Predictability Enables Parallelization of Nonlinear State Space Models

Xavier Gonzalez*, Leo Kozachkov*,
David M. Zoltowski, Kenneth L. Clarkson,
Scott W. Linderman

Nonlinear State Space Models (nSSMs) are everywhere!

$$s_t = f_t(s_{t-1}), \;\; s_t \in \mathbb{R}^D$$

$$\begin{tikzcd}
s_0 \arrow[r, "f_1"] & s_1 \arrow[r, "f_2"] & s_2 & \cdots & s_{T-1} \arrow[r, "f_T"] & s_T
\end{tikzcd}$$

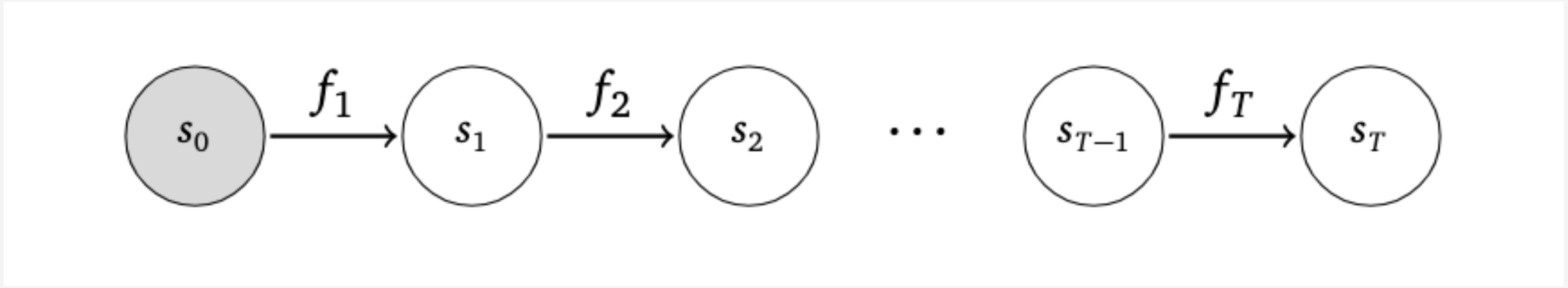
We can include inputs by defining $f_t(s_{t-1}) = f(s_{t-1}, u_t)$

$$\begin{tikzcd}
s_0 \arrow[r, "f_1"] & s_1 \arrow[r, "f_2"] & s_2 & \cdots & s_{T-1} \arrow[r, "f_T"] & s_T \\
 & u_1 \arrow[u] & u_2 \arrow[u] & \cdots & u_{T-1} \arrow[u] & u_T \arrow[u]
\end{tikzcd}$$

Examples include

- Recurrent Neural Networks (RNNs)
- Markov chain Monte Carlo (MCMC)
- Sampling from a diffusion model
- The blocks of a transformer model *over depth*
- Gradient descent
- State in reinforcement learning agent
- …and more…

Evaluating nonlinear SSMs seemed to be "inherently sequential"…



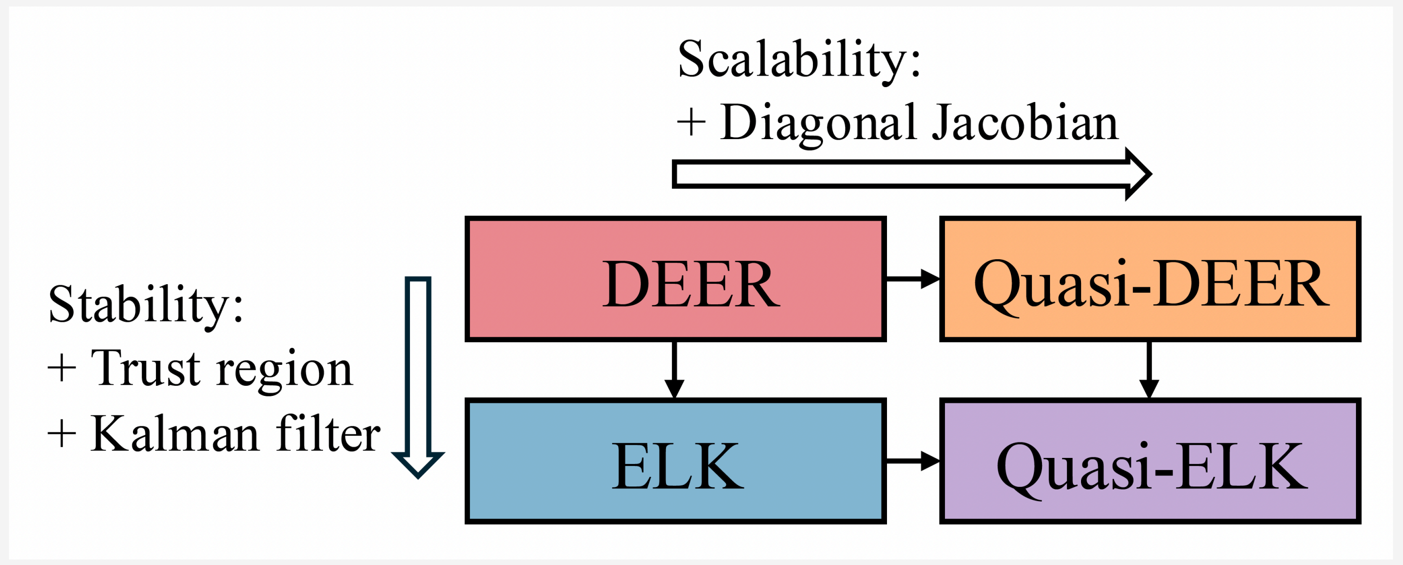
$$\begin{tikzcd} s_0 \arrow[r, "f_1"] & s_1 \arrow[r, "f_2"] & s_2 & \cdots & s_{T-1} \arrow[r, "f_T"] & s_T \end{tikzcd}$$

…but turns out that nonlinear SSMs can be parallelized!

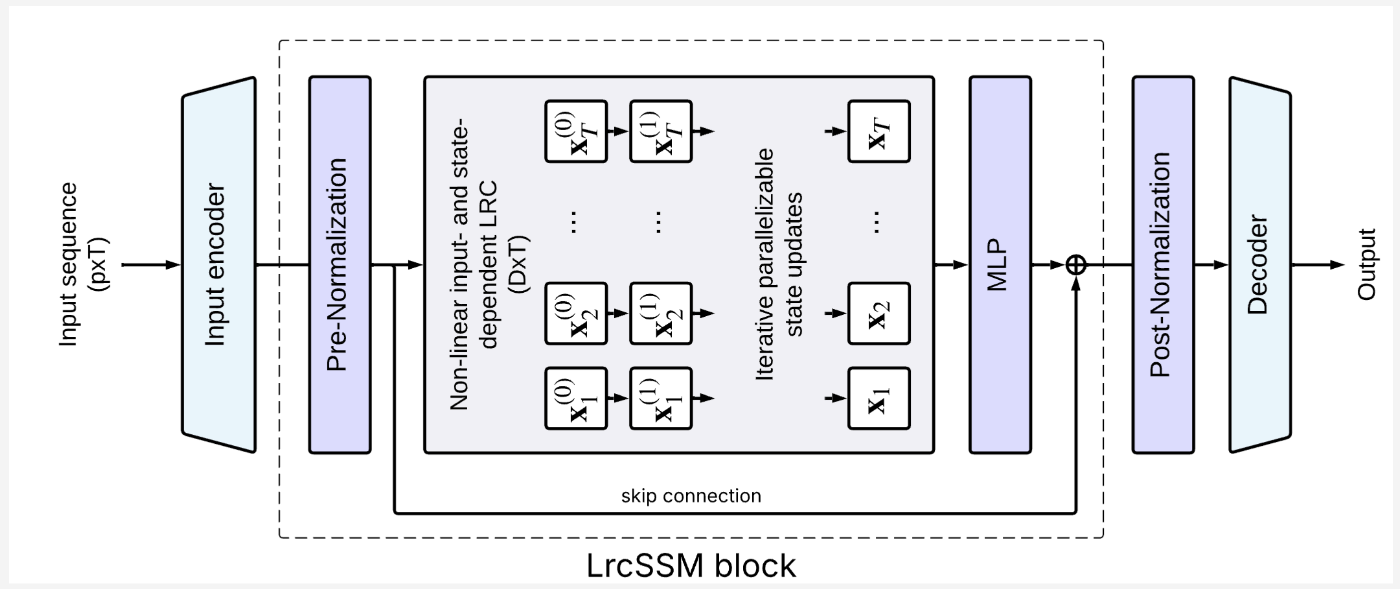
Danieli et al, NeurIPS ’23

$$\begin{bmatrix} I & & & \\ -J_{f_1}\big|_{z_0^k} & I & & \\ & \ddots & \ddots & \\ & & -J_{f_L}\big|_{z_{L-1}^k} & I \end{bmatrix} \begin{bmatrix} \delta z_0^k \\ \delta z_1^k \\ \vdots \\ \delta z_L^k \end{bmatrix} = \begin{bmatrix} f_0(x) - z_0^k \\ f_1(z_0^k) - z_1^k \\ \vdots \\ f_L(z_{L-1}^k) - z_L^k \end{bmatrix}$$

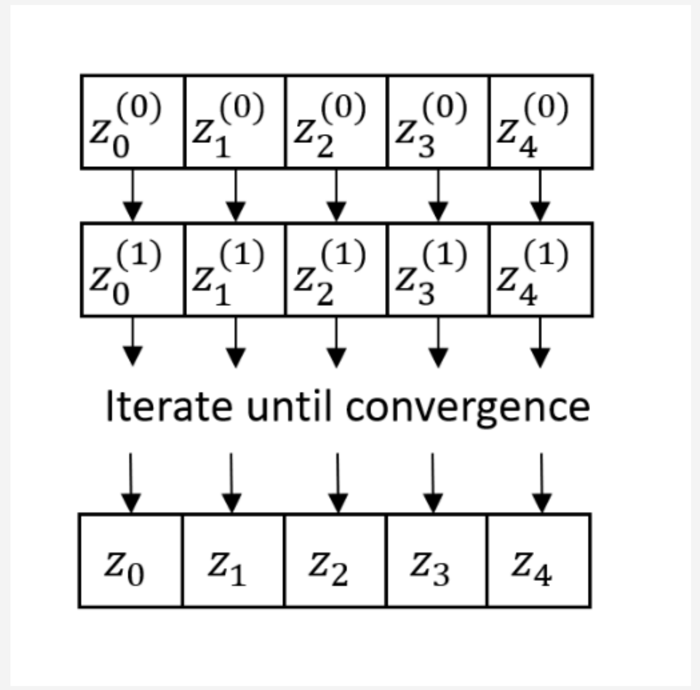
Gonzalez et al, NeurIPS ’24



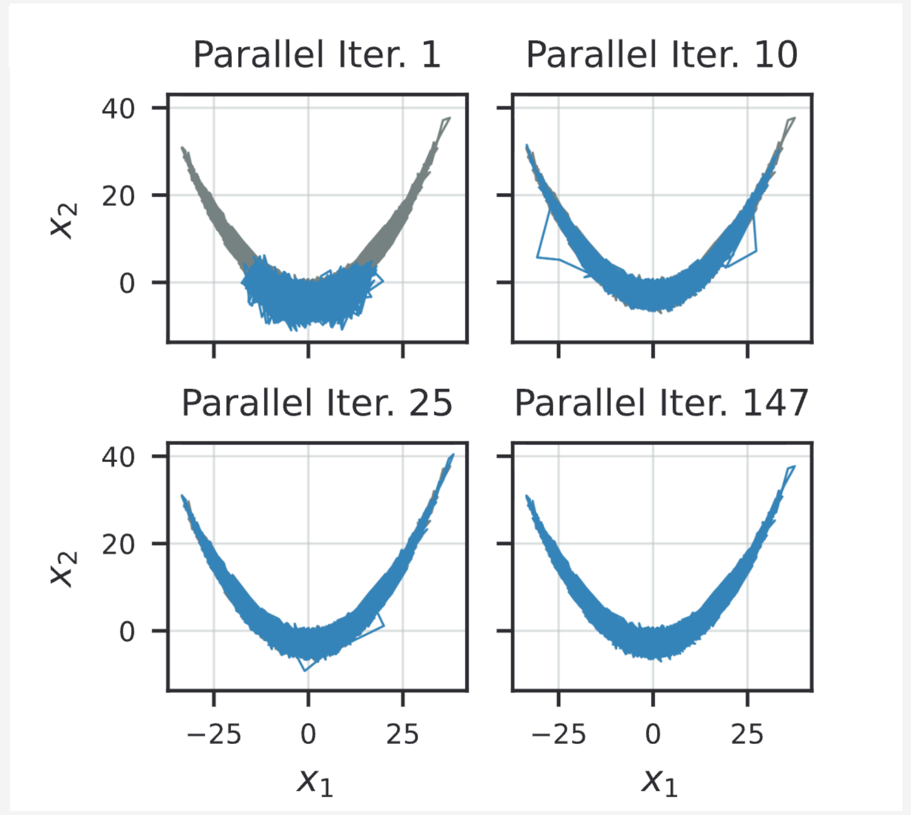
Farsang et al, NeurIPS ’25



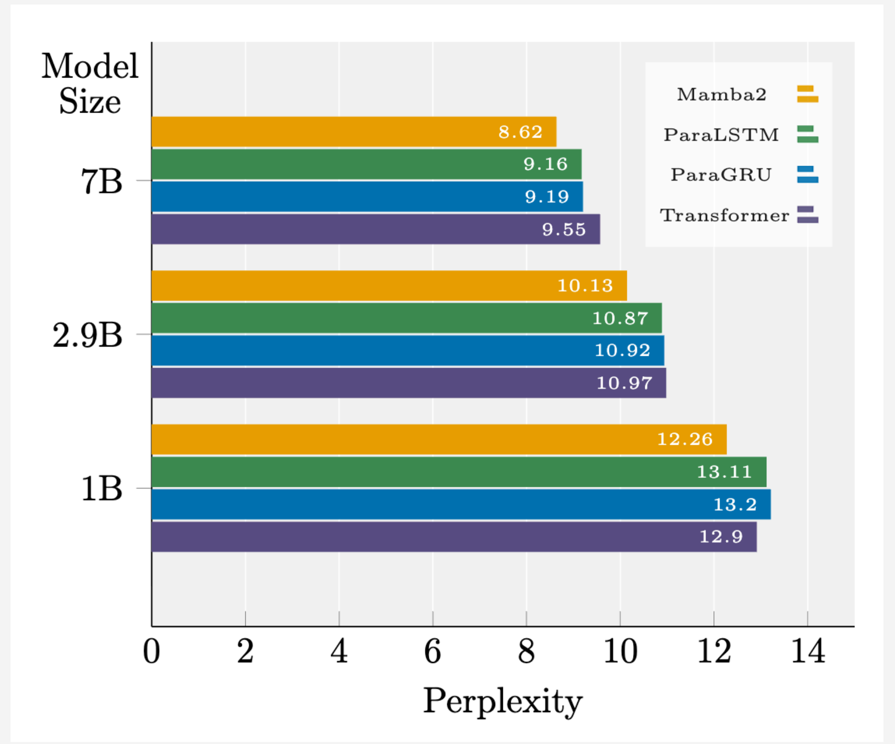
Lim et al, ICLR ’24


$z_0^{(0)}$ $z_1^{(0)}$ $z_2^{(0)}$ $z_3^{(0)}$ $z_4^{(0)}$
$z_0^{(1)}$ $z_1^{(1)}$ $z_2^{(1)}$ $z_3^{(1)}$ $z_4^{(1)}$
Iterate until convergence
z_0 z_1 z_2 z_3 z_4

Zoltowski et al, NeurIPS ’25



Danieli et al, ’25



We can parallelize nSSMs by going from sequential evaluation to a high-dimensional optimization problem

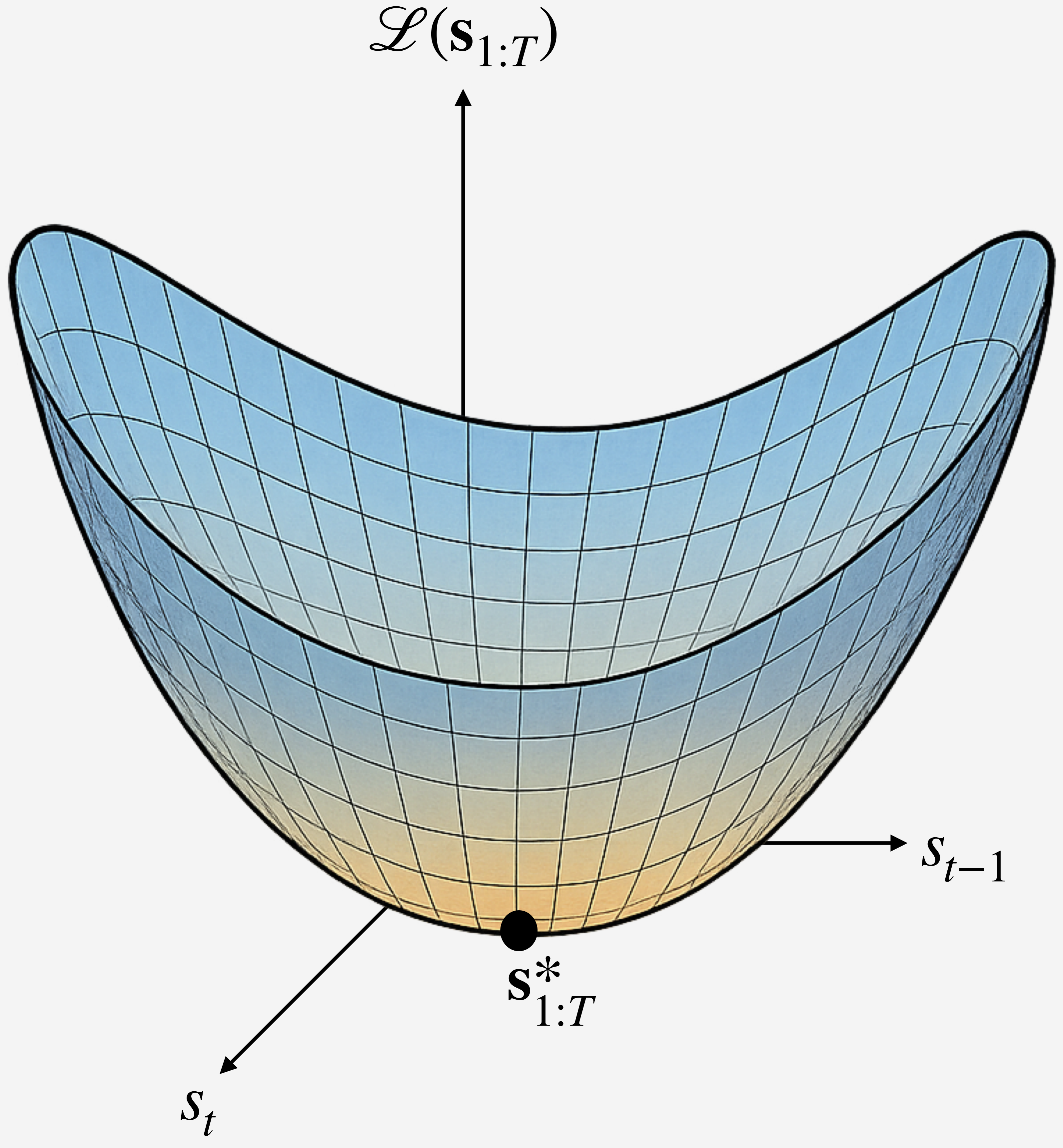
Recall the nSSM satisfies $s_t = f_t(s_{t-1})$, starting from some initial condition $s_0 \in \mathbb{R}^D$.

Treat as our optimization variable *the entire trajectory* $\mathbf{s}_{1:T} \in \mathbb{R}^{TD}$.

Define the one-step prediction error as
$r_t := s_t - f_t(s_{t-1})$.

Define the "merit function" as
$\mathscr{L}(\mathbf{s}_{1:T}) := \|r_1\|^2 + \|r_2\|^2 + \ldots + \|r_T\|^2$.

Then the unique minimizer of $\mathscr{L}$ is the true roll-out from the nSSM.



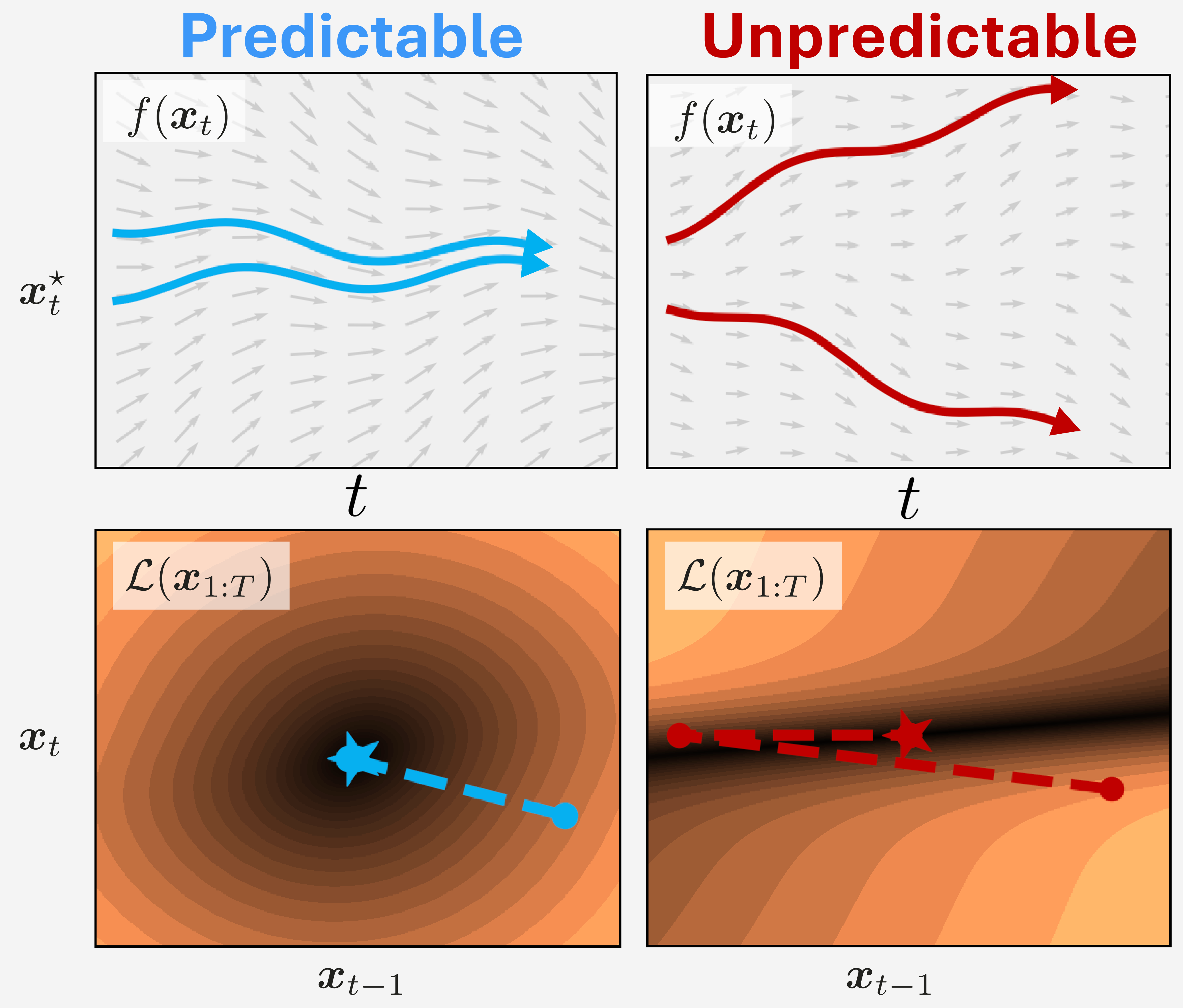
But how do we know when our solver will converge quickly?

We show that "predictability" enables parallelization.

We connect the "predictability" of the state-space *dynamics* to the conditioning of the *optimization problem*.

State-space Dynamics

Optimization



Key quantity

<u>Largest Lyapunov Exponent (LLE, or λ)</u>

- $\lambda < 0$, predictable
- $\lambda > 0$, unpredictable

<u>PL constant μ</u>

- $\mu \gg 0$, well-conditioned, fast
- $\mu \approx 0$, ill-conditioned, slow

Main Theorem

Theorem 2 of our paper (informal):

Let μ be the PL constant, representing the conditioning of the optimization problem. Larger μ is a better-conditioned optimization problem that can be solved more quickly.

Let λ be the Largest Lyapunov Exponent (LLE), representing the predicability of the nSSM. $\lambda < 0$ means predicable dynamics, while $\lambda > 0$ means unpredictable dynamics.

Then, under regularity assumptions,

$$\boxed{\frac{e^{\lambda} - 1}{e^{\lambda T} - 1} \leq \sqrt{\mu} \leq \frac{1}{e^{\lambda(T-1)}} .}$$

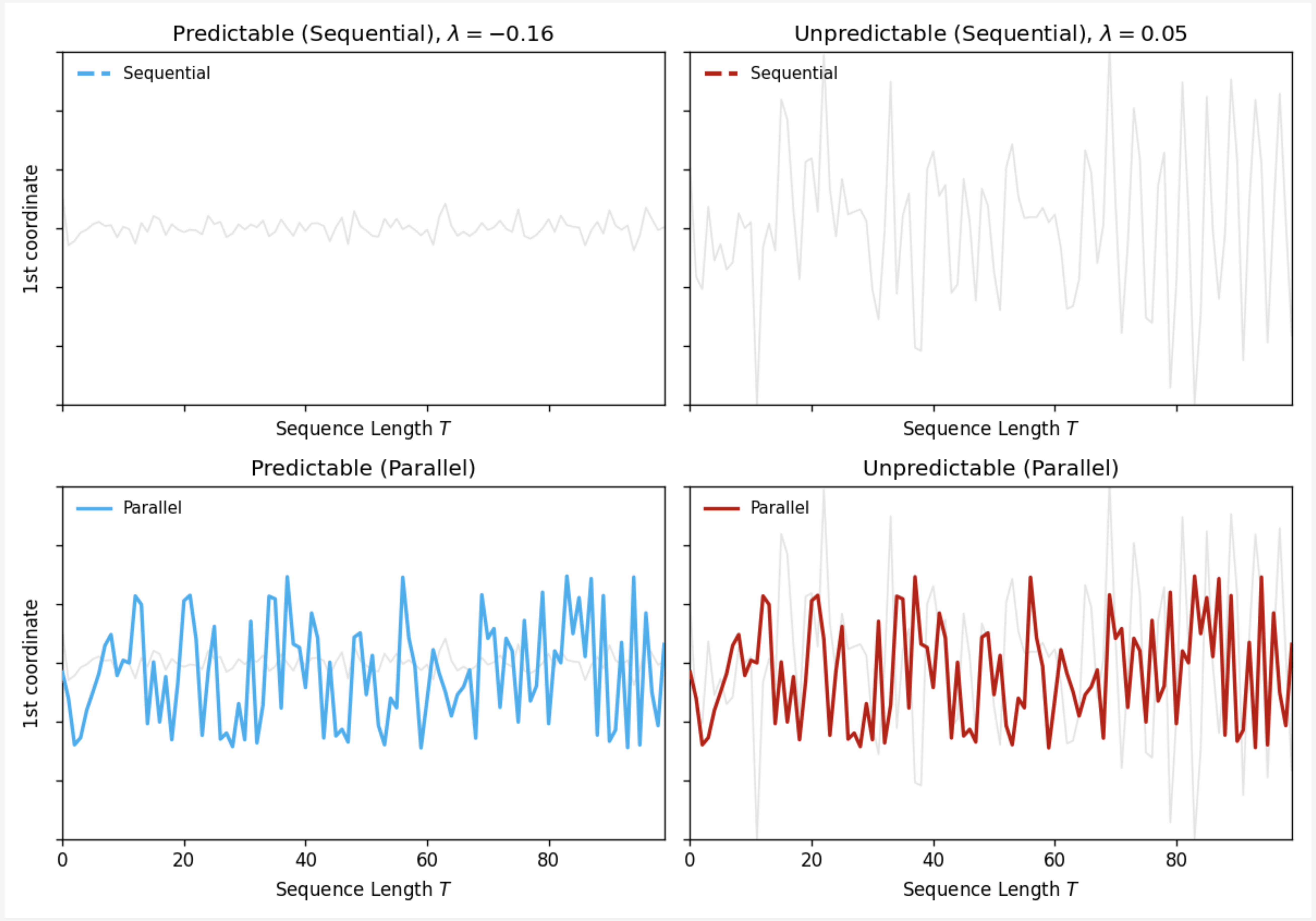
Predictable ($\lambda < 0$)

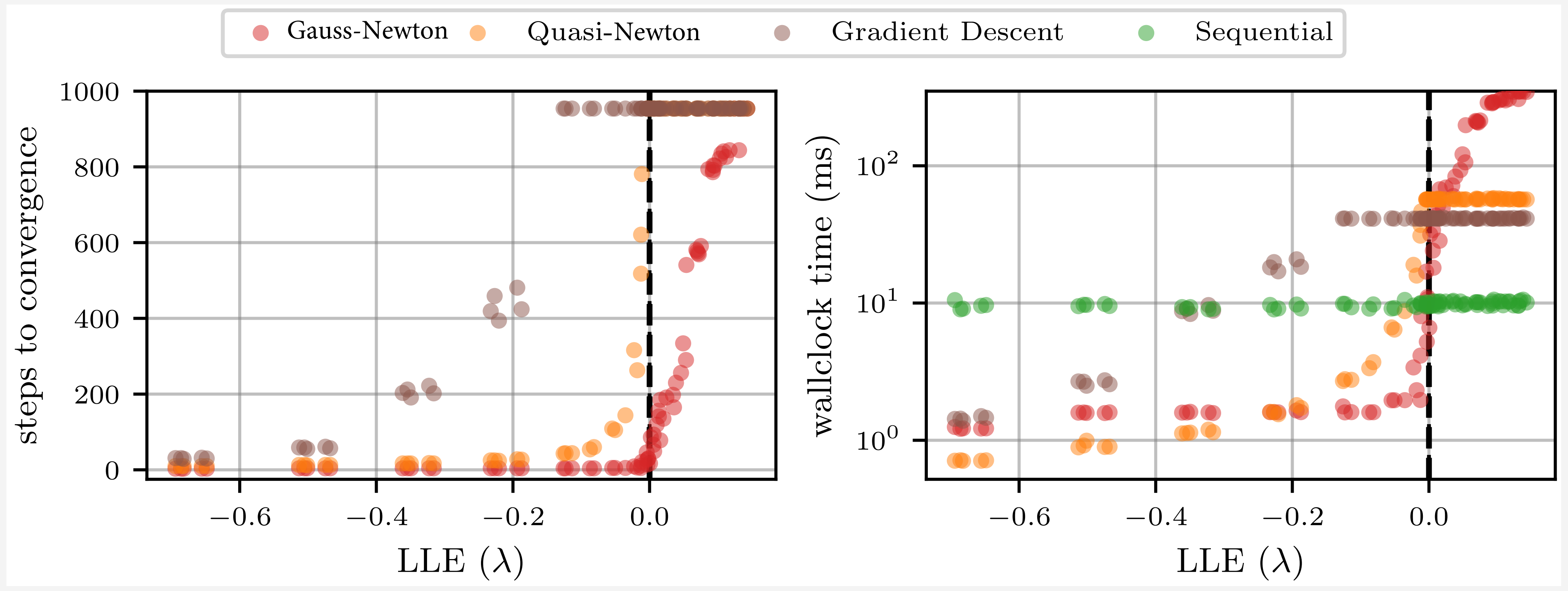
- For long sequences, PL constant μ is bounded away from zero.
- Fast, well-conditioned optimization
- Speed-ups from parallelization

Unpredictable ($\lambda > 0$)

- For long sequences, PL constant $\mu \to 0$ goes to zero exponentially quickly
- Slow, ill-conditioned optimization
- No speed-ups from parallelization :(

Experiments



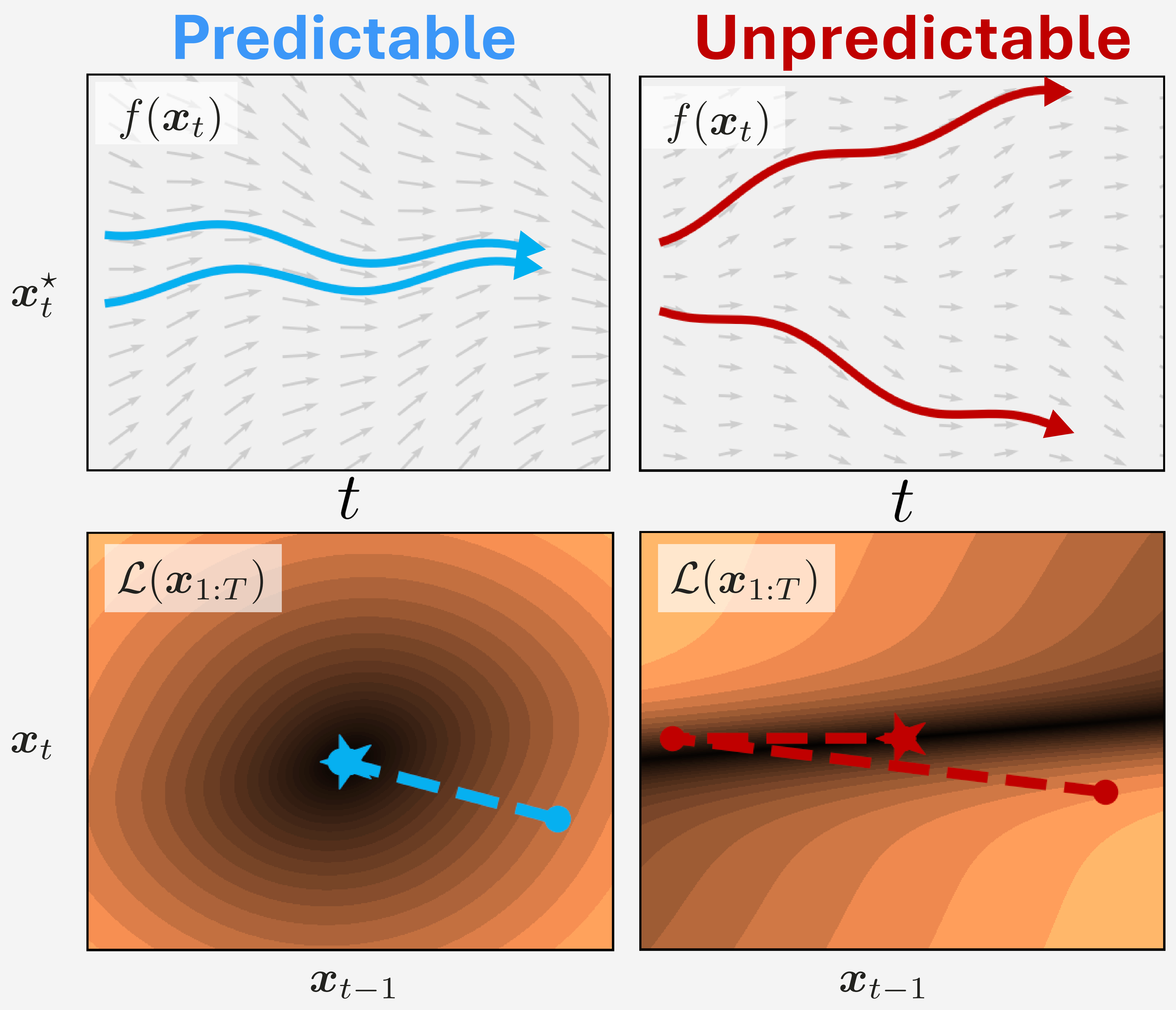


Implications

- Predictable state space models are used in many applications (particularly in robotics and control), and <u>you should parallelize them!</u>

- If you want to build a sequence modeling architecture (RNN) so that it enjoys parallelized training, <u>you should design it to be predictable and therefore parallelizable.</u>

Come by our poster to learn more!



- Paper: https://arxiv.org/abs/2508.16817
- Code: https://github.com/lindermanlab/predictability_enables_parallelization
- Longer Talk: https://www.youtube.com/watch?v=C9AqgW51-B4