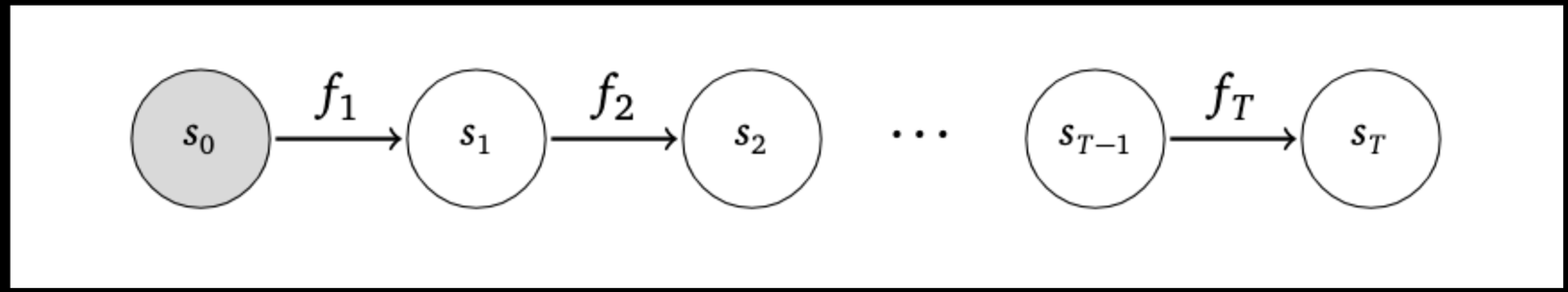
Parallelizing “Inherently Sequential” Processes

Parallel Newton methods for nonlinear state space models

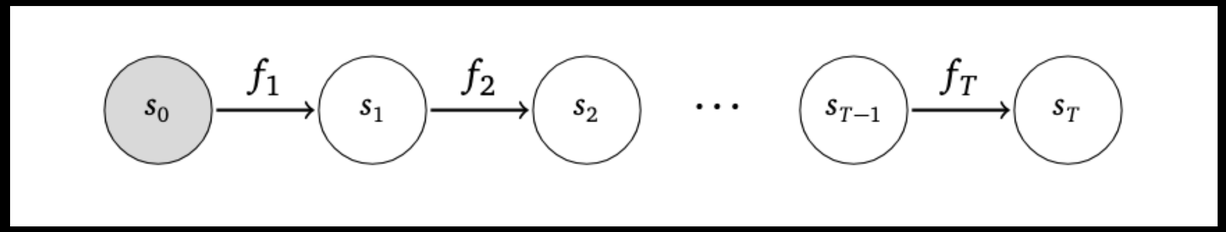
Xavier Gonzalez and Leo Kozachkov, ASAP seminar, 9/30

State Space Model

$$s_t = f_t(s_{t-1})$$



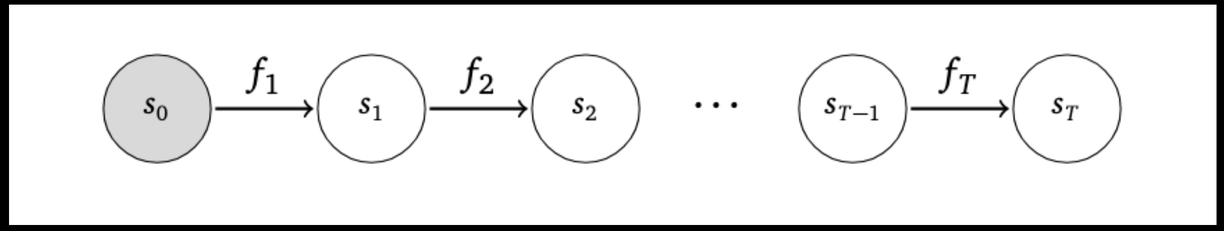
State space models are everywhere



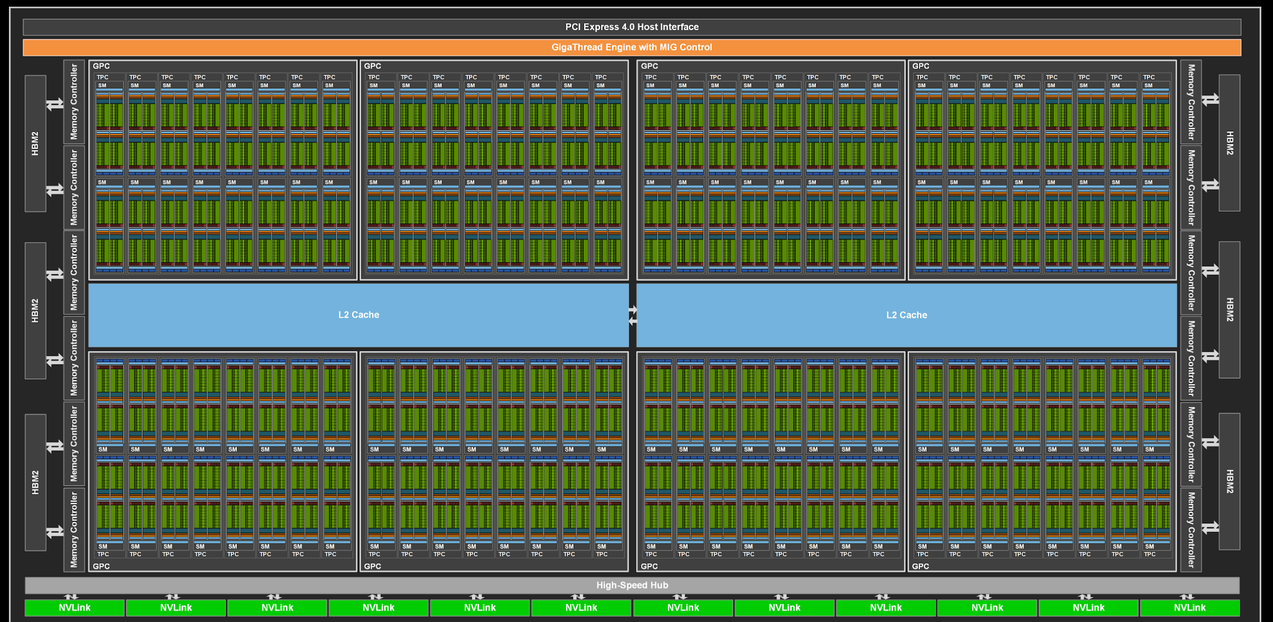
$$s_t = f_t(s_{t-1})$$

- Nonlinear Recurrent Neural Networks (GRU, LSTM and family)
- Markov chain Monte Carlo
- Iterative optimization techniques (like gradient descent)
- The blocks of a transformer model *in depth* (“recurrent depth” as in Schone et al and Geipeng et al ’25)
- Sampling from a diffusion model
- …

Problem: people thought nonlinear SSMs were inherently sequential



Not good for GPU parallelization!



We can parallelize nonlinear SSMs!

DEER: Differential Equation as fixed point itERation

Accepted as a conference paper at ICLR 2024

PARALLELIZING NON-LINEAR SEQUENTIAL MODELS OVER THE SEQUENCE LENGTH

Y. H. Lim[1], **Q. Zhu** [1], **J. Selfridge**[2]*, **M. F. Kasim**[1]†
[1] Machine Discovery Ltd., UK, [2] University of Oxford, UK
{yi.heng, qi.zhu, muhammad}@machine-discovery.com
joshua.selfridge@trinity.ox.ac.uk

ABSTRACT

Sequential models, such as Recurrent Neural Networks and Neural Ordinary Differential Equations, have long suffered from slow training due to their inherent sequential nature. For many years this bottleneck has persisted, as many thought sequential models could not be parallelized. We challenge this long-held belief with our parallel algorithm that accelerates GPU evaluation of sequential models by up to 3 orders of magnitude faster without compromising output accuracy. The algorithm does not need any special structure in the sequential models' architecture, making it applicable to a wide range of architectures. Using our method, training sequential models can be more than 10 times faster than the common sequential method without any meaningful difference in the training results. Leveraging this accelerated training, we discovered the efficacy of the Gated Recurrent Unit in a long time series classification problem with 17k time samples. By overcoming the training bottleneck, our work serves as the first step to unlock the potential of non-linear sequential models for long sequence problems.

1 INTRODUCTION

Parallelization is arguably a main workhorse in driving the rapid progress in deep learning over the past decade. Through specialized hardware accelerators such as GPU and TPU, matrix multiplications which are prevalent in deep learning can be evaluated swiftly, enabling rapid trial-and-error in research. Despite the widespread use of parallelization in deep learning, sequential models such as Recurrent Neural Networks (RNN) (Hochreiter & Schmidhuber, 1997; Cho et al., 2014) and Neural Ordinary Differential Equations (NeuralODE) (Chen et al., 2018; Kidger et al., 2020) have not fully benefited from it due to their inherent need for serial evaluations over sequence lengths.

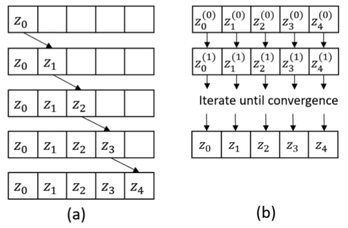


Figure 1: Evaluating sequential models using (a) sequential method and (b) iterative method that is parallelizable.

Serial evaluations have become the bottleneck in training sequential deep learning models. This bottleneck might have diverted research away from se-

DeepPCR: Parallelizing Sequential Operations in Neural Networks

Federico Danieli **Miguel Sarabia** **Xavier Suau** **Pau Rodríguez** **Luca Zappella**
Apple
{f_danieli, miguelsdc, xsuaucuadros, pau.rodriguez, lzappella}@apple.com

Abstract

Parallelization techniques have become ubiquitous for accelerating inference and training of deep neural networks. Despite this, several operations are still performed in a sequential manner. For instance, the forward and backward passes are executed layer-by-layer, and the output of diffusion models is produced by applying a sequence of denoising steps. This sequential approach results in a computational cost proportional to the number of steps involved, presenting a potential bottleneck as the number of steps increases. In this work, we introduce DeepPCR, a novel algorithm which *parallelizes typically sequential operations* in order to speed up inference and training of neural networks. DeepPCR is based on interpreting a sequence of L steps as the solution of a specific system of equations, which we recover using the *Parallel Cyclic Reduction* algorithm. This reduces the complexity of computing the sequential operations from $\mathcal{O}(L)$ to $\mathcal{O}(\log_2 L)$, thus yielding a speedup for large L. To verify the theoretical lower complexity of the algorithm, and to identify regimes for speedup, we test the effectiveness of DeepPCR in parallelizing the forward and backward pass in multi-layer perceptrons, and reach speedups of up to $30\times$ for the forward, and $200\times$ for the backward pass. We additionally showcase the flexibility of DeepPCR by parallelizing training of ResNets with as many as 1024 layers, and generation in diffusion models, enabling up to $7\times$ faster training and $11\times$ faster generation, respectively, when compared to the sequential approach.

1 Introduction

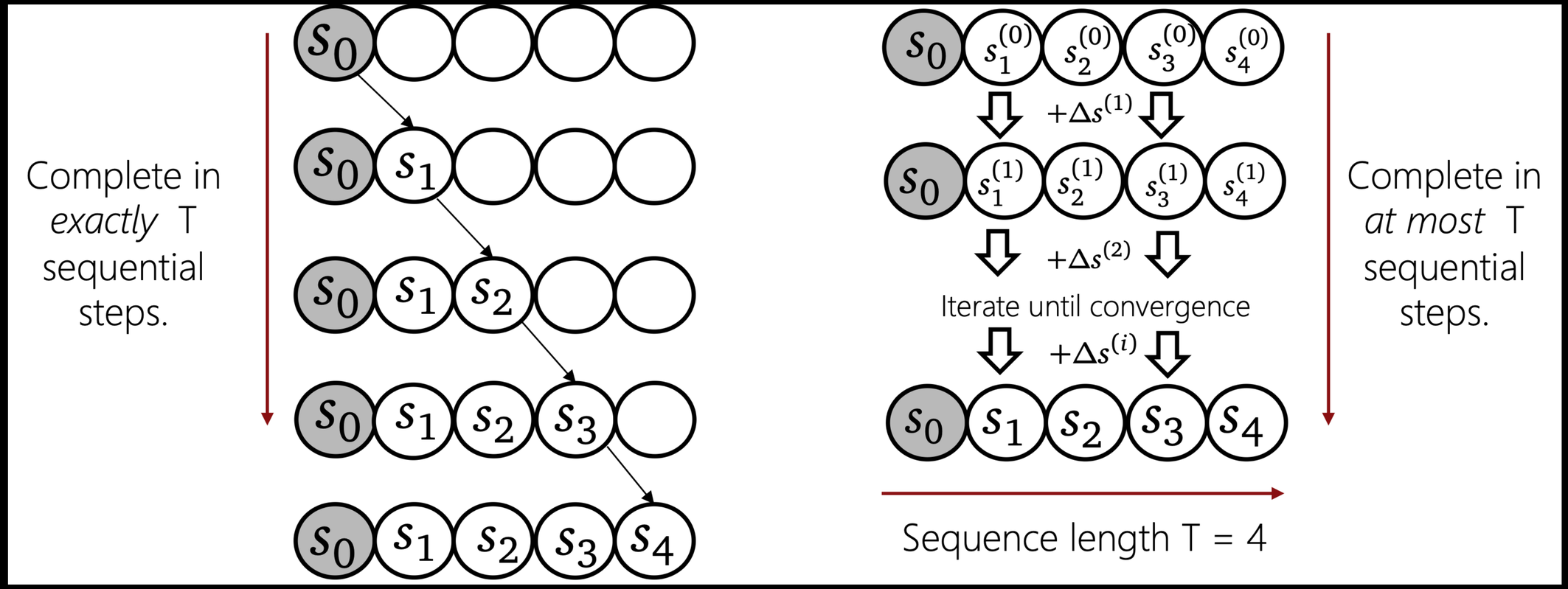
Neural Networks (NNs) have proven very effective at solving complex tasks, such as classification [26, 14], segmentation [5, 30], and image or text generation [26]. Training NNs, however, is a computationally demanding task, often requiring wall-clock times in the order of days, or even weeks [35, 18], before attaining satisfactory results. Even inference in pre-trained models can be slow, particularly when complex architectures are involved [4]. To reduce training times, a great effort has been invested into speeding up inference, whether by developing dedicated software and hardware [7, 22, 23], or by investigating algorithmic techniques such as (early) pruning [28, 40, 20, 27, 43, 9].

Another possibility for reducing wall-clock time, and the one we focus on in this work, consists in parallelizing computations that would otherwise be performed sequentially. The most intuitive approach to parallelization involves identifying sets of operations which are (almost entirely) independent, and executing them concurrently. Two paradigms that follow this principle are *data-parallelization*, where multiple datapoints are processed simultaneously in batches; and *model-parallelization*, where the model is split among multiple computational units, which perform their evaluations in parallel [1].

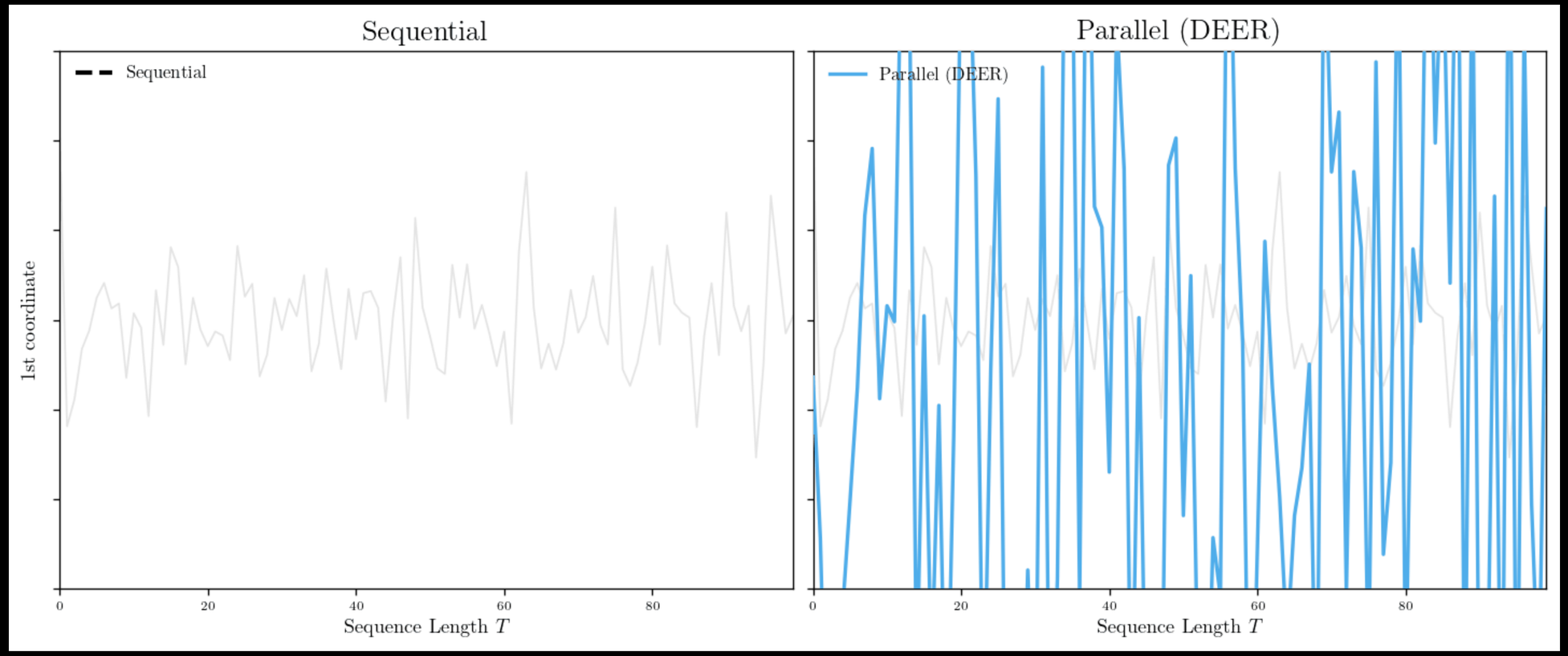
Still, certain operations which are key for training and inference in NNs have a sequential structure. The forward and backward pass of a NN are examples of such operations, where activations

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

Sequential vs. Parallel (Iterative) Evaluation



A simple demo: parallelizing a vanilla RNN

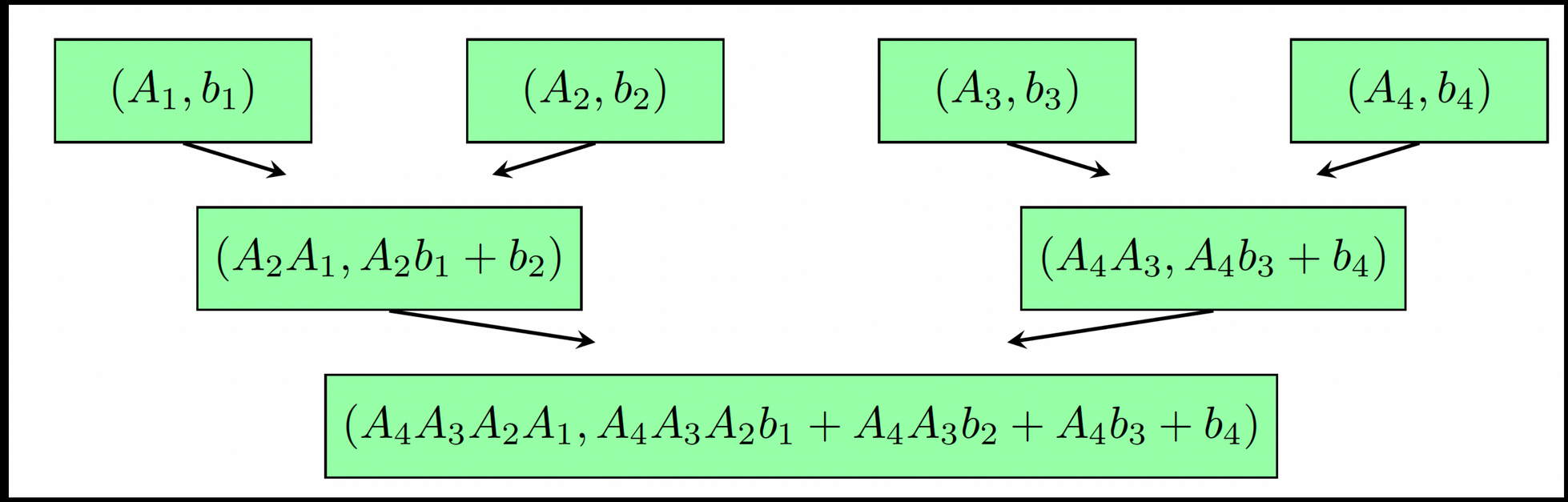


How is this working?

The secret sauce: parallel associative scan

Consider a *linear* SSM

$$s_t = A_t s_{t-1} + b_t$$



With sufficient parallel processors, we can evaluate a linear SSM in $\mathcal{O}(\log T)$ time. (Blelloch, '90)

What about an arbitrary SSM?

Goal: parallelize *nonlinear* SSMs like nonlinear RNNs and MCMC

Problem: How do we get the pscan to work for arbitrary nonlinear dynamics f

Crazy idea: What if we just alternatively:

1. Linearize the nonlinear dynamics f around current guesses for the states $\mathbf{s}_{1:T}^{(i)}$
2. Evaluate the linearized dynamics in parallel to get new guesses $\mathbf{s}_{1:T}^{(i+1)}$

This crazy idea is DEER!

DEER is equivalent to Gauss-Newton optimization

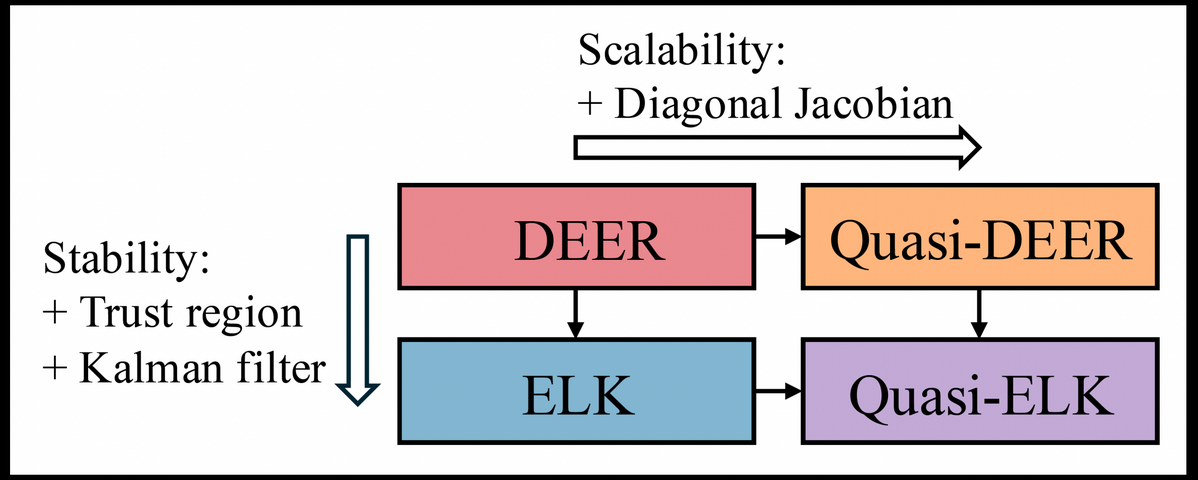
DEER is so cool!
But of course it has limitations.

Limitation #1: Scalability and Stability

Solution:

Towards Scalable and Stable Parallelization of Nonlinear RNNS

Xavier Gonzalez Andrew Warrington Jimmy Smith Scott Linderman



Limitation #2: Provable Guidance When to use DEER?

Solution:

Predictability Enables Parallelization of Nonlinear SSMs

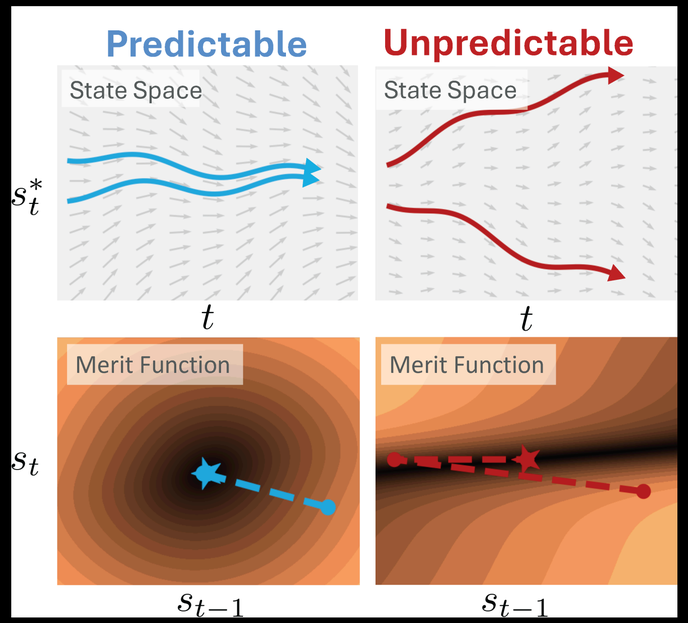
Xavier Gonzalez*

Leo Kozachkov*

David Zoltowski

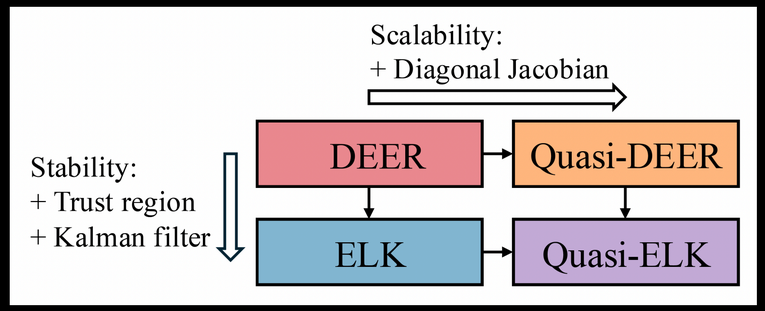
Kenneth Clarkson

Scott Linderman

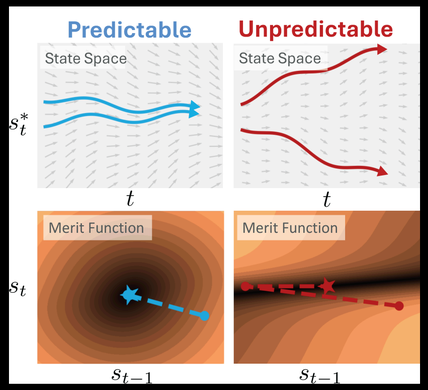


Outline

1. Introduction to DEER (Leo)
   - DEER is the Gauss-Newton method
   - Parallelization comes from the parallel scan (Blelloch ’90)
2. Scaling and Stabilizing DEER (Xavier)



3. When to use DEER (Leo)



1. Introduction to DEER

How to Parallelize Nonlinear SSMs

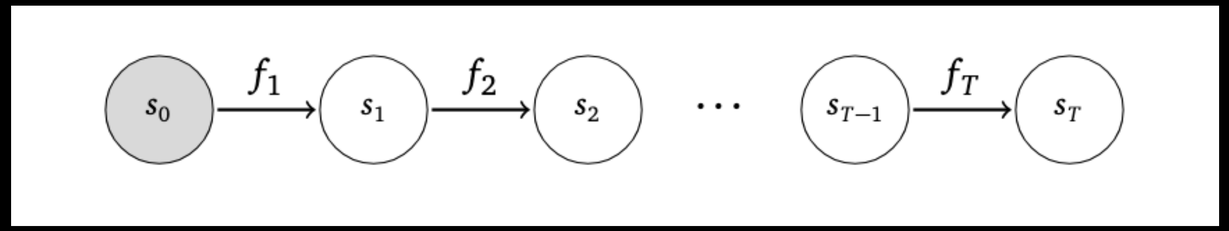
Convert The Forward Pass to an Optimization Problem

- Start with an initial state $s_0 \in \mathbb{R}^D$
- Write down an loss function that has the “true” trajectory as a solution, i.e.,

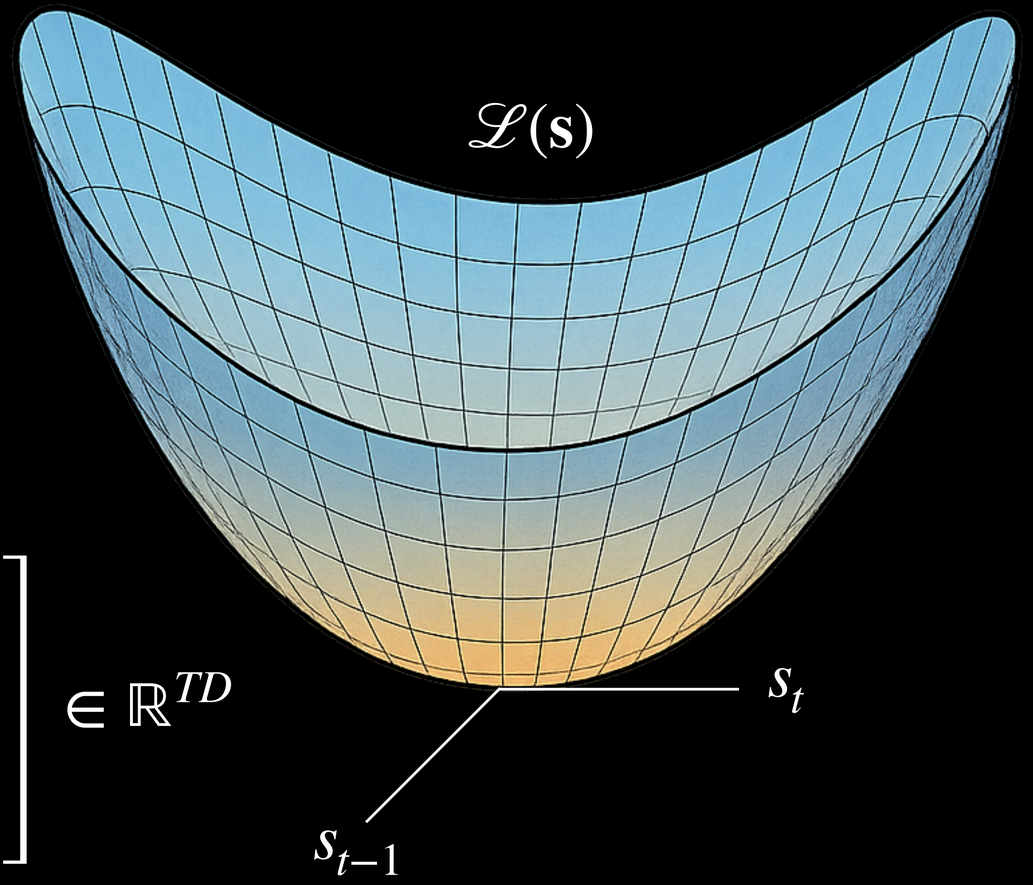
$$\mathscr{L} = 0 \iff s_t = f_t(s_{t-1})$$

- Find a way to very quickly optimize this loss function, so that evaluation is faster than sequential.

Sequential Evaluation of nSSM



Optimization-Based Evaluation of nSSM



$$\mathbf{s} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_T \end{bmatrix} \in \mathbb{R}^{TD}$$

How to Parallelize Nonlinear SSMs

Converting The Forward Pass to an Optimization Problem

$$s_t = \tanh(Ws_{t-1} + u_t)$$

We want to find a sequence of states that satisfy $s_t^* = f_t(s_{t-1}^*) \qquad s_0$

Define the deviation from desired by a residual: $r_t = s_t - f_t(s_{t-1})$

Find an optimizer that minimizes the sum-of-squares loss: $\mathscr{L} = \|r_1\|^2 + \|r_2\|^2 + \ldots + \|r_T\|^2$

Note that when the loss is zero, the original nonlinear SSM equation is verified

Fast Optimization-Based Evaluation

Gauss-Newton Algorithm

- Treat the whole trajectory as an optimization variable $\mathbf{s} \in \mathbb{R}^{TD}$
- Gauss-Newton step $\Delta \mathbf{s} = -\mathbf{J}^{-1}(\mathbf{s})\,\mathbf{r}(\mathbf{s})$
- Naively, inverting this matrix is very hard. But using the special structure of the problem, parallel scans come to the rescue!

$$\Delta s_t = J_t(s_{t-1})\,\Delta\, s_{t-1} - r_t \qquad \mathcal{O}(\log(T))$$

- So each step of GN can be parallelized—but what controls the number of steps? More on this later!

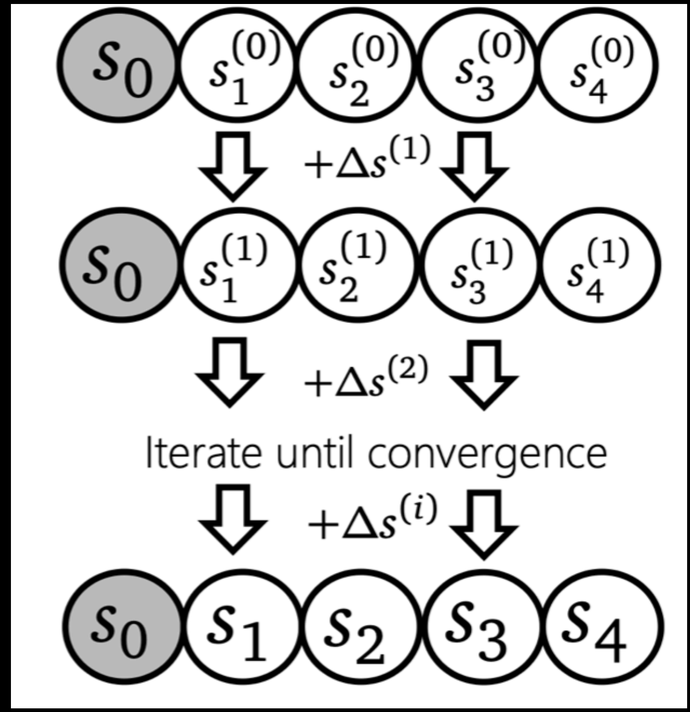
Each DEER step is a linear SSM!

$$s_t^{(i+1)} = f_t(s_{t-1}^{(i)}) + J_t^{(i)} \left(s_{t-1}^{(i+1)} - s_{t-1}^{(i)} \right)$$

procedure DEER(f, s_0, initial guess $\mathbf{s}_{1:T}^{(0)}$, tolerance ϵ)
  for $i = 0, 1, \ldots, T$ **do**
    $J_{1:T} \leftarrow \textsc{LinearizeDynamics}(f, \mathbf{s}_{0:T}^{(i)})$
    $\mathbf{s}_{1:T}^{(i+1)} \leftarrow \textsc{ParallelScan}(J_{1:T}, f, \mathbf{s}_{0:T}^{(i)})$
    if $\textsc{ComputeError}(f, \mathbf{s}_{0:T}^{(i+1)}) < \epsilon$ **then**
      break
  end for
  return $\mathbf{s}_{1:T}^{(i+1)}$
end procedure

DEER recap

1. Make an initial guess $\mathbf{s}_{1:T}^{(0)}$ for the states
2. Linearize the dynamics: evaluate $J_t^{(i)} := \frac{df_t}{ds_{t-1}}(s_{t-1}^{(i)})$ for all t
3. Evaluate the resulting linear SSM using a parallel associative scan to get the Gauss-Newton step $\Delta\mathbf{s}^{(i)}$

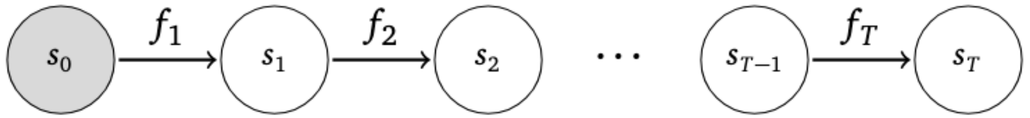


2. How to improve DEER? Scalability and Stability

“Toward Stable and Scalable Parallelization of Nonlinear RNNs.”
Gonzalez, Warrington, Smith, Linderman.
NeurIPS ’24. https://arxiv.org/abs/2407.19115

Code: https://github.com/lindermanlab/elk

DEER limitations

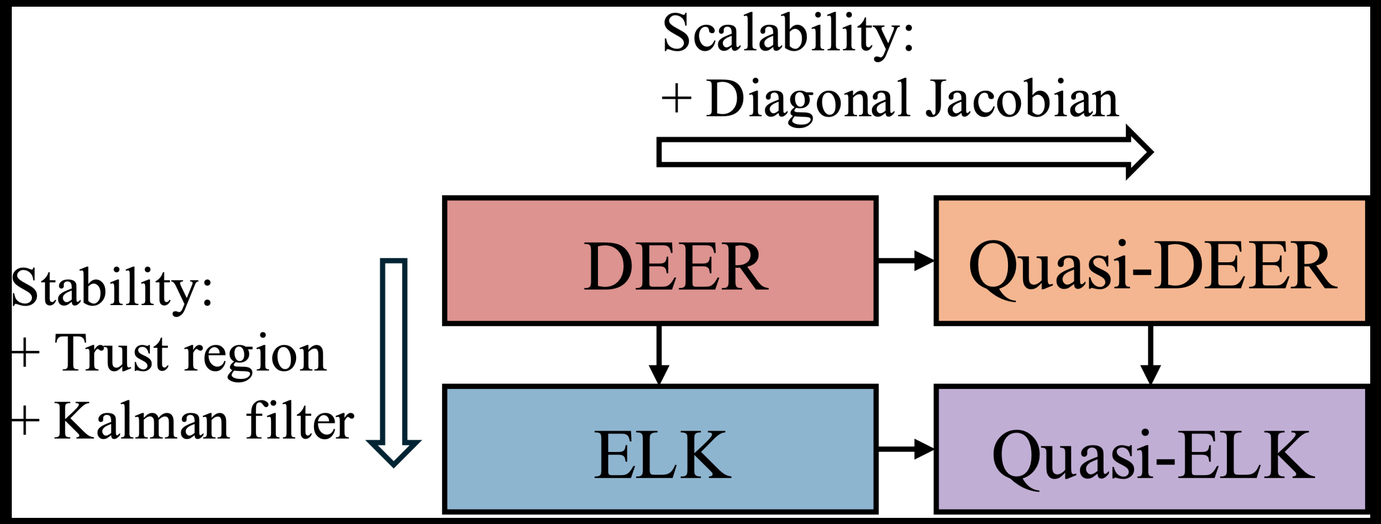


$$s_t^{(i+1)} = f_t(s_{t-1}^{(i)}) + \underbrace{J_t^{(i)}}_{\in \mathbb{R}^{D \times D}} \left(s_{t-1}^{(i+1)} - s_{t-1}^{(i)} \right)$$

Problems

- **Compute**: $\mathcal{O}(TD^3)$ work
- **Memory**: $\mathcal{O}(TD^2)$ memory
- **Stability**: Imagine if many J_t have big eigenvalues!

The ungulates



Method	Desiderata			
	Parallel	Work	Memory	Stability
Sequential	No	$\mathcal{O}(TD^2)$	$\mathcal{O}(D)$	Very high
DEER	Yes	$\mathcal{O}(TD^3)$	$\mathcal{O}(TD^2)$	Low
Quasi-DEER	Yes	$\mathcal{O}(TD)$	$\mathcal{O}(TD)$	Low
ELK	Yes	$\mathcal{O}(TD^3)$	$\mathcal{O}(TD^2)$	High
Quasi-ELK	Yes	$\mathcal{O}(TD)$	$\mathcal{O}(TD)$	Moderate

Scalability: quasi-Newton methods

- DEER:

$$s_t^{(i+1)} = f_t(s_{t-1}^{(i)}) + J_t \left(s_{t-1}^{(i+1)} - s_{t-1}^{(i)} \right)$$

- quasi-DEER:

$$s_t^{(i+1)} = f_t(s_{t-1}^{(i)}) + \mathrm{diag}(J_t) \left(s_{t-1}^{(i+1)} - s_{t-1}^{(i)} \right)$$

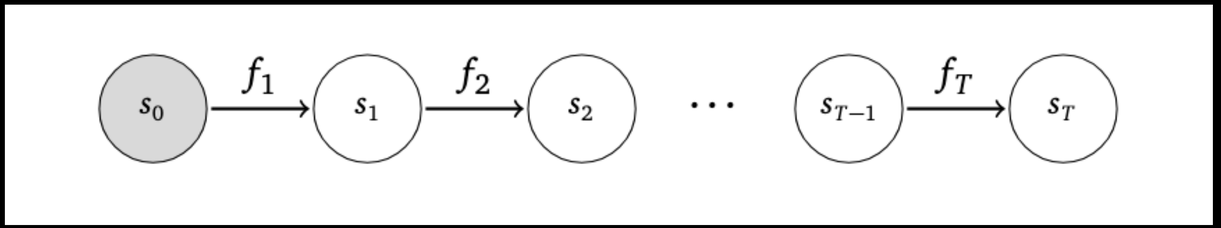
Global Convergence of DEER and quasi-DEER

Proposition 1 of Gonzalez et al, ’24

Both DEER and quasi-DEER converge to the true solution $\mathbf{s}^*_{1:T}$ in at most T Newton iterations, for <u>any</u> initial guess $\mathbf{s}^{(0)}_{1:T}$

- In general problems, Gauss-Newton can fail to converge
- This result is specific to using GN on a loss function coming from one-step prediction errors (i.e. $r_t = s_t - f(s_{t-1})$

Proof of Proposition 1 by Induction



$$s_{t+1}^{(i+1)} = f_{t+1}(s_t^{(i)}) + J_{t+1} \underbrace{\left(s_t^{(i+1)} - s_t^{(i)}\right)}_{\Delta s_t}$$

- Assume that at iteration (i), for all $t \leq i$, we have $s_t^{(i)} = s_t^*$.
- Then $\Delta s_t = 0$ for all $t \leq i$.
- So, $s_{i+1}^{(i+1)} = f_{i+1}(s_i^{(i)}) = f_{i+1}(s_i^*) = s_{i+1}^*$
- By induction, $\mathbf{s}_{1:T}^{(T)} = \mathbf{s}_{1:T}^*$

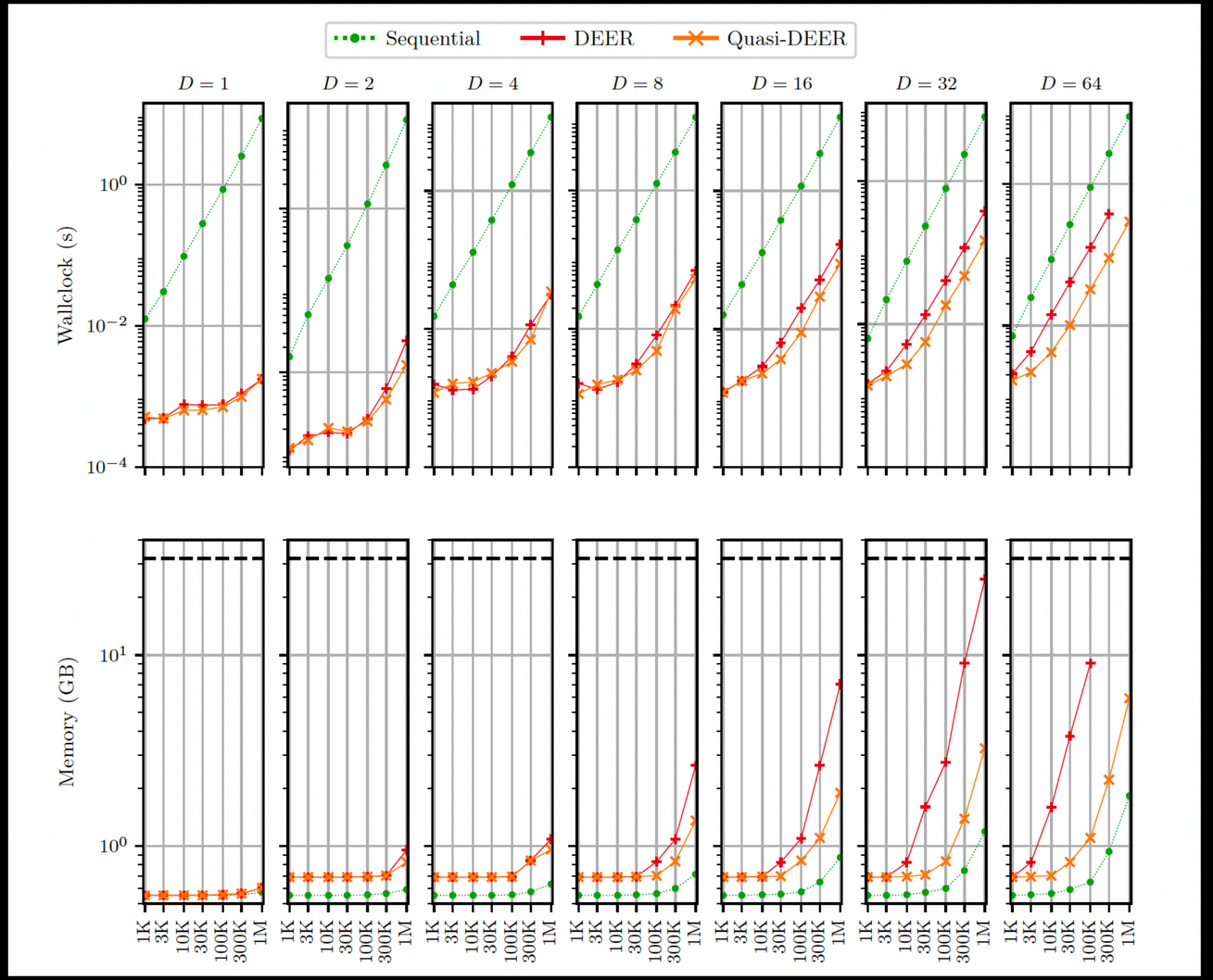
Global Convergence has powerful consequences

Consequences of the Proof of Proposition 1

- We can work with *arbitrary* approximations to J_t.
  - Quasi-DEER uses $\mathrm{diag}(J_t)$ for simplicity.
  - We can even work with *nondifferentiable* f via surrogate gradient tricks
  - For example: accept-reject steps in MCMC! As in “Parallelizing MCMC over the sequence length,” Zoltowski*, Wu*, **Gonzalez, Kozachkov**, Linderman.
  - Could be useful in other discrete settings (autoregressive token generation)
- Because the proof is by induction, we can reset later steps in the trace to arbitrary values and still get global convergence (useful for numerical overflows)

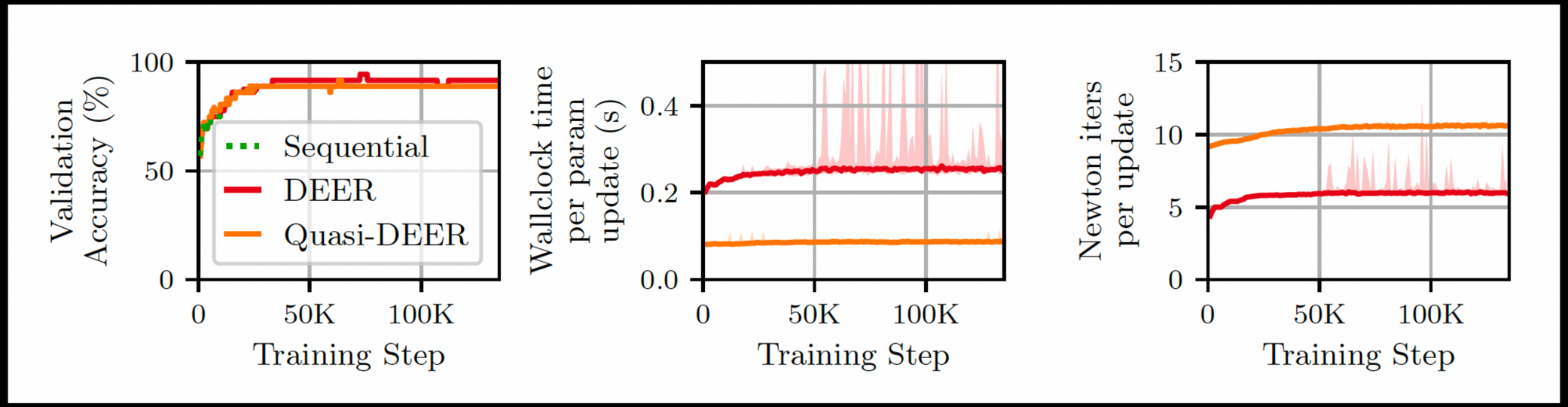
Benchmarking Wallclock Time and Memory

Forward pass (inference)



Quasi-DEER for Training

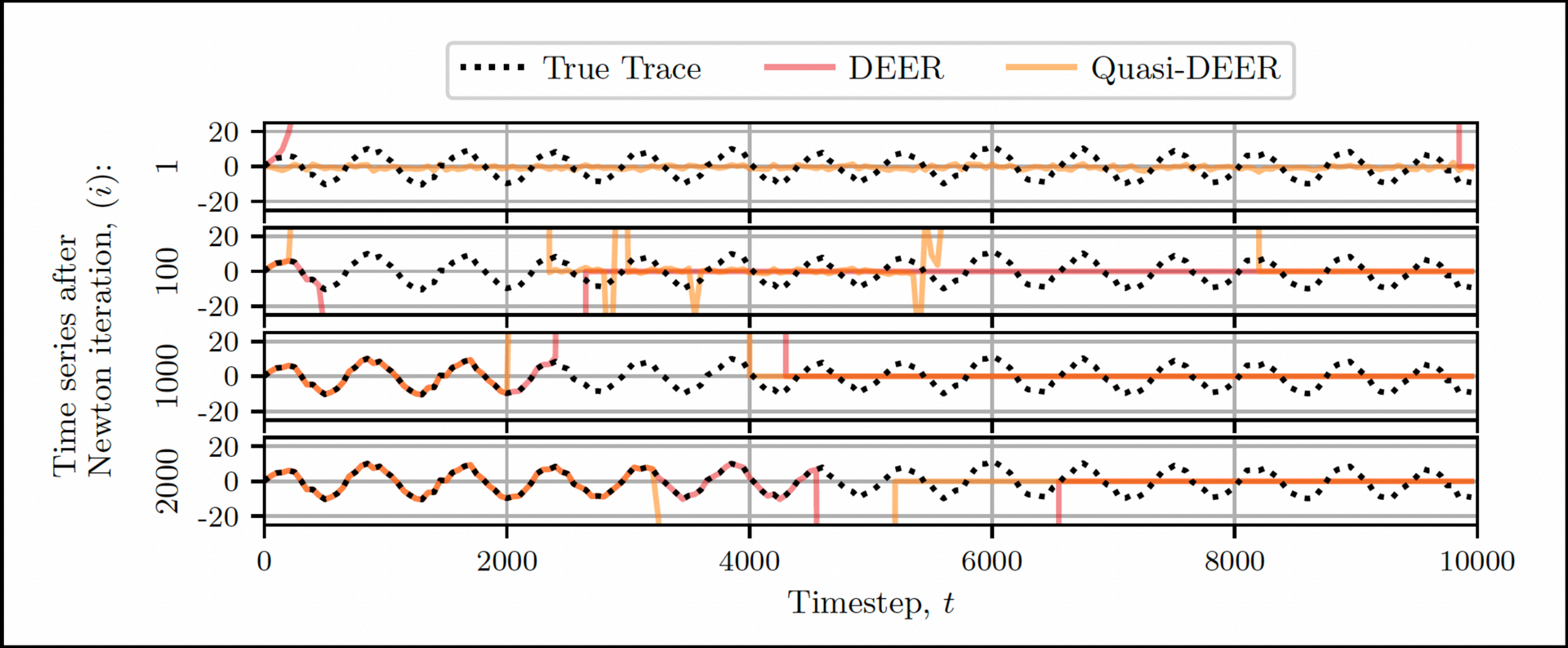
"Eigenworms" task, $T = 17{,}984$, longest task from UEA MTSC dataset



Architecture: 5 layer GRU, hidden state size of $D = 32$

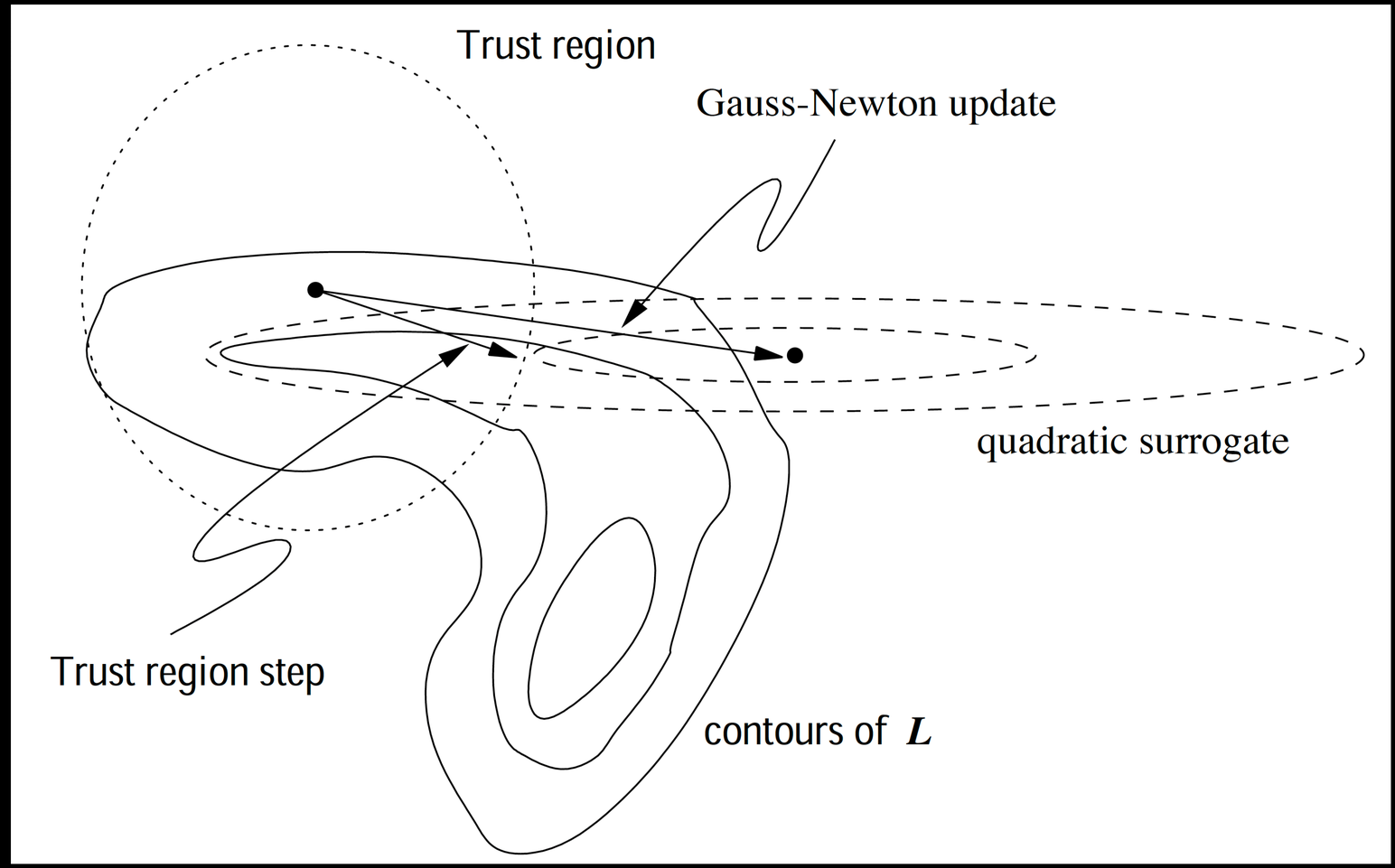
Quasi-DEER and DEER can be unstable

Training a GRU to generate a sine wave



Trust region optimization (Levenberg-Marquardt)

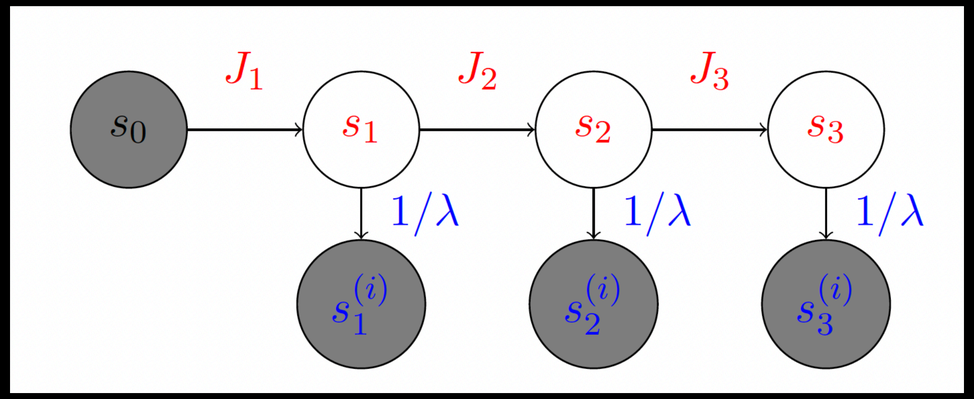
Chapter 4 of *Numerical Optimization* by Nocedal and Wright



ELK

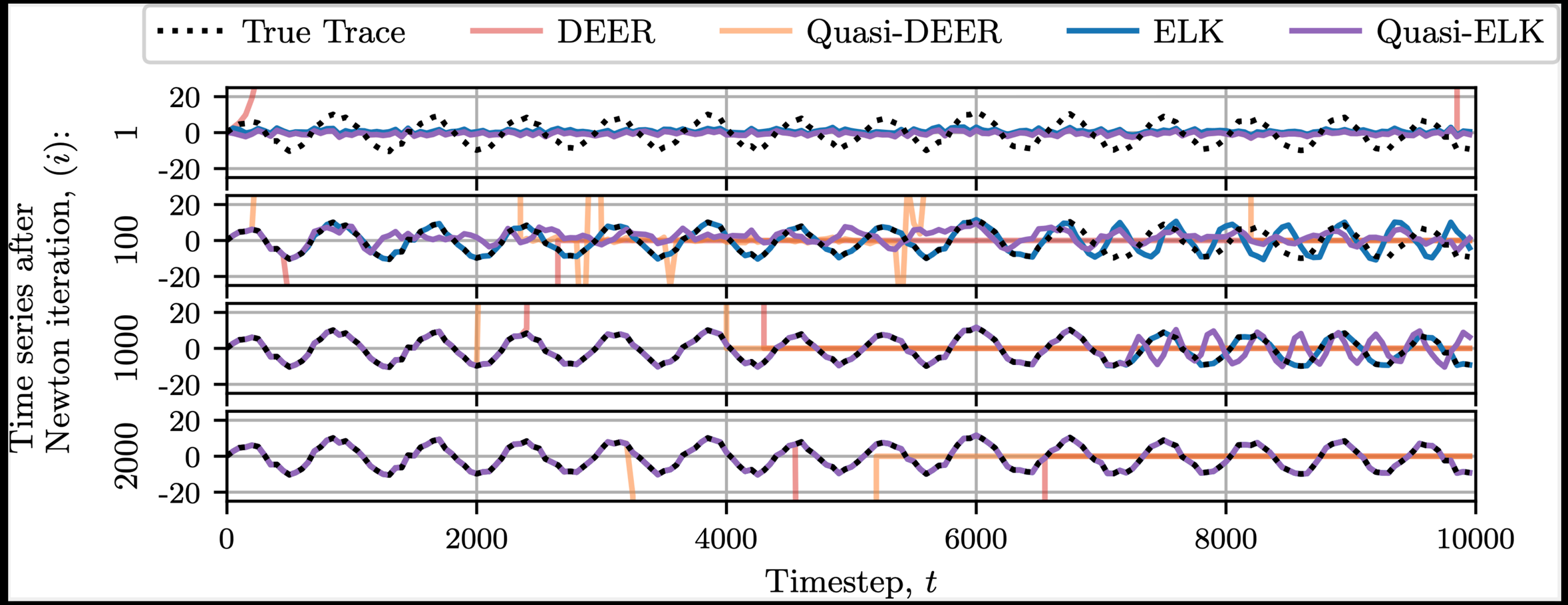
Evaluating Levenberg-Marquardt with Kalman

Trust region approach (Levenberg-Marquardt) can be reformulated as the solution to a Kalman smoother



Kalman smoothers can be evaluated in parallel! (Sarkka and Garcia-Fernandez ’20)

ELK Results


True Trace
DEER
Quasi-DEER
ELK
Quasi-ELK
Time series after Newton iteration, (i):
1
100
1000
2000
20
0
-20
0
2000
4000
6000
8000
10000
Timestep, t

Ungulates struggle to parallelize oscillations

Algorithm	Average time per Newton iteration $\pm$ std (ms)	Number of Newton iterations to convergence	Total time to convergence (ms)
Sequential Evaluation			
Sequential	NA	NA	96.2
Parallelized Methods			
DEER	0.282 ± 0.0005	4449	1255
quasi-DEER	0.087 ± 0.0002	7383	642
ELK	3.600 ± 0.067	172	619
quasi-ELK	0.141 ± 0.0004	1566	221

Parallelizing at the edge of stability

Do I just need a

- Bigger machine?
- Longer sequence?
- Smarter approach?

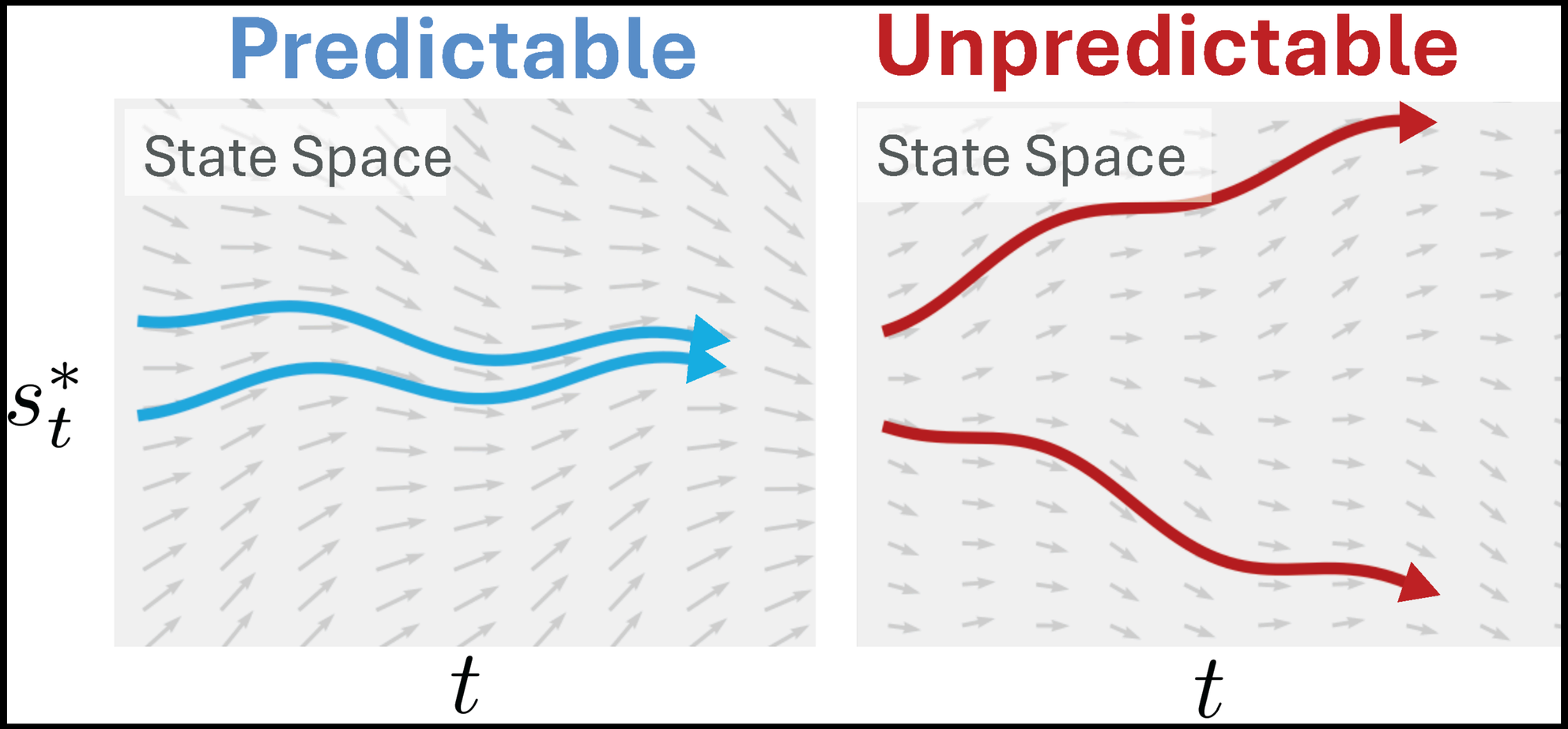
Or are there limits on what sequences I can *efficiently* parallelize?

3. When to use DEER?
Parallelize stable systems.
Do NOT parallelize unstable systems.

“Predictability Enables Parallelization of Nonlinear State Space Models.”
Gonzalez*, Kozachkov*, Zoltowski, Clarkson, Linderman ’25
https://arxiv.org/abs/2508.16817

Parallelize Stable Systems

Sequentially Evaluate Unstable Ones



Definition: Predictability and Unpredictability

Consider an SSM $s_t = f_t(s_{t-1})$ with derivatives $J_t := \dfrac{\partial f_t}{\partial s_{t-1}}$.

The Largest Lyapunov Exponent (LLE or λ) of this SSM is

$$\text{LLE} := \lim_{T\to\infty} \frac{1}{T} \log\left(\|J_T J_{T-1} \dots J_1\|\right).$$

The displacement between trajectories scales as

$$\|s_t - s_t'\| \sim e^{\lambda t} \|s_0 - s_0'\|.$$

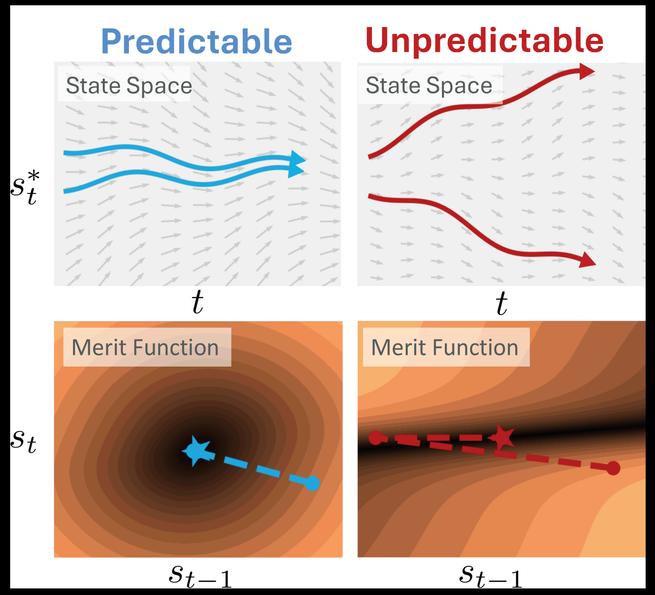
- $\lambda < 0$: predictable
- $\lambda > 0$: unpredictable

SSM Stability Determines Loss Landscape

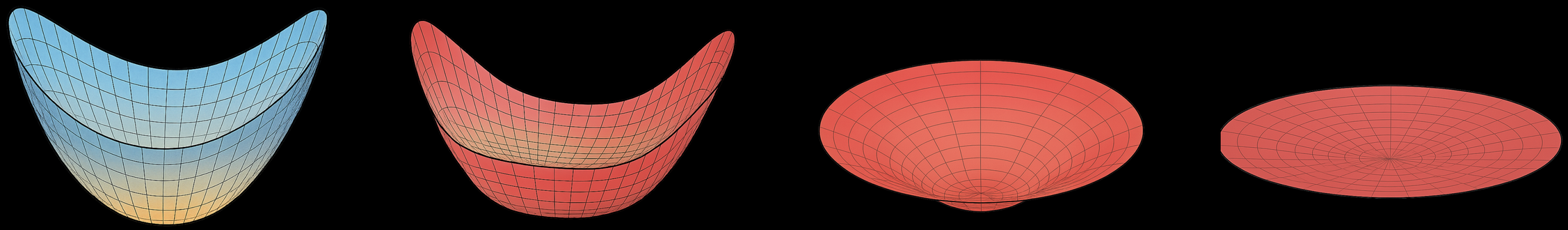
Remember the loss function we wanted to minimize?

$$\mathscr{L}(\mathbf{s}_{1:T}) = \frac{1}{2}\|\mathbf{r}(\mathbf{s}_{1:T})\|^2.$$

The stability of an SSM (measured by its LLE λ) controls the flatness of its corresponding loss function



Geometry of Instability



Easy to Optimize → Hard to Optimize

How do we quantify flatness?

Polyak-Łojasiewicz or PL constant

The PL condition requires that the gradient never get too small relative to the loss:

$$\frac{1}{2}\|\nabla \mathscr{L}\|^2 \geq \mu \mathscr{L} \, .$$

Proposition (Nesterov and Polyak, ’06)

The PL constant μ of $\mathscr{L}$ is given by

$$\mu = \inf_{\mathbf{s}} \sigma_{\min}^2 \mathbf{J}(\mathbf{s})$$

Proposition (Karimi et al, ’20)

Since $\mathscr{L}$ is PL, gradient descent converges at linear rate.
The precise rate is controlled by the value of the PL constant μ.

Predictability Enables Parallelization

Theorem (Gonzalez*, Kozachkov*, et al, ’25)

(Informal). Under regularity assumptions,

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

Predictable/Stable Systems ($\lambda < 0$)

- As $T \to \infty$, lower bound of $\sqrt{\mu}$ approaches $1 - e^{\lambda}$
- So $\sqrt{\mu}$ is bounded away from 0, rate of convergence doesn’t degrade with T
- You should parallelize!

Unpredictable/Unstable Systems ($\lambda > 0$)

- $\lim_{T\to\infty} \mu = 0$
- Loss conditioning / rate of convergence degrades exponentially quickly with T
- You should not parallelize

Predictability Enables Parallelization

Theorem (Gonzalez*, Kozachkov*, et al, ’25)

(Informal). Under regularity assumptions,

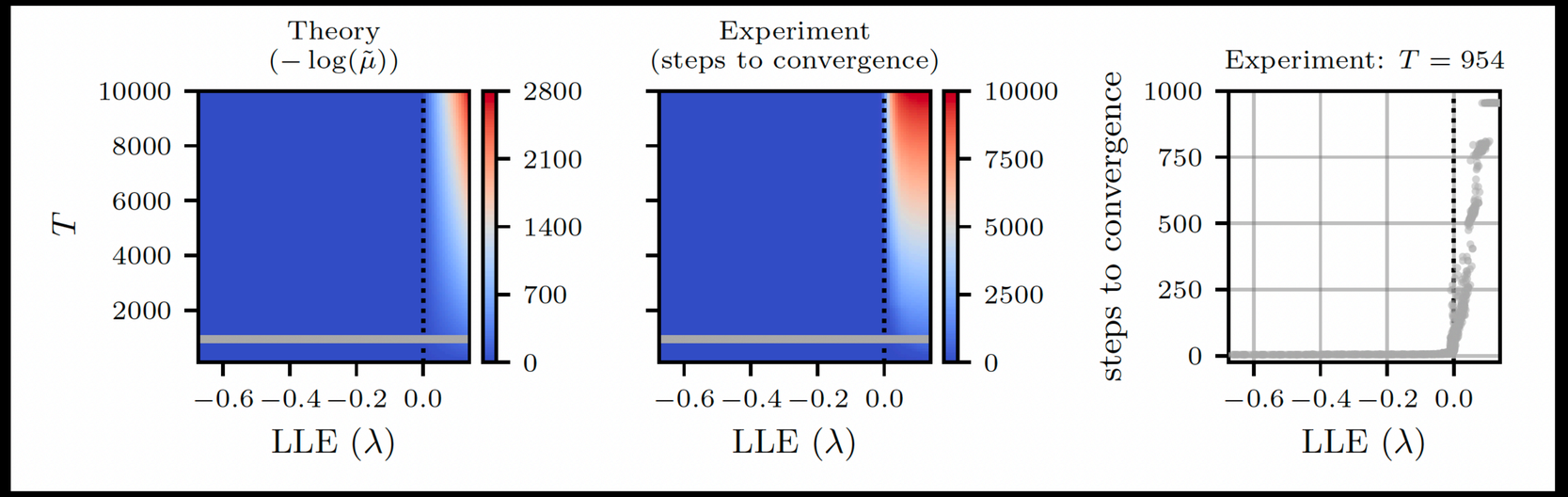
$$\frac{e^{\lambda}-1}{e^{\lambda T}-1} \leq \sqrt{\mu} \leq \frac{1}{e^{\lambda(T-1)}}.$$

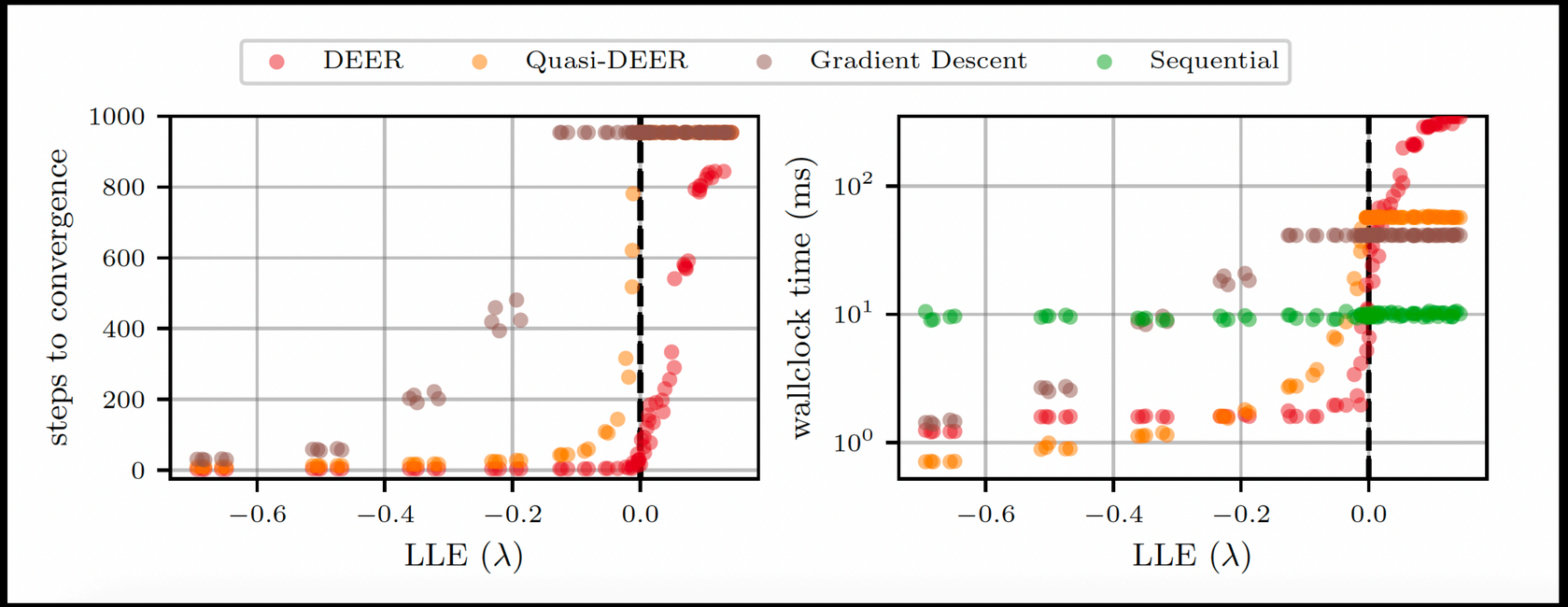
Proof Sketch

- $\sigma_{\min}(\mathbf{J}) = 1/\sigma_{\max}(\mathbf{J}^{-1})$
- In the setting where $T = 4$,

$$\mathbf{J}^{-1} = \begin{pmatrix} I_D & 0 & 0 & 0 \\ J_2 & I_D & 0 & 0 \\ J_3 J_2 & J_3 & I_D & 0 \\ J_4 J_3 J_2 & J_4 J_3 & J_4 & I_D \end{pmatrix}$$

Empirical Evidence in a Parametric Family of RNNs





Summary

1. Markov processes (nonlinear RNNs, Markov chain Monte Carlo, sequential evaluation of transformer blocks, …) are everywhere in machine learning, and **DEER can parallelize them**

2. Use **quasi-Newton** approaches at **scale**

3. The ungulates work for **stable (contractive) systems.** If you have a stable dynamical system, you should parallelize it. Also, you should *design* systems to be stable if you want to parallelize them.