Transformers for Financial Time Series Forecasting

Transformers for forecasting

Oualid Missaoui

Agenda

- Time series forecasting key challenges
- Direct forecasting with transformers
- Auto-regressive (AR) forecasting with transformers
- Foundation Models for Time Series
- References

Time-Series Forecasting - Key Challenges

- Nonstationarity & regime shifts:
    - changing trends/volatility (drift)
- Seasonality & calendar effects:
    - multi-scale (daily/weekly/annual)
- Long-range dependencies:
    - need wide receptive fields efficiently
- Exogenous drivers:
    - static + known-future + observed-past covariates
- Missing/irregular sampling & cold starts:
    - gaps, new series, short histories
- Noise & outliers:
    - heavy tails; robust losses/likelihoods needed
- Data leakage & evaluation:
    - strict causal splits; rolling-origin backtests
- Scale mismatch:
    - per-series normalization (RevIN/instance norms)
- Horizon alignment:
    - target vs covariate timing; avoid peeking
- Uncertainty quantification:
    - intervals, quantiles, proper scoring rules

Direct (Multi-Horizon) Forecasting - Training

Goal. Learn a single mapping from a past context to the entire horizon in one shot:

$$\hat{\mathbf{y}}_{t+1:t+H} = f_\theta\big(\mathbf{y}_{t-L+1:t},\ \mathbf{x}^{\text{past}}_{t-L+1:t},\ \mathbf{x}^{\text{known}}_{t+1:t+H},\ \mathbf{s}\big)$$

- $\mathbf{y}$: target series; **(L)** = context length; **(H)** = forecast horizon
- $\mathbf{x}^{\text{past}}$: observed-past covariates (lags, rolling stats, sensors)
- $\mathbf{x}^{\text{known}}$: **known-future** covariates (calendar, tariffs, promos)
- $\mathbf{s}$: static features (entity metadata)

Data windows & batching

- Each training example is **(past window, future window)** $\to H$ labels.
- Slide windows over time/entities; maintain **causality** (no future target leakage).

Architecture pattern (Transformer)

- Encoder (or encoder–decoder) over past tokens + **future tokens** (one per step or per patch).
- **Masking:** block attention to true future targets; allow cross-horizon attention among future tokens.

Losses (horizon-aware)

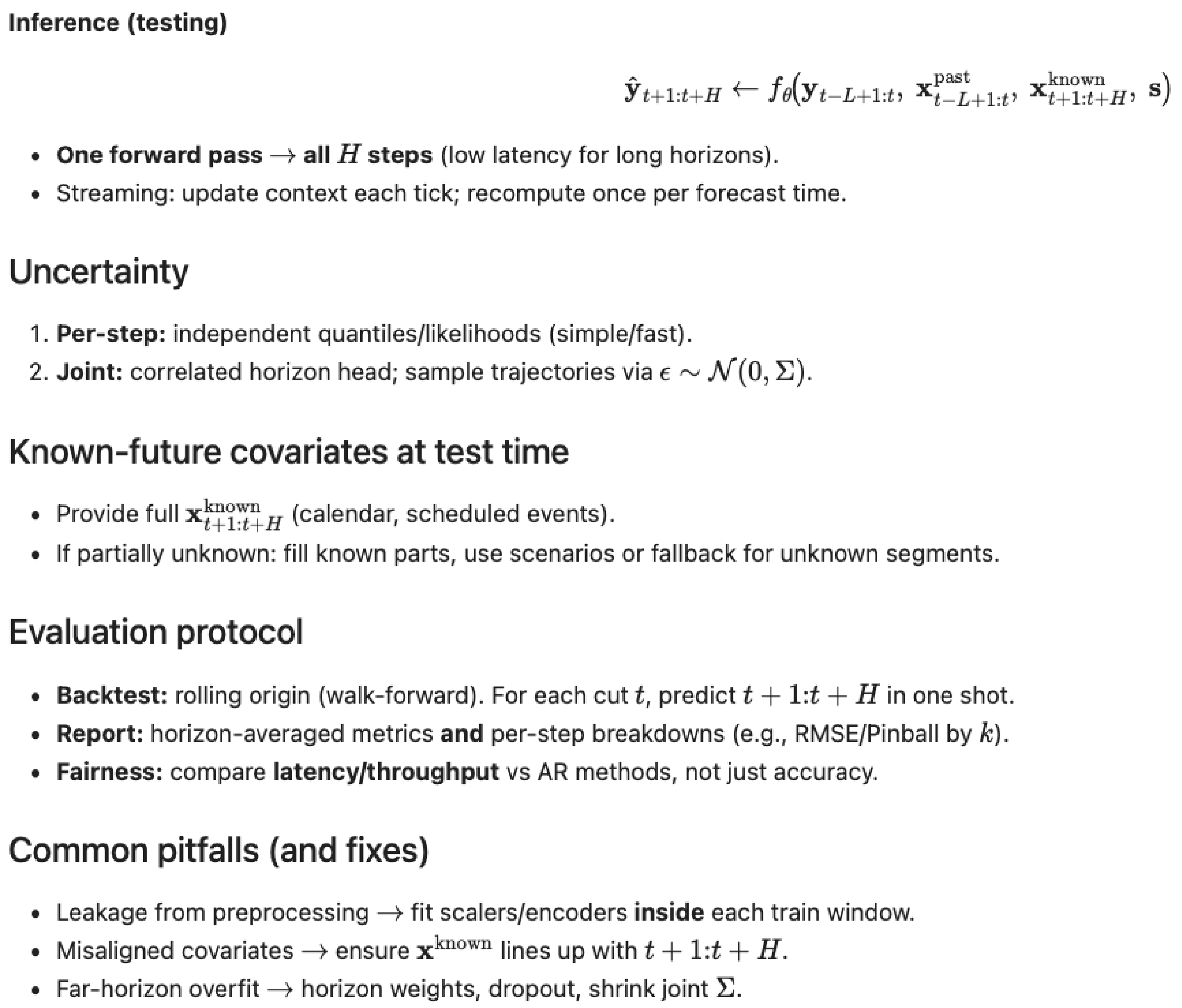
$$\mathcal{L}(\theta) = \sum_{k=1}^{H} \ell\big(\hat{y}_{t+k},\ y_{t+k}\big) \quad \text{(MSE/MAE/Pinball/CRPS)}$$

- Optional **joint** head (multivariate Gaussian/low-rank Σ) to model step-to-step correlations.
- Horizon weights w_k to emphasize near- vs far-term steps.

Regularization & tricks

- Dropout on future tokens; temporal smoothing across $\hat{y}_{t+1:t+H}$.
- Patch the past (Patch-style tokens) for long contexts; multi-scale attention; label noise.
- **Cross-validation:** rolling origin / blocked folds to avoid leakage.

Direct (Multi-Horizon) Forecasting - Testing



Inference (testing)

$$\hat{\mathbf{y}}_{t+1:t+H} \leftarrow f_\theta\big(\mathbf{y}_{t-L+1:t},\ \mathbf{x}^{\text{past}}_{t-L+1:t},\ \mathbf{x}^{\text{known}}_{t+1:t+H},\ \mathbf{s}\big)$$

- **One forward pass** $\rightarrow$ **all** H **steps** (low latency for long horizons).
- Streaming: update context each tick; recompute once per forecast time.

Uncertainty

1. **Per-step:** independent quantiles/likelihoods (simple/fast).
2. **Joint:** correlated horizon head; sample trajectories via $\epsilon \sim \mathcal{N}(0, \Sigma)$.

Known-future covariates at test time

- Provide full $\mathbf{x}^{\text{known}}_{t+1:t+H}$ (calendar, scheduled events).
- If partially unknown: fill known parts, use scenarios or fallback for unknown segments.

Evaluation protocol

- **Backtest:** rolling origin (walk-forward). For each cut t, predict $t+1{:}t+H$ in one shot.
- **Report:** horizon-averaged metrics **and** per-step breakdowns (e.g., RMSE/Pinball by k).
- **Fairness:** compare **latency/throughput** vs AR methods, not just accuracy.

Common pitfalls (and fixes)

- Leakage from preprocessing $\rightarrow$ fit scalers/encoders **inside** each train window.
- Misaligned covariates $\rightarrow$ ensure $\mathbf{x}^{\text{known}}$ lines up with $t+1{:}t+H$.
- Far-horizon overfit $\rightarrow$ horizon weights, dropout, shrink joint Σ.

DGM Autoregressive (AR) Forecasting - Training

Goal. Learn the one-step transition, then generate multi-step forecasts by rolling forward:

$$p_\theta(\mathbf{y}_{t+1:t+H} \mid \mathcal{H}_t) = \prod_{k=1}^{H} p_\theta\big(y_{t+k} \,\big|\, \mathcal{H}_t,\ y_{t+1:t+k-1},\ \mathbf{x}_{t+1:t+k}\big)$$

- $\mathcal{H}_t$: observed history up to t (targets + covariates).
- $\mathbf{x}$: covariates (past observed + possibly **known-future** at each step).

Training batches (teacher forcing)

- Inputs: past window $\mathbf{y}_{t-L+1:t}$, covariates up to $t+1$.
- Target: **next step** y_{t+1}.
- Shift and repeat over time/entities to form batches of $(\text{context} \to y_{t+1})$.

Architecture & masking (Transformer as DGM)

- **Decoder-only** or encoder–decoder with **strict causal mask** along time.
- Each query (for $t+1$) attends to tokens $\leq t$ only.
- For probabilistic DGMs, the head parameterizes a distribution (e.g., Gaussian/Quantiles/Discrete tokens).

Losses (next-step likelihood/quantile)

$$\mathcal{L}(\theta) = -\log p_\theta\big(y_{t+1} \mid \mathbf{y}_{t-L+1:t},\ \mathbf{x}_{t-L+1:t+1}\big) \quad \text{(or Pinball/CRPS for quantiles)}$$

DGM Autoregressive (AR) Forecasting - Testing

Rollout (testing)

1. Warm-start with observed history $\mathbf{y}_{t-L+1:t}$.
2. For $k = 1..H$:

$$\hat{y}_{t+k} \sim p_\theta\big(\cdot \mid \hat{\mathbf{y}}_{t-L+1:t+k-1},\ \mathbf{x}_{t+1:t+k}\big)$$

   Append $\hat{y}_{t+k}$ to the context and continue.

- **Latency/compute**: cost scales with H (one decode per step, unless using block decoding).
- **Streaming-friendly**: emit one step per tick; no need to re-forecast the full horizon each time.

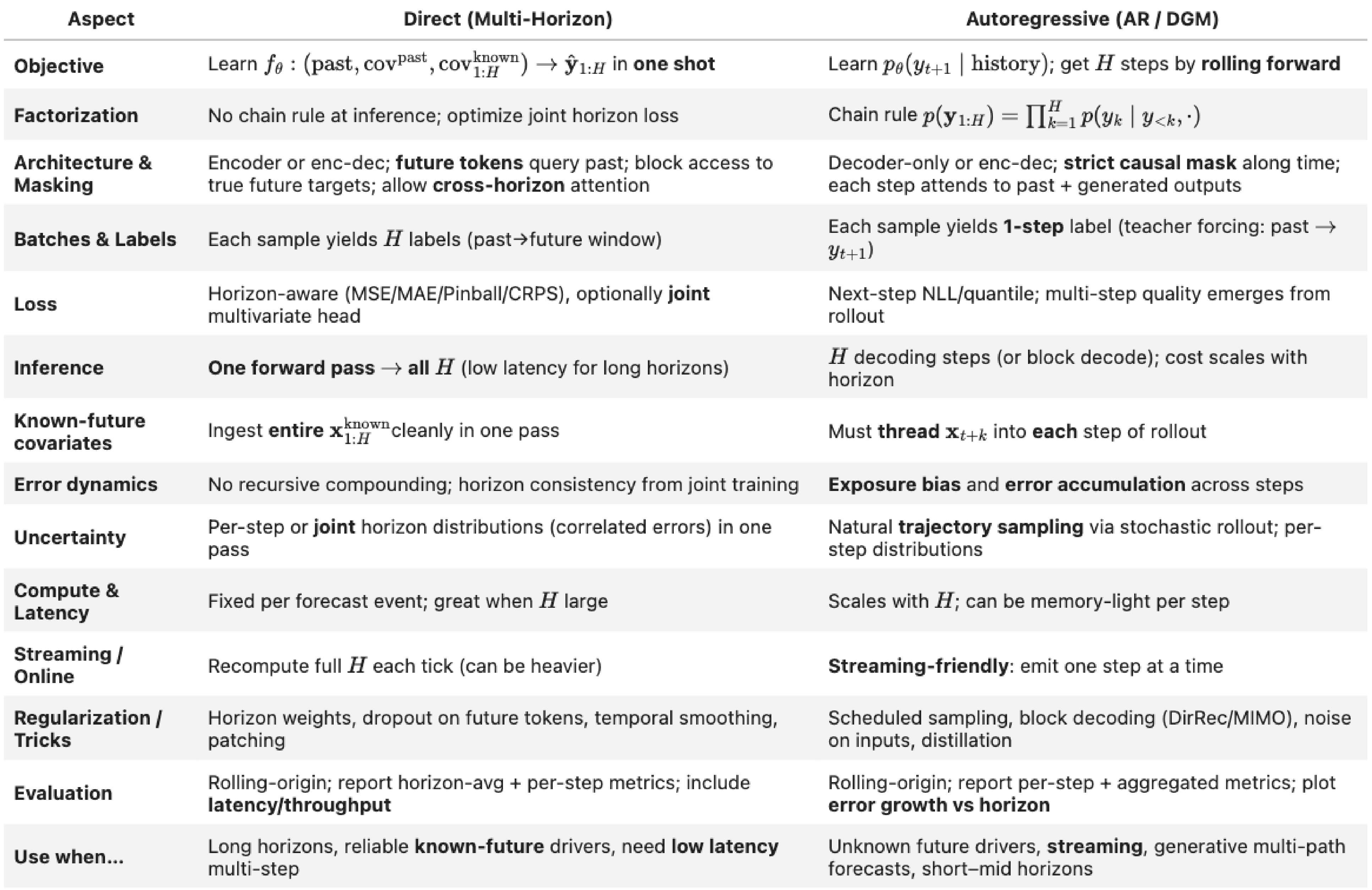
Uncertainty

- Naturally **trajectory-level**: sample multiple rollouts (MC) from p_θ to get fan charts/intervals.
- Distributional heads (quantiles/MoG/tokenized) yield calibrated per-step predictive distributions.

Known-future covariates at test time

- Thread $\mathbf{x}_{t+k}$ into **each step** of the rollout.
- If partially known: scenario trees (branch per covariate scenario), or condition only on observed drivers.

Direct vs Autoregressive (AR) Transformers

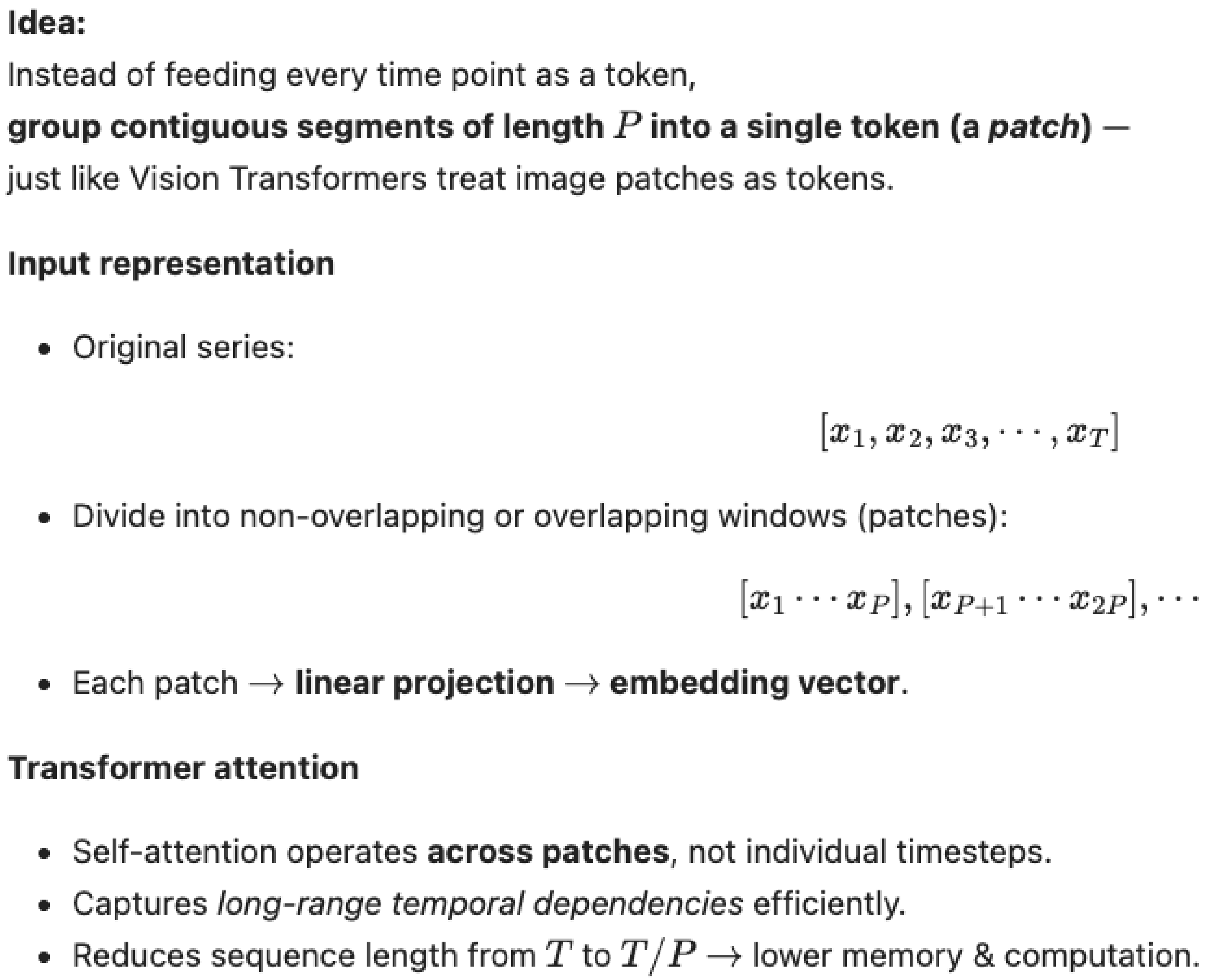


Aspect	Direct (Multi-Horizon)	Autoregressive (AR / DGM)
Objective	Learn $f_\theta : (\text{past}, \text{cov}^{\text{past}}, \text{cov}^{\text{known}}_{1:H}) \to \hat{\mathbf{y}}_{1:H}$ in **one shot**	Learn $p_\theta(y_{t+1} \mid \text{history})$; get H steps by **rolling forward**
Factorization	No chain rule at inference; optimize joint horizon loss	Chain rule $p(\mathbf{y}_{1:H}) = \prod_{k=1}^{H} p(y_k \mid y_{<k}, \cdot)$
Architecture & Masking	Encoder or enc-dec; **future tokens** query past; block access to true future targets; allow **cross-horizon** attention	Decoder-only or enc-dec; **strict causal mask** along time; each step attends to past + generated outputs
Batches & Labels	Each sample yields H labels (past→future window)	Each sample yields **1-step** label (teacher forcing: past $\to$ y_{t+1})
Loss	Horizon-aware (MSE/MAE/Pinball/CRPS), optionally **joint** multivariate head	Next-step NLL/quantile; multi-step quality emerges from rollout
Inference	**One forward pass** $\to$ **all** H (low latency for long horizons)	H decoding steps (or block decode); cost scales with horizon
Known-future covariates	Ingest **entire** $\mathbf{x}^{\text{known}}_{1:H}$ cleanly in one pass	Must **thread** $\mathbf{x}_{t+k}$ into **each** step of rollout
Error dynamics	No recursive compounding; horizon consistency from joint training	**Exposure bias** and **error accumulation** across steps
Uncertainty	Per-step or **joint** horizon distributions (correlated errors) in one pass	Natural **trajectory sampling** via stochastic rollout; per-step distributions
Compute & Latency	Fixed per forecast event; great when H large	Scales with H; can be memory-light per step
Streaming / Online	Recompute full H each tick (can be heavier)	**Streaming-friendly**: emit one step at a time
Regularization / Tricks	Horizon weights, dropout on future tokens, temporal smoothing, patching	Scheduled sampling, block decoding (DirRec/MIMO), noise on inputs, distillation
Evaluation	Rolling-origin; report horizon-avg + per-step metrics; include **latency/throughput**	Rolling-origin; report per-step + aggregated metrics; plot **error growth vs horizon**
Use when...	Long horizons, reliable **known-future** drivers, need **low latency** multi-step	Unknown future drivers, **streaming**, generative multi-path forecasts, short–mid horizons

Foundation Models for Time Series

- Moirai (2024, Salesforce)
- Lag-Llama (2024, Meta FAIR)
- Chronos (2024, Amazon)
- TimesFM (2024, Google DeepMind)

“Patching” in Time-series Transformers



Idea:
Instead of feeding every time point as a token,
group contiguous segments of length P into a single token (a *patch*) —
just like Vision Transformers treat image patches as tokens.

Input representation

- Original series:

$$[x_1, x_2, x_3, \cdots, x_T]$$

- Divide into non-overlapping or overlapping windows (patches):

$$[x_1 \cdots x_P], [x_{P+1} \cdots x_{2P}], \cdots$$

- Each patch $\rightarrow$ **linear projection** $\rightarrow$ **embedding vector**.

Transformer attention

- Self-attention operates **across patches**, not individual timesteps.
- Captures *long-range temporal dependencies* efficiently.
- Reduces sequence length from T to $T/P \rightarrow$ lower memory & computation.

Moirai

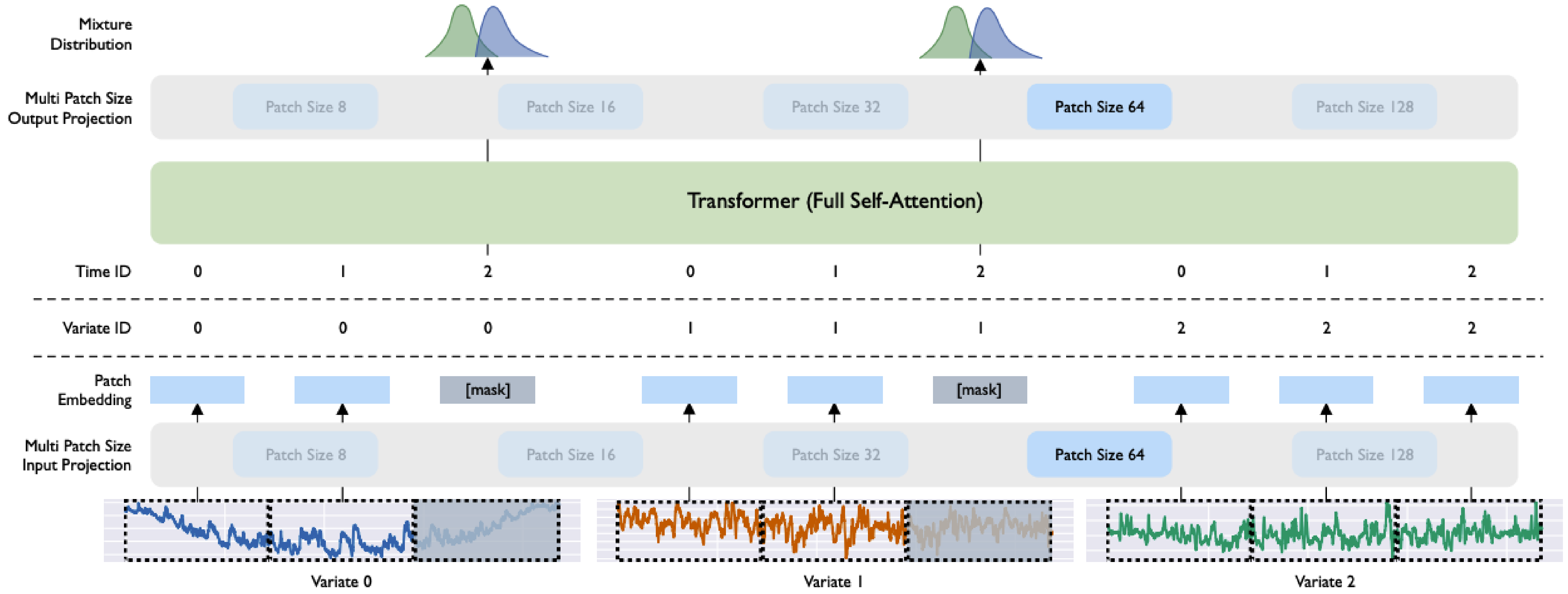


Figure 2. Overall architecture of MOIRAI. Visualized is a 3-variate time series, where variates 0 and 1 are target variables (i.e. to be forecasted, and variate 2 is a dynamic covariate (values in forecast horizon known). Based on a patch size of 64, each variate is patchified into 3 tokens. The patch embeddings along with sequence and variate id are fed into the Transformer. The shaded patches represent the forecast horizon to be forecasted, whose corresponding output representations are mapped into the mixture distribution parameters.

Lag-Llama

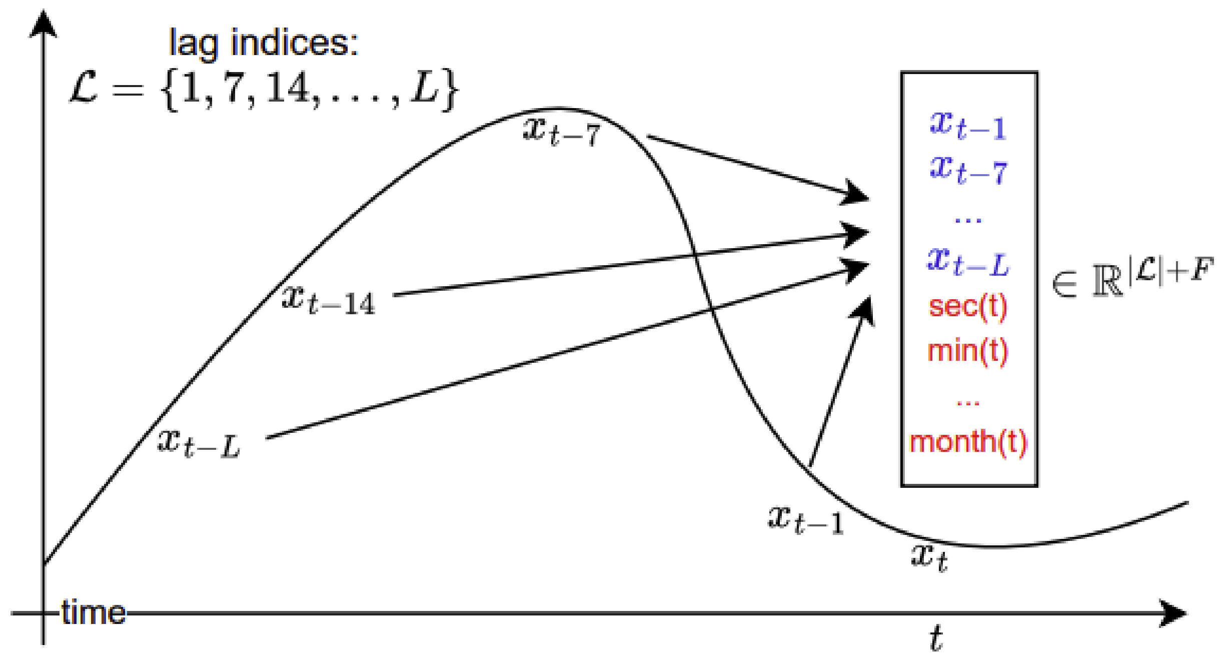


Figure 1: For a time series, we depict the tokenization at the timestep t of the value x_t which contains lag features constructed using an example set of lag indices $\mathcal{L}$, where each value in the vector is from the *past* of x_t (in blue), and F possible temporal covariates (date-time features) constructed from timestamp t (red).

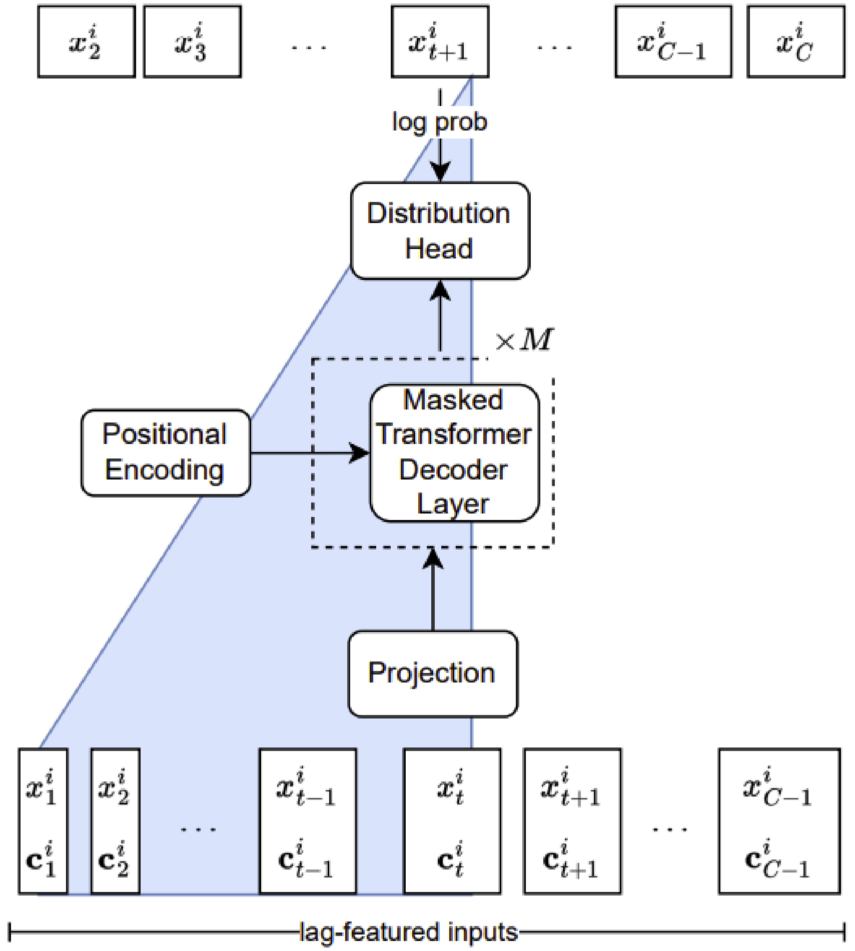


Figure 2: The `Lag-Llama` architecture. `Lag-Llama` learns to output a distribution over the values of the next time step based on lagged input features. The input to the model is the token of a univariate time series i at a given timestep, $\mathbf{x}_t^i$, constructed as described in Sec.4.1. Here, we use $\mathbf{c}_t^i$ to refer to all additional covariates used along with the value at a timestep t, which include the $|\mathcal{L}|$ lags, F date-time features, and summary statistics. The inputs are projected through M masked decoder layers. The features are then passed through the distribution head and trained to predict the parameters of the forecast distribution of the next timestep.

Chronos

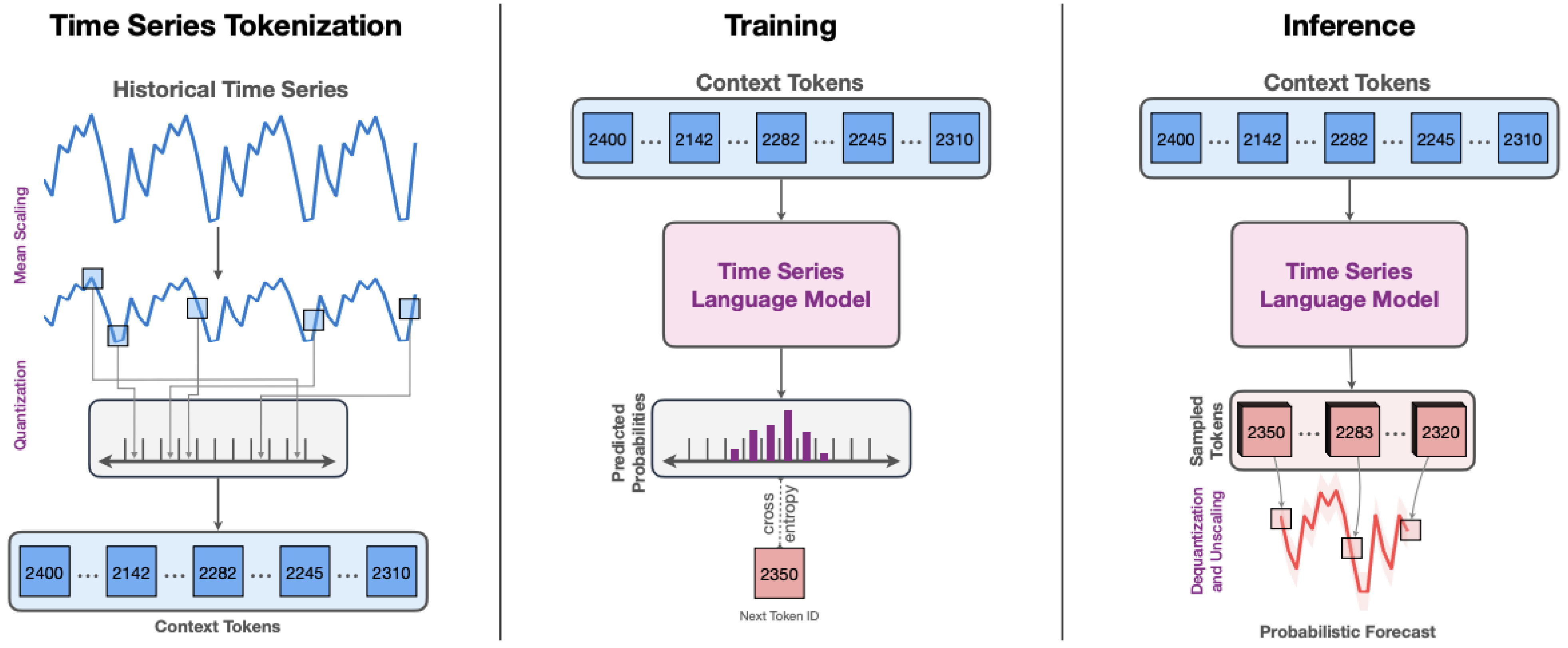


Figure 1: High-level depiction of CHRONOS. (**Left**) The input time series is scaled and quantized to obtain a sequence of tokens. (**Center**) The tokens are fed into a language model which may either be an encoder-decoder or a decoder-only model. The model is trained using the cross-entropy loss. (**Right**) During inference, we autoregressively sample tokens from the model and map them back to numerical values. Multiple trajectories are sampled to obtain a predictive distribution.

Abdul Fatir Ansari (2024), Chronos: Learning the Language of Time Series

TimesFM

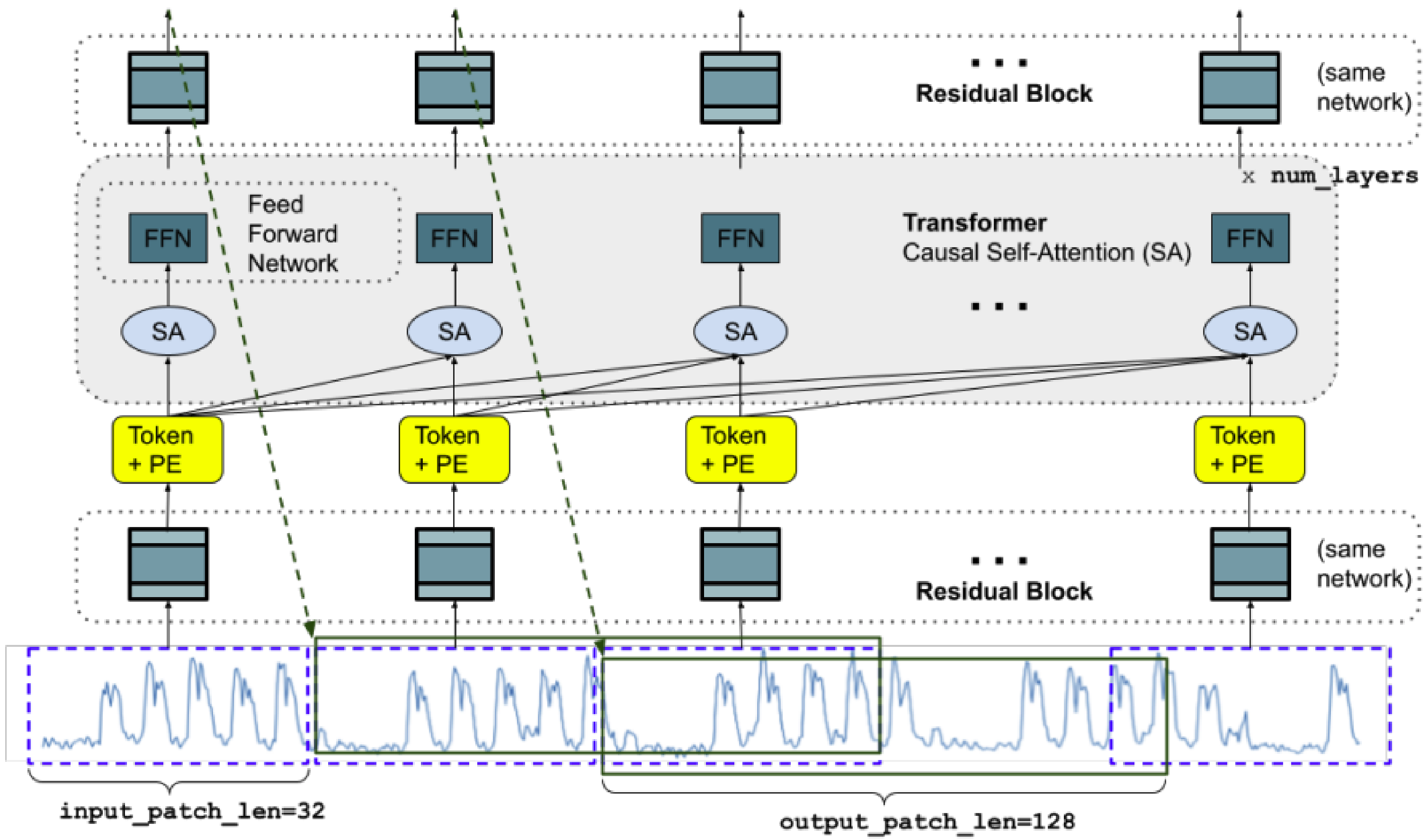
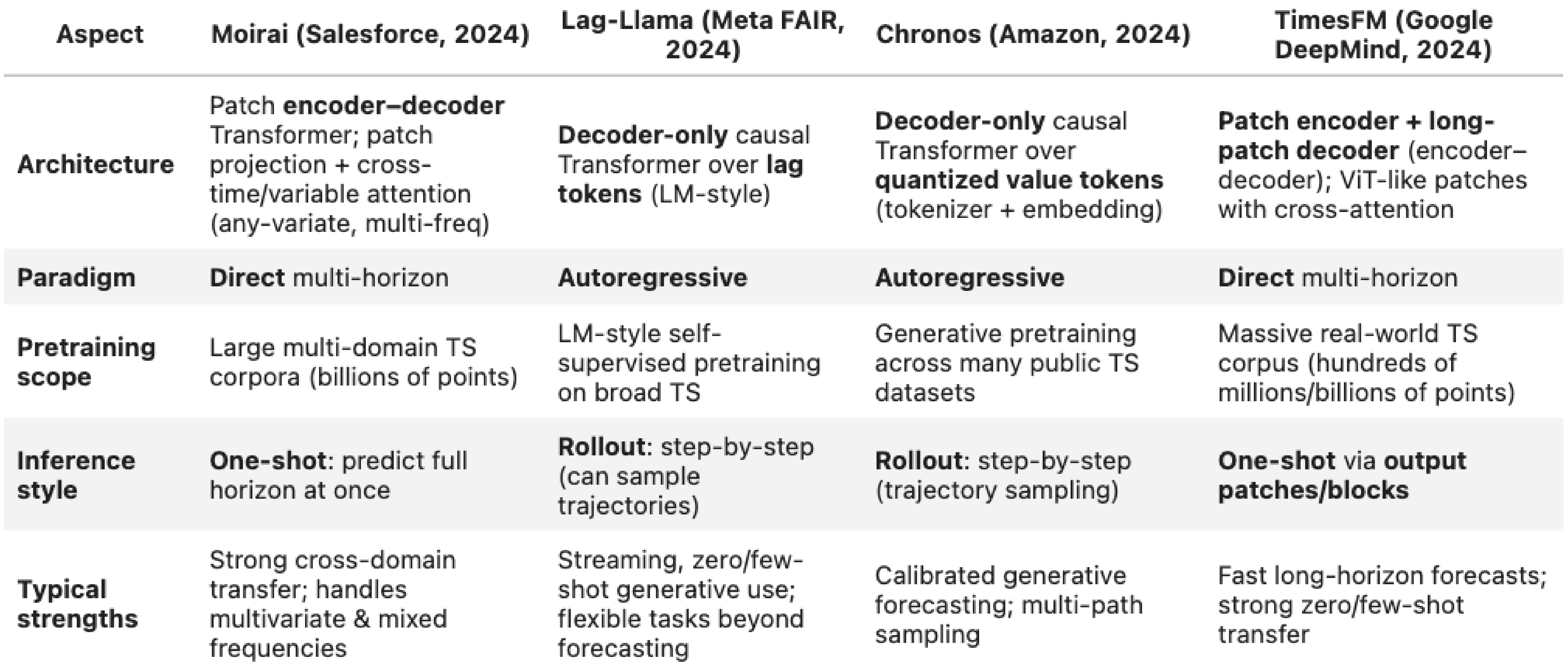


Figure 1: We provide an illustration of the TimesFM model architecture during training, where we show a input time-series of a specific length that can be broken down into input patches. Each patch along is processed into a vector by a residual block (as defined in the model definition) to the model dimension of the transformer layers. The vector is then added to positional encodings and fed into n_l stacked transformer layers. SA refers to self-attention (note that we use multi-head causal attention) and FFN is the fully connected layer in the transformer. The output tokens are then mapped through a residual block to an output of size `output_patch_len`, which is the forecast for the time window following the last input patch seen by the model so far.

Comparing Time-Series Foundation Models



Aspect	Moirai (Salesforce, 2024)	Lag-Llama (Meta FAIR, 2024)	Chronos (Amazon, 2024)	TimesFM (Google DeepMind, 2024)
Architecture	Patch **encoder–decoder** Transformer; patch projection + cross-time/variable attention (any-variate, multi-freq)	**Decoder-only** causal Transformer over **lag tokens** (LM-style)	**Decoder-only** causal Transformer over **quantized value tokens** (tokenizer + embedding)	**Patch encoder + long-patch decoder** (encoder–decoder); ViT-like patches with cross-attention
Paradigm	**Direct** multi-horizon	**Autoregressive**	**Autoregressive**	**Direct** multi-horizon
Pretraining scope	Large multi-domain TS corpora (billions of points)	LM-style self-supervised pretraining on broad TS	Generative pretraining across many public TS datasets	Massive real-world TS corpus (hundreds of millions/billions of points)
Inference style	**One-shot**: predict full horizon at once	**Rollout**: step-by-step (can sample trajectories)	**Rollout**: step-by-step (trajectory sampling)	**One-shot** via **output patches/blocks**
Typical strengths	Strong cross-domain transfer; handles multivariate & mixed frequencies	Streaming, zero/few-shot generative use; flexible tasks beyond forecasting	Calibrated generative forecasting; multi-path sampling	Fast long-horizon forecasts; strong zero/few-shot transfer

References

- Siva Rama Krishna Kottapalli et al (2025), Foundation Models for Time Series: A Survey
- Gerald Woo et al (2024), Unified Training of Universal Time Series Forecasting Transformers
- Abhimanyu Das et al (2024), A decoder-only foundation model for times series forecasting
- Abdul Fatir Ansari (2024), Chronos: Learning the Language of Time Series
- Kasif Rasul et al (2024), Lag-Llama: Towards Foundation Models for Probabilistic Time Series Forecasting