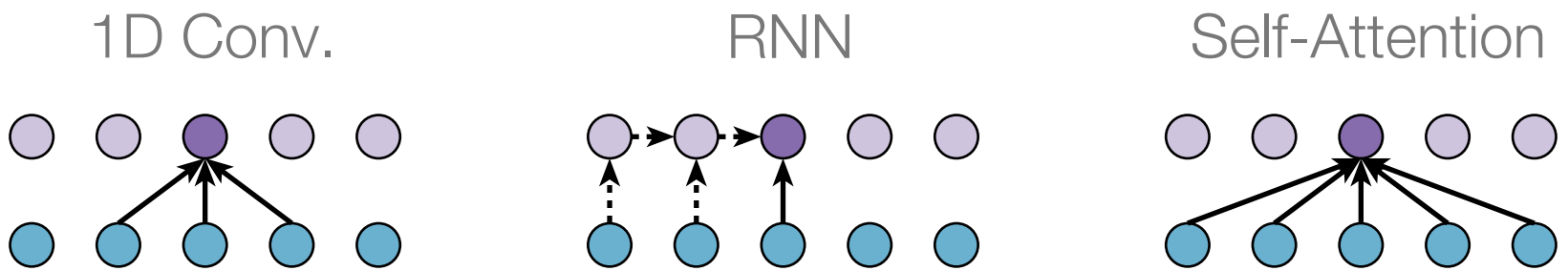
Deep Learning for Time Series

Session 6: State Space Models

Romain Tavenard

Motivation

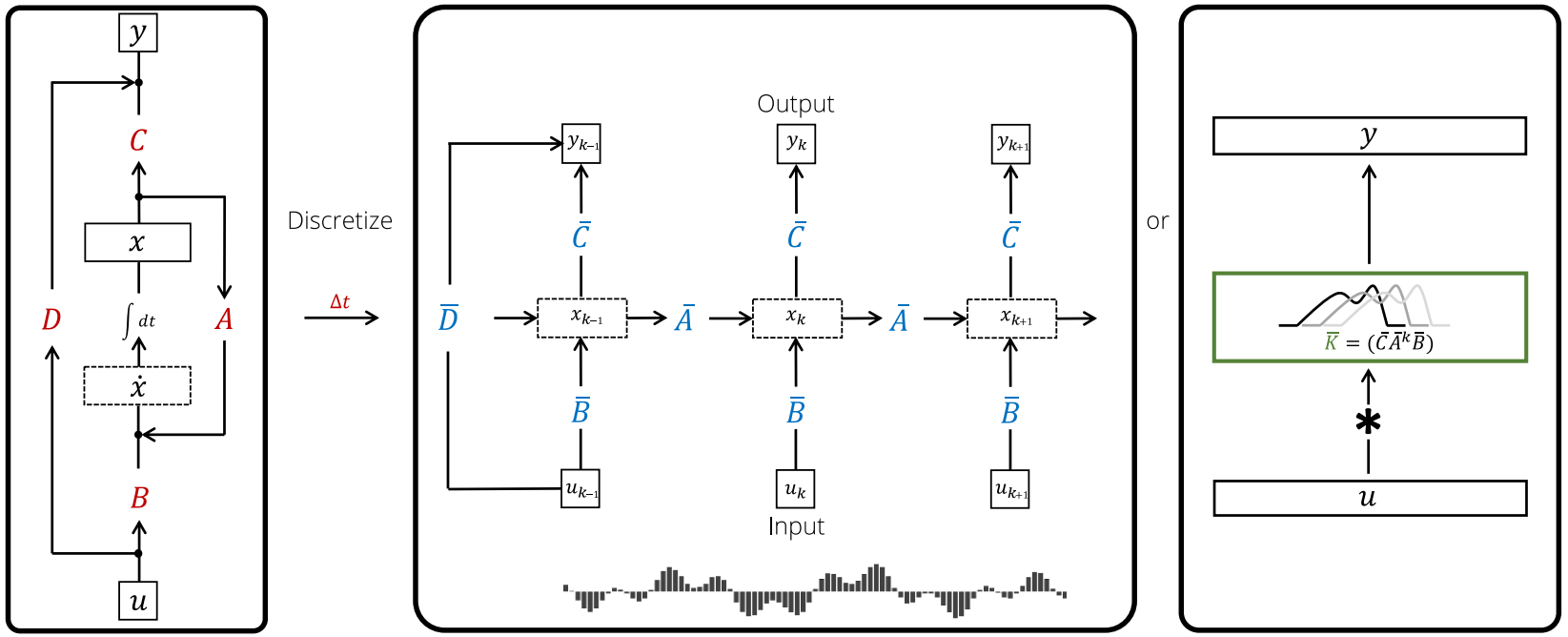
Caveats of previously introduced models



- RNNs: sequential, limits parallelism
- CNNs: local receptive field
- Attention-based models: $O(T^2)$ complexity

What State Space Models (SSMs) can offer

- Continuous-time dynamics
- Long-range dependencies
- Linear (or near-linear) complexity



Source: “Combining Recurrent, Convolutional, and Continuous-time Models with Linear State-Space Layers”, NeurIPS’21, by Albert Gu et al.

State Space Models

Classical SSM formulation

Latent state $h_t \in \mathbb{R}^d$ evolves over time following:

$$\begin{cases} \dot{h}(t) = Ah(t) + Bx(t) \\ o(t) = Ch(t) + \cancel{Dx(t)} \end{cases}$$

- $x(t)$: input
- $o(t)$: output
- $h(t)$: latent state

In practice, we often set $D = 0$ for simplicity (can be later recovered through residual connection in Deep SSM anyway).

SSMs vs Neural ODEs

SSMs

$$\dot{h}(t) = Ah(t) + Bx(t)$$

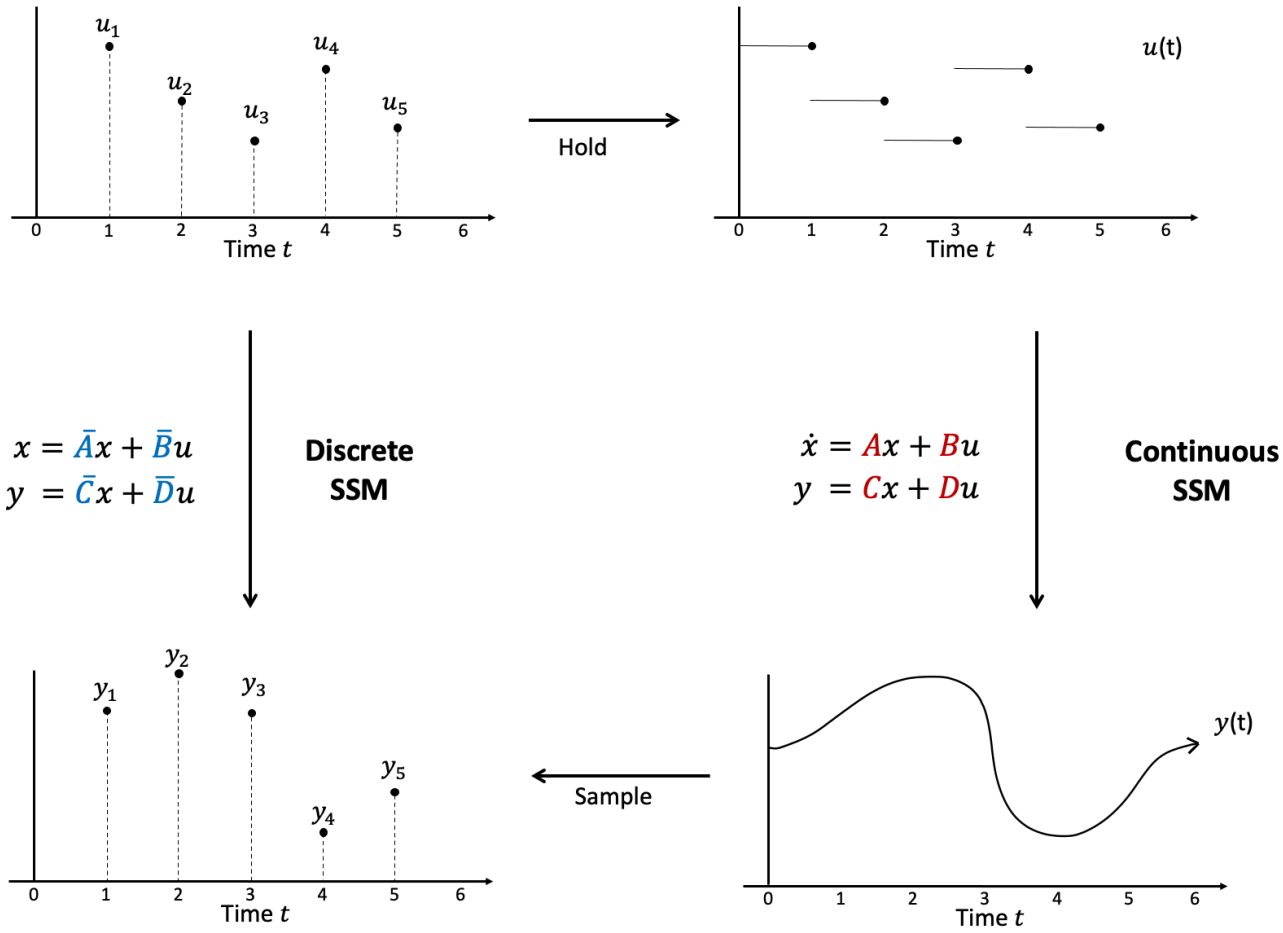
- Linear dynamics specified with respect to the hidden state $h(t)$

Neural ODEs

$$\dot{x}(t) = f_\theta(x(t))$$

- Non-linear dynamics specified with respect to the input $x(t)$

Step 1: Time discretization (S4)



Source: "Structured State Spaces: Combining Continuous-Time, Recurrent, and Convolutional Models", a blogpost by by Albert Gu et al. (2022)

Step 1: Time discretization (S4)

$$\dot{h}(t) = Ah(t) + Bx(t)$$

Goal: discretize time with step Δ between indexes `t-1` and `t`.

1. Textbook ODE solving gives the following form for $h(t)$:

$$h(t) = e^{A\Delta} h(t - \Delta) + \int_0^{\Delta} e^{As} Bx(t - s)\,\mathrm{d}s$$

Step 1: Time discretization (S4)

$$\dot{h}(t) = Ah(t) + Bx(t)$$

Goal: discretize time with step Δ between indexes `t-1` and `t`.

1. Textbook ODE solving gives the following form for $h(t)$:

$$h(t) = e^{A\Delta} h(t - \Delta) + \int_0^{\Delta} e^{As} B x(t - s) \, \mathrm{d}s$$

2. Assume $x(t)$ is constant between $t - \Delta$ and t:

$$h(t) = e^{A\Delta} h(t - \Delta) + \left(\int_0^{\Delta} e^{As} \, \mathrm{d}s \right) B x(t)$$

Step 1: Time discretization (S4)

$$\dot{h}(t) = Ah(t) + Bx(t)$$

Goal: discretize time with step Δ between indexes `t-1` and `t`.

1. Textbook ODE solving gives the following form for $h(t)$:

$$h(t) = e^{A\Delta} h(t - \Delta) + \int_0^{\Delta} e^{As} Bx(t - s)\,\mathrm{d}s$$

2. Assume $x(t)$ is constant between $t - \Delta$ and t:

$$h(t) = e^{A\Delta} h(t - \Delta) + \left(\int_0^{\Delta} e^{As}\,\mathrm{d}s \right) Bx(t)$$

3. Get the discrete update rule:

$$h_t = \overline{A} h_{t-1} + \overline{B} x_t$$

Step 1: Time discretization (S4)

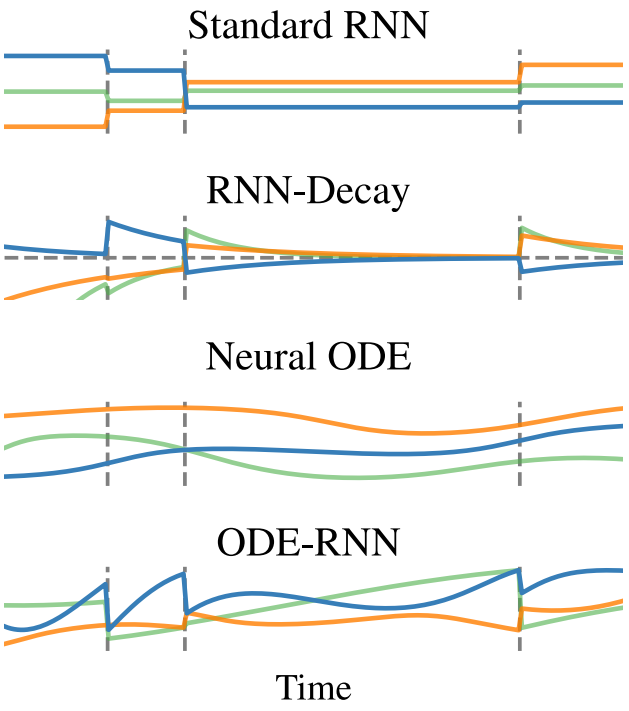
Now we have the discrete system:

$$\begin{cases} h_t = \overline{A} h_{t-1} + \overline{B} x_t \\ o_t = \overline{C} h_t \end{cases}$$

with:

- $\overline{A} = e^{A\Delta}$
- $\overline{B} = \left(\int_0^{\Delta} e^{As} \, \mathrm{d}s \right) B$

  $= A^{-1} \left(e^{A\Delta} - I \right) B$
- $\overline{C} = C$



Source: "Latent ODEs for Irregularly-Sampled Time Series", NeurIPS'20

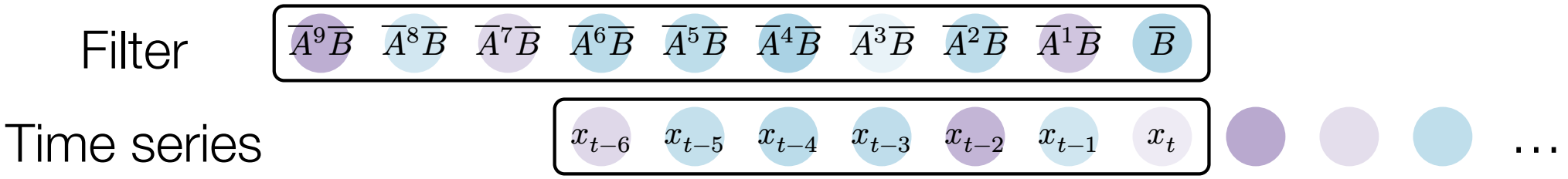
Main idea behind S4: Parametrize A, B, C such that the resulting discrete system has good properties (efficient computation, stable training, long memory).

Step 1: Time discretization (S4)

Efficient computation: The convolutional trick

Let us assume $h_0 = 0$, then we get:

$$h_t = \sum_{k=1}^{t} \overline{A}^{t-k} \overline{B} x_k$$



Efficient implementation:

- Easy-to-compute powers of $\overline{A}$ (cf. parametrization, next slide)
- Compute the convolution using FFT

Step 1: Time discretization (S4)

Long-term memory: The parametrization trick

- Risk of vanishing / exploding for long-range terms ($\overline{A}^k \overline{B}$)
- S4's solution: smart parametrization of A

Step 1: Time discretization (S4)

Long-term memory: The parametrization trick

- Risk of vanishing / exploding for long-range terms ($\overline{A}^k \overline{B}$)
- S4's solution: smart parametrization of A
  - Typical choice: $A = V \Lambda V^{-1}$ with eigenvalues of the form

$$\lambda_j(A) = \underbrace{-\alpha_j}_{>0} + i\omega_j$$

  - $\overline{A} = e^{A\Delta} = V e^{\Lambda \Delta} V^{-1}$ with $\lambda_j\left(\overline{A}\right) = e^{-\alpha_j \Delta} e^{i\omega_j \Delta}$

Step 1: Time discretization (S4)

Long-term memory: The parametrization trick

- Risk of vanishing / exploding for long-range terms ($\overline{A}^k \overline{B}$)
- S4's solution: smart parametrization of A
  - Typical choice: $A = V \Lambda V^{-1}$ with eigenvalues of the form

$$\lambda_j(A) = -\underbrace{\alpha_j}_{>0} + i\omega_j$$

  - $\overline{A} = e^{A\Delta} = V e^{\Lambda\Delta} V^{-1}$ with $\lambda_j\left(\overline{A}\right) = e^{-\alpha_j \Delta} e^{i\omega_j \Delta}$
  - $\overline{A}^k = V\left(e^{\Lambda\Delta}\right)^k V^{-1}$
  - Small $\alpha_j \Rightarrow e^{-\alpha_j \Delta} \approx 1 \Rightarrow$ long-term memory

Step 2: Input-dependent dynamics (Mamba)

Pure linearity in SSMs:

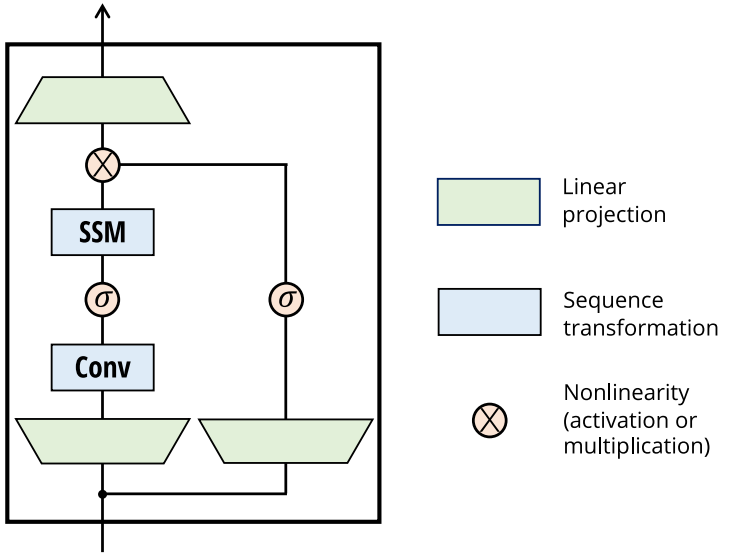
✓ Great long-range memory

✕ Limited expressivity

Solution: Make dynamics input-dependent

$$\begin{cases} h_t = \overline{A}_t h_{t-1} + \overline{B}_t x_t \\ o_t = \overline{C}_t h_t \end{cases}$$

Now the state transition adapts to the input as in attention-based models



Source: "Mamba: Linear-Time Sequence Modeling with Selective State Spaces", COLM'24

Step 2: Input-dependent dynamics (Mamba)

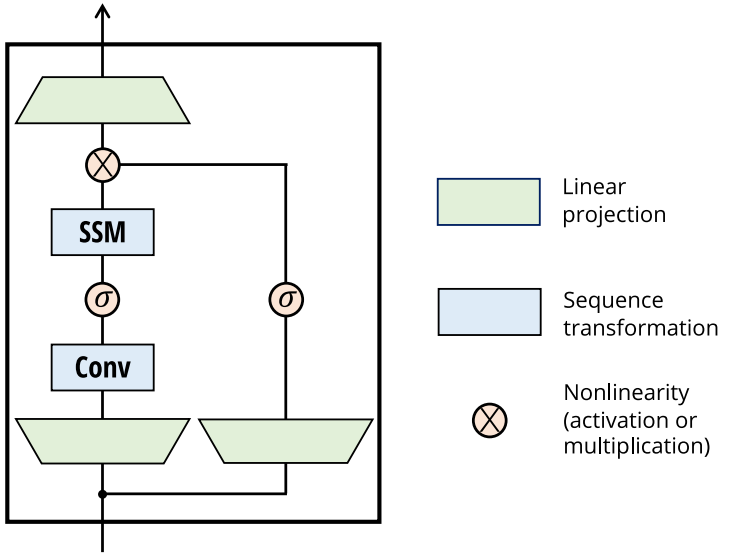
$$\begin{cases} h_t = \overline{A}_t h_{t-1} + \overline{B}_t x_t \\ o_t = \overline{C}_t h_t \end{cases}$$

where:

$$\overline{A}_t = \exp(\Delta_t A)$$

$$\overline{B}_t = (\exp(\Delta_t A) - I)(\Delta_t A)^{-1} \Delta_t B$$

and Δ_t, $\overline{C}_t$ are functions of x_t.



Source: "Mamba: Linear-Time Sequence Modeling with Selective State Spaces", COLM'24

Mamba: Efficiency without convs

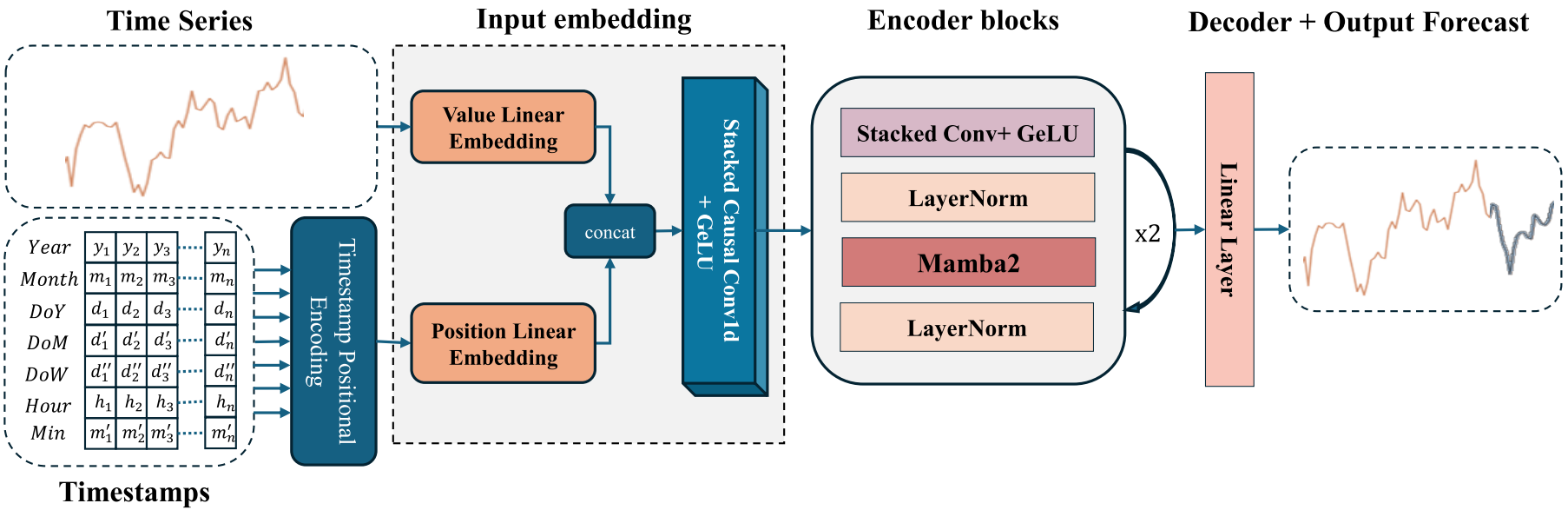
The challenge: input-dependent parameters break the convolutional structure.

Mamba's design choices:

1. Scan-based computation
   - Process sequentially (like RNN)
   - Still efficient thanks to low-level optimizations (e.g., GPU kernels) using the linearity of the state update
2. Gating mechanisms
   - Controls what information is retained in the state
   - Similar to LSTM/GRU gates

$\Rightarrow O(T)$ complexity

Mamba4Cast: a foundation model for (univariate) TS forecasting



Source: “Mamba4Cast: Efficient Zero-Shot Time Series Forecasting with State Space Models”, NeurIPS’24 Workshop