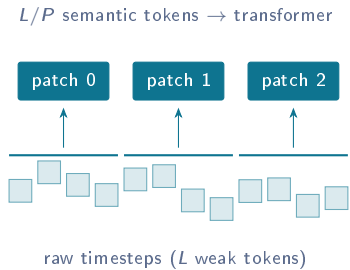
The Fix: Patches as Tokens

PatchTST (ICLR 2023)

- Split each series into patches of P steps; embed each patch as one token.
- Tokens become *meaningful local shapes*; context length shrinks by P, so attention cost drops by P^2.
- Channel independence: one shared backbone applied per variate.

iTransformer (ICLR 2024): invert the axes — treat each *variate* as a token and attend across variables.

Patching restored transformer SOTA — and became the input layer of nearly every foundation model in Module 3.



MODULE 3 1:50 – 2:45

Time Series Foundation Models

One pretrained model, any series · tokenizing time · the 2026 model zoo

The Zero-Shot Paradigm Shift

Before

One model per dataset (or per series).

Feature engineering, hyperparameter tuning, retraining pipelines.

Cold-start series are a hard problem.

After

One network pretrained on huge, diverse corpora (e.g. LOTSA: ∼27B observations) plus synthetic series.

Load it, feed history, get a probabilistic forecast — *zero-shot*.

Forecasting becomes model *selection*, not model *training*.

Direct analogy to LLMs — but time series add nonstationarity, mixed frequencies, irregular sampling, and very different value ranges across domains.

Tokenizing Time: Two Camps

Values as a vocabulary

Scale series, quantize values into discrete bins, train a language model on the tokens.

- Chronos-T5: forecasting literally *as* language modeling; probabilistic forecasts by sampling trajectories.
- Pro: reuses the whole LLM toolbox. Con: quantization error, long sequences.

Real-valued patch embeddings

PatchTST-style patches in, distribution parameters or quantiles out.

- TimesFM: decoder-only over patches; Moirai: any-variate attention, mixture-distribution head; Chronos-Bolt/Chronos-2: patches + direct quantile heads.
- Pro: compact, fast, native uncertainty. Now the dominant design.

Output heads to know: autoregressive sampling · direct quantiles · parametric mixtures · (new) generative flow/diffusion heads — a bridge to Module 4.

The Model Zoo (state of play, 2026)

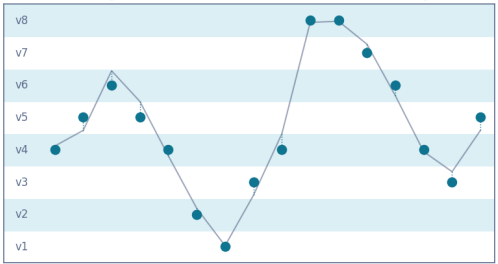
Model	Org	Architecture	Size	Notable for
Chronos-2 (2025)	Amazon	encoder-only, patches	∼120M	Multivariate + covariates, in-context learning, strong zero-shot; successor to Chronos-T5/Bolt
TimesFM 2.5	Google	decoder-only, patches	∼200M	16k context, continuous quantile head, battle-tested in production
TimesFM-3 (2026)	Google	decoder-only, patches	∼330M	Native multivariate pretraining, past + future covariates, 9-quantile head
Moirai-2 (2026)	Salesforce	decoder-only	compact	Any-variate, messy real-world data focus; Moirai-MoE adds sparse experts
Toto	Datadog	decoder-only	∼150M	Observability/DevOps metrics specialization
TimeGPT	Nixtla	encoder–decoder	API	First commercial TSFM (2023); API-only
Lag-Llama	ServiceNow+	decoder-only	small	Lag features as tokens; open probabilistic baseline
Time-MoE / Sundial	THU et al.	MoE / flow head	up to 2.4B	Scaling studies; Sundial pairs a transformer with a generative flow-matching head

Open weights for most (Hugging Face); AutoGluon–TimeSeries and GluonTS wrap several behind one interface.

Tokenizing Time: A Picture

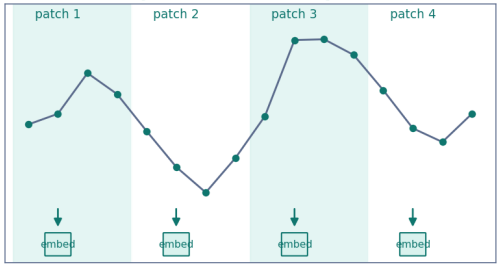
Two ways to turn a time series into tokens

Camp 1 — Tokenize into a vocabulary



scale → quantize each value into a bin → token IDs
then run a language model: v3 v5 v7 v6 ... (Chronos)

Camp 2 — Real-valued patches



group timesteps into patches → embed each as a vector
keep real values; output quantiles directly (TimesFM, Moirai)

Left (Chronos): quantize each value into a fixed bin — the series becomes a sentence of tokens any language model can continue (binning caps resolution). **Right (TimesFM, Moirai)**: keep real values, embed *patches* of timesteps — compact, fast, probabilistic by default.

Do They Actually Work?

- On public leaderboards (GIFT-Eval, fev-bench), top TSFMs **zero-shot beat tuned statistical models and most task-specific deep models** on aggregate — as of September 2026 the GIFT-Eval board carried 127 models, with TimesFM-3 and a Synapse-based arbitration entry among the top entries.
- Fine-tuning helps, but *how much* is now disputed: earlier comparisons found the gain modest and dataset-dependent, while Laglil et al. (2026) report consistent improvement from fine-tuning selected models — zero-shot is still the cheap first thing to try.
- Per-series inference is fast (hundreds of forecasts/sec on one GPU for small variants); small models run on CPU.

Caveats to teach your team: benchmark contamination is hard to rule out for pretrained models · covariate support is still uneven · irregular sampling and very long horizons remain weak spots · domain shift (e.g. finance) can erase the advantage — always backtest on *your* data.

Live Demo: Zero-Shot in a Few Lines

```
# pip install chronos-forecasting
import pandas as pd, torch
from chronos import BaseChronosPipeline

pipe = BaseChronosPipeline.from_pretrained(
    "amazon/chronos-2")

y = pd.read_csv("electricity.csv")["load"]
ctx = torch.tensor(y.values[-512:])

q, mean = pipe.predict_quantiles(
    context=ctx, prediction_length=48,
    quantile_levels=[0.1, 0.5, 0.9])
```

What to look at during the demo

- Forecast quality with *zero* training.
- Calibration of the 80% interval over a rolling backtest.
- Comparison against seasonal-naive and AutoETS on the same split.
- Runtime on CPU vs GPU.

We repeat this pattern in the Module 5 lab with your own data.

MODULE 4 2:55 – 3:55 (AFTER A 10-MINUTE BREAK)

Diffusion Models for Forecasting

Denoising math from scratch · TimeGrad · CSDI & SSSD · the generative frontier

Why Generative Models for Forecasting?

- A forecast is a **distribution over futures**, not a number. Decisions (inventory, trading, staffing) consume quantiles and scenarios.
- Quantile heads give marginal uncertainty per timestep, but **no coherent joint trajectories**: you cannot ask "what does a plausible bad week look like?"
- Generative models sample *whole future paths*: correlated across time and across variables — exactly what risk analysis, scenario planning, and downstream simulation need.
- Diffusion models are the current best-in-class generative family (images, audio, video) — Module 4 ports them to time series.

From Gaussian VARMA Densities to Sampled Scenarios

Classical — Gaussian VARMA

$$y_{t+h} \mid \mathcal{F}_t \sim \mathcal{N}(\hat{y}_{t+h}, \Sigma_h)$$

- Closed-form density, Σ_h built from the shock covariance W.
- But the shape is *fixed to be Normal* — no skew, no multi-modality.

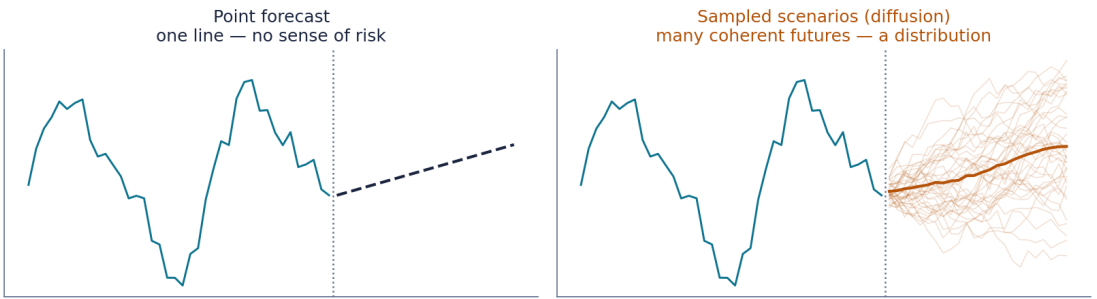
Modern — diffusion

$$y_{t+1:t+H} \sim p_\theta(\cdot \mid \mathcal{F}_t)$$

- *Learn to sample* whole future paths — no Gaussian assumption.
- Multi-modal, heavy-tailed, and *coherent* across time and variables.

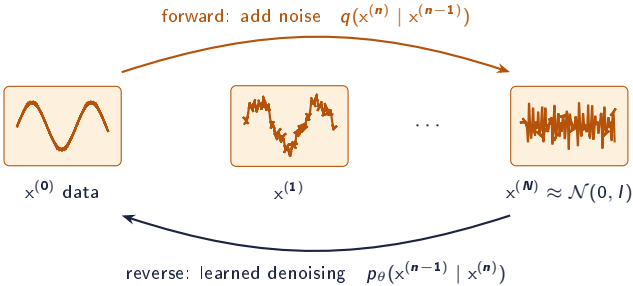
Link: diffusion learns the joint predictive law that a Gaussian VARMA approximates with a single covariance Σ_h. **But** neither neural nor foundation models enforce $I(1)$ structure or a cointegrating equilibrium $\beta^\top y_t = 0$ — when a known long-run relation must hold, a classical **ECM** is still the right tool.

Why Generative: A Picture



A point forecast is one line with no sense of risk. A diffusion model samples *many coherent future paths* — a distribution you can read quantiles and worst-cases from. **This is what generative forecasting buys you.**

Diffusion Basics I: Destroy, Then Learn to Rebuild



- **Forward process (fixed):** $q(x^{(n)} \mid x^{(n-1)}) = \mathcal{N}\big(\sqrt{1-\beta_n}\,x^{(n-1)},\, \beta_n I\big)$ — gradually drown the signal in Gaussian noise.
- **Shortcut:** $x^{(n)} = \sqrt{\bar{\alpha}_n}\,x^{(0)} + \sqrt{1-\bar{\alpha}_n}\,\epsilon$, with $\bar{\alpha}_n = \prod_{i \le n}(1-\beta_i)$ — jump to any noise level in one step.
- **Generation:** start from pure noise, apply the learned reverse steps to synthesize new data.

Diffusion Basics II: Training and Sampling

Training reduces (after simplifying the ELBO) to noise prediction:

$$\mathcal{L}(\theta) \;=\; \mathbb{E}_{\mathrm{x}^{(0)},\, n,\, \boldsymbol{\epsilon}\sim\mathcal{N}(0,I)} \Big\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\big(\sqrt{\bar{\alpha}_n}\,\mathrm{x}^{(0)} + \sqrt{1-\bar{\alpha}_n}\,\boldsymbol{\epsilon},\; n\big) \Big\|^2$$

i.e. a denoising network learns "which noise was added?" at every corruption level. Equivalent view: it learns the *score* $\nabla_{\mathrm{x}} \log p(\mathrm{x})$ — the direction back toward the data manifold.

Sampling

- Iterate $\mathrm{x}^{(n)} \to \mathrm{x}^{(n-1)}$ using $\boldsymbol{\epsilon}_\theta$; classic DDPM uses 50–1000 steps.
- Fast samplers (DDIM, DPM-Solver, distillation, flow matching) cut this to a handful of steps.

Conditioning — the key for forecasting

- Make the denoiser $\boldsymbol{\epsilon}_\theta(\mathrm{x}^{(n)}, n, \mathrm{c})$ where c encodes the observed history (and covariates).
- Classifier-free guidance: train with and without c, interpolate at sampling time to sharpen conditional samples.

Forecasting as conditional generation: sample $\mathrm{x}_{T+1:T+H} \sim p_\theta(\cdot \mid \mathrm{x}_{1:T})$ many times $\Rightarrow$ an empirical predictive distribution of full trajectories.

Diffusion Math I: The Forward Process in Closed Form

A fixed Markov chain adds Gaussian noise over N steps with schedule $\beta_1, \ldots, \beta_N$:

$$q(\mathrm{x}^{(n)} \mid \mathrm{x}^{(n-1)}) = \mathcal{N}\big(\mathrm{x}^{(n)};\ \sqrt{1-\beta_n}\,\mathrm{x}^{(n-1)},\ \beta_n I\big)$$

Let $\alpha_n = 1 - \beta_n$ and $\bar{\alpha}_n = \prod_{i=1}^{n} \alpha_i$. Composing Gaussians gives a one-shot jump to any level:

$$\boxed{q(\mathrm{x}^{(n)} \mid \mathrm{x}^{(0)}) = \mathcal{N}\big(\sqrt{\bar{\alpha}_n}\,\mathrm{x}^{(0)},\ (1-\bar{\alpha}_n) I\big)}$$

$$\Rightarrow \quad \mathrm{x}^{(n)} = \sqrt{\bar{\alpha}_n}\,\mathrm{x}^{(0)} + \sqrt{1-\bar{\alpha}_n}\,\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$$

Why this matters

Training never simulates the whole chain: pick a random step n, draw one $\boldsymbol{\epsilon}$, and form $\mathrm{x}^{(n)}$ directly. As $n \to N$, $\bar{\alpha}_n \to 0$ and $\mathrm{x}^{(N)} \approx \mathcal{N}(0, I)$ — a tractable prior to start sampling from.

The true reverse step is also Gaussian when conditioned on $\mathrm{x}^{(0)}$:
$q(\mathrm{x}^{(n-1)} \mid \mathrm{x}^{(n)}, \mathrm{x}^{(0)}) = \mathcal{N}(\tilde{\boldsymbol{\mu}}_n, \tilde{\beta}_n I)$ — the target the network learns to match.

Diffusion Math II: Reverse Process and Objective

The model learns the reverse chain with a denoising network $\boldsymbol{\epsilon}_\theta$:

$$p_\theta(\mathrm{x}^{(n-1)} \mid \mathrm{x}^{(n)}) = \mathcal{N}\big(\boldsymbol{\mu}_\theta(\mathrm{x}^{(n)}, n),\, \Sigma_n\big), \qquad \boldsymbol{\mu}_\theta = \frac{1}{\sqrt{\alpha_n}}\Big(\mathrm{x}^{(n)} - \tfrac{\beta_n}{\sqrt{1-\bar{\alpha}_n}}\,\boldsymbol{\epsilon}_\theta(\mathrm{x}^{(n)}, n)\Big)$$

Variational bound. Maximizing the data likelihood reduces to a sum of KL terms matching p_θ to the true Gaussian posteriors:

$$\mathcal{L}_{\mathrm{vlb}} = \mathbb{E}_q\Big[\textstyle\sum_n \mathrm{KL}\big(q(\mathrm{x}^{(n-1)} \mid \mathrm{x}^{(n)}, \mathrm{x}^{(0)}) \,\|\, p_\theta\big)\Big]$$

Simplified loss (DDPM). Reparameterizing collapses each KL to a plain regression on the injected noise:

$$\boxed{\mathcal{L} = \mathbb{E}_{\mathrm{x}^{(0)}, n, \boldsymbol{\epsilon}}\big\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathrm{x}^{(n)}, n)\big\|^2}$$

Score-based view. Predicting noise is (up to scale) estimating the score:

$$\boldsymbol{\epsilon}_\theta(\mathrm{x}^{(n)}, n) \approx -\sqrt{1-\bar{\alpha}_n}\,\nabla_{\mathrm{x}} \log q(\mathrm{x}^{(n)})$$

Sampling = following the score back to the data manifold, step by step.

Sampling step: $\mathrm{x}^{(n-1)} = \boldsymbol{\mu}_\theta(\mathrm{x}^{(n)}, n) + \sqrt{\Sigma_n}\,\mathrm{z}$, $\mathrm{z} \sim \mathcal{N}(0, I)$, from $n = N$ down to 1.

The Diffusion Idea, Visually

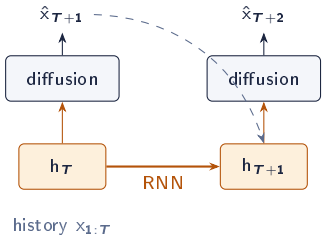
Left to right, a *fixed* forward process buries the series in noise. A network learns to run it *backwards* (right to left). Generation starts from pure noise and denoises into a sample. **Forecasting = condition that denoiser on the observed past.**

TimeGrad (2021): The First Diffusion Forecaster

- An RNN summarizes history into hidden state h_t; a **conditional diffusion head** models $p(x_{t+1} \mid h_t)$.
- Applied **autoregressively**: generate one multivariate step, feed it back, repeat over the horizon.
- Strong CRPS on high-dimensional multivariate benchmarks (e.g. thousands of correlated series).

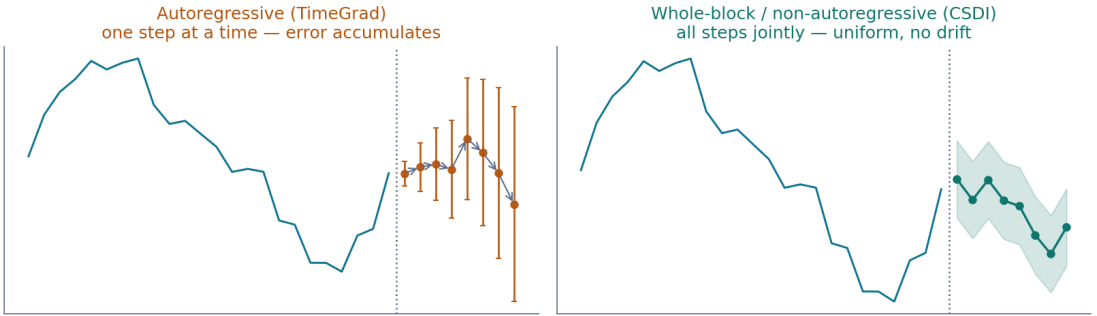
Limitations

- Slow: a full reverse diffusion *per timestep*.
- Error accumulation over long horizons.



One reverse-diffusion sampling loop per forecast step, conditioned on the recurrent state.

Autoregressive vs. Whole-Block



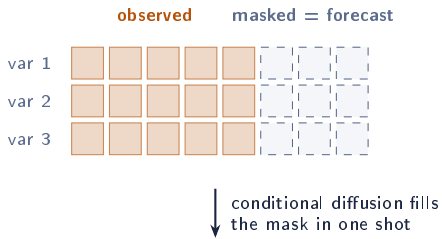
TimeGrad generates one step at a time, so the uncertainty band *grows* and errors compound. Generating the whole horizon *jointly* (next slide) keeps the band uniform and avoids drift — the key move behind CSDI/SSSD.

CSDI & SSSD: Forecasting as Imputation

CSDI (NeurIPS 2021)

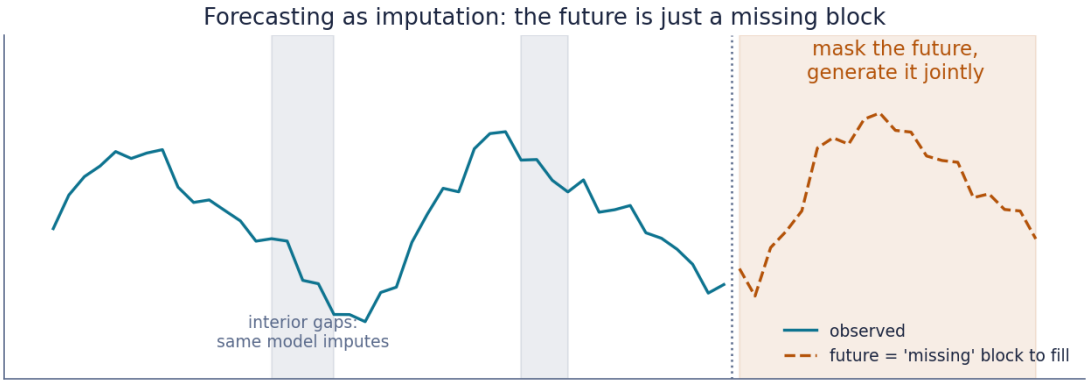
- Treat the future as **missing values**: mask it, condition the diffusion on the observed part, generate the whole masked block *non-autoregressively*.
- Two-dimensional attention over time and features; one model serves imputation *and* forecasting.

SSSD (2022): same recipe, but transformer layers replaced by **S4 state-space layers** — cheaper on long sequences.



No step-by-step decoding ⇒ no error accumulation; handles arbitrary missingness patterns for free.

Forecasting as Imputation



The trick behind CSDI/SSSD: treat the *future as a missing block*, then generate it jointly, conditioned on the observed past. The same model fills interior gaps too — forecasting and imputation become one problem.

Sharper Conditioning: TimeDiff and TSDiff

TimeDiff (ICML 2023)

Non-autoregressive, with *time-series-aware* conditioning:

Future mixup — blend ground-truth future into the condition during training.

AR initialization — start the reverse process from a cheap linear forecast instead of pure noise.

Faster and more accurate than masking-style designs on long horizons.

TSDiff (NeurIPS 2023)

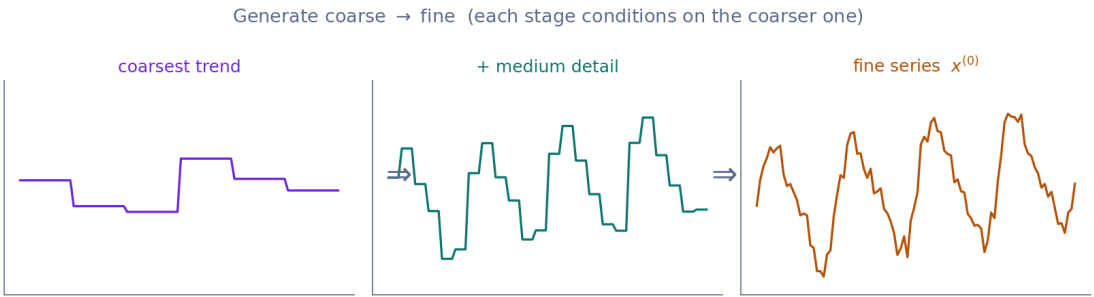
One *unconditional* diffusion model per domain.

Self-guidance at inference conditions it on any observation pattern — forecasting, imputation, refinement — without retraining.

Doubles as a generator of **synthetic training data** for downstream models.

Design axes to remember: autoregressive vs whole-segment · where conditioning enters · data space vs latent space · how many sampling steps you can afford.

Coarse-to-Fine Generation



Multi-scale diffusion (MG-TSD, mr-Diff) generates the *coarse trend first*, then adds detail — because noising a series resembles smoothing it. Get the big shape right, then fill in the wiggles. **Easy-to-hard, and more stable.**

The Generative Frontier (2024–2026)

- **Multi-scale & decomposition:** MG-TSD, mr-Diff — denoise coarse-to-fine.
- **Non-stationary diffusion:** NsDiff (ICML 2025) adapts the noise process to drifting variance.
- **Latent & multimodal:** diffuse in a learned latent space; condition on text or vision (LDM4TS, MCD-TSF).
- **Few-step sampling:** DDIM/DPM-Solver, distillation, and **flow matching** make generation cheap enough for production.
- **Convergence with Module 3:** generative heads inside foundation models (e.g. Sundial's flow-matching head) — pretrained backbone, sampled trajectories.

Synthesis: transformers solved *representation*; foundation models solved *generalization*; diffusion solves *uncertainty*. The frontier is all three in one model.

MODULE 5 3:55 – 4:45

Hands-On & Practice

Lab · honest evaluation · choosing a model · open problems

Hands-On Lab — Six Exercises (25 min)

1. **Build the data** — synthetic hourly load, or load your own CSV; train/test split.
2. **Classical VAR** — simulate a VAR(1), recover A by OLS, run a Granger F-test, plot the IRF.
3. **Baselines & MASE** — seasonal-naive with a backtest-based 80% interval; add AutoETS.
4. **Self-attention from scratch** — 12 lines of numpy; add a causal mask and multiple heads.
5. **Diffusion forward process** — noise a window with the closed-form q; stub the reverse step.
6. **Probabilistic scoring** — sample paths, CRPS, coverage; then beat it with Chronos-2 zero-shot.

Everything runs on **CPU** with `numpy + matplotlib` (`exercises.py`). The zero-shot stretch goal adds `pip install chronos-forecasting gluonts statsforecast`.

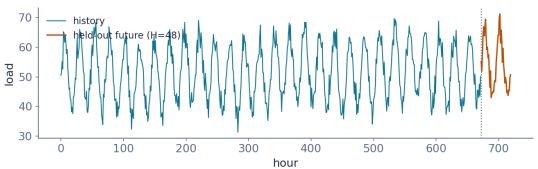
Exercise 1 — Build the Dataset

```python
import numpy as np

def make_series(n=24*30, seed=2):
    rng = np.random.default_rng(seed)
    t = np.arange(n)
    trend  = 50 + 0.005*t
    daily  = 12*np.sin(2*np.pi*t/24)
    weekly =  2*np.sin(2*np.pi*t/(24*7))
    noise  = rng.normal(0, 2.5, n)
    return t, trend+daily+weekly+noise

t, y = make_series()
H = 48                              # horizon
y_train, y_test = y[:-H], y[-H:]
```

Your turn: swap in a real CSV (`pd.read_csv`); identify the dominant seasonal period with an autocorrelation plot.



Hourly synthetic load: trend + daily + weekly + noise. The last 48 hours (amber) are held out for every model in this lab.

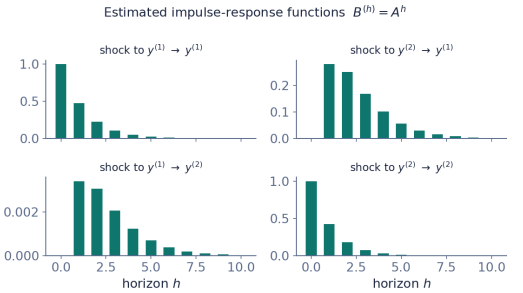
Exercise 2 — Classical VAR(1)

```
A_true = np.array([[0.5, 0.3],
                   [0.0, 0.4]])    # y2 not driven
                                     by y1

def simulate_var1(A, T=400, seed=3):
    rng = np.random.default_rng(seed)
    Y = np.zeros((T, A.shape[0]))
    for s in range(1, T):
        Y[s] = A@Y[s-1] + rng.normal(0,1,A.shape
               [0])
    return Y

def fit_var1(Y):                      # OLS: Y_t = A Y_
    {t-1}
    X, Z = Y[1:], Y[:-1]
    return np.linalg.lstsq(Z, X, rcond=None)[0].T

Y = simulate_var1(A_true)
A_hat = fit_var1(Y)                   # ~ A_true
B = [np.linalg.matrix_power(A_hat, h)
     for h in range(11)]              # IRF: B_h = A^h
```



Your turn: implement the Granger F-test of H_0 : lags of y_1 don't enter the y_2 equation. You should find $F_{y_1 \to y_2} \approx 0$ but $F_{y_2 \to y_1} \gg 0$.

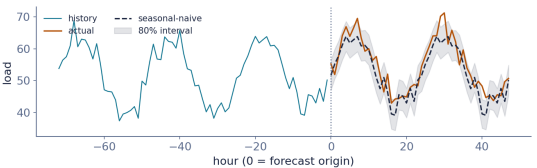
Exercise 3 — Baselines and MASE

```python
m = 24                                  # daily period
sn = y_train[-m:][np.arange(H) % m]

# rolling-origin backtest -> honest spread
def bt_resid(y, m, H, n=120):
    R = []
    for o in range(len(y)-H-n, len(y)-H):
        f = y[o-m:o][np.arange(H) % m]
        R.append(y[o:o+H] - f)
    return np.array(R)

sig = bt_resid(y_train, m, H).std(0)
lo, hi = sn - 1.28*sig, sn + 1.28*sig    # 80%

mase = (np.abs(y_test - sn).mean() /
        np.abs(y_train[m:]-y_train[:-m]).mean())
print(mase)    # ~1.06
```



Your turn: add an AutoETS forecast (`statsforecast`); report MASE for both. Why build the interval from *backtest* residuals, not one-step errors?

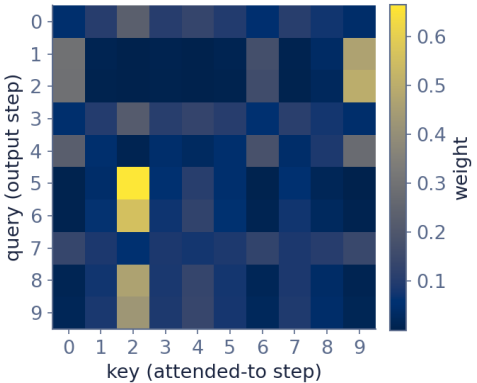
Exercise 4 — Self-Attention from Scratch

```python
def softmax(x, axis=-1):
    x = x - x.max(axis, keepdims=True)
    e = np.exp(x)
    return e / e.sum(axis, keepdims=True)

def attention(X, Wq, Wk, Wv):
    Q, K, V = X@Wq, X@Wk, X@Wv
    dk = Q.shape[-1]
    S = Q @ K.T / np.sqrt(dk)      # scores
    A = softmax(S)                 # weights
    return A @ V, A                # output, map

Z, A = attention(X, Wq, Wk, Wv)
```

Your turn: (1) add a causal mask ($S_{ij} = -\infty$ for $j > i$); (2) split into h heads and concatenate. Confirm each row of `A` sums to 1.



Attention matrix A over 10 daily tokens: bright cells show which past days each output step leans on.

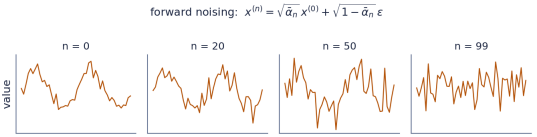
Exercise 5 — Diffusion Forward Process

```
N = 100
betas  = np.linspace(1e-4, 0.08, N)
abar   = np.cumprod(1 - betas)    # alpha-bar

def q_sample(x0, n, rng):
    eps = rng.standard_normal(len(x0))
    return (np.sqrt(abar[n])*x0 +
            np.sqrt(1-abar[n])*eps)

x0 = (y_test - y_test.mean())/y_test.std()
rng = np.random.default_rng(0)
xn = {n: q_sample(x0, n, rng)
      for n in (0, 20, 50, 99)}
```

Your turn: given a trained `eps_theta(x,n)`, implement one reverse step $x^{(n-1)} = \frac{1}{\sqrt{\alpha_n}}(x^{(n)} - \frac{\beta_n}{\sqrt{1-\bar{\alpha}_n}}\epsilon_\theta) + \sqrt{\beta_n}z$ and sample from noise.



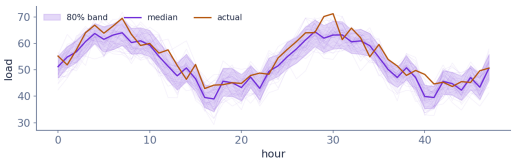
A clean window dissolves into Gaussian noise as $n \to N$. Training asks a network to invert one step of this at a random n.

Exercise 6 — Probabilistic Forecast & Scoring

```
# 200 correlated sample paths (AR(1) noise)
S, phi = 200, 0.6
ar = np.zeros((S, H)); g = np.random.default_rng
     (7)
ar[:,0] = g.normal(0, sig[0], S)
for h in range(1, H):
    e = g.normal(0, sig[h]*np.sqrt(1-phi**2), S)
    ar[:,h] = phi*ar[:,h-1] + e
samples = sn[None,:] + ar
q10,q50,q90 = np.percentile(samples,[10,50,90],0)

def crps(samps, obs):            # sample CRPS
    a = np.abs(samps-obs).mean(0)
    b = np.abs(samps[:,None]-samps[None]).mean
        ((0,1))
    return (a - 0.5*b).mean()

cov = np.mean((y_test>=q10)&(y_test<=q90))
print(crps(samples, y_test), cov)  # ~2.3, ~0.7
```



Your turn: an 80% band covering ∼70% is mildly *under*-calibrated — diagnose and fix. Then beat this CRPS with Chronos-2 zero-shot.

Evaluating Forecasts Without Fooling Yourself

Metrics

- **Point:** MASE (scale-free, robust), sMAPE; never plain MAPE near zeros.
- **Probabilistic:** CRPS, weighted quantile loss, interval coverage — a sharp but miscalibrated model is a liability.

$$\mathrm{CRPS}(F, y) = \int \big(F(z) - 1\{y \le z\}\big)^2 \, dz$$

Backtesting traps

- Use **rolling-origin** evaluation, not one fixed split.
- Fit scalers/normalizers on the training window only.
- Beware overlapping windows inflating sample counts.
- For TSFMs: your "test set" may resemble their pretraining data — validate on genuinely private series.

Choosing a Model in Practice

Your situation	Reach for
A few series, interpretability matters	ETS / ARIMA / Theta
Many related series, rich covariates, tabular culture	Gradient boosting or PatchTST-class model
Cold start, no ML team, fast time-to-value	Foundation model, zero-shot (Chronos-2, TimesFM)
Risk decisions need coherent scenario paths	Diffusion forecaster, or a TSFM with a generative head
Tight latency / edge deployment	Linear (DLinear) or distilled small model

Default workflow: seasonal-naive → statistical baseline → zero-shot TSFM → fine-tune or go generative only if the backtest says the gap is worth it.

Open Problems & Where the Field Is Going

- **Irregular & multimodal data:** events, text, and exogenous signals fused with numeric series.
- **Very long horizons** and hierarchical/grouped coherence constraints.
- **Calibration under drift:** guarantees when the world changes mid-deployment.
- **Benchmark hygiene:** contamination-free evaluation for pretrained models.
- **Arbitration over scaling:** adaptively picking among several complementary pretrained models (Synapse) instead of training one ever-larger model.
- **Agentic pipelines:** LLM agents that select, run, and monitor forecasting models end-to-end.

References I — Classical, Transformers, Foundation Models

Classical multivariate time series

- Sims, *Macroeconomics and Reality*, Econometrica 1980.
- Granger, *Investigating Causal Relations*, Econometrica 1969.
- Engle & Granger, *Co-integration and Error Correction*, Econometrica 1987.
- Johansen, *Estimation and Testing of Cointegration Vectors*, Econometrica 1991.
- Lütkepohl, *New Introduction to Multiple Time Series*, 2005; Hamilton, *Time Series Analysis*, 1994.

Transformers for forecasting

- Vaswani et al., *Attention Is All You Need*, NeurIPS 2017.
- Zhou et al., *Informer*, AAAI 2021; Wu et al., *Autoformer*, NeurIPS 2021; Zhou et al., *FEDformer*, ICML 2022.

Transformers (cont.)

- Zeng et al., *Are Transformers Effective?* (DLinear), AAAI 2023.
- Nie et al., *PatchTST*, ICLR 2023; Liu et al., *iTransformer*, ICLR 2024.
- Salinas et al., *DeepAR*, IJF 2020; Oreshkin et al., *N-BEATS*, ICLR 2020; Kim et al., *RevIN*, ICLR 2022.

Foundation models

- Ansari et al., *Chronos*, TMLR 2024; Das et al., *TimesFM*, ICML 2024; Woo et al., *Moirai* & LOTSA, ICML 2024.
- Garza & Mergenthaler-Canseco, *TimeGPT-1*, 2023; Rasul et al., *Lag-Llama*, 2023; Shi et al., *Time-MoE*, ICLR 2025.
- Aksu et al., *GIFT-Eval*, 2024.
- Jain & Sen, *TimesFM-3*, Google Research blog, 2026.
- Laglil et al., review of foundation-model architectures, pretraining & fine-tuning, 2026.

References II — Diffusion, Surveys, Evaluation

Diffusion & score-based models

- Ho et al., *DDPM*, NeurIPS 2020; Song et al., *Score-based SDEs*, ICLR 2021.
- Song et al., *DDIM*, ICLR 2021; Ho & Salimans, *Classifier-Free Guidance*, 2022; Lipman et al., *Flow Matching*, ICLR 2023.
- Gu et al., *S4 (structured state spaces)*, ICLR 2022.

Diffusion for forecasting

- Rasul et al., *TimeGrad*, ICML 2021; Tashiro et al., *CSDI*, NeurIPS 2021.
- Lopez Alcaraz & Strodthoff, *SSSD*, TMLR 2023; Shen & Kwok, *TimeDiff*, ICML 2023.

Diffusion for forecasting (cont.)

- Kollovieh et al., *TSDiff*, NeurIPS 2023.
- Fan et al., *MG-TSD*, ICLR 2024; Shen et al., *mr-Diff*, ICLR 2024.

Surveys & evaluation

- Yang et al., *Survey on Diffusion Models for Time Series*, 2024.
- Meijer & Chen, *The Rise of Diffusion Models in TS Forecasting*, 2024.
- Makridakis et al., *M4*, IJF 2020; *M5*, IJF 2022.
- Gneiting & Raftery, *Strictly Proper Scoring Rules*, JASA 2007.

Full BibTeX in `references.bib`; typeset list in `references.pdf`.

Thank You

Core reading

Vaswani et al. 2017 · Zhou et al. (Informer) 2021 · Zeng et al. (DLinear) 2023 · Nie et al. (PatchTST) 2023

Ansari et al. (Chronos) 2024 · Das et al. (TimesFM) 2024 · Woo et al. (Moirai) 2024

Rasul et al. (TimeGrad) 2021 · Tashiro et al. (CSDI) 2021 · Shen & Kwok (TimeDiff) 2023 · Kollovieh et al. (TSDiff) 2023

Su et al. (Diffusion-for-TSF survey) 2025 · Meijer & Chen (survey) 2024

Slides, outline, and lab notebook accompany this deck.