



Time Series Analysis and Forecasting

Chapter 11: LLMs and Foundation Models for Time Series



Daniel Traian PELE

Bucharest University of Economic Studies

IDA Institute Digital Assets

Blockchain Research Center

AI4EFin Artificial Intelligence for Energy Finance

Romanian Academy, Institute for Economic Forecasting

MSCA Digital Finance

Learning Objectives

By the end of this chapter, you will be able to:

1. **Explain** the Transformer architecture and self-attention mechanism
2. **Distinguish** encoder-only, decoder-only, and encoder-decoder architectures
3. **Describe** the foundation model paradigm: pre-train, fine-tune, deploy
4. **Compare** tokenization strategies: patching, quantization, and embeddings
5. **Evaluate** major foundation models (Chronos, TimesFM, TimeGPT, Lag-Llama, Moirai)
6. **Decide** when to use zero-shot inference vs. fine-tuning
7. **Apply** foundation models to real-world forecasting tasks in Python
8. **Assess** limitations, ethical concerns, and future directions

Outline

- Motivation
- Transformer Architecture
- From Transformers to Foundation Models
- Deep Dive into Models
- Zero-Shot vs Fine-Tuning
- Practical Use Cases
- Benchmarks and Model Selection
- Limitations, Ethics, and Future
- AI Use Case
- Summary
- Quiz

From Task-Specific to Foundation Models

Era 1: Classical Statistical Models (1970s–2010s)

- ▣ ARIMA, ETS, GARCH — hand-crafted per series, strong assumptions
- ▣ Strengths: interpretable, well-understood theory, works with small data

Era 2: Deep Learning (2015–2022)

- ▣ LSTM, TCN, DeepAR, N-BEATS — trained on individual datasets
- ▣ Strengths: capture nonlinear dynamics; Weakness: require large labeled data per task

Era 3: Foundation Models (2023–present)

- ▣ TimeGPT, Chronos, TimesFM, Lag-Llama, Moirai — pre-trained on billions of time points
- ▣ **Key insight:** train *once* on massive diverse data, deploy *everywhere* (zero-shot)

The Scale Advantage

Foundation Models: Scale Comparison

Model	Params	Pre-training Data	Architecture	Zero-Shot?
TimeGPT	N/A (proprietary)	100B+ points	Transformer enc-dec	Yes (API)
Chronos	20M–710M	84B points	T5 enc-dec	Yes
TimesFM	200M	100B points	Decoder-only	Yes
Lag-Llama	352M	Multiple datasets	LLaMA decoder	Yes
Moirai	14M–311M	27B points	BERT encoder	Yes
PatchTST	1M–10M	Task-specific	Transformer enc	No (supervised)

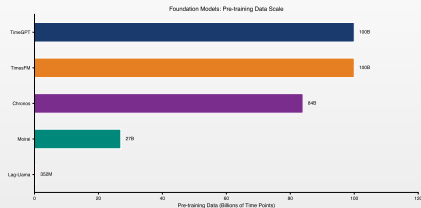
Key Observation

Foundation models leverage **cross-domain knowledge transfer**: patterns learned from energy data help forecast retail sales, and vice versa. This is analogous to how GPT-3 learns grammar from diverse text corpora.

Chapter Roadmap

What We Will Cover

1. **Motivation** — the three eras of forecasting
2. **Transformer Architecture** — attention, positional encoding
3. **From Transformers to Foundation Models** — tokenization, patching
4. **Deep Dive into Models** — Chronos, TimesFM, TimeGPT, Lag-Llama, Moirai
5. **Zero-Shot vs Fine-Tuning** — when and how
6. **Practical Use Cases** — EUR/RON, energy, volatility
7. **Benchmarks and Model Selection** — Monash, GiftEval
8. **Limitations, Ethics, and Future** — what lies ahead



Pre-training data scale across foundation models (log scale).

Researcher Spotlight: Vaswani et al. (2017)

“Attention Is All You Need”

- ▣ Published at NeurIPS 2017 by researchers at Google Brain
- ▣ Over **130,000 citations** — one of the most influential ML papers ever
- ▣ Originally designed for **machine translation** (NLP)
- ▣ Key innovation: replaced recurrence entirely with **self-attention**

Why It Matters for Time Series

- ▣ Time series are **sequential** data — same fundamental challenge as language
- ▣ Transformers handle **variable-length** sequences naturally
- ▣ **Parallel computation**: unlike RNNs, all positions processed simultaneously
- ▣ Foundation models (GPT, BERT, T5) all build on this architecture

Impact Timeline

2017: Attention Is All You Need → 2018: BERT, GPT → 2020: GPT-3 → 2023: TimeGPT, Chronos → 2024: TimesFM, Moirai

Self-Attention: Intuition

Why Attention Replaces Recurrence

- ▣ **RNNs:** process tokens sequentially $\Rightarrow O(T)$ sequential steps, information bottleneck
- ▣ **Self-Attention:** every position attends to every other position **in parallel**
- ▣ Direct connections between distant positions: no vanishing gradients

RNN: Sequential

- ▣ Path length between x_1 and x_T : $O(T)$ steps
- ▣ Cannot parallelize across time steps
- ▣ Gradient must flow through T multiplications
- ▣ Information “compresses” through hidden state

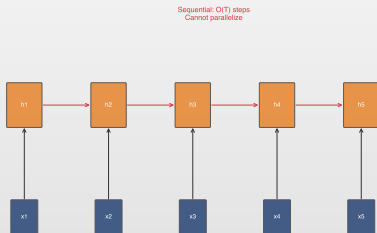
Attention: Parallel

- ▣ Path length between any two positions: $O(1)$
- ▣ Fully parallelizable $\Rightarrow$ faster on GPUs
- ▣ Each position directly “sees” all others
- ▣ Learns which positions to attend to (data-driven)

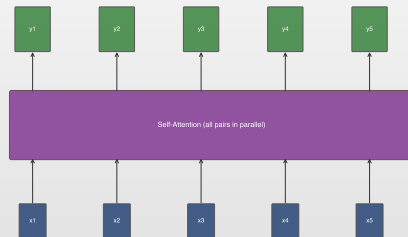
Self-Attention: Intuition (Diagram)

RNN vs Transformer: Processing Paradigm

(a) RNN: Sequential Processing



(b) Transformer: Parallel Attention



RNN processes tokens sequentially ($O(T)$ steps); Transformer attends to all positions in parallel ($O(1)$ path length).

Self-Attention: Mathematics

Scaled Dot-Product Attention

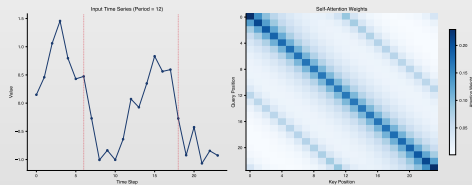
Given input embeddings $X \in \mathbb{R}^{T \times d}$, compute:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

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

Interpretation

- Q (Queries): “What am I looking for?”
- K (Keys): “What do I contain?”
- V (Values): “What information do I provide?”
- $\sqrt{d_k}$: scaling prevents softmax saturation
- Output: weighted average of values, weights = compatibility scores



Attention weights heatmap: rows attend to columns.

Multi-Head Attention

Multiple Attention Heads Capture Diverse Patterns

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

Each head projects to a lower dimension: $W_i^Q, W_i^K \in \mathbb{R}^{d \times d_k}$, $W_i^V \in \mathbb{R}^{d \times d_v}$ with $d_k = d_v = d/h$.

What Different Heads Learn

- ▣ Head 1: **Trend** — attend to recent values
- ▣ Head 2: **Seasonality** — attend to same period last cycle
- ▣ Head 3: **Outliers** — attend to extreme values
- ▣ Head 4: **Long-range** — attend to distant correlations

Computational Cost

- ▣ Self-attention complexity: $O(T^2 \cdot d)$
- ▣ Memory: $O(T^2)$ for the attention matrix
- ▣ For long sequences ($T > 4096$): quadratic cost becomes prohibitive
- ▣ Solutions: patching, sparse attention, linear attention

Positional Encoding

Why Position Matters

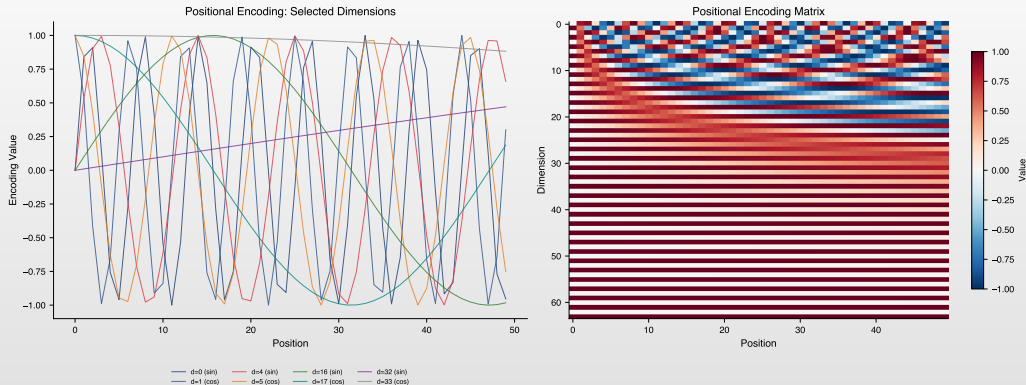
Self-attention is **permutation-equivariant**: without positional info, it treats $\{x_1, x_2, x_3\}$ and $\{x_3, x_1, x_2\}$ identically. Time series **order matters**, so we inject position.

Sinusoidal Positional Encoding (Vaswani et al., 2017)

$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$
$$PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

Embedding is on pos and diff. frequency $\Rightarrow$ unique positional signature. Relative positions can be

Positional Encoding (Visualization)



Each dimension oscillates at a different frequency, creating a unique positional signature. Relative positions can be computed via linear transformations.

The Full Transformer Block

Transformer Block Components

1. **Input embedding** + Positional encoding
2. **Layer Normalization** (pre-norm variant)
3. **Multi-Head Self-Attention**
4. **Residual connection:** $x + \text{Attn}(x)$
5. **Layer Normalization**
6. **Feed-Forward Network (FFN):** two linear layers with GELU/ReLU

$$\text{FFN}(x) = W_2 \cdot \text{GELU}(W_1 x + b_1) + b_2$$

7. **Residual connection:** $x + \text{FFN}(x)$

Stacking N blocks ($N = 6\text{--}24$) gives depth. Residual connections enable gradient flow.



Diagram: the standard Transformer block with pre-norm layout.

Encoder vs Decoder Architectures

Three Architectural Paradigms

	Encoder-only	Decoder-only	Encoder-Decoder
NLP origin	BERT	GPT	T5, BART
Attention	Bidirectional	Causal (masked)	Cross-attention
TS models	Moirai, PatchTST	TimesFM, Lag-Llama	Chronos, TimeGPT
Strengths	Learns rich representations	Natural for autoregressive generation	Flexible input-output mapping
TS use	Representation, classification	Forecasting, generation	Forecasting, seq2seq

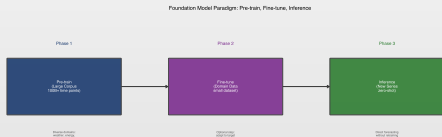
Key Distinction

Causal masking (decoder): position t can only attend to $\{1, \dots, t\}$. Prevents “peeking” into the future — natural fit for time series forecasting. **Bidirectional** (encoder): every position sees all others — useful for representation learning and classification.

The Foundation Model Paradigm

Three-Stage Pipeline

1. **Pre-training:** Train on massive, diverse corpora (billions of time points across domains)
2. **Fine-tuning** (optional): Adapt to a specific domain or dataset with a small learning rate
3. **Inference:** Deploy on new, unseen series — zero-shot or few-shot



Key Insight (Bommasani et al., 2021)

- “A foundation model is any model trained on broad data that can be adapted to a wide range of downstream tasks.”
- For TS: train on *energy + finance + retail + health + weather* $\Rightarrow$ forecast *anything*
- Critical assumption: **shared temporal patterns** exist across domains (trend, seasonality, noise)

Tokenization Strategies for Time Series

Challenge: Continuous Values $\rightarrow$ Discrete Tokens

NLP models process discrete tokens (words). Time series are **continuous** $\Rightarrow$ need a mapping strategy.

Strategy 1: Raw Value Embeddings

- ▣ Project each $x_t \in \mathbb{R}$ to $\mathbb{R}^d$ via linear layer
- ▣ Used by: vanilla Transformer TS models
- ▣ Simple but ignores local structure

Strategy 2: Scaling + Quantization

- ▣ Normalize, then map to B discrete bins
- ▣ Used by: **Chronos** (4096 bins)
- ▣ Leverages NLP machinery directly

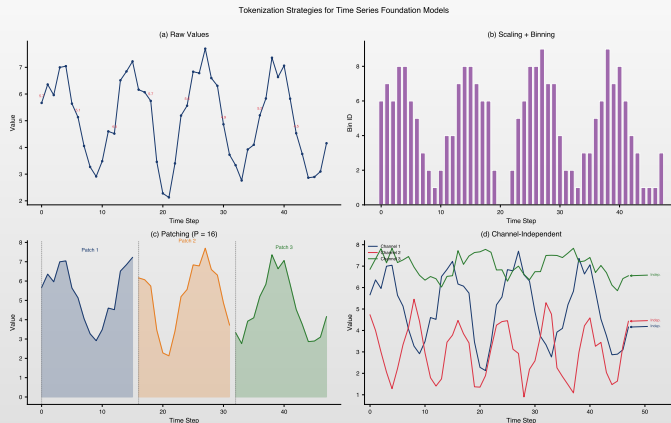
Strategy 3: Patching

- ▣ Group P consecutive points into one token
- ▣ Used by: **PatchTST**, **TimesFM**
- ▣ Reduces sequence length by factor P

Strategy 4: Variate-Independent Channels

- ▣ Each variable processed independently
- ▣ Used by: **PatchTST**, **Moirai**
- ▣ Scales to any number of variates

Tokenization Strategies (Visualization)



Four approaches to converting continuous time series into discrete tokens for Transformer models.

Patching: Why It Works

PatchTST Insight (Nie et al., 2023)

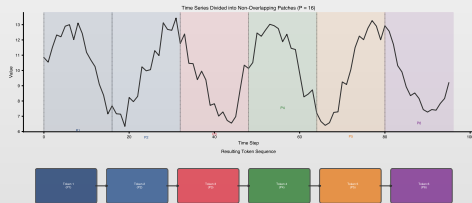
Divide a time series of length L into $N = \lfloor L/P \rfloor$ non-overlapping patches of size P :

$$p_i = (x_{(i-1)P+1}, x_{(i-1)P+2}, \dots, x_{iP}) \in \mathbb{R}^P, \quad i = 1, \dots, N$$

Each patch is linearly projected to dimension d and fed as a single token to the Transformer.

Benefits of Patching

- **Reduces complexity:** attention $O(T^2) \rightarrow O((T/P)^2)$
- **Local semantics:** each patch captures a short sub-pattern (e.g., daily cycle)
- **Longer context:** with $P = 32$, a 512-token window covers 16,384 time steps
- **Better representations:** patch-level tokens are more informative than point-level



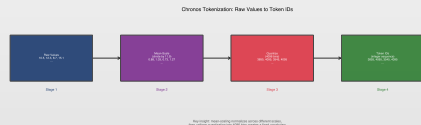
Quantization: The Chronos Approach

Scale-and-Quantize Tokenization (Ansari et al., 2024)

1. **Normalize:** $\tilde{x}_t = \frac{x_t - \bar{x}}{s}$ where $\bar{x}$ is mean and s is the mean absolute value
2. **Quantize:** map to B bins via the CDF: $b_t = \lfloor (B - 1) \cdot \Phi(\tilde{x}_t) \rfloor$ where Φ is the standard normal CDF
3. **Embed:** look up bin b_t in the T5 embedding table ($B = 4096$)

Why Quantize?

- ▣ Reuses **NLP vocabulary machinery:** embeddings, softmax output layer
- ▣ Cross-entropy loss (categorical) is well-understood
- ▣ Probabilistic: output is a distribution over bins $\Rightarrow$ full predictive distribution
- ▣ Trade-off: resolution limited by number of bins



Pre-Training Objectives

Paradigm 1: Next-Token Prediction (Autoregressive, GPT-style)

- ▣ Predict x_{t+1} given $x_1, \dots, x_t$ via causal masking. Loss: $\mathcal{L} = - \sum_{t=1}^{T-1} \log p(x_{t+1} | x_1, \dots, x_t)$
- ▣ Used by: **TimesFM**, **Lag-Llama**, **TimeGPT**

Paradigm 2: Masked Prediction (BERT-style)

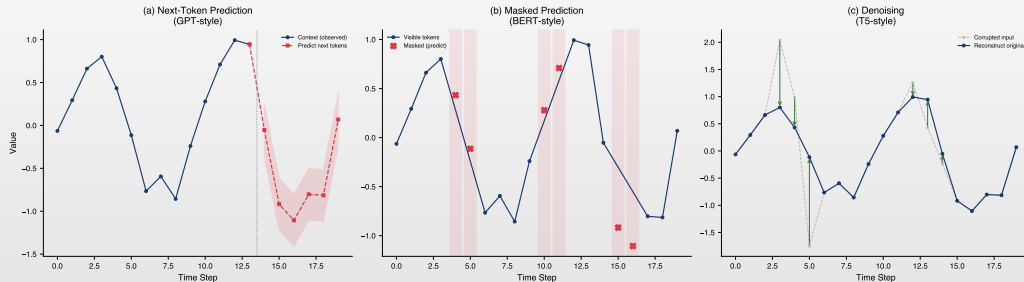
- ▣ Randomly mask 15–40% of tokens, predict masked ones. Loss: $\mathcal{L} = - \sum_{t \in \mathcal{M}} \log p(x_t | x_{\setminus \mathcal{M}})$
- ▣ Used by: **Moirai**, **PatchTST** (self-supervised)

Paradigm 3: Denoising / Seq2Seq (T5-style)

- ▣ Encoder processes context, decoder generates forecast. Loss: cross-entropy over quantized tokens
- ▣ Used by: **Chronos**

Pre-Training Objectives (Visualization)

Pre-training Objectives for Time Series Foundation Models



Three pre-training paradigms: autoregressive (GPT-style), masked prediction (BERT-style), and denoising/seq2seq (T5-style).

Data Augmentation for Pre-Training

KernelSynth (Ansari et al., 2024)

- Generate **synthetic** time series by composing Gaussian Process (GP) kernels:

$$k(t, t') = k_{\text{RBF}}(t, t') + k_{\text{periodic}}(t, t') + k_{\text{linear}}(t, t') + \dots$$

- Produces realistic patterns: trends, seasonality, changepoints, noise
- Chronos uses **billions** of synthetic series to augment real data

TSMix (TimesFM)

- Mix existing series: $\tilde{x} = \lambda x_a + (1 - \lambda)x_b$
- Augments diversity without needing GP expertise
- Combined with window slicing and frequency mixing

Why Augmentation Matters

- Prevents overfitting to specific domains
- Increases pattern diversity during pre-training
- Chronos: 50% real data + 50% synthetic $\Rightarrow$ **better** than 100% real

Chronos: Architecture

Chronos (Ansari et al., 2024 — Amazon)

- ▣ **Architecture:** T5 encoder-decoder (pre-trained language model backbone)
- ▣ **Model sizes:** Mini (20M), Small (46M), Base (200M), Large (710M)
- ▣ **Tokenization:** Mean-scale normalization + quantization into **4096 bins**
- ▣ **Pre-training data:** 84B time points from 28 public datasets + KernelSynth
- ▣ **Output:** Categorical distribution over bins $\Rightarrow$ full probabilistic forecast

Key Innovations

- ▣ Forecasting as **language modeling** over quantized tokens
- ▣ No architecture changes to T5 — only tokenization is new
- ▣ Probabilistic: sample N trajectories, compute quantiles
- ▣ Open-source, multiple sizes

Limitations

- ▣ Resolution limited by 4096 bins (quantization error)
- ▣ Univariate only (each variable separately)
- ▣ Context window: 512 tokens
- ▣ Slower inference than decoder-only (enc+dec)

Chronos: How It Forecasts

Step-by-Step Pipeline

1. **Input:** raw time series $(x_1, x_2, \dots, x_T)$
2. **Normalize:** mean-scale $\tilde{x}_t = x_t / |x|$
3. **Quantize:** map each $\tilde{x}_t$ to one of 4096 bins via Φ (normal CDF)
4. **Encode:** T5 encoder processes context tokens $\rightarrow$ hidden representations
5. **Decode:** T5 decoder generates forecast tokens **autoregressively**
6. **Sample:** repeat decoding N times (e.g., $N = 20$) with temperature sampling
7. **Dequantize:** convert bin indices back to continuous values
8. **Output:** median forecast + quantile intervals (10%, 90%)

Chronos Inference Pipeline



Probabilistic forecasting via categorical distribution over quantized bins

Chronos: Benchmark Results

Performance on Monash Benchmark (Ansari et al., 2024)

- Evaluated on 27 datasets from the Monash Time Series Forecasting Archive
- Metrics: **WQL** (Weighted Quantile Loss) and **MASE** (Mean Absolute Scaled Error)
- Chronos-Large achieves best or second-best on 16/27 datasets **in zero-shot**
- Outperforms task-specific deep learning models (DeepAR, TFT, N-BEATS) on average



WQL scores: lower is better. Chronos-Large competitive with or better than supervised baselines.

TimesFM: Architecture

TimesFM (Das et al., 2024 — Google Research)

- ▣ **Architecture:** Decoder-only Transformer (200M parameters)
- ▣ **Input/output patching:** patch size $P = 32$, projects patch to model dimension
- ▣ **Pre-training data:** 100B real-world + synthetic data points
- ▣ **Multi-resolution:** trained on diverse frequencies (minutely to yearly)
- ▣ **Context window:** up to 512 patches (= 16,384 time steps)

Key Innovations

- ▣ **Decoder-only:** simpler and faster than enc-dec
- ▣ **Output patches:** predict P values at once
- ▣ **Frequency token:** encodes data frequency
- ▣ Point forecast + fine-tuning for quantiles

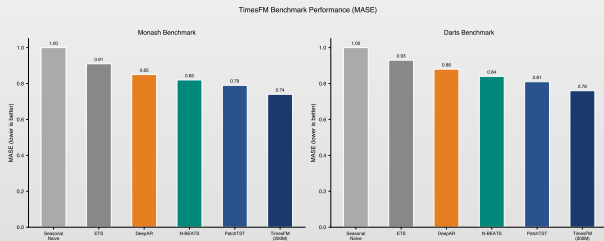
Comparison with Chronos

- ▣ Continuous values (no quantization)
- ▣ Faster inference (decoder-only, patch output)
- ▣ Less naturally probabilistic
- ▣ Closed weights until 2024, now open

TimesFM: Zero-Shot Performance

Benchmark Results (Das et al., 2024)

- ▣ Evaluated on Monash, M3, M4, and long-horizon benchmarks (ETTh, Weather, Electricity)
- ▣ **Zero-shot** TimesFM beats supervised models on the majority of Monash datasets
- ▣ Particularly strong on **long-horizon** forecasting (96–720 steps)
- ▣ Competitive with task-specific PatchTST and iTransformer even without fine-tuning



TimesFM zero-shot MAE across benchmark datasets. Bars show relative improvement over best supervised baseline.

TimeGPT: API-First Foundation Model

TimeGPT (Garza & Mergenthaler-Canseco, 2024 — Nixtla)

- ▣ **Architecture:** Proprietary Transformer encoder-decoder
- ▣ **Pre-training:** largest known corpus — **100+ billion** time points from diverse domains
- ▣ **Access:** API-only (no open weights), commercial product
- ▣ **Features:** zero-shot forecasting, conformal prediction intervals, anomaly detection, fine-tuning endpoint, exogenous variable support

Advantages

- ▣ **Easiest to use:** 3 lines of Python code
- ▣ **Conformal prediction:** calibrated uncertainty
- ▣ **Exogenous variables:** unlike most FM
- ▣ Regular model updates by Nixtla team

Limitations

- ▣ **Black box:** proprietary, no reproducibility
- ▣ **API cost:** per-call pricing
- ▣ **Data privacy:** data sent to Nixtla servers
- ▣ **Vendor lock-in:** dependent on service availability

Lag-Llama: LLM Backbone for Time Series

Lag-Llama (Rasul et al., 2024)

- ▣ **Architecture:** LLaMA decoder adapted for time series (352M parameters)
- ▣ **Key innovation: Lag features as token embeddings** — lags $\{1, 2, \dots, 7, 14, 28, \dots\}$
- ▣ **Distribution head:** outputs Student- t parameters (μ, σ, ν) instead of token probabilities
- ▣ **Pre-training:** Multiple publicly available time series datasets

Why LLaMA for Time Series?

- ▣ LLaMA backbone: proven at scale in NLP
- ▣ RoPE (Rotary Positional Encoding) for temporal positions
- ▣ GQA (Grouped Query Attention) for efficient inference
- ▣ Lag tokens capture known temporal structures (weekly, monthly)

Distribution Head

- ▣ Instead of softmax over vocabulary:

$$p(x_{t+1} \mid \cdot) = \text{Student-}t(\mu, \sigma, \nu)$$

- ▣ Naturally handles heavy tails (financial data)
- ▣ Probabilistic: sample for prediction intervals
- ▣ Open-source and fine-tunable

Moirai: Any-Variate, Any-Frequency

Moirai (Woo et al., 2024 — Salesforce)

- ▣ **Architecture:** BERT-style masked encoder (14M, 91M, 311M parameters)
- ▣ **Key innovation:** handles **any number of variates, any frequency, any prediction length**
- ▣ **Pre-training:** 27B observations from **LOTSA** (Large-scale Open Time Series Archive, 9 domains)
- ▣ **Output:** mixture distribution (parameters of a flexible output distribution)

Universal Design

- ▣ **Any-variate:** flattens multivariate into patches, uses variate IDs
- ▣ **Any-frequency:** frequency embeddings (hourly, daily, etc.)
- ▣ **Any-length:** masked attention for variable output lengths
- ▣ Closest to a “universal” TS foundation model

LOTSA Dataset

- ▣ 9 domains: energy, transport, finance, healthcare, nature, web, sales, climate, social
- ▣ 27B observations, curated for quality
- ▣ Open-source dataset collection
- ▣ Enables reproducible pre-training

PatchTST: The Architectural Blueprint

PatchTST (Nie et al., 2023)

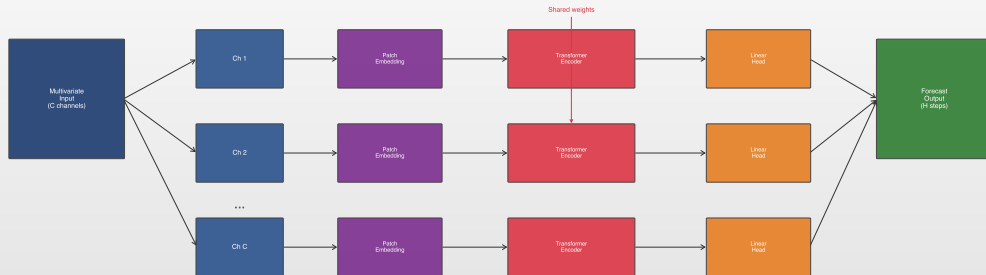
- ▣ **Architecture:** Transformer encoder with **channel-independent patching**
- ▣ Not a foundation model, but introduced two key ideas adopted by all later models: **(1)** Patching: group P consecutive time points into one token; **(2)** Channel independence: each variate processed independently
- ▣ Parameters: 1M–10M (task-specific, much smaller than foundation models)

Legacy

- ▣ Showed Transformers can beat linear models on long-horizon forecasting
- ▣ Patching adopted by: TimesFM, Moirai, MOMENT
- ▣ Channel independence: Moirai, iTransformer
- ▣ Self-supervised variant: masked patching

PatchTST: Architecture Diagram

PatchTST Architecture: Channel-Independent Patched Transformer



PatchTST processes each channel independently, groups consecutive time steps into patches, and feeds them through a shared Transformer encoder.

Architecture Comparison Table

Comprehensive Model Comparison

Model	Architecture	Params	Tokenization	Output	Pre-training
Chronos	T5 enc-dec	20M–710M	Quantization (4096)	Categorical dist.	84B pts + KernelSynth
TimesFM	Decoder-only	200M	Patching ($P=32$)	Point + quantiles	100B real + synthetic
TimeGPT	Enc-dec (prop.)	N/A	Proprietary	Point + conformal	100B+ (proprietary)
Lag-Llama	LLaMA decoder	352M	Lag features	Student- t	Public datasets
Moirai	BERT encoder	14M–311M	Patching + variate ID	Mixture dist.	27B (LOTSAs)
PatchTST	Transformer enc	1M–10M	Patching ($P=16$)	Point (MSE)	Task-specific

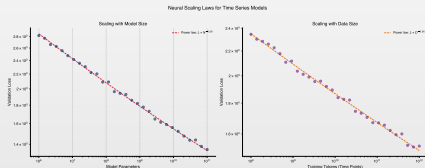
Key Takeaway

No single architecture dominates. The choice depends on: (1) whether you need probabilistic forecasts, (2) multivariate support, (3) compute budget, and (4) whether open-source matters.

The Scaling Hypothesis for Time Series

Does Scaling Work for Time Series?

In NLP, scaling laws (Kaplan et al., 2020): $\text{loss} \propto N^{-\alpha}$ where N = parameters. For time series: evidence is mixed.



Why TS Scaling Differs from NLP

- **Less data diversity:** text more diverse than TS
- **Simpler patterns:** many TS well-described by AR + seasonality
- **Diminishing returns:** Chronos-Large $\approx$ Chronos-Base
- **Data quality:** noisy TS limit learning
- **Evidence:** Chronos 710M $\approx$ 200M; TimesFM 200M competitive with larger models; data quality > parameter count

When Foundation Models Fail

Failure Modes to Watch For

1. **Distribution shift:** Financial regime changes (COVID crash, rate hikes). Pre-training data may not cover such regimes.
2. **Missing exogenous variables:** Weather affects energy demand; policy changes affect economics. Most foundation models are **univariate**.
3. **Rare frequencies:** Irregular sampling, event-driven or tick-by-tick data. Pre-training favors regular frequencies.
4. **Very short series:** Fewer than ~ 50 observations $\Rightarrow$ insufficient context. Classical methods (ETS, ARIMA) often dominate.
5. **Multivariate dependencies:** Cross-variable correlations (e.g., VAR) ignored by univariate models. Only Moirai attempts multivariate support.

Rule of Thumb

If your data has known causal structure, domain-specific features, or regime changes, consider combining foundation model forecasts with domain-informed models (ensemble or hierarchical).

Zero-Shot: When and Why

Zero-Shot Forecasting

Use the pre-trained model directly on a new series without any training.

When to Use Zero-Shot

- ▣ **Limited labeled data:** new product launch, cold-start
- ▣ **Rapid prototyping:** quick baseline in minutes
- ▣ **Cross-domain deployment:** same model for finance, retail, energy
- ▣ **Many series:** 10,000+ series to forecast, no time to train each
- ▣ **No ML expertise:** simple API call



Surprising Finding

Ansari et al. (2024): Chronos zero-shot outperforms task-specific models (DeepAR, N-BEATS) that were *trained on the target data*. Pre-training transfers temporal knowledge effectively.

Fine-Tuning: Domain Adaptation

Full Fine-Tuning vs LoRA

- ▣ **Full fine-tuning**: update all parameters on domain data (expensive, risk of forgetting)
- ▣ **LoRA** (Low-Rank Adaptation, Hu et al., 2022): freeze base weights, add small trainable matrices

$$W' = W + \Delta W = W + BA, \quad B \in \mathbb{R}^{d \times r}, \quad A \in \mathbb{R}^{r \times d}, \quad r \ll d$$

- ▣ LoRA: 0.1–1% of trainable parameters, similar performance, much cheaper

Risks of Fine-Tuning

- ▣ **Catastrophic forgetting**: model loses pre-trained knowledge
- ▣ **Overfitting**: small domain dataset $\Rightarrow$ poor generalization
- ▣ **Hyperparameter sensitivity**: learning rate, epochs, warmup critical

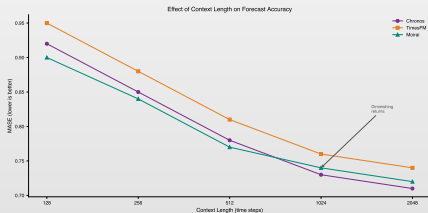
Best Practices

- ▣ Start with **low learning rate** (10^{-5} – 10^{-4})
- ▣ Use **learning rate warmup** (500–1000 steps)
- ▣ **Early stopping** on validation loss
- ▣ **Freeze early layers**, fine-tune later layers
- ▣ Prefer **LoRA** for small datasets ($<10K$ series)

Few-Shot and In-Context Learning

Context Length as “Few-Shot” for Time Series

- In NLP, “few-shot” = provide examples in the prompt. In TS, “context” = historical observations
- **Longer context** → **better forecasts** (up to the model's context window)
- The model conditions on historical patterns to infer trend, seasonality, and noise structure



Forecast error vs. context length: diminishing returns beyond the model's window.

Context Windows by Model

- Chronos: 512 tokens (≈ 512 time steps)
- TimesFM: 512 patches ($\approx 16,384$ steps)
- Moirai: 5000 tokens (configurable)
- Lag-Llama: 1024 steps (with lags extending further)
- TimeGPT: configurable via API

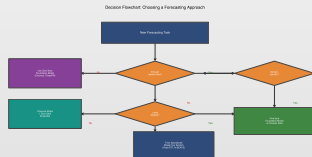
Practical Tip

Provide as much context as the model allows. For daily data, ≥ 1 year. For monthly, ≥ 5 years.

Decision Framework

When to Choose Zero-Shot, Fine-Tuning, or Classical

- **Step 1:** Do you have labeled domain data? No → Zero-shot
- **Step 2:** Is zero-shot performance sufficient? Yes → Deploy; No → Fine-tune
- **Step 3:** Is the domain very different from pre-training data? Yes → Full fine-tune; No → LoRA
- **Step 4:** Need interpretability or causal reasoning? Yes → Classical models (ARIMA, VAR)



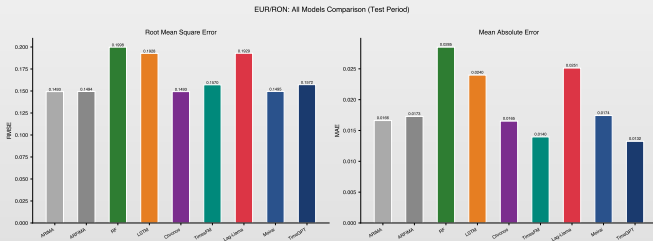
Rules of Thumb

- <50 observations: ARIMA / ETS
- 50–500 observations: zero-shot foundation model
- 500–10K observations: fine-tuned foundation model
- >10K observations: task-specific deep learning may match or beat FM
- Always benchmark against ARIMA as baseline

Use Case 1: EUR/RON with All Foundation Models

Extending the Chapter 8 Case Study

- ▣ **Data:** EUR/RON daily exchange rate (2020–2024, ~1,200 observations)
- ▣ **Task:** 30-day-ahead forecast, 80/20 temporal split
- ▣ **Models:** ARIMA, ARFIMA, RF, LSTM, **Chronos**, **TimesFM**, **Lag-Llama**, **Moirai**, **TimeGPT**
- ▣ Foundation models used in **zero-shot** mode (no training on EUR/RON data)



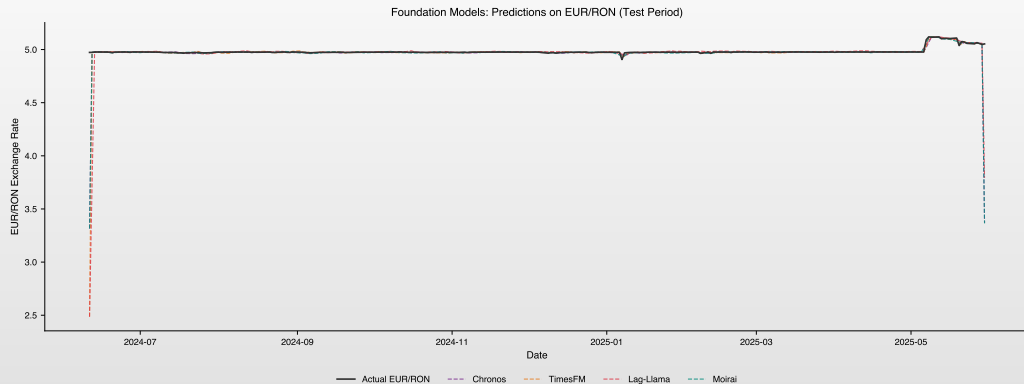
EUR/RON: Detailed Results

Forecast Performance Comparison

Model	RMSE	MAE	MAPE (%)	Training Time
ARIMA(1,1,1)	—	—	—	<1s
ARFIMA	—	—	—	<1s
Random Forest	—	—	—	~2s
LSTM (2-layer)	—	—	—	~60s
Chronos-Base	—	—	—	0s (zero-shot)
TimesFM	—	—	—	0s (zero-shot)
Lag-Llama	—	—	—	0s (zero-shot)
Moirai-Base	—	—	—	0s (zero-shot)
TimeGPT	—	—	—	0s (API call)

Note: Metric values (—) populated at runtime from the Quantlet code. See the companion notebook for exact numbers.

EUR/RON: Prediction Comparison

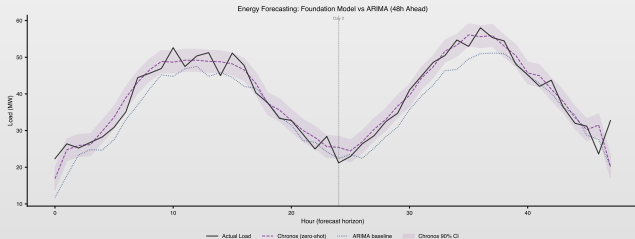


Foundation model predictions overlaid on EUR/RON exchange rate. All foundation models use zero-shot inference.

Use Case 2: Energy Demand Forecasting

German Electricity Demand (Hourly, Multi-Seasonal)

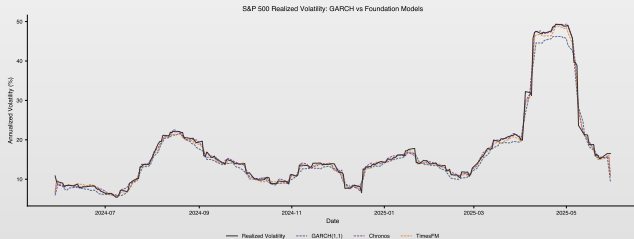
- ▣ **Data:** German hourly electricity consumption (2015–2023, 70K+ observations)
- ▣ **Challenge:** multiple seasonalities (24h daily + 168h weekly + annual)
- ▣ **Models:** TBATS (Ch9), Prophet (Ch9), LSTM (Ch8), Chronos, TimesFM, Moirai
- ▣ Foundation models handle multi-seasonal patterns **without explicit specification**



Use Case 3: Financial Volatility

S&P 500 Realized Volatility

- ▣ **Data:** S&P 500 daily realized volatility (2000–2024)
- ▣ **Challenge:** Clustering, heavy tails, regime changes (2008 crisis, 2020 COVID)
- ▣ **Models:** GARCH(1,1) (Ch5), HAR-RV, Chronos, TimesFM, Lag-Llama
- ▣ **Question:** Can foundation models capture volatility dynamics without GARCH structure?



Python Code Patterns

Chronos (3 lines)

```
from chronos import ChronosPipeline
pipe = ChronosPipeline.from_pretrained(
    "amazon/chronos-t5-base")
forecast = pipe.predict(
    context=torch.tensor(data),
    prediction_length=30)
```

TimesFM (3 lines)

```
import timesfm
tfm = timesfm.TimesFm(hparams=
    timesfm.TimesFmHparams())
tfm.load_from_checkpoint()
preds = tfm.forecast(data)
```

TimeGPT (3 lines)

```
from nixtla import NixtlaClient
nixtla = NixtlaClient(api_key="...")
fcst = nixtla.forecast(
    df=df, h=30, model="timegpt-1")
```

Lag-Llama (4 lines)

```
from lag_llama.gluon.estimator import
    LagLlamaEstimator
estimator = LagLlamaEstimator(
    prediction_length=30)
predictor = estimator.train(train_ds)
forecasts = list(
    predictor.predict(test_ds))
```

Major Benchmarks

Monash Time Series Forecasting Archive

- ▣ **30K+ series** across 30 datasets (tourism, electricity, traffic, weather, finance, etc.)
- ▣ Standard evaluation protocol: fixed train/test splits, WQL and MASE metrics
- ▣ Most widely used benchmark for foundation models (Chronos, TimesFM, Moirai all report here)

M-Competitions

- ▣ **M3** (2000): 3,003 series, monthly/quarterly/yearly
- ▣ **M4** (2018): 100,000 series, 6 frequencies
- ▣ **M5** (2020): Walmart sales, hierarchical
- ▣ Finding: simpler methods often competitive

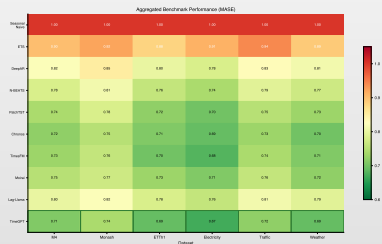
GiftEval (2024)

- ▣ **General Time Series Forecasting Evaluation**
- ▣ Designed specifically for foundation model evaluation
- ▣ Diverse frequencies, domains, horizons
- ▣ Calibration and reliability metrics

Aggregated Results Across Benchmarks

Cross-Benchmark Performance Summary

- Aggregated rankings across Monash, M3, M4, ETT, Weather, Electricity, and Traffic
- No single model wins everywhere — **model selection depends on the use case**
- Foundation models are **most impressive in zero-shot**, often matching supervised baselines



Heatmap: model (rows) $\times$ benchmark (columns). Color = relative ranking (green = top, red = bottom). [TSA_ch11_benchmarks](#)

When to Use What: Decision Table

Scenario-Based Recommendations

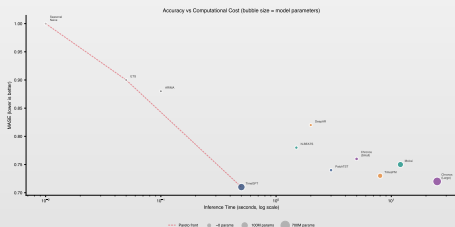
Scenario	ARIMA	Chronos	TimesFM	Moirai	TimeGPT
Limited data (<100 obs)	✓✓	✓	✓	~	✓
Real-time / low latency	✓✓	~	✓	~	✓
Interpretability needed	✓✓	×	×	×	×
Multivariate dependencies	VAR	×	×	✓✓	~
Many series (10K+)	~	✓✓	✓✓	✓✓	✓
No ML expertise	✓	✓	~	~	✓✓
Open source required	✓✓	✓✓	✓	✓✓	×
Probabilistic forecasts	✓	✓✓	~	✓✓	✓

✓✓ = excellent, ✓ = good, ~ = acceptable, × = poor fit

Computational Cost Analysis

Pareto Frontier: Accuracy vs. Cost

- Foundation models trade **compute for convenience**: no training needed, but inference is heavier
- The “best” model lies on the Pareto frontier of accuracy and cost



Cost Considerations

- ARIMA**: CPU, milliseconds, free
- Chronos-Mini**: 1 GPU, ~0.1s/series
- Chronos-Large**: 1 GPU (A100), ~1s/series
- TimesFM**: 1 GPU, ~0.05s/series (fast)
- TimeGPT**: API, \$0.01–\$0.10/call
- Pre-training from scratch**: 100+ GPU-hours (Chronos-Large)

Current Limitations

Five Key Limitations of Foundation Models for Time Series

1. **Most models are univariate only:** Cannot capture cross-variable interactions (e.g., price-volume, temperature-demand). Only Moirai attempts multivariate.
2. **Limited context windows (512–4096 tokens):** Cannot leverage very long histories. For daily data with yearly seasonality, 512 steps $\approx$ 1.5 years only.
3. **No causal reasoning:** Models learn correlations, not causation. Cannot answer “what if interest rates rise by 1%?” — no structural model underneath.
4. **Black box nature:** Attention weights are not easily interpretable. Difficult to explain *why* a particular forecast was made — problematic for regulated industries.
5. **“Hallucination” in probabilistic forecasts:** Generated trajectories may exhibit unrealistic patterns (e.g., negative prices, explosive growth) that violate domain constraints.

Mitigation

Post-processing (clipping, reconciliation), ensemble with classical models, conformal calibration.

Ethical Considerations

Data Provenance

- ▣ Pre-training data may include **private or proprietary** time series (e.g., internal sales data scraped from APIs)
- ▣ Lack of transparency: most models do not fully disclose training data composition
- ▣ LOTSA (Moirai) is a positive example: open, curated, documented sources

Fairness and Bias

- ▣ Pre-training skews toward **data-rich domains** (energy, retail)
- ▣ Under-represented: healthcare in developing countries, agriculture in Africa
- ▣ Biased training → biased forecasts → biased decisions

Environmental and Regulatory

- ▣ **Carbon footprint:** pre-training Chronos-Large $\approx$ 100+ GPU-hours on A100s
- ▣ **EU AI Act:** high-risk AI systems require explainability — foundation models may not qualify
- ▣ **Accountability:** who is responsible when a forecast-based decision fails?

Future Directions

Where Is the Field Heading?

1. **Multimodal TS models:** Combine time series with text, images, and metadata
2. **Agentic forecasting:** LLM agents that *select, run, and evaluate* forecasting models autonomously
3. **Continual pre-training:** Update foundation models with new data without full retraining
4. **Multivariate foundation models:** Learn cross-variable dynamics and spatial correlations
5. **Domain-specific foundation models:** FinanceFM, EnergyFM, HealthFM — smaller, specialized, more accurate

The Next 3–5 Years

Foundation models will likely become the **default starting point** for forecasting, much as pre-trained word embeddings became standard in NLP. Classical methods will remain essential for interpretability, causal inference, and small-data regimes.

A Balanced Perspective

When to Use Classical Models

- ▣ **Interpretability required:** regulatory settings, scientific research, policy analysis
- ▣ **Causal inference:** structural models (VAR, SVAR) for “what if” analysis
- ▣ **Small data:** fewer than ~ 50 observations, seasonal structure known a priori
- ▣ **Computational constraints:** edge devices, real-time systems, no GPU access
- ▣ **Explainability mandate:** EU AI Act, financial regulation (MiFID II, Basel III)

When to Use Foundation Models

- ▣ **Rapid prototyping:** get a strong baseline in minutes, not days
- ▣ **Cross-domain:** same model for retail, energy, finance — no domain expertise needed
- ▣ **Zero-shot / cold-start:** new product, new market, no historical data
- ▣ **Scale:** 10,000+ series to forecast — foundation models amortize the cost
- ▣ **Probabilistic forecasts:** native uncertainty quantification (quantiles, prediction intervals)

Bottom line: Classical and foundation models are complementary, not competing. The best practitioners use both.

AI Exercise: Foundation Model Selection

Prompt to test in ChatGPT / Claude / Copilot

“Compare ARIMA, Prophet, and a foundation model (Chronos) for forecasting monthly airline passengers (the classic Box-Jenkins dataset, 144 observations). For each model: (1) fit the model, (2) forecast 24 months ahead, (3) plot actual vs predicted with 95% confidence intervals, (4) compute RMSE and MAPE. Which model would you recommend and why? Provide complete Python code.”

Exercise:

1. Run the prompt in an LLM of your choice and critically analyze the response.
2. Does the code properly handle the **multiplicative seasonality** in airline data?
3. How does Chronos handle seasonality **without being told** the period is 12?
4. Are confidence intervals from Chronos (quantile-based) comparable to ARIMA's (parametric)?
5. Does the LLM discuss the **trade-off**: ARIMA is interpretable, Chronos is zero-shot?

Warning: AI-generated code may run without errors and look professional. *That does not mean it is correct.* Always validate against known benchmarks.

Key Takeaways

What We Learned

- ▣ **Transformers** replace recurrence with self-attention: parallel, scalable, long-range dependencies
- ▣ **Foundation models** are pre-trained on billions of diverse time points and deployed zero-shot
- ▣ **Tokenization** strategies (patching, quantization) bridge continuous TS and discrete Transformer inputs
- ▣ **Chronos** (T5, quantized), **TimesFM** (decoder, patched), **Lag-Llama** (LLaMA, lags), **Moirai** (BERT, any-variate) each have distinct design philosophies
- ▣ **Zero-shot** performance is surprisingly strong — often matching supervised baselines
- ▣ **Fine-tuning** (especially LoRA) helps when domain data is available but must be done carefully

Practical Recommendations

- ▣ Always benchmark against **ARIMA/ETS** as a classical baseline
- ▣ Use **foundation models** for rapid prototyping and when domain expertise is limited
- ▣ Classical and foundation models are **complementary** — combine them via ensembles

Question 1

Question

□ What is the key advantage of self-attention over recurrence in sequence modeling?

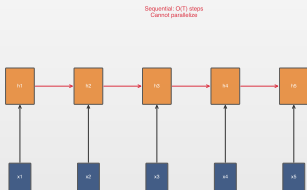
Answer Choices

- (A) Self-attention uses fewer parameters than RNNs
- (B) Self-attention processes all positions in parallel with $O(1)$ path length between any two positions
- (C) Self-attention does not require positional encoding
- (D) Self-attention is always faster than recurrence regardless of sequence length

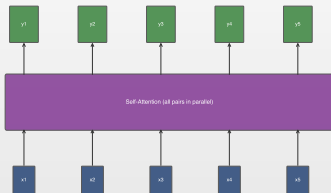
Question 1: Answer

RNN vs Transformer: Processing Paradigm

(a) RNN: Sequential Processing



(b) Transformer: Parallel Attention



Answer: (B)

- Self-attention connects every position to every other position directly (path length $O(1)$), enabling full parallelization. RNNs require $O(T)$ sequential steps, causing vanishing gradients and slow training.

Question 2

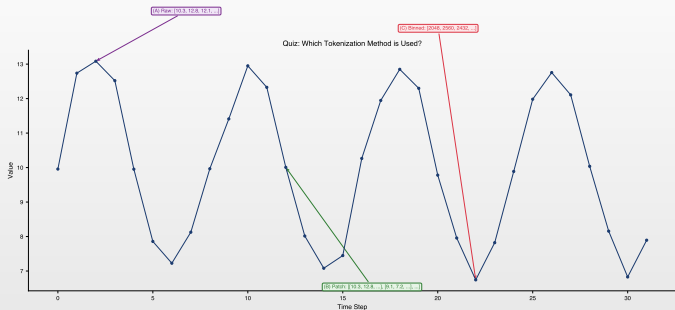
Question

- ☐ Which tokenization strategy does Chronos use to convert continuous time series into discrete tokens?

Answer Choices

- (A) Patching: grouping consecutive time points into patches
- (B) Scaling and quantization into discrete bins using the normal CDF
- (C) Direct linear embedding of raw values
- (D) Byte-pair encoding (BPE) borrowed from NLP

Question 2: Answer



Answer: (B)

- ☐ Chronos normalizes values (mean-scaling), then maps to 4096 bins via $b_t = \lfloor (B - 1) \cdot \Phi(\tilde{x}_t) \rfloor$. This allows reuse of the T5 vocabulary and cross-entropy loss, making forecasting a language modeling problem over quantized tokens.

Question 3

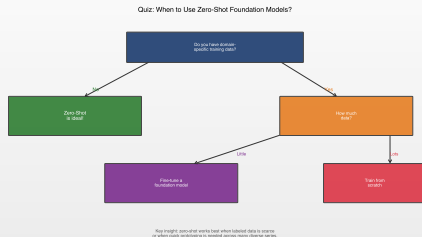
Question

- ☐ When should you prefer zero-shot inference over fine-tuning a foundation model?

Answer Choices

- (A) When you have millions of labeled domain-specific observations
- (B) When you need the highest possible accuracy on a single dataset
- (C) When you have limited labeled data, need rapid deployment, or face cold-start problems
- (D) When computational resources are not a concern

Question 3: Answer



Answer: (C)

- Zero-shot excels with limited data, cold-start, and rapid prototyping. The model's pre-trained knowledge transfers temporal patterns from billions of diverse series. Fine-tuning is preferred when sufficient domain data is available and marginal accuracy gains justify the cost.

Question 4

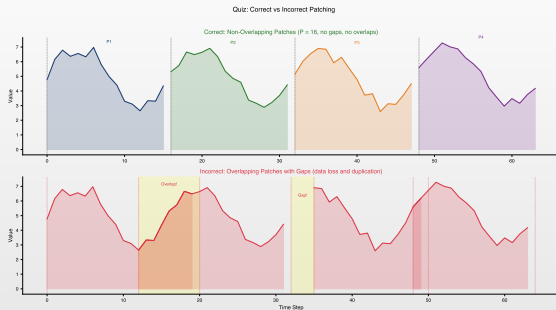
Question

□ What is “patching” in the context of Transformer-based time series models, and why does it help?

Answer Choices

- (A) Patching means filling missing values in the time series
- (B) Patching groups P consecutive points into one token, reducing attention complexity from $O(T^2)$ to $O((T/P)^2)$ while capturing local patterns
- (C) Patching is a data augmentation technique that creates new training series
- (D) Patching refers to updating model weights after deployment

Question 4: Answer



Answer: (B)

- Patching (PatchTST, 2023) groups P consecutive points into a single token. This reduces the effective sequence length for attention, captures local temporal patterns within each patch, and enables the model to handle much longer context windows.

Question 5

Question

- ☐ You need to forecast a multivariate energy dataset with 50 correlated variables at hourly frequency. Which foundation model is the best fit?

Answer Choices

- (A) Chronos — because it has the most model sizes to choose from
- (B) TimeGPT — because it is the easiest to use via API
- (C) Moirai — because it is the only model designed to handle any number of variates and any frequency natively
- (D) Lag-Llama — because it uses lag features that capture temporal dependencies

Question 5: Answer

Quiz: Model Selection by Scenario

Scenario	Data Size	Domain	Latency Requirement	Recommended Model
1. Retail demand forecasting (300k of SKUs)	Large (>1M points)	Retail	Low	TruqPTT (TruqPTL point-to-point)
2. Hospital patient admissions	Medium (10M points)	Healthcare	Medium	Chronos (over 100k)
3. High-frequency stock returns	Very Large (>10M points)	Finance	Very Low	TruqPTT (TruqPTL point-to-point)
4. Climate temperature forecasting	Small (<1M points)	Energy	High	Chronos (medium point-to-point)
5. IoT sensor anomaly detection	Medium (10M points)	Industrial	Low	TruqPTL (over 100k)

Answer: (C)

- Moirai is the only foundation model designed for **any-variate** forecasting. It uses variate IDs and flexible patching to handle 50 correlated variables simultaneously. Chronos and Lag-Llama are univariate; they would process each variable independently, ignoring cross-variable correlations.

Bibliography I

Foundational Papers

- Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention Is All You Need, *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 5998–6008.
- Ansari, A.F., Stella, L., Turkmen, C., et al. (2024). Chronos: Learning the Language of Time Series, *arXiv preprint arXiv:2403.07815*.
- Das, A., Kong, W., Leber, A., et al. (2024). A Decoder-Only Foundation Model for Time-Series Forecasting, *Proceedings of ICML 2024*.

Foundation Models

- Garza, A., & Mergenthaler-Canseco, M. (2024). TimeGPT-1, *International Journal of Forecasting*, forthcoming.
- Rasul, K., Ashok, A., Williams, A.R., et al. (2024). Lag-Llama: Towards Foundation Models for Probabilistic Time Series Forecasting, *Proceedings of NeurIPS 2024*.

Bibliography II

Architecture and Pre-Training

- Woo, G., Liu, C., Kumar, A., et al. (2024). Unified Training of Universal Time Series Forecasting Transformers, *Proceedings of ICML 2024*.
- Nie, Y., Nguyen, N.H., Sinthong, P., & Kalagnanam, J. (2023). A Time Series is Worth 64 Words: Long-term Forecasting with Transformers, *Proceedings of ICLR 2023*.
- Bommasani, R., Hudson, D.A., Adeli, E., et al. (2021). On the Opportunities and Risks of Foundation Models, *arXiv preprint arXiv:2108.07258*.

Benchmarks and Evaluation

- Godahewa, R., Bergmeir, C., Webb, G., et al. (2021). Monash Time Series Forecasting Archive, *Proceedings of NeurIPS 2021 Datasets and Benchmarks Track*.
- Hu, E.J., Shen, Y., Wallis, P., et al. (2022). LoRA: Low-Rank Adaptation of Large Language Models, *Proceedings of ICLR 2022*.

Bibliography III

Additional References

- ▣ Kaplan, J., McCandlish, S., Henighan, T., et al. (2020). Scaling Laws for Neural Language Models, *arXiv preprint arXiv:2001.08361*.
- ▣ Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018). The M4 Competition: 100,000 Time Series and 61 Forecasting Methods, *International Journal of Forecasting*, 36(1), 54–74.
- ▣ Petropoulos, F., et al. (2022). Forecasting: Theory and Practice, *International Journal of Forecasting*, 38(3), 845–1054.

Online Resources and Code

- ▣ **Quantlet:** <https://quantlet.com> – Code platform for quantitative methods
- ▣ **Quantinar:** <https://quantinar.com> – Learning platform for quantitative methods
- ▣ **GitHub TSA:** https://github.com/danpele/TSA/tree/main/TSA_ch11 – Python code for this chapter

Thank You!

Questions?

Course materials available at: <https://danpele.github.io/Time-Series-Analysis/>



Quantlet



Quantinar

